On the Computation of Approximations of Database Queries

Flavio A. Ferrarotti **José M. Turull-Torres**

Information Science Research Centre
Department of Information Systems
Massey University
PO Box 756, Wellington, New Zealand
Email: [J.M.Turull|F.A.Ferrarotti]@massey.ac.nz

Abstract

Reflective Relational Machines were introduced by S. Abiteboul, C. Papadimitriou and V. Vianu in 1994, as variations of Turing machines which are suitable for the computation of queries to relational databases. The machines are equipped with a relational store which can be accessed by means of dynamically built first order logic queries. We initiate a study on approximations of computable queries, defining, for every natural k, the k-approximation of an arbitrary computable query q. This, in turn, motivates us to define a new variation of Reflective Relational Machines by considering two different logics to express dynamic queries: one for queries and a possibly different one for updates to the relational store. We prove several results relating k-approximations of queries with the new machines, and also with classes of queries defined in terms of preservation of equality of FO^k theories. Finally, we summarize a few open problems related to our work.

Keywords: query computability, reflective machines, query approximations.

1 Introduction

In 1980, A. Chandra and D. Harel initiated the construction of a Computability Theory for queries to relational databases (Chandra A. K., Harel D. 1980). They came out with a formalization of the notion of *computable query*, and also with a formal programming language QL which characterized exactly the class of computable queries, denoted as $\mathcal{CQ}$. As to the classical computability theory for functions defined in the set of natural numbers (as in (Hopcroft J., Ullman J. 1979)) it was shown to be inadequate for the purpose of studying computable queries. The reason is that from a conceptual point of view it is desirable for a model of computation of queries to be *representation independent* (Ullman J. D. 1988). This means, roughly, that queries to databases which represent the "same" reality should evaluate to the same result. This concept was partially captured in the definition of computable query by using the mathematical concept of isomorphism. Specifically, the definition of computable query requires that queries on isomorphic databases must evaluate to the same result. Hence, when using a Turing machine to compute a query, the resulting relations computed by the machine on isomorphic databases should be isomorphic. And Turing machines are not defined in that way. Turing machines are *not* supposed to preserve isomorphisms, since they are built to compute functions on natural numbers, rather than structures, i.e., relational databases. So, another kind of formal machine was needed as a model of query computation.

By now, there are different formalisms which may be used to compute, or express, queries. Among the most important we can include formal programming languages, formal machines and logics with finite interpretations. In the present work we will concentrate our attention in formal machines, specifically in the *reflective relational machine* model and some variations of it.

The *reflective relational machine* (RRM from now on) was developed in (Abiteboul S., Papadimitriou C., Vianu V. 1994) as an extension of the relational machine introduced in (Abiteboul S., Vianu V. 1991), where they have been called *loosely coupled generic machines* and denoted as GM^{loose}. They have been further studied in (Abiteboul S., Vianu V. 1995) and (Abiteboul S., Vardi M. Y., Vianu V. 1995), where they were renamed as *relational machines* (RM from now on), and also in (Abiteboul S., Vardi M. Y., Vianu V. 1997) and (Abiteboul S., Papadimitriou C., Vianu V. 1998).

Both models are based on Turing machines interacting with a database using First-Order queries. RMs are similar in spirit to the computation carried out in practical languages such as C plus SQL, while the $RRMs$ allow dynamic generation of queries in the style of reflective programming languages like Prolog or Lisp. In such languages, clauses or functions can be generated and evaluated in execution time, respectively.

Several properties of relational databases which are related to the notion of homogeneity in classical Model Theory were defined an studied in (Turull Torres J. M. 2001). Such properties demonstrated to be useful as a means to increase the computational power of sub-classes of the $RRMs$ of bounded variable complexity (which are known to be incomplete). In the same paper the classes $\mathcal{QCQ}^k$ of computable queries were defined. They provide a semantic characterization of the computation power of sub-classes of RRM defined by limiting the number of variables in the First-Order queries used by the machine to interact with the database.

In (Turull Torres J. M. 2002) a variation of the reflective relational machine called *reflective counting machine* (RCM) was defined together with a characterization of its expressive power denoted as $\mathcal{QCQ}^{C^\omega}$.

Both semantic classifications provide an alternative measure of the computability power of these formal machines. They also have been studied in a different setting in the context of query optimization (see (Link S., Turull Torres J. M. 2003)). They used the fact that by definition, the answer of any query in

the classes $\mathcal{QCQ}^k$ and $\mathcal{QCQ}^{C^k}$, for every $k \geq 1$, when evaluated on an arbitrary database, is the union of some equivalence classes in the relation of equality of FO^k and C^k types, respectively. Roughly, FO^k is the fragment of first order logic (FO) of the formulas with up to k different variables, and C^k is FO^k plus counting quantifiers of the form $\exists^{\geq m}x$ for all m. What makes these classes interesting in the context of query optimization is that such equivalence relations can be computed efficiently, since it is known (Otto M. 1997) that equality of FO^k types and of C^k types is decidable in $PTIME$. The concept of type, in the Model Theoretic sense, is central in this setting, and in the present work we make heavy use of it. In Subsection 2.3 we give the needed background on types, as well as a more intuitive introduction to that notion.

Aiming to initiate a study on approximations of computable queries, we define in this article the k-approximation of an arbitrary computable query q, as the closure under equality of FO^k types of the actual query q. That is, if $q(I)$ is the answer to a query q on a database instance I, then the answer to the k-approximation of q on I (denoted as $q_k(I)$) will contain not only the tuples in $q(I)$ but also those tuples which have the same properties (up to those expressible in FO^k) on I, as the tuples in $q(I)$.

The intuition behind this notion is that we think in computing a query in such a way that the database can be appreciated in all its details (i.e., we can see all its FO types, or isomorphism types), but our capability to generate the answer to the query is not precise, in the sense that we cannot distinguish among tuples beyond FO^k.

Intuitively, it could seem that for every k the k-approximations to all computable queries are in the class $\mathcal{QCQ}^k$, but we prove here that this is not the case. That is, the classes $\mathcal{QCQ}^k$ are not suitable to characterize k-approximations of queries.

This, in turn, motivates us to define a variation of RRM by considering two different logics to express dynamic queries: one for Boolean queries ($\mathcal{L}_q$) and a possibly different one for r-ary queries, or updates ($\mathcal{L}_u$). We denote this new machine as $RRM(\mathcal{L}_q, \mathcal{L}_u)$.

We prove that for every $k \geq 1$, $RRM(FO, FO^k)$ includes the class of k-approximations to all computable queries. We still do not know, though, whether the inclusion is strict. We also study the relation between the RRM machines with variable complexity k and $RRM(FO, FO^k)$, and we give some initial results (see Propositions 7 and 8).

Although the concept of query approximation has been previously studied in fields as decision support systems, OLAP and data mining, the approach usually taken in such fields has been quite different to the approach proposed here. In those fields, the emphasis has been on the problem of generating fast approximate answers to queries posed to large databases and the methods more frequently used are statistical or histogram based (see for instance (Ioannidis Y., Poosala V. 1999, Pavlov D., Mannila H., Smyth P. 2001)). In contrast, in this work the emphasis is on the formal definition and study of the concept of approximation to computable queries.

Another characteristic of the concept of approximation used in such fields as data mining is that the error of an approximate answer is measured in terms of metric or quasi-metric distances. Such distances do not take into account the properties of the database.

As we use for our purpose a concept of similarity based in equality of FO^k types, in this setting an approximation of a query is measured in a semantic way, taking into account the properties of the tuples in the database. An interesting consequence of this is that by choosing different bounds in the amount of variables allowed, it is possible to graduate the precision of the approximation to a given query. Furthermore such precision measure is determined by the properties of the tuples in the database.

We believe that the definition of query approximation presented in this paper can also be useful in areas as artificial intelligence or data mining whenever a "fuzzy" answer rather than an exactly answer is pursued.

Now we proceed to outline the organization of the paper.

In Section 2 we give a brief background of the main technical concepts regarding computable queries, and Finite Model Theory and its relation to Database Theory. We also introduce some known results which we use in our present work.

In Section 3, we include the formal definitions of relational machine and reflective relational machine as well as a brief and more intuitive explanation. We also include there two different *semantic* classifications of computable queries (i.e., classifications defined in terms of properties of queries as functions) which have been defined and studied in (Turull Torres J. M. 2001) and (Turull Torres J. M. 2002).

We define in Section 4 our concept of approximation of computable queries as well as a new variation of RRM intended as a syntactic characterization of k-approximations. Initial results concerning this new approach to query approximation and its relation to the classes $\mathcal{QCQ}^k$ are presented in this section.

In Section 5, we summarize a few open problems related to our work on semantic characterizations of $RRMs$ and query approximations.

Finally, in Section 6, we outline some directions we are aiming to follow in our research programme with respect to query approximations.

2 Preliminaries

Unless otherwise stated, in the present article we will follow the usual notation in Finite Model Theory, as in (Ebbinghaus H., Flum J. 1999).

We use ω to denote the set on natural numbers, as it is usual in Model Theory.

We define a *relational database schema*, or simply *schema*, as a set of relation symbols with associated arities. We don't allow constraints in the schema, and we don't allow constant symbols either. If $\sigma = \langle R_1, \ldots, R_s \rangle$ is a schema with arities $r_1, \ldots, r_s$, respectively, a *database instance*, or simply *database* over the schema σ is a structure $I = \langle D^I, R_1^I, \ldots, R_s^I \rangle$ where D^I is a finite set, which contains exactly all the elements of the database, and for $1 \leq i \leq s$, R_i^I is a relation of arity r_i, i.e., $R_i^I \subseteq (D^I)^{r_i}$.

When we use graphs as examples of databases, we will refer to the *schema of the graphs*. By that we mean a schema with only one relation symbol (besides the symbol for equality), and where that relation symbol is of arity 2.

We will use $\simeq$ to denote isomorphism. That is, if I and J are two databases of some schema σ, we denote as $I \simeq J$ the existence of an *isomorphism* between I and J, i.e., a bijection which preserves all the relations in σ. An *automorphism* in I is a bijection from $dom(I)$ onto $dom(I)$ which preserves all the relations in σ.

We will denote as $\mathcal{B}_\sigma$ the class of all the databases of schema σ.

2.1 Computable Queries

Following (Chandra A. K., Harel D. 1980) we next define the formal notion of *computable query*.

Definition 1 *Fixing a schema σ and an integer $r > 0$, we define a* computable query of arity r and schema σ, *as a partial function $q : \mathcal{B}_\sigma \to \mathcal{B}_{\langle R \rangle}$, where R is a relation symbol of arity r, and where q has the following properties:*

i. For every database I of schema σ on which q is defined, $dom(q(I)) \subseteq dom(I)$.

ii. q is a partial recursive function for some linear encoding of the input databases.

iii. q preserves isomorphisms, i.e., for every pair of databases of schema σ, I and J, and for every isomorphism $h : dom(I) \longrightarrow dom(J)$, either $q(J) = h(q(I))$, or q is undefined on both I and J.

A *computable Boolean query* is a 0-ary computable query.

Note that a query may be *partial*, that is, it may be defined on a proper sub-class of $\mathcal{B}_\sigma$. Otherwise, if the query is defined on the whole class of databases of schema σ, we say that it is a *total query*.

We denote the class of computable queries of schema σ as $\mathcal{CQ}_\sigma$. Consequently, $\mathcal{CQ}$ denotes the whole class of computable queries of all schemas.

2.2 Finite Model Theory and Databases

We refer the reader to (Ebbinghaus H., Flum J., Thomas W. 1984) as a source book for mathematical logic, to (Ebbinghaus H., Flum J. 1999) and (Immerman N. 1999) for finite model theory, and (Abiteboul S., Hull R., Vianu V. 1994) for the relation between databases and finite model theory.

We will use the notion of a *logic* in a general sense, without a formal definition, which would be complicated and unnecessary to present our work. As it is usual in this setting, we will regard a logic as a language, that is, as a set of formulas.

We will only consider signatures, or vocabularies, which are purely *relational*, i.e., there are no constant symbols nor function symbols. We will assume always that the signature includes a symbol for equality.

We consider *only finite* structures. So that our structures will always be *finite relational structures*. Consequently, if $\mathcal{L}$ is a logic, the notions of *satisfaction*, denoted as $\models_\mathcal{L}$ and *equivalence* between sentences, denoted as $\equiv_\mathcal{L}$ will be related to only finite structures.

If $\mathcal{L}$ is a logic, and σ is a signature, we will denote as $\mathcal{L}_\sigma$ the class of formulas from $\mathcal{L}$ with signature σ. If I is a structure of signature σ, or *σ-structure*, we define the *$\mathcal{L}$ theory of* I, as $Th_\mathcal{L}(I) = \{\varphi \in \mathcal{L}_\sigma : I \models_\mathcal{L} \varphi\}$. That is, $Th_\mathcal{L}(I)$ is the class of all Boolean queries expressible in the logic $\mathcal{L}$ which are TRUE on I.

A *database schema* will be regarded as a *relational signature*, and a *database instance* of some schema σ, as a finite and relational *σ-structure*. Consequently, the class of all finite σ-structures is exactly what we denoted as $\mathcal{B}_\sigma$ at the beginning of this section.

By $\varphi(x_1, \ldots, x_r)$ we denote a formula of some logic, whose free variables are *exactly* $\{x_1, \ldots, x_r\}$. Let $free(\varphi)$ be the set of free variables of the formula φ. If $\varphi(x_1, \ldots, x_k) \in \mathcal{L}_\sigma$, $I \in \mathcal{B}_\sigma$, $\bar{a}_k = (a_1, \ldots, a_k)$ is a k-tuple over I, let $I \models \varphi(x_1, \ldots, x_k)[a_1, \ldots, a_k]$ denote that φ is TRUE, when interpreted by I, under a valuation v, where for $1 \leq i \leq k$, $v(x_i) = a_i$. That is, we consider the tuple $\bar{a}_k$ as a (partial) *valuation*. We will use the same notation also when the set of free variables of the formula is *strictly* included in $\{x_1, \ldots, x_k\}$, even when the formula is a sentence.

Then we can consider the set of all such valuations, as follows:

$$\varphi^I = \{(a_1, \ldots, a_k) : a_1, \ldots, a_k \in dom(I) \wedge$$
$$I \models \varphi(x_1, \ldots, x_k)[a_1, \ldots, a_k]\}$$

That is, φ^I is the relation defined by φ in the structure I, and its arity is given by the number of free variables in φ.

Formally, we say that a formula $\varphi(x_1, \ldots, x_k)$ of signature σ, *expresses* a query q of schema σ, if for every database I of schema σ, is $q(I) = \varphi^I$. Similarly, a sentence φ expresses a Boolean query q if for every database I of schema σ, is $q(I) = 1$ iff $I \models \varphi$.

Note that when using logics as query languages only *total* queries are considered.

We denote as FO^k, for some integer $k > 0$, the fragment of first order logic where only formulas whose variables are in $\{x_1, \ldots, x_k\}$ are allowed. In this setting, FO^k is itself a logic. This logic is clearly *less expressive* than FO, i.e., the class of queries which can be expressed in FO^k is strictly included in the class of queries which can be expressed in FO.

As an example, recall that a *k-clique*, denoted as K_k, is a complete graph with k nodes. The Boolean query, *"is the graph a $(k+1)$-clique?"*, in the schema of graphs, is not expressible in FO^k (see (Immerman N. 1999)). On the other hand the query is easily expressible in FO^{k+1}.

We denote as C^k, for some integer $k > 0$, the logic which we get by adding to FO^k *counting quantifiers*, i.e., all the existential quantifiers of the form $\exists^{\geq m} x$, for every $m \geq 1$. Informally, the semantics of $\exists^{\geq m} x(\varphi)$ is that there exists at least m *different* elements in the structure or database, which satisfy φ.

2.3 Types, or Properties of tuples

Given a database I and a k-tuple over I, $\bar{a}_k = (a_1, \ldots, a_k)$, we want to consider *all* the properties of $\bar{a}_k$ in the database. One possible such property is whether the tuple belongs to a given k-ary relation in I. But there are many more properties that $\bar{a}_k$ may satisfy, given the components of the tuple and the relations which form the database. Different sub-tuples of $\bar{a}_k$ may belong to relations of arity $< k$; two different components of $\bar{a}_k$ may belong to two different tuples in some relation, which in turn have some common components, and so on. As we want to consider *all* such properties, we can use a logic, say FO, and see which properties of the tuple can be expressed by formulas in that logic. Then we can ask ourselves whether the chosen logic is "complete" in this sense, that is, whether all the properties which we want to consider are expressible through that logic. Note that we should not consider only formulas with exactly k free variables, because we want to include also the properties of the different components of the tuple, as well as of the different sub-tuples.

It turns out that there is a very interesting formal concept in model theory which is quite suitable for our purpose, namely, the notion of *type*. We refer the interested reader to (Dawar, A. 1993), (Dawar, A., Lindell, S., Weinstein, S. 1995) and (Otto M. 1997) where the subject of types is studied in depth. In this sub-section we will usually follow the notation as in (Otto M. 1997).

Definition 2 *Let $\mathcal{L}$ be a logic. Let I be a database of some schema σ, let $\bar{a}_k = (a_1, \ldots, a_k)$ be a k-tuple over I. The* $\mathcal{L}$ type *of $\bar{a}_k$ in I, denoted $tp_I^{\mathcal{L}}(\bar{a}_k)$, is the set of formulas in $\mathcal{L}_\sigma$ with free variables* among *$\{x_1, \ldots, x_k\}$, such that every formula in the set is TRUE when interpreted by I for any valuation which assigns the i-th component of $\bar{a}_k$ to the variable x_i, for every $1 \leq i \leq k$. In symbols $tp_I^{\mathcal{L}}(\bar{a}_k) = \{\varphi \in \mathcal{L}_\sigma : free(\varphi) \subseteq \{x_1, \ldots, x_k\} \wedge I \models \varphi[a_1, \ldots, a_k]\}$*

It is note worthy that, according this definition, the $\mathcal{L}$ *theory* of the database I, i.e., $Th_{\mathcal{L}}(I)$, is included in the $\mathcal{L}$ type of *every* tuple over I. That is, the class of all the properties of a given tuple, which are expressible in $\mathcal{L}$, includes not only the properties of all its sub-tuples, but also the properties of the database itself.

Note that the type of two different k-tuples of the same database may be different, even if the types of the corresponding components are the same. Think of a complete binary tree of depth h. If we consider types for single elements (i.e., $k = 1$), then we have only $h+1$ different types, because all the nodes of the same depth satisfy the same properties. They can be exchanged by an automorphism of the tree. Now let us consider types for pairs ($k = 2$). We can build two pairs, such that the type of the two first components is the same, and the type of the two second components is also the same, but the types of the two pairs are different. Just take for one pair a node in some depth > 0 and one of its sons, and for the other pair we take a node of the same depth as the first component of the first pair, and another node in the next level of the tree, but which is *not* the son of the first component.

Considering the logic $\mathcal{L}$ as a query language, we may also regard an $\mathcal{L}$ type as a *set of queries*. The $\mathcal{L}$ type of a tuple $\bar{a}_k$ over a database I is the set of all l-ary queries, with $0 \leq l \leq k$, which are expressible in $\mathcal{L}$, and such that the tuple $\bar{a}_k$ (or the corresponding sub-tuple, according to the free variables in the corresponding formula) belongs to the answer of all of them, when evaluated on the database I. Moreover, the *(infinitary) conjunction* of all the $\mathcal{L}$ formulas which form the $\mathcal{L}$ type of $\bar{a}_k$, is itself a query, though not necessarily expressible in $\mathcal{L}$. The answer to this query when evaluated on I is the set of k-tuples over I whose $\mathcal{L}$ type is $tp_I^{\mathcal{L}}(\bar{a}_k)$.

We can also think of a type of a tuple, *without having a particular database in mind*, just by adding properties (formulas with the appropriate free variables) to a set as long as it remains consistent. That is, we can first define a type for a given schema and a given length of tuple, and then ask whether in a given database there is a tuple of that length and of that type. This is the idea behind the next definition. First, let us denote as $Tp^{\mathcal{L}}(\sigma, k)$, for some $k > 0$, the class of *all* $\mathcal{L}$ types for k-tuples over databases of schema σ. In symbols

$$Tp^{\mathcal{L}}(\sigma, k) = \{tp_I^{\mathcal{L}}(\bar{a}_k) : I \in \mathcal{B}_\sigma \wedge \bar{a}_k \in (dom(I))^k\}$$

That is, $Tp^{\mathcal{L}}(\sigma, k)$ is a class of properties, or a set of sets of formulas.

Definition 3 *Let $\mathcal{L}$ be a logic. Let σ be a schema, let $k > 0$, and let $\alpha \in Tp^{\mathcal{L}}(\sigma, k)$ (i.e., α is the $\mathcal{L}$ type of some k-tuple over some database in $\mathcal{B}_\sigma$). Let I be a database of schema σ. We say that I* realizes *the type α if there is a k-tuple $\bar{a}_k$ over I whose $\mathcal{L}$ type is α. That is, if $tp_I^{\mathcal{L}}(\bar{a}_k) = \alpha$.*

Let I be a database of some schema σ, and let $k > 0$. We denote as $Tp^{\mathcal{L}}(I, k)$ the class of the properties of all the k-tuples over the database I, which can be expressed in $\mathcal{L}$. In symbols

$$Tp^{\mathcal{L}}(I, k) = \{tp_I^{\mathcal{L}}(\bar{a}_k) : \bar{a}_k \in (dom(I))^k\}$$

The next is a well known result, and means that when two databases realize the same (FO^k or C^k) types for tuples, it does not mind which length of tuples we consider; for proofs see among others (Turull Torres J. M. 2001) and (Turull Torres J. M. 2002).

Fact 1 *For every $k \geq 1$, for every schema σ, for every pair of databases I, J of schema σ, and for every $1 \leq l \leq k$, the following holds:*

- $Tp^{FO^k}(I, k) = Tp^{FO^k}(J, k) \iff Tp^{FO^k}(I, l) = Tp^{FO^k}(J, l)$
- $Tp^{C^k}(I, k) = Tp^{C^k}(J, k) \iff Tp^{C^k}(I, l) = Tp^{C^k}(J, l)$

The relation between types of tuples and theories of databases for the logics FO^k and C^k is stated in the following fact, which among other sources, can be found in (Otto M. 1997).

Fact 2 *For every $k > 0$, for every schema σ, and for every pair of databases I, J of schema σ, the following holds: $I \equiv_{FO^k} J \iff Tp^{FO^k}(I, k) = Tp^{FO^k}(J, k)$. Similarly, for every $k > 0$, for every schema σ, and for every pair of databases I, J of schema σ, the following holds: $I \equiv_{C^k} J \iff Tp^{C^k}(I, k) = Tp^{C^k}(J, k)$.*

Although types are infinite sets of formulas, due to the next results of A. Dawar and of M. Otto, respectively, a *single* FO^k (C^k) formula is equivalent to the FO^k (C^k) type of a tuple over a given database. And this equivalence holds for all the databases of the same schema.

Proposition 1 *(Dawar, A., Lindell, S., Weinstein, S. 1995) For every schema σ, for every database I of schema σ, for every $k > 0$, for every $1 \leq l \leq k$, and for every l-tuple $\bar{a}_l$ over I, there is an FO^k formula $\phi \in tp_I^{FO^k}(\bar{a}_l)$ such that for any database J of schema σ and for every l-tuple $\bar{b}_l$ over J*

$$J \models \phi[\bar{b}_l] \iff tp_I^{FO^k}(\bar{a}_l) = tp_J^{FO^k}(\bar{b}_l)$$

Moreover, such a formula ϕ can be built inductively for a given database. If an FO^k formula ϕ satisfies the condition of Proposition 1, we say that ϕ is an *isolating formula* for $tp_I^{FO^k}(\bar{a}_l)$.

Similarly, a C^k formula is equivalent to the C^k type of a tuple over a given database.

Proposition 2 *(Otto M. 1996) For every schema σ, for every database I of schema σ, for every $k \geq 1$, for every $1 \leq l \leq k$, and for every l-tuple $\bar{a}_l$ over I, there is a C^k formula $\chi \in tp_I^{C^k}(\bar{a}_l)$ such that for any database J of schema σ and for every l-tuple $\bar{b}_l$ over J*

$$J \models \chi[\bar{b}_l] \iff tp_I^{C^k}(\bar{a}_l) = tp_J^{C^k}(\bar{b}_l)$$

Again, such a formula χ can be built inductively for a given database. If a C^k formula χ satisfies the condition of Proposition 2, we say that χ is an *isolating formula* for $tp_I^{C^k}(\bar{a}_l)$.

Given a logic $\mathcal{L}$, and a database I, it may be the case that $\mathcal{L}$ is not powerful enough to express *all* the properties of *every* tuple over I. That is, two tuples

over the database may differ in some property, which is not expressible in the logic $\mathcal{L}$.

In order to study in a formal setting whether a given logic is expressive enough regarding types, we need to define types considering the *algebraic* properties of the tuples in a given database, i.e., having no logic in mind.

Definition 4 *Let σ be a schema, let I be a database of schema σ, let $k > 0$, and let $\bar{a}_k = (a_1, \ldots, a_k)$ be a k-tuple over I. Let $\mathcal{L}$ be a logic. We say that the type $tp_I^{\mathcal{L}}(\bar{a}_k)$ is an* automorphism type *if for every k-tuple $\bar{b}_k = (b_1, \ldots, b_k)$ over I, if $tp_I^{\mathcal{L}}(\bar{a}_k) = tp_I^{\mathcal{L}}(\bar{b}_k)$, then there exists an automorphism f in the database I which maps $\bar{a}_k$ onto $\bar{b}_k$, i.e., for $1 \leq i \leq k$, $f(a_i) = b_i$.*

Thus, regarding the tuple $\bar{a}_k$ in the database I, the logic $\mathcal{L}$ is sufficiently expressive as to include all the properties which might make $\bar{a}_k$ distinguishable from other k-tuples in the database I.

We now give a stronger definition, which states that a given type is not only expressive enough in the given database, but also in the *whole* class $\mathcal{B}_\sigma$ of all the databases of the same schema.

Definition 5 *Let σ be a schema, let I be a database of schema σ, let $k > 0$, and let $\bar{a}_k = (a_1, \ldots, a_k)$ be a k-tuple over I. Let $\mathcal{L}$ be a logic. We say that the type $tp_I^{\mathcal{L}}(\bar{a}_k)$ is an* isomorphism type *if for every database $J \in \mathcal{B}_\sigma$, and for every k-tuple $\bar{b}_k = (b_1, \ldots, b_k)$ over J, if $tp_I^{\mathcal{L}}(\bar{a}_k) = tp_J^{\mathcal{L}}(\bar{b}_k)$, then there exists an isomorphism $f : dom(I) \to dom(J)$ which maps $\bar{a}_k$ in I onto $\bar{b}_k$ in J, i.e., for $1 \leq i \leq k$, $f(a_i) = b_i$.*

The following is a well known result, which, among other sources, can be found as Proposition 2.1.1 in (Ebbinghaus H., Flum J. 1999).

Proposition 3 *For every schema σ, and for every pair of (finite) databases I, J of schema σ, the following holds: $I \equiv_{FO} J \iff I \simeq J$.*

Moreover, the FO theory of every database is equivalent to a single FO^{n+1} sentence where n is the size of the database.

This means that, given a database, FO is expressive enough to describe the database up to isomorphism, i.e., there exists a sentence $\Delta_I \in FO$ such that $MOD(\Delta_I) = \{J \in \mathcal{B}_\sigma : J \backsimeq I\}$. Such Δ sentence is usually called the diagram of the database.

Finally, we define two equivalence relations.

Definition 6 *Let k, l be positive integers, such that $k \geq l \geq 1$. We denote by $\equiv_k$ the (equivalence) relation induced in the set of l-tuples over a given database I, by the equality in the FO^k types of the l-tuples. That is, for every pair of l-tuples $\bar{a}_l$ and $\bar{b}_l$ over I, $\bar{a}_l \equiv_k \bar{b}_l$ iff $tp_I^{FO^k}(\bar{a}_l) = tp_I^{FO^k}(\bar{b}_l)$. Similarly, we denote by $\equiv_{C^k}$ the (equivalence) relation induced in the set of l-tuples over a given database I, by the equality in the C^k types of the l-tuples. That is, for every pair of l-tuples $\bar{a}_l$ and $\bar{b}_l$ over I, $\bar{a}_l \equiv_{C^k} \bar{b}_l$ iff $tp_I^{C^k}(\bar{a}_l) = tp_I^{C^k}(\bar{b}_l)$.*

3 Machines for Computing Queries

FO provides a rich class of database queries. However, some plausible queries are not FO expressible. Aho and Ullman (Aho A. V., Ullman J. D. 1979) proved that the transitive closure query cannot be expressed in relational algebra and so neither in FO.

In this section and the next, we will consider more powerful languages which model arbitrary computations interacting with a database using a finite set of FO queries. Such computations are modelled by a formal machine, the *Loosely Coupled Generic Machine* (GM^{loose}) introduced in (Abiteboul S., Vianu V. 1991), today renamed as *relational machine* (RM).

A RM consists of a TM augmented with a finite set of fixed-arity relations forming a *relational store* (rs). Designated relations contain initially the input database, and one specific relation holds the output at the end of the computation. In a transition the rs can be modified through a FO query. Thus, RM provides arbitrary computation interacting with the database through FO queries. RMs can be thought of as database application programs, which often use a relational language like SQL embedded in a programming language like C. The TM part of the RM corresponds to the use of a programming language and the FO queries on the rs correspond to the SQL queries.

Formally, a RM is an eleven-tuple

$$M = \langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \sigma, \tau, T, \Omega, \Phi \rangle$$

where:

1. $Q, \Sigma, q_0, \mathfrak{b}$ and F are defined as in the TM model;
2. σ is the vocabulary of the rs;
3. $\tau \subseteq \sigma$ is the vocabulary of the input structure;
4. $T \in \sigma$ is the output relation;
5. Ω is a finite set of FO sentences of vocabulary σ;
6. Φ is a finite set of FO formulas with free variables of vocabulary σ, and
7. $\delta : Q \times \Sigma \times \Omega \to \Sigma \times Q \times \{R, L\} \times \Phi \times \sigma$ is the transition function of M.

Note that at each step of a computation the contents of the rs is a σ-structure, and that the query computed by the machine is of the form $q : \mathcal{B}_\tau \to \mathcal{B}_{\langle T \rangle}$.

An *instantaneous description* of a RM consists of an ID of the TM component, together with the current content of the relational store. Note that transitions are based on:

i. the current state,

ii. the contents of the current tape cell, and

iii. the answer to a *yes/no* FO query on the store.

Let I_1 be the σ-structure which is the contents of the rs. Each transition involves the following actions:

i. move right (R) or left (L) the head on the tape,

ii. overwrite the current tape cell with some tape symbol,

iii. change state, and

iv. update the relational store so that the new contents of it is the structure I_2 obtained from I_1 by changing the contents of one relation in σ according to the result of the evaluation of one FO formula in Φ on the structure I_1 (denoted from now on by $R \leftarrow \varphi$, where $R \in \sigma$ and $\varphi \in \Phi$).

The machine starts in the start state, with the input in the designated relations of the relational store, and an empty tape. It halts when a halting state is reached.

The *arity* of a RM is the maximum number of variables used in its FO queries.

It is easily seen that the relational machines define only *generic mappings* from the input structure to the output relation, i.e., all RM computations preserve isomorphisms.

The RM model is strictly more powerful than FO. Relational machines can compute transitive closure queries, as shown in the following example.

Example 1 *"Set of pairs (a, b) of cities such that b can be reached from a with an arbitrary number of stopovers".*

Let M be the following RM where $\sigma = \langle F, T, S\rangle$ is the vocabulary of the rs*, $\tau = \langle F\rangle$ is the vocabulary of the input structure and T is the output relation.*

1. $T \leftarrow F(x, y)$; $S \leftarrow F(x, y)$;
2. $T \leftarrow T(x, y) \vee \exists z(T(x, z) \wedge F(z, y))$;
3. *If* $\neg(\forall x, y(T(x, y) \leftrightarrow S(x, y))$ *then*
   $S \leftarrow T(x, y)$
   Goto step 2;
4. *Halt.*

When M halts, the result of the query is the contents of the relation T in rs*. Note that the machine does the composition of F with itself until no new pair of cities is added to T.*

Relational machines are restricted to use a fixed set of queries. In particular it implies a constant bound on the number of variables, a fact with important consequences in the expressive power.

Nowadays database programming environments do away with this limitation, and allow *programmable queries* constructed *dynamically* by the program, and not explicitly by the programmer. RM as defined previously fail to model this *reflective* style of database programming.

In (Abiteboul S., Papadimitriou C., Vianu V. 1994) the relational machines are extended so that they interact with the relational store via a writable query tape. Such machines are called reflective relational machines (RRM).

A RRM M is a RM extended with a *writable query tape* used to interact with the *relational store* (rs). The rs is capable of storing an unbounded number of arbitrary relations $R_0, R_1, R_2, \ldots$. Initially, the relational store contains only the *input relations* $R_0, \ldots, R_n$ whose arities $r_0, \ldots, r_n$ depends only on M, all tapes are empty and the machine is in the initial state. M computes as an ordinary two tape Turing machine, with the following unique exception: when the "query state" is reached, then the contents of the query tape is interpreted as a formula. Two kinds of formulas are allowed.

1. yes-no queries of the form

   $$\varphi(R_{i_1}, \ldots, R_{i_k})$$

   where $i_1, \ldots, i_k$ are natural numbers, and φ is a FO sentence over the vocabulary $\langle R_{i_1}, \ldots, R_{i_k}\rangle$. The query is evaluated *in a single step* on the relational store. Depending on the answer, the machines moves to one of the particular states *yes*, *no*.

2. Updates of the form

   $$R_j \leftarrow \varphi(R_{i_1}, \ldots, R_{i_k})$$

   where $j, i_1, \ldots, i_k$ are natural numbers, and φ is a FO formula with free variables over the vocabulary $\langle R_{i_1}, \ldots, R_{i_k}\rangle$. The result of query φ is assigned to the relation R_j. The machine moves to a particular state, say *update done*.

If R_j does not yet exist in the rs, it is added to the store.

When some error during the evaluation of a query posed on query tape occurs the machine halts and rejects.

The *variable complexity* of a reflective machine M is the largest number of variables ever used in a query.

It is immediate that reflective relational machines are more powerful than relational machines. There is a RRM M which computes the query

$$\text{"}even(S) = \textbf{true} \text{ iff } |S| \text{ is even"}$$

on a unary relation S. M goes on asking queries of the form

$$\exists x_1, \ldots, x_m \left(\bigwedge_{1 \le i < j \le m} (x_i \neq x_j) \wedge \bigwedge_{1 \le i \le m} S(x_i) \right)$$

for $m = 1, 2, \ldots$ until it receives a negative answer.

The *variable complexity* of a RRM is the maximum number of variables which may be used in the dynamic queries which are generated by the machine throughout any computation. We will denote as RRM^k, with $k > 0$, the sub-class of reflective relational machines with variable complexity k. That is, machines in RRM^k may use at most k different variables in the FO formulas which are generated in the query tape during any computation. $RRM^{O(1)}$ will denote the sub-class of RRM with *bounded variable complexity*, i.e., RRM whose variable complexity is bounded by some integer. In symbols $RRM^{O(1)} = \bigcup_{k>0} RRM^k$.

In (Abiteboul S., Papadimitriou C., Vianu V. 1998) it was shown that the RRM model is *complete*. However, it was also shown there that the sub-class $RRM^{O(1)}$ is incomplete. Moreover, it is equivalent to the RM model (Abiteboul S., Vardi M. Y., Vianu V. 1997).

In (Turull Torres J. M. 2001) a semantic characterization of the sub-classes of total queries computed by $RRMs$ with variable complexity k ($total(RRM^k)$ from now on), for every natural number k, was studied in order to determine the increment of such subclasses of $RRMs$ working on homogeneous databases.

Such characterization was achieved by defining, for every $k \geq 1$, the class $\mathcal{QCQ}^k \subset total(\mathcal{CQ})$, as the total queries which cannot distinguish between two non isomorphic databases when the properties which make them different need more than k variables to be expressed in FO. The way in which the classes $\mathcal{QCQ}^k$ was defined is to some extent implicit in (Abiteboul S., Vianu V. 1995) and (Abiteboul S., Vardi M. Y., Vianu V. 1995).

Definition 7 *Let σ be a database schema, and let $k \geq 1$ and $k \geq r \geq 0$. Then we define*

$$\mathcal{QCQ}^k_\sigma = \{f^r \in \mathcal{CQ}_\sigma | \forall I, J \in \mathcal{B}_\sigma :$$

$$Tp^{FO^k}(I, k) = Tp^{FO^k}(J, k) \Longrightarrow$$

$$Tp^{FO^k}(\langle I, f(I)\rangle, k) = Tp^{FO^k}(\langle J, f(J)\rangle, k)\}$$

where $\langle I, f(I)\rangle$ and $\langle J, f(J)\rangle$ are databases of schema $\sigma \cup \{R\}$, with R being a relation symbol of arity r; we also require that $f(I)$ be closed under $\equiv_k$. Further, we define $\mathcal{QCQ}^k = \bigcup_\sigma \mathcal{QCQ}^k_\sigma$, and $\mathcal{QCQ}^\omega = \bigcup_{k\geq 1} \mathcal{QCQ}^k$.

That is, a query is in the sub-class $\mathcal{QCQ}^k$ if for every pair of databases of the corresponding schema, say σ, and for every FO^k type for k-tuples if either both databases have no k-tuple of that type, or both databases have a non empty subset of k-tuples of that type then the relations defined by the query in each database also agree in the same sense, considering the schema of the databases with the addition of a relation symbol with the arity of the query.

Intuitively, the different sub-classes $\mathcal{QCQ}^k$ in the hierarchy $\mathcal{QCQ}^\omega$, correspond to different degrees of "precision", say, with which the queries in the given sub-class *need* to consider the database for them to be evaluated.

The next result states that the classes $\mathcal{QCQ}^k$ form a strict hierarchy inside $\mathcal{CQ}$

Proposition 4 *(Turull Torres J. M. 2001) For every $k \geq 1$, $\mathcal{QCQ}^{k+1} \supset \mathcal{QCQ}^k$.*

As an example of a query which belongs to the sub-class $(\mathcal{QCQ}^{k+1} - \mathcal{QCQ}^k)$, consider the query "*nodes in (k + 1)-cliques which are subgraphs of the input graph*", defined in the schema of the graphs.

Now we give a *syntactic* characterization of the sub-classes $\mathcal{QCQ}^k$, by means of a computation model.

Theorem 1 *(Turull Torres J. M. 2001) For every $k \geq 1$, the class of total queries which are computable by RRM^k machines is exactly the class $\mathcal{QCQ}^k$.*

Corollary 1 $\mathcal{QCQ}^\omega = total(RRM^{O(1)})$.

Corollary 2 $\mathcal{QCQ}^\omega \subset \mathcal{CQ}$.

In (Otto M. 1996), M. Otto defined a new model of computation of queries, inspired in the *RM* of (Abiteboul S., Vianu V. 1995), to characterize the expressive power of fixed point logic with the addition of *counting terms* (see (Immerman N. 1999)). The finite set of operations which the machine of Otto can perform includes the capability to count, where the parameter m of the counting quantifier $\exists^{\geq m}$ is dynamically defined.

Next, we define a machine introduced in (Turull Torres J. M. 2002) which is similar to the model of Otto, but which is inspired in the *RRM* of (Abiteboul S., Papadimitriou C., Vianu V. 1998), instead.

Definition 8 *We define, for every $k \geq 1$, the* reflective counting machine of variable complexity k, *which we denote by RCM^k, as a reflective relational machine RRM^k, where dynamic queries are C^k formulas, instead of FO^k formulas. In all other aspects, our model works in exactly the same way as RRM^k. We define $RCM^{O(1)} = \bigcup_{k\geq 1} RCM^k$.*

In the same spirit than in previous section, we present now a semantic characterization of the sub-classes of $total(RCM^k)$ for every natural number k. The hierarchy of sub-classes that we consider for this purpose was originally defined in (Turull Torres J. M. 2002).

Definition 9 *Let σ be a database schema, and let $k \geq 1$ and $k \geq r \geq 0$. Then we define*

$$\mathcal{QCQ}_\sigma^{C^k} = \{f^r \in \mathcal{CQ}_\sigma | \forall I, J \in \mathcal{B}_\sigma :$$

$$Tp^{C^k}(I,k) = Tp^{C^k}(J,k) \Longrightarrow$$

$$Tp^{C^k}(\langle I, f(I)\rangle, k) = Tp^{C^k}(\langle J, f(J)\rangle, k)\}$$

where $\langle I, f(I)\rangle$ and $\langle J, f(J)\rangle$ are databases of schema $\sigma \cup \{R\}$, with R being a relation symbol of arity r; we also require that $f(I)$ be closed under $\equiv_{C^k}$. We define, further $\mathcal{QCQ}^{C^k} = \bigcup_\sigma \mathcal{QCQ}_\sigma^{C^k}$, and $\mathcal{QCQ}^{C^\omega} = \bigcup_{k\geq 1} \mathcal{QCQ}^{C^k}$.

Proposition 5 *(Turull Torres J. M. 2002) For every $k \geq 1$, there is some $h > k$ such that $\mathcal{QCQ}^{C^h} \supset \mathcal{QCQ}^{C^k}$.*

Proposition 6 *(Turull Torres J. M. 2002)* $\mathcal{QCQ}^{C^\omega} \subset \mathcal{CQ}$.

Finally, the characterization of the expressive power of the model RCM^k is provided by the next theorem.

Theorem 2 *(Turull Torres J. M. 2002) For every $k \geq 1$, the class of total queries which are computable by RCM^k machines is exactly the class $\mathcal{QCQ}^{C^k}$.*

Corollary 3 $RCM^{O(1)} = \mathcal{QCQ}^{C^\omega}$.

4 Computing Approximations of Queries

In a general setting, we can consider an approximation of a query q as a query q' which defines for every database I a relation $q'(I)$ "close" to $q(I)$.

More precisely, in this section we will consider that q_k is a *k-approximation* of a query q of arity $r \leq k$ iff for every database I, the relation $q_k(I)$ defined by q_k over I includes exactly those r-tuples in $dom(I)^r$ which have the FO^k type of some tuple in $q(I)$.

We present next an example which shows how the concept of k-approximation works.

Example 2 *Think of a query q which, given a tree, asks for those nodes of a given depth d with exactly two sons. It is easy to express such query in FO^4.*

$$\begin{aligned}\varphi(x) \equiv\ & \psi_d(x) \wedge \exists y, z(E(x,y) \wedge E(x,z) \wedge (y \neq z) \wedge \\ & \forall w(E(x,w) \rightarrow ((w = y) \vee (w = z))))\end{aligned}$$

where $\psi_d(x)$ is true iff x is a node of depth d.

$$\begin{aligned}\psi_0(x) &\equiv \forall y(\neg E(y,x)) \\ \psi_{d+1}(x) &\equiv \exists y(\psi_d(y) \wedge E(y,x))\end{aligned}$$

Clearly if we posse such query q over the database (tree) $\mathcal{T}$ of Figure 1 asking for the nodes of depth 2, we will obtain the unary relation $q(\mathcal{T}) = \{a_{21}, a_{24}\}$ as answer.

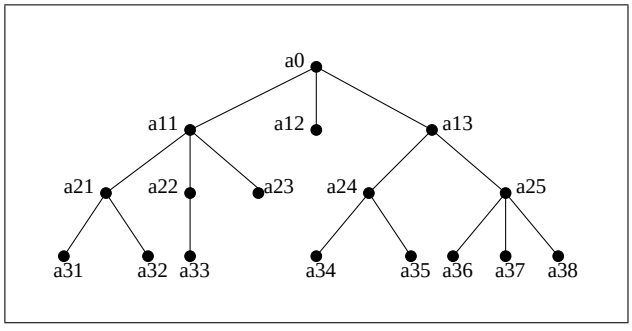

Figure 1: Tree $\mathcal{T}$

Taking into account that at least four variables are required to express that a given node in a tree has exactly two sons, we apply our concept of approximation to this example. It turns out that the q_3 approximation of q includes the node a_{25} which actually has tree sons, i.e., $q_3(\mathcal{T}) = \{a_{21}, a_{24}, a_{25}\}$. Note that a_{21}, a_{24} and a_{25} have the same FO^3 type.

Now, we proceed with the formal definition of our concept of approximation.

Definition 10 *Let q be a total computable query of arity r and of schema σ. Let $k \geq r$. Let $I \in \mathcal{B}_\sigma$. Then we define $q_k(I)$ as follows:*

$$q_k(I) = \{\bar{a} \in dom(I)^r : \exists \bar{b} \in q(I) \wedge \bar{b} \equiv_k \bar{a}\}$$

The following are well known facts (for proofs, among other sources see (Turull Torres J. M. 2001)).

Fact 3 *For every $k \geq 1$ equivalence classes in $\equiv_k$ for k-tuples are unions of equivalence classes in $\equiv_{k+1}$ for k-tuples.*

Fact 4 *For every $k \geq 1$ equivalence classes in $\equiv_k$ for k-tuples are unions of equivalence classes in $\equiv_\simeq$ for k-tuples".*

From Facts 3 and 4, the following result is straightforward

Fact 5 *For every database I, for every query q and for every $k \geq 1$, $q_k(I) \supseteq q_{k+1}(I) \supseteq q(I)$.*

Note that, each time we decrease the index k, we add zero or more tuples to $q_k(I)$ but we never delete a tuple. Conversely, each time we increment k, we get either a better approximation of q or the same approximation as before. If we look at the Example 2 at the beginning of this section, we can appreciate this clearly.

$$q_1(\mathcal{T}) = dom(\mathcal{T}) \supset q_2(\mathcal{T}) = \{a_{21}, a_{22}, a_{24}, a_{25}\} \supset$$

$$q_3(\mathcal{T}) = \{a_{21}, a_{24}, a_{25}\} \supset q_4(\mathcal{T}) = q(\mathcal{T})$$

Intuitively, it could seem that for an arbitrary k and an arbitrary computable query q, the k-approximation of q belongs to the class $\mathcal{QCQ}^k$. But this turns out to be not true. The next result means that the classes $\mathcal{QCQ}^k$ are not suitable to characterize k-approximations of queries.

Proposition 7 *It is not true that for every computable query q and for every k, $q_k \in \mathcal{QCQ}^k$ or equivalent $q_k \in total(RRM^k)$.*

Proof: We construct a counterexample. Let q be a query that is defined as the set of nodes in the biggest clique of a graph that is the disjoint union of two cliques. If the cliques have equal size or the graph is not the disjoint union of two cliques then q is defined as the empty set. Let $G_1 = K_k \dot{\cup} K_{k+1}$ be the disjoint union of a clique of size k with a clique of size $k+1$ and $G_2 = K_k \dot{\cup} K_k$ be the disjoint union of two cliques of size k. It follows that $q(G_1) = V(K_{k+1})$ and $q_k(G_1) = V(G_1)$ while $q(G_2) = \emptyset$ and $q_k(G_2) = \emptyset$, but as $G_1 \equiv_{FO^k} G_2$ it is clear that q_k cannot be computed by any RRM^k and thus $q_k \notin \mathcal{QCQ}^k$.

□

The previous result also has a positive reading. It indicates that we are in presence of probably interesting classes of queries which need further study. Having this in mind, we can think in a different approach to this topic by defining a new model of computation similar to the RRM model but with the addition of an extra tape for updates (the *update tape*) and where the original *query tape* cannot be used for this purpose. Using such schema we can define machines in which the logic used to interact with the *rs* does not need to be the same for both, queries and updates.

Definition 11 *We define, for a given pair $\mathcal{L}_q$ and $\mathcal{L}_u$ of abstract logics, the $RRM(\mathcal{L}_q, \mathcal{L}_u)$ machine, as a reflective relational machine extended with a* writable update tape*. This model works in exactly the same way as RRM with the following important exceptions:*

i when the "query state" is reached, then the contents of the query tape *is interpreted as a formula of the form*

$$\varphi(R_{i_1}, \ldots, R_{i_k})$$

where $i_1, \ldots, i_k$ are natural numbers, and φ is a $\mathcal{L}_q$ sentence *over the vocabulary $\langle R_{i_1}, \ldots, R_{i_k}\rangle$, i.e., φ has no free variables,*

ii when the "update state" is reached, then the contents of the update tape *is interpreted as a formula of the form*

$$R_j \leftarrow \varphi(R_{i_1}, \ldots, R_{i_k})$$

where $j, i_1, \ldots, i_k$ are natural numbers, and φ is a $\mathcal{L}_u$ formula with free variables *over the vocabulary $\langle R_{i_1}, \ldots, R_{i_k}\rangle$. The result of query φ is assigned to the relation R_j.*

For every k, we denote as $RRM^{(k)}$ the class of machines $RRM(FO, FO^k)$. We point out that $RRM^k = RRM(FO^k, FO^k)$ and that $RRM = RRM(FO, FO)$.

Finally, we present two results on the class of queries computed by $RRM^{(k)}$ machines. The first result shows that the new $RRM^{(k)}$ model as defined above is effectively different than the RRM^k model regarding the class of queries computed by each of them, i.e., we are really talking of two different machines.

Proposition 8 $total(RRM^{(k)}) \supset total(RRM^k)$

Proof:

a. To prove ($total(RRM^{(k)}) \supseteq total(RRM^k)$) is trivial, since for every $q \in total(RRM^k)$ and for every database I, $q(I)$ is closed under FO^k types.

b. To prove that $total(RRM^k)$ is strictly included in $total(RRM^{(k)})$, we consider the k-approximations q_k of the query "set of nodes in the biggest clique" defined in the proof of Lemma 7 and we build for every k a $RRM^{(k)}$ machine M_{q_k} which computes q_k. M_{q_k} works as follows:

   1. Determine the size n of the domain of the database. M_{q_k} goes on asking queries of the form

$$\exists x_1, \ldots, x_m \left(\bigwedge_{1 \leq i < j \leq m} (x_i \neq x_j) \right)$$

   for $m = 1, 2, \ldots$ until it receives a negative answer. Then $n = m - 1$.

2. For every $m_1 \neq m_2$ such that $m_1 + m_2 = n$, M_{q_k} goes on asking if the input database I is equal to the disjoint union of K_{m_1} and K_{m_2}.
3. In case M_{q_k} does not obtain any positive answer in the previous step, then it updates (using the *update tape*) the designated unary output relation with the empty set and halts. Otherwise, M_{q_k} keeps on updating auxiliary relations of the appropriate arity with the formula

$$\varphi(x_1, \ldots, x_m) \equiv \bigwedge_{1 \leq i < j \leq m} E(x_i, x_j)$$

from $m = k$ down-to 1, until some of the resulting relations is not empty. Then M_{q_k} updates the output relation with the formula

$\psi(x) \equiv \exists x_2, \ldots, x_m(\varphi(x, x_2, \ldots, x_m) \vee \ldots \vee$

$\exists x_1, \ldots, x_{m-1}(\varphi(x_1, \ldots, x_{m-1}, x)))$ and halts.

As we saw in Proposition 7 that $q_k \notin total(RRM^k)$, then the inclusion is strict.

□

The second result shows that the $RRM^{(k)}$ machines defined in the present paper have enough computation power as to compute the class of k-approximations of *all* queries in $total(\mathcal{CQ})$.

Proposition 9 *$total(RRM^{(k)}) \supseteq \{q_k : q \in total(\mathcal{CQ})\}$, i.e., the class of total computable queries which are computable by $RRM^{(k)}$ machines includes the class of k-approximations of queries in $total(\mathcal{CQ})$.*

Proof: Let q_k be an r-ary query which is the k-approximation of a query $q \in total(\mathcal{CQ})$ of schema σ. We will build a $RRM^{(k)}$ machine M_{q_k}, which will compute q_k. M_{q_k} works as follows:

1. M_{q_k} computes the size of the domain n.
2. M_{q_k} goes on writing down diagrams Δ_i in its *query tape* for databases in $\mathcal{B}_\sigma$ of size n till the database I in the *rs* models Δ_i ($I \models \Delta_i$). When it happens, it means that the diagram Δ_i in the query tape is the diagram Δ_I of the database I in the *rs*.
3. Using Δ_I, M_{q_k} builds a canonical representation I' of I in its TM tape.
4. Now M_{q_k} computes $q(I')$ in its TM tape.
5. For every r-tuple s_i over I', M_{q_k} builds in its TM tape the isolating formula ϕ_{s_i} as in Proposition 1 for the FO^k type of s_i and in this way we get isolating formulas for the types in $Tp^{FO^k}(I', r)$.
6. M_{q_k} evaluates as queries (in the *update tape*) the formulas ϕ_{s_i}, for every i, which are in FO^k, and assigns the results to the working r-ary relation symbols S_i in the relational store, respectively (note that these queries are evaluated on the input database).
7. Finally, M_{q_k} assigns to the output relation in the *rs* the tuples in the relations S_i corresponding to the tuples which are in the output relation of the query q in the TM tape.

□

We do not know yet whether the $RRM^{(k)}$ machines are able to compute queries which are not in the respective class of k-approximations or not. Indeed, this is our first open problem in the next section.

5 Open Problems

In this section we include a small list of open problems in the area of semantic classification of database queries in which we are working at present.

Problem 1 *We know that $total(RRM^{(k)}) \supseteq \{q_k : q \in total(\mathcal{CQ})\}$ by Proposition 9. What we do not know yet is whether the inclusion is strict, i.e., whether $total(RRM^{(k)}) \subseteq \{q_k : q \in total(\mathcal{CQ})\}$ holds.*

Problem 2 *We already know that $RRM^{(k)} \supset RRM^k$, then it would be interesting to find a characterization of the expressive power of the machines $RRM^{(k)}$.*

Problem 3 *Find a characterization of the class of queries q such that its k-approximations fall in the classes QCQ^k, i.e., the class of queries q such that for every $k \geq 1$, $q_k \in QCQ^k$.*

There are also several open problems related to the expressive power of the RRM^k and the RCM^k working in homogeneous databases; see (Turull Torres J. M. 2001) and (Turull Torres J. M. 2002). We present next an open problem of this kind.

Let $k \geq 1$. A database I is *C^k-homogeneous* if for every pair of k-tuples $\bar{a}_k$ and $\bar{b}_k$ over I, if $\bar{a}_k \equiv_k \bar{b}_k$, then $\bar{a}_k \equiv_{\simeq} \bar{b}_k$; where $\equiv_{\simeq}$ denotes the (equivalence) relation defined by the existence of an automorphism in the database I mapping one k-tuple onto the other. That is, $\bar{a}_k \equiv_{\simeq} \bar{b}_k$ iff there exists an automorphism f on I such that, for every $1 \leq i \leq k$, $f(a_i) = b_i$.

We define next a presumably stronger notion regarding homogeneity: *strong C^k-homogeneity*. To the authors' knowledge it is not known whether there exist examples of classes of databases which are C^k-homogeneous for some k, but which are not strongly C^k-homogeneous. However, the consideration of strong C^k-homogeneity makes sense not only because of the intuitive appeal of the notion, but also because this is the property proved in (Turull Torres J. M. 2002) to be a lower bound with respect to the increment in computation power of the machine RCM^k when working on strongly C^k-homogeneous databases. Up to know, we could not prove that this result holds for C^k-homogeneous databases as well. Let $k \geq 1$. A database I is *strongly C^k-homogeneous* if it is C^r-homogeneous for every $r \geq k$.

Definition 12 *For any schema σ, let us denote as $arity(\sigma)$ the maximum arity of a relation symbol in the schema. We define the class $\mathcal{QCQ}^C$ as the class of queries $f \in \mathcal{CQ}$ of some schema σ and of any arity, for which there exists an integer $n \geq max\{arity(f), arity(\sigma)\}$ such that for every pair of databases I, J in $\mathcal{B}_\sigma$, if $Tp^{C^h}(I, h) = Tp^{C^h}(J, h)$, then $Tp^{C^h}(\langle I, f(I)\rangle, h) = Tp^{C^h}(\langle J, f(J)\rangle, h)\}$ where $\langle I, f(I)\rangle$ and $\langle J, f(J)\rangle$ are databases of schema $\sigma \cup \{R\}$ with R being a relation symbol with the arity of the query f, and $h = max\{n, min\{k : I$ and J are strongly C^k-homogeneous$\}\}$.*

Problem 4 *Clearly, $\mathcal{CQ} \supseteq \mathcal{QCQ}^C \supseteq \mathcal{QCQ}^{C^\omega}$. But, unlike the analogous class $\mathcal{QCQ}$ in (Turull Torres J. M. 2001), we do not know whether the inclusions are strict.*

6 Conclusions and Future Work

We defined a formal notion of approximation of computable query, making use of the Model Theoretic concept of FO^k type. The use of FO^k types as an approximation measure also seems to be new. We believe that such notion is interesting not only from a theoretical perspective (as it was explored in this paper) but also from a practical perspective. For instance, it can be useful in areas as decision support systems. Note that, we use a concept of similarity based in properties of the tuples, which in turn can have applications in the search of common behavior patterns.

In an attempt to characterize syntactically the classes of k-approximations defined in this article, we defined a new variation of Reflective Relational Machine. We found that such machine includes the classes of k-approximations for every natural number k but we still do not know whether such inclusion is strict or not. The intuitive idea behind this new machine is that it has capability to appreciate the database in all its details but its capability to generate answers to queries is not precise. We proved in this article that such kind of machines are not subsumed by the Reflective Relational Machines. We think that they are interesting on their own right and that they can provide more insight on the nature of query computation.

Our concept of k-approximation as closure under equality of FO^k types can be generalized and applied to other logics as long as they are less expressive than FO. Indeed, the study of C^k types in this context is part of our research programme in query approximations. That is, we can define for every k, the notion of C^k approximation of a query q as the closure under equality of C^k types of the actual query q. More precisely, we will consider that q_{C^k} is a C^k *approximation* of a query q of arity $r \leq k$ iff for every database I, the relation $q_{C^k}(I)$ defined by q_{C^k} over I includes exactly those r-tuples in $dom(I)^r$ which have the C^k type of some tuple in $q(I)$. And then, we can do a similar study with machines $RRM(FO, C^k)$.

In a different perspective, we are also planning to explore approximations of queries, using the notion of *almost everywhere equivalence* of Logics defined in (Hella L., Kolaitis P., Luosto K. 1996). This notion was previously used in the context of database queries in (Turull Torres J. M. 2002). Intuitively, two queries q and q' are equivalent *almost everywhere* if as we consider databases of bigger sizes, the ratio of databases of the same size were q and q' coincides grows tending to one. Formally, let σ be a schema, let q, q' be two computable queries of schema σ and let $\mu_{(q=q')}$ be as follows:

$$\mu_{(q=q')} = lim_{n\to\infty}$$

$$\frac{|\{I \in \mathcal{B}_\sigma : dom(I) = \{1,\ldots,n\} \wedge q(I) = q'(I)\}|}{|\{I \in \mathcal{B}_\sigma : dom(I) = \{1,\ldots,n\}\}|}$$

Then, we say that q' coincides with q over *almost all* databases iff $\mu_{(q=q')} = 1$. Clearly, this concept can be interpreted as a notion of approximation of queries where $\mu_{(q=q')} = 1$ means that q' is an approximation of the query q.

References

Abiteboul S., Hull R., Vianu V. (1994), *Foundations of Databases*. Addison-Wesley.

Abiteboul S., Papadimitriou C. and Vianu V. (1994), *The power of reflective relational machines*. Proceedings 9th Symposium on Logic in Computer Science, pp. 230-240.

Abiteboul S., Papadimitriou C., Vianu V. (1998), *Reflective Relational Machines*. Information and Computation 143, pp. 110-136.

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

Abiteboul S., Vianu V. (1995), *Computing with First-Order Logic*. JCSS 50(2), pages 309-335.

Abiteboul S., Vardi M.Y., Vianu V. (1995), *Computing with Infinitary Logic*. Theoretical Computer Science 149(1), pp. 101-128.

Abiteboul S., Vardi M.Y., Vianu V. (1997), *Fixpoint logic, relational machines, and computational complexity*. Journal of ACM 44(1), pp. 30-56.

Aho A.V., Ullman J.D. (1979), *Universality of Data Retrieval Languages*. Proceedings 6th ACM Symp. on Principles of Programming Languages.

Chandra A.K., Harel D. (1980), *Computable Queries for Relational Data Bases*. Journal of Computer and System Sciences 21(2), pp. 156-178.

Dawar, A. (1993), *Feasible Computation through Model Theory*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Dawar, A., Lindell, S., Weinstein, S. (1995), *Infinitary Logic and Inductive Definability over Finite Structures*. Information and Computation 119(2), pp. 160-175.

Ebbinghaus H., Flum J. (1999), *Finite Model Theory*. Springer, 2nd. ed.

Ebbinghaus H., Flum J., Thomas W. (1984), *Mathematical Logic*. Springer.

Hella L., Kolaitis P., Luosto K. (1996), *Almost Everywhere Equivalence of Logics in Finite Model Theory*. The Bulletin of Symbolic Logic **2, 4** 422–443.

Hopcroft J., Ullman J. (1979), *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Immerman N. (1999), *Descriptive Complexity*. Springer.

Ioannidis Y., Poosala V. (1999), *Histogram-Based Approximation of Set-Valued Query Answers*, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland.

Link S., Turull Torres J.M. (2003), *Towards optimising query evaluation with quotient databases*. Advances in Databases and Information Systems, 7th East European Conference, ADBIS 2003, Dresden, Germany, Local Proceedings, pp.28-37.

Otto M. (1996), *The Expressive Power of Fixed Point Logic with Counting*. Journal of Symbolic Logic 61,1 147–176.

Otto M. (1997), *Bounded Variable Logics and Counting*. Springer.

Pavlov D., Mannila H., Smyth P. (2001), *Beyond independence: probabilistic methods for query approximation on binary transaction data.* In Technical report ICS TR-01-09, Information and Computer Science Department, UC Irvine, 2001.

Turull Torres J. M. (2001), *A Study of Homogeneity in Relational Databases.* Annals of Mathematics and Artificial Intelligence, 33(2), p.379-414.

Turull Torres J.M. (2002), *Relational Databases and Homogeneity in Logics with Counting.* Foundations of Information and Knowledge Systems: Second International Symposium, FoIKS 2002, Germany, February 2002, Springer LNCS.

Ullman J.D. (1988), *Principles of database and knowledge Base Systems: Volume I and II.* Computer Science Press.