Untyped Queries, Untyped Reflective Machines and Conditional Quantifiers

JOSE MARIA TURULL TORRES[1] [2]

Universidad Nacional de San Luis
Fac. Cs. Fís-Matemáticas - Depto. de Informática

Universidad Tecnológica Nacional
Fac. Regional Buenos Aires - Depto. de Sistemas

Peron 2315, piso 4, depto. P
1040 Capital Federal, ARGENTINA

email: turull@iamba.edu.ar turull@unsl.edu.ar

April 1997

Abstract

In the definition of the class $\mathcal{CQ}$ of computable queries, Chandra and Harel included *untyped queries*, that is, queries whose answers are relations of possibly different arities on different structures. However, it seems that in the quite wide and important work which followed on the subject, untyped queries were not considered at all, neither in the abstract machines side (*Relational Machines* and *Reflective Machines*), nor in the logic framework. We propose to re-introduce these queries in the study of Query Computability and Complexity. One important application which we face is the computation of numerical functions, that is, functions which range over the naturals. We denote this class as $\mathcal{NQ}$ (*numerical queries*). For that purpose we define an extension of the Reflective Machine of [APV,94], which we called *untyped Reflective Machine*, and we prove that this model is *complete* considering also the untyped queries. As a consequence of this proof we extend a result of Tyskiewicz about the possibility of restricting the class of dynamic queries, and still not loosing expressive power, and we get a *Normal Form* for the untyped reflective machines. In the logic framework, we define a new quantifier, which we call *conditional quantifier*, and we build with it an infinitary logic which we denote by $\mathcal{L}^c_{\omega_1\omega}$. The difference between this logic and $\mathcal{L}_{\omega_1\omega}$ is that a formula of our logic can induce relations of different arity on different structures. Then we prove that $\mathcal{L}^c_{\omega_1\omega}$ strictly includes the whole sub-class $total(\mathcal{CQ})$, including untyped queries. Finally we define a fragment of $\mathcal{L}^c_{\omega_1\omega}$ which we prove that exactly captures $total(\mathcal{CQ})$, but which is undecidable. We also define an undecidable fragment of $\mathcal{L}_{\omega_1\omega}$ which exactly captures $total\text{-}typed(\mathcal{CQ})$.

[1]The work presented here was done during a visit to the Department of Mathematics of the University of Helsinki.

[2]This work has been partially supported by Grant 1011049 of the Academy of Finland and by a Grant of Universidad CAECE, Argentina.

Section 1: Introduction.

The class of *Computable Queries*, denoted by $\mathcal{CQ}$, was first defined in [CH,80]. Roughly, a *computable query* is a function from classes of finite relational stuctures of a given finite relational signature to classes of relations on the corresponding structure, such that it is partial recursive and it preserves isomorphisms. Chandra and Harel defined a formal programming language for computing queries in that article, called QL, and they proved that this language is *complete*, that is, the set of QL programs computes (or "expresses") the whole class $\mathcal{CQ}$.

After that, in 1991, Abiteboul and Vianu defined in [AV,91] another kind of formalism to characterize computable queries. They defined an abstract machine, which they called *Generic Machine*, denoted as GM^{loose}, which they proved to be strictly included in QL, considering the class of queries which may be computed by programs or machines in each formalism. They also proved that the class of GM^{loose} machines "behaves" as complete when working on classes of *ordered* structures. They defined later the so called *Reflective Machine* (RM from now on), together with Papadimitriou in [APV,94]. RM is a generic machine which works as an acceptor device, and which allows the use of FO queries that can be generated during a computation. These queries are called *dynamic queries* and are evaluated by the machine in one single step. This machine is not restricted to have a finite signature for its *relational store* and this store can hold arbitrary many relations of arbitrary arity. Nevertheless, it is important to note that we are not talking of relations of variable arity as output, since RM is an acceptor device. They observed that if the dynamic queries are allowed to use $O(n)$ variables, this model is complete (as to boolean queries). We can think also of this model as a transducer machine with a finite signature in its relational store, and with one of its relation symbols designated as output, following the way in which this model was defined in [Tur,95]. That is, we can define this machine as an extension of GM^{loose}. It is easy to see that the model we get in this way is complete with respect to the whole class of computable queries considered by these authors. Let's denote it by GM^n.

In the meantime, and from outside of the field of Computer Science, Gurevich defined in 1984 the concept of *global predicate* ([Gur,84]), which he pointed out is equivalent to that of a query. This is a concept which naturallly arises from Tarski semantics of Classical Logic. In this way, the notion of a *logic*, in an abstract sense, turned into a different kind of formalism for computing queries. There are by now a lot of results characterizing different subclasses of the class of computable global predicates (or computable queries, as they are treated as equivalent in the literature) by different logics. And one of them is the well known fact that the *infinitary logic* $\mathcal{L}_{\omega_1\omega}$ strictly includes the whole class $total(\mathcal{CQ})$. The strict inclusion follows from the fact that in that logic *any* global predicate can be expressed, including those global predicates which are non recursive. We are referring in this context only to total queries because this is the usual way of thinking of logicians in this field. They usually don't consider partial queries. We make some considerations to this regard at the end of Section 4.

Now, there are some aspects in the relation among the three different formalisms which we mentioned, which are worthwhile to be noted:

- In the definition of computable query in [CH,80] the result of applying a query to different structures are relations which are *not* restricted to be of a fixed arity. That is, the same query applied over two different structures of a given fixed signature can have different arities. So the class $\mathcal{CQ}$ includes computable queries of *varying arities*. Moreover, the authors point out this fact in the article, and they briefly describe a slight modification to the definition of the language QL, so that it can include only the queries of fixed arity (which they call *typed queries*).

- In [AV,91] the authors define computable queries as *fixed arity* queries, besides the other properties already defined in [CH,80]. But they don't refer to these queries as *typed*. However, they make explicit reference to the work of Chandra and Harel, and they mention the language QL (which they call $while^{unsorted}$ in their paper) and they point out that this was the first *complete* language. In other articles, like [AV,92a], [AV,92b] and [APV,94] the authors keep on considering only fixed arity queries. And this seems to be the general feeling in the literature among the authors who come from the field of Computer Science, as well.

- The concept of global predicate by Gurevich, defined in [Gur,84], requires also a *fixed arity* of the resulting relation for the whole class of structures where it is defined. And it seems to be the case that all the logicians who work in this area consider the same restriction as to the arity of the global predicates or queries.

  We should point out to this regard that this is a quite natural restriction arising from Tarski semantics.

Let us summarize the situation of the field to this regard as follows:

a) $QL = \mathcal{CQ} \supset typed(\mathcal{CQ}) \supset total\text{-}typed(\mathcal{CQ})$

b) $GM = (GM^{loose} + \leq) = typed(\mathcal{CQ}) = GM^n$

c) $\mathcal{L}_{\omega_1\omega} \supset total\text{-}typed(\mathcal{CQ})$

d) $\mathcal{L}_{\omega_1\omega} \not\supseteq total(\mathcal{CQ})$ because it cannot express untyped queries.

Note that there are interesting untyped queries which are meaningful in the Model Theory setting. Lets consider here two examples:

- *The set of automorphisms of a structure.* This query induces a $2n$-ary relation in every structure I of size n. Every $2n$-tuple represents one ordering of $dom(I)$, in the first n-subtuple, together with one of the automorphisms in I for that ordering, in the second n-subtuple. In symbols:

$$q(I) = \{(t_1, \ldots, t_n, t_{n+1}, \ldots, t_{2n}) | f : t_i \mapsto t_{n+i},\ for\ 1 \leq i \leq n,$$

$$is\ an\ automorphism\ on\ I\}$$

  In the Example 4.1 we show how to express this query.

- *The set of tuples such that every component is an element of a different equivalence class in the relation of commutability of elements under some automorphism.* It is clear that this relation depends on the structure, and so does the number of equivalence classes which it induces, so that the arity of the relation induced by the query is not fixed, but depends on the structure.

On the other hand, in real world data bases it is quite usual to ask queries whose answers are expected to be (natural) numbers, instead of relations. Consider queries like "*how many clients have bought the article 101 in the last 12 months?*" or "*how many salesmen accomplished with their sales budget last year?*". They should be considered as computable queries, but they are not because, according to the definition, a query is expected to range over relations defined on the input structure (or data base), and not over numbers.

Some extensions to the formalism of [CH,80] have been defined to consider these kind of queries. One was defined in the same article and another was recently defined by Gurevich and Grädel in 1995 ([GG,95]). Roughly speaking, both of them propose to extend the finite relational structures which represent data base instances with the infinite domain of the natural numbers. And, of course, this trend takes us out of the field of Finite Model Theory, a theoretical framework which is becoming more and more solid in the present decade.

So, in order to remain within the current framework of FMT, and in the consideration of the mismatch which was exposed above, we propose to extend the concept of computable query which is being used both in the formalism of abstract machines and in the formalism of abstract logic. The idea is to consider a part of the subclass of the *untyped queries* as what we call *numerical queries*. That is, queries which range over the Naturals. We will do this by encoding the resulting number as the arity of the output relation of an untyped query.

For the purpose of computing both untyped queries, and particularly numerical queries, we extend the Reflective Machine to what we call Untyped Reflective Machine (GM_u). Then we prove that it computes exactly the *whole* class $\mathcal{CQ}$, including untyped queries. The way in which we prove this result gives rise to a *Normal Form* for GM_u machines, which is not the same as the one defined in [AV,91] and extended in [APV,94] for a subclass of RM.

In [Tys,96], it is shown that relational machines working as acceptors, bounded to polynomial time, computes exactly the same class of boolean queries (PSPACE) as what the author defines as Existential RM (EFO-machines). This amounts to allowing for dynamic queries only a fragment of Existential FO, namely he allows input relations and equality to be negated in the queries, but not updated, and all of the remaining relations in the relational store are allowed to be updated but must appear unnegated in the queries. It means that if we restrict RM in this way, we are *not* restricting the expressive power of the subclass of RM which work in polynomial time.

Here, as a Corollary to one of the main results, and considering just expressibility but not complexity, we extend that result, stating that if we consider the *whole* class of RM, including transducers as well as acceptors (actually, it is a wider class, because we consider the model which we define as GM_u, which strictly includes RM), they are not restricted in expressive power if we allow as dynamic queries only the same fragment of FO as in [Tys,96]. We actually restrict the class of dynamic queries to a still smaller class of Existential FO formulae (see Corollary 3.2).

However, if we consider the complexity of the machines, it *does* change, because our construction includes the computation of a canonical representation of the input structure, which is exponential in time.

As to the Logic Formalism, we introduce a new kind of quantifier, which we call *conditional quantifier*, which allows the same formula to induce relations of different arities on different structures. So, we define the infinitary logic $\mathcal{L}^c_{\omega_1\omega}$, augmenting the well known logic $\mathcal{L}_{\omega_1\omega}$, and we prove that it strictly includes the *whole* sub-class $total(\mathcal{CQ})$, including the untyped queries, and, hence, the numerical queries. Finally, we define a fragment of $\mathcal{L}^c_{\omega_1\omega}$, which we denote by $\mathcal{L}^c_*$, and we prove that it captures exactly the sub-class $total(\mathcal{CQ})$, but unfortunately it is undecidable. And we also get, as a Corollary to the former, an undecidable fragment $\mathcal{L}_*$ of the logic $\mathcal{L}_{\omega_1\omega}$ which exactly captures the sub-class *total-typed*($\mathcal{CQ}$). Nevertheless, we hope that these results can give some intuition on the search of such decidable fragments, though we are not optimistic about its existence for the sub-class $total(\mathcal{CQ})$ because of the close relation between it and the class of the Turing Machines which compute total functions, which is well known to be undecidable.

Section 2: Numerical Queries.

As we said in Section 1, we want to consider a part of the subclass of the *untyped queries* as what we call *numerical queries*. That is, queries which range over the Naturals. We will do this by encoding the resulting number as the arity of the output relation of a untyped query.

We will restrict the content of the output relations so that their cardinality will be minimal. Besides, we also want to emphasize the idea of a number being encoded by a tuple (though we will actually have in the output relation as many tuples as elements in the domain of the stucture), by asking that every tuple be built with repetitions of the same element, giving thus a unary representation of the natural number.

In the present article we will use the standard notation for Finite Model Theory, as in [EF,95], unless otherwise stated. By $\mathcal{E}_{\sigma,fin}$ we will refer to the class of finite structures with signature σ.

Definition 2.1:
A *numerical query* is a function

$$q : \mathcal{E}_{\sigma,fin} \longrightarrow \bigcup_{r\in\omega} \mathcal{E}_{\langle R^r\rangle,fin}$$

for some finite relational signature σ, such that:

- q is partial recursive in some codification.
- q preserves isomorphisms.
- For all $I \in \mathcal{E}_{\sigma,fin}$, if $q(I)$ is defined then $|q(I)| = |dom(I)|$ and every tuple of the relation $q(I)$ is formed with only one element of $dom(I)$ by repeating it according to the arity of the relation. If the arity of $q(I)$ is 0 then it will be a 0-ary relation with the value TRUE.

We will denote the class of numerical queries as $\mathcal{NQ}$. Then, it is clear that:

- $\mathcal{NQ} \not\subseteq typed(\mathcal{CQ})$
- $\mathcal{NQ} \subset \mathcal{CQ}$

Note that there are also interesting numerical queries which naturally arise in the Model Theory setting. Just consider any numerical invariant of finite relational structures, such as the size of a structure, the size of its automorphisms group, the diameter of a graph, the size of the maximal complete subgraph in a graph, etc.

Section 3: The Abstract Machine Formalism.

We will define an extension of the generic machine GM^n, as it was defined in [Tur,95].

Definition 3.1:

An *untyped reflective machine*, denoted by GM_u, is a transducer reflective machine which allows dynamic queries in FO (see [APV,94]) with any number of variables, with the following characteristics:

- The signature of the output structure, ρ, is countably infinite, having exactly one relation symbol T_i for every $i \in \omega$. Note that this includes a 0-ary relation symbol T_0.
- There are countably many relation symbols of each finite arity in the *relational store*. So that the signature of the relational store, σ, is countably infinite.
- The output structure of every computation has at most one relation symbol T_i whose content at the end of the computation is non empty (for some $i \geq 0$). In the case of $i = 0$, this means that the content of the relation symbol T_0 is a 0-ary relation with the value TRUE.
- The machine has a distinguished halting state which indicates that in the output structure all the relations T_i are empty, for $i \geq 0$. In the case of $i = 0$, this means that the content of the relation symbol T_0 is a 0-ary relation with the value FALSE.

Note that, as every computation is finite for an input structure which belongs to the domain of the query which is being computed by the machine, then though the signature of the relational store is countably infinite, only a finite number of relation symbols will be used in every ending computation.

Theorem 3.1: $GM_u = \mathcal{CQ}$.

Proof:

One direction is trivial, since clearly every query computed by a GM_u machine is in $\mathcal{CQ}$.

As to the other direction, let $q \in \mathcal{CQ}$. We will show how to build a GM_u machine $\mathcal{M}_q$ which computes the query q.

First, we can find out by using dynamic queries with up to $(n+1)$ variables the size n of the input structure stored in the relational store. We do this by generating the following dynamic query, increasing n until we get a value FALSE:

$$\exists x_1 \dots x_n (x_1 \neq \dots \neq x_n)$$

Next, we will build a sentence Ψ with n variables of the form

$$\Psi \equiv \exists x_1 \dots x_n (\psi(x_1, \dots, x_n))$$

where $\psi(x_1, \dots, x_n)$ is a conjunction of atomic and negation of atomic formulae of the form $R_i(x_{j_1} \dots x_{j_{r_i}})$, with R_i being an input relation and r_i its arity. The sentence Ψ will assign a unique variable (that is, a unique index) to every element in the domain of the input structure, in such a way that we will be able to use this index to represent that element up to isomorphism. To build the isomorphism preserving mapping between the domain of the input structure and the set of integers $\{1, \dots, n\}$, we will build for Ψ every possible configuration of the input structure.

Let's say that the input signature is $\langle R_1, \dots, R_k \rangle$, with arities $(r_1, \dots, r_k)$, respectively. For, say, relation R_i, we will build every possible string of length $(n^{r_i} \cdot r_i + 1)$ from the alphabet $\{1, \dots, n, *\}$. We will restrict the set of strings, by asking that the symbol * be allowed only at the first (leftmost) position of the string, or at any position $(h \cdot r_i + 1)$, for some $1 \leq h \leq n^{r_i}$. Further, there must be exactly one occurrence of the symbol * in the string. The symbol * will separate the tuples of R_i from the tuples of $\neg R_i$. So that every string built in this way will represent a possible content of the relation R_i for a given input structure as well as a possible content for its complement. Then, when building the formula $\psi(x_1, \dots, x_n)$ we will use the integers in the string before the * as indices for the variables in the atomic formulae for R_i, and the integers in the string after the * as indices for the variables in the negation of the atomic formulae for R_i. For instance, let u_i be one of such possible strings:

$$u_i = (j_1, \dots, j_{r_i}, \dots, j_{d_i - r_i}, \dots, j_{d_i}, \dots, j_{d_i + r_i}, \dots, j_{n^{r_i} \cdot r_i - r_i + 2}, \dots, j_{n^{r_i} \cdot r_i + 1})$$

where $j_{d_i} = *$ with $d_i = h \cdot r_i + 1$, for some $1 \leq h < n^{r_i}$, in this example. Then we build the sub-formula $\varphi_i(x_1, \dots, x_n)$ as follows:

$$\varphi_i(x_1, \dots, x_n) \equiv R_i(x_{j_1}, \dots, x_{j_{r_i}}) \wedge \dots \wedge R_i(x_{j_{d_i - r_i}}, \dots, x_{j_{d_i - 1}}) \wedge$$

$$\neg R_i(x_{j_{d_i+1}}, \ldots, x_{j_{d_i+r_i}}) \wedge \ldots \wedge \neg R_i(x_{j_{n^{r_i} \cdot r_i - r_i + 2}}, \ldots, x_{j_{n^{r_i} \cdot r_i + 1}})$$

Now, as we have k relations in the input structure, we will build also every possible string with the same restriction, and, of course, considering the corresponding arity, for every relation in $\langle R_1, \ldots, R_k \rangle$. And we build the corresponding sub-formulae $\varphi_1, \ldots, \varphi_k$. We will concatenate the k strings following the order of the indices 1 to k, so that we will get strings of length

$$(n^{r_1} \cdot r_1 + \cdots + n^{r_k} \cdot r_k + k)$$

Then we build the corresponding formula ψ for every string with the conjunction of the sub-formulae $\varphi_1, \ldots, \varphi_k$ which we built according to every substring, respectively. Note that we use the TM tape of $\mathcal{M}_q$ to keep track of the generation of every possible string, and also to build the strings, while for every string we build the corresponding formula Ψ in the query tape of $\mathcal{M}_q$.

In this way, by considering every possible string as we defined them, we will eventually get a value of TRUE when evaluating the sentence Ψ on the input structure. And this will mean that the mapping being represented by that sentence, of the form $f : dom(I) \longrightarrow \{1, \ldots, n\}$, is one of the possible mappings which preserve isomorphisms.

We generate the different strings in lexicographical order, so that we are actually building a *canonical encoding* of the input structure.

Now we write an encoding of the input structure on the TM tape of $\mathcal{M}_q$. And we use the string which gave us the value TRUE for Ψ as that codification. The only difference is that we don't consider for this matter the complement of the relations. So, we stop writing the content of every relation in the tape when we find the * in the string.

At this stage we have the input structure written in the TM tape, so we make $\mathcal{M}_q$ work as a Turing Machine, which exists because q is partial recursive. Note that the TM will preserve isomorphisms by the definition of computable query.

When the TM stops we have the output relation, say T, written in the TM tape. And we must translate it to a relation, say T_s if the arity of T is s, in the relational store. To do this, for every s-tuple $(t_1, \ldots, t_s)$ in T we build a formula $\alpha_{(t_1,\ldots,t_s)}$ as follows:

$$\alpha_{(t_1,\ldots,t_s)}(z_1, \ldots, z_s) \equiv \exists x_1 \ldots x_n(\psi(x_1, \ldots, x_n) \wedge z_1 = x_{t_1} \wedge \ldots \wedge z_s = x_{t_s})$$

With this formula we build a dynamic query to compute the union between the s-ary relation which α induces on the input structure in the relational store, and the previous content of the s-ary relation symbol T_s in this store:

$$T_s(z_1, \ldots, z_s) \vee \alpha_{(t_1,\ldots,t_s)}(z_1, \ldots, z_s)$$

And we assign the result of the evaluation of this dynamic query to the relation symbol T_s in the relational store of $\mathcal{M}_q$.

Here it should be noted that for every s-tuple $(t_1, \ldots, t_s)$ in T the whole set of s-tuples with the same automorphism type (extended to s-tuples) will be added to T_s, which is correct to preserve isomorphisms. $\blacksquare$

Corollary 3.1: $GM_u \supset \mathcal{NQ}$. $\blacksquare$

Remark 3.1:

Note that in the sub-formula α of the dynamic query which we used in the Proof of Theorem 3.1, we may have used more than $O(n)$ variables. Actually, we used a number of variables which cannot be bounded, because according to the considerations we made in Section 2, the arity of the resulting relation is to be interpreted as the result of the numerical query. So that it cannot be bounded by any recursive function on n.

So, this means that when dealing with the *whole* class $\mathcal{CQ}$ (that is, when considering also untyped queries) it is no longer true that Reflective Machines with $O(n)$ variables in their dynamic queries are complete.

If we consider the Proof of Theorem 3.1, we can see that we have actually used a very restricted version of the machine GM_u. The Proof also gives us a *Normal Form* for computations of queries in the whole class $\mathcal{CQ}$, consisting of three stages. The first stage is the construction of a canonical representation of the input structure in the TM tape of the machine. The second stage is a pure TM computation, in which the relational store is not used at all, and leaves an encoding of the resulting relation in the TM tape. Finally, the third stage consists just of the decoding of the output relation from the tape, and its storage in the relational store.

However, we should point out that this is important as a matter of expressibility or computability, but not as a matter of complexity, since the canonical encoding requires exponential time. This should be contrasted with the Normal Form of [AV,91] for Relational Machines, where an encoding which is built in the TM tape in the first stage of the computation can be computed in polynomial time (see also [DLW,95]). And also with the extension of this Normal Form defined in [APV,94] for Reflective Machines with variable complexity $O(f(n))$, with f being sublinear and computable in PSPACE, for which the authors proved that the first phase of the computation, which includes an encoding of an ordered structure in the TM tape, can be carried on in polynomial space. In both cases the models are not complete.

So the next result follows immediately.

Corollary 3.2: *The model GM_u remains complete, as to the whole class $\mathcal{CQ}$, if* *we restrict the untyped machines as follows:*

a) *Only dynamic queries in the positive existential fragment of FO are allowed, that is, formulae of the form:*

$$\exists x_1 \ldots x_s(\varphi(x_1, \ldots, x_s))$$

Where φ is quantifier free.

b) *The dynamic queries must be either boolean queries, or their result must be assigned to one output relation T_i, which must be the same in the whole computation of the machine for a given input.*

c) *We don't allow auxiliary relations in the relational store, besides the input relations and the output relations T_i, for every $i \in \omega$.*

d) *Only one output relation can be used in every computation of the machine.* ■

Section 4: The Logic Formalism.

According to Tarski semantics for First Order Logic, which in turn has been usually followed in the construction of other logics, every formula with, say, r free variables defines an r-ary relation in every structure of the corresponding signature. This means that we cannot express untyped queries with the usual semantics and syntax. So, we will change the notion of free variable in a formula by making the property of an occurrence of a variable being free or bounded be a semantic notion, instead of a purely syntactic one.

To this regard, we will introduce a new kind of quantifier, which does not fit in the general frame of Lindström Quantifiers (see [Ebi,85] and [Flu,85]), and which we will call *conditional quantifier*. The main idea behind it is that the structure which in a given moment interprets a formula is which actually defines which variables are free and which are bound in that formula and for that structure. Then, as the set, and hence the number, of free variables of a given formula will depend on the structure, the formula will define in that structure a relation whose arity will depend on the structure as well, and not only on the formula.

One important restriction to note at this point is that if we use finitary logics then the arities that the untyped queries can take will be bounded by a constant. So we will add the conditional quantifier to an infinitary logic, namely $\mathcal{L}_{\omega_1\omega}$.

Definition 4.1:

Let $\mathcal{L}^c_{\omega_1\omega}$ be the infinitary logic $\mathcal{L}_{\omega_1\omega}$ augmented with the following formation rule for formulae:

- If δ is a sentence of $\mathcal{L}_{\omega_1\omega}$, φ is a formula of $\mathcal{L}^c_{\omega_1\omega}$ and x_i is a variable, then $\exists^c x_i(\delta,\varphi)$ is also a formula of $\mathcal{L}^c_{\omega_1\omega}$.

We will usually write $\exists^c x_i(\delta,\varphi(x_i))$ to emphasize the fact that the variable x_i is free in φ in the purely syntactic sense (i.e., in the classical sense).

Definition 4.2:

The semantics of the conditional quantifier is as follows. Recall that a *valuation* v is a total function from the countable set of variables in a logic to the domain of a structure. We say that two valuations v and v' are i-equivalent, for an arbitrary positive integer i, if it is the case that $v(x_j)=v'(x_j)$ for every positive integer $j\neq i$.

Then, if I is a structure of the same signature that δ and φ, and v is a valuation on the domain of I, we say that

$$I,v\models\exists^c x_i(\delta,\varphi(x_i))\iff$$

either

$I\models\delta$ and for some valuation v' which is i-equivalent to v it is $I,v'\models\varphi(x_i)$

or

$I\not\models\delta$ and $I,v\models\varphi(x_i)$

Now we will define the relation induced in a structure by a formula of the logic $\mathcal{L}^c_{\omega_1\omega}$. For that purpose, for every formula $\psi\in\mathcal{L}^c_{\omega_1\omega}$ with signature σ, and for every σ-structure I, we will denote by $\hat{\psi}_I$ the formula in $\mathcal{L}_{\omega_1\omega}$ which we get from ψ by replacing every subformula of the form $\exists^c x(\delta,\varphi(x))$ in ψ, either by the subformula $\exists x(\varphi(x))$, if $I\models\delta$, or by the subformula $\varphi(x)$ (with x as a free variable) if $I\not\models\delta$. Next, we will define it formally, by induction.

Definition 4.3:

- For every formula $\psi\in\mathcal{L}^c_{\omega_1\omega}$ with signature σ, and for every σ-structure I, we define $\hat{\psi}_I$ as follows:
  - if ψ is atomic, then $\hat{\psi}_I=\psi$.
  - if $\psi\equiv\psi_1\wedge\psi_2$, then $\hat{\psi}_I=\hat{\psi}_{1,I}\wedge\hat{\psi}_{2,I}$.
  - if $\psi\equiv\psi_1\vee\psi_2$, then $\hat{\psi}_I=\hat{\psi}_{1,I}\vee\hat{\psi}_{2,I}$.
  - if $\psi\equiv\neg\psi_1$, then $\hat{\psi}_I=\neg\hat{\psi}_{1,I}$.
  - if $\psi\equiv\exists x(\psi_1)$, then $\hat{\psi}_I=\exists x(\hat{\psi}_{1,I})$.

– if $\psi \equiv \exists^c x(\delta, \psi_1(x))$, then $\hat{\psi}_I = \exists x(\hat{\psi}_{1,I}(x))$, if $I \models \delta$, and $\hat{\psi}_I = \hat{\psi}_{1,I}(x)$, with x as a free variable in $\hat{\psi}_{1,I}$, if $I \not\models \delta$.

- Then, if $\hat{\psi}_I$ is t-ary, for some $t \geq 1$, the relation induced in I by ψ is defined as follows:

$$\psi^I = \{(v(x_1), \ldots, v(x_t)) : \\ v \text{ is a valuation on the domain of } I \text{ and} \\ I, v \models \hat{\psi}_I(x_1, \ldots, x_t)\}$$

Otherwise, if $\hat{\psi}_I$ is 0-ary, i.e, if it is a sentence, then

$$\psi^I = \text{TRUE} \qquad \Longleftrightarrow \qquad I \models \hat{\psi}_I$$

Note that $\hat{\psi}_I$ may have countably many free variables, though we have considered above only the case where it has a finite number of them.

The same kind of extension can be defined over FO, getting the logic FO^c. We will refer to this logic in Definition 5.1 in Section 5.

Example 4.1:
We will show how to express the query which we mentioned as the first example in Section 1 in the logic $\mathcal{L}^c_{\omega_1\omega}$. That is, for a given signature σ, and for every σ-structure I, we want the set of automorphisms of I.

For this matter we will define a $2n$-ary relation, with n being the size of the structure I, such that every $2n$-tuple t represents one automorphism on one of the possible orderings of $dom(I)$. The first n-subtuple of t will represent an ordering, and the second n-subtuple of t will represent one automorphism f as follows: for every $1 \leq i \leq n \quad f(t_i) = t_{n+i}$, where t_i denotes the i-th component of the tuple t. Note that as the size of the automorphisms group of I is from 1 to $n!$, and as there are $n!$ different orderings of $dom(I)$, the size of the resulting relation will give us some more information about I:

- The size of the automorphisms group of I is r, for some $1 \leq r \leq n!$, iff the size of the resulting relation for I is $r \cdot n!$.

- In particular, the structure I is *rigid* iff the size of the resulting relation for I is $n!$.

Let $\sigma = \langle R_1^{r_1}, \ldots, R_k^{r_k} \rangle$. Then the following formula Ψ of $\mathcal{L}^c_{\omega_1\omega}$ expresses the query:

$$\Psi \equiv \bigvee_{n\in\omega} \exists^c z_1 \ldots z_{2n}($$

$$\neg(\exists x_1 \ldots x_n (x_1 \neq \ldots \neq x_n \wedge \forall x_{n+1}(x_{n+1} = x_1 \vee \cdots \vee x_{n+1} = x_n))),$$

$$\bigwedge_{i=1}^{k} \quad \bigwedge_{(y_1,\ldots,y_{r_i})\in\{z_1,\ldots,z_n\}^{r_i}} \quad R_i(y_1,\ldots,y_{r_i}) \Longleftrightarrow R_i({y'}_1,\ldots,{y'}_{r_i}))$$

where for $1 \leq j \leq r_i$ and for $1 \leq l \leq n$, if $y_j = z_l$, then ${y'}_j = z_{l+n}$. ■

Example 4.2:

Let $\psi(x) \in \mathcal{L}_{\omega_1\omega}$ be a unary computable query. Let γ be a numerical query defined as follows, for every structure I of a given signature:

$$\gamma(I) = |\{x : \psi^I(x)\}|$$

First, we define the following sentences of $\mathcal{L}_{\omega_1\omega}$:

$$\delta_0 \equiv \neg\exists x(\psi(x))$$

and for $i \geq 1$:

$$\delta_i \equiv \exists x_1 \ldots x_i(\psi(x_1) \wedge \ldots \wedge \psi(x_i) \wedge$$

$$x_1 \neq \ldots \neq x_i \wedge \forall x_{i+1}(\psi(x_{i+1}) \Longrightarrow (x_{i+1} = x_1 \vee \ldots \vee x_{i+1} = x_i)))$$

δ_i means "there are exactly i different elements of the structure which satisfy $\psi(x)$".

Now, for every $i \geq 1$ we define the following formula of $\mathcal{L}^c_{\omega_1\omega}$:

$$\varphi_i \equiv \exists^c z_1 \ldots z_i(\neg\delta_i, z_1 = \ldots = z_i)$$

Note that if $I \models \delta_i$ then φ_i is equivalent to $z_1 = \ldots = z_i$, so that the relation φ_i^I will be i-ary.

Finally, we write the numerical query γ as follows:

$$\gamma \equiv \delta_0 \vee \bigvee_{i\geq 1} \varphi_i \qquad ■$$

Theorem 4.1: $\mathcal{L}^c_{\omega_1\omega} \supset (\mathcal{NQ} \cup total(\mathcal{CQ}))$.

Proof:

Let $f \in \mathcal{CQ}$, then by Theorem 3.1 there is a GM_u $\mathcal{M}_f$ which computes it, as in the proof of the theorem. We modify $\mathcal{M}_f$ as follows:

- We add four constant symbols to the input signature σ. So we have

$$\sigma' = \langle R_1^{r_1}, \ldots, R_k^{r_k}, @, -, 1, * \rangle$$

- We add to $\mathcal{M}_f$ two new distinguished states: q_I and q_T.

- When $\mathcal{M}_f$ ends with the encoding of the input structure $I \in \mathcal{E}_{\sigma', fin}$ in the TM tape, it moves the tape head to the left of the beginning of the encoding of I, and then it switches to the state q_I.

- $\mathcal{M}_f$ will work in the TM tape in unary notation, using the symbol 1. It will use the symbol * as a separator, the symbol $-$ as a space (i.e., the content of a tape cell not yet used), and two occurrences of the symbol @ will enclose a unary encoding of the current state of $\mathcal{M}_f$, to the left of the current cell. $\mathcal{M}_f$ will keep the tape updated in this way. That is, every "real" step in the computation of the machine (we mean by this a step which actually counts in the computation of the query f) will be followed by the update of the head position and the state in the way we have described.

- The relations which form I will we encoded in the tape in the order $D, R_1, \ldots, R_k$ (D being the domain of I). The elements of the encoding of D will be written in ascending order $1, \ldots, n$ (recall that they are integers, according to the proof of Theorem 3.1). $\mathcal{M}_f$ will write the string * following every component of every tuple, including the last component of the tuple. It will write the string ** following every tuple of every relation, including the last tuple of the relation. And it will write the string *** following every relation. So that every tuple in the tape will end with the string ***, and every relation will end with the string ******.

- $\mathcal{M}_f$ writes the arity of the resulting relation, enclosed between two strings ******, just before the relation itself.

- If the resulting relation T is 0-ary, then $\mathcal{M}_f$ writes the string ***1***, if the value is TRUE, or it writes the string ***$-$***, if it is FALSE. This is to distinguish this case from that of T being a unary relation with one single tuple with the element 1, in which case it would be encoded as 1******.

- When $\mathcal{M}_f$ ends with the pure TM part of the computation, it moves the tape head either to the left of the beginning of the first symbol 1 of the encoding of the arity of T, if it is $\neq 0$, or to the left of the second of the two strings ****** which enclose the arity, if it is zero, and then it switches to the state q_T.

We will build now a formula $\Psi_f \in \mathcal{L}^c_{\omega_1\omega}$ which expresses the query f. The formula will affirm the existence of a Turing Machine which computes the query f, and which is actually the pure TM part of $\mathcal{M}_f$.

In Ψ_f we will use second order variables, to write it in a simpler way. This is well known to be possible in the infinitary logic $\mathcal{L}_{\omega_1\omega}$ on finite models, and hence in $\mathcal{L}^c_{\omega_1\omega}$. To see this, think of the SO formula $\exists S^r(\varphi(S))$. Then, it is clearly equivalent to the following formula in $\mathcal{L}_{\omega_1\omega}$:

$$\bigvee_{n\in\omega} \exists x_1 \ldots x_n (x_1 \neq \cdots \neq x_n \wedge \forall x_{n+1}(x_{n+1} = x_1 \vee \cdots \vee x_{n+1} = x_n) \wedge$$

$$\exists z_1 \ldots z_{n^r \cdot r} (\bigvee z_1 = x_{i_1} \wedge \cdots \wedge z_{(k+1)\cdot r} = x_{i_{(k+1)\cdot r}} \wedge$$

$$\varphi'(z_1, \ldots, z_{(k+1)\cdot r})))$$

Where $0 \leq k < n^r$, and $\{x_{i_1}, \ldots, x_{i_{n^r \cdot r}}\} \subseteq \{x_1, \ldots, x_n\}$. If I is the structure interpreting the formula $\exists S^r(\varphi(S))$, then the second disjunction ranges over the power set of $(dom(I))^r$. So that every sequence of variables $(x_{i_1}, \ldots, x_{i_{(k+1)\cdot r}})$ represents one subset of $(dom(I))^r$, which is encoded by the string $(i_1, \ldots, i_{(k+1)\cdot r})$. The formula $\varphi'(z_1, \ldots, z_{(k+1)\cdot r})$ can be built from the formula $\varphi(S)$ considering that the sequence of variables $(z_1, \ldots, z_{(k+1)\cdot r})$ holds one possible r-ary relation over $dom(I)$, with $(k+1)$ tuples. The case of S being empty has to be independently considered.

The SO variables which will be used in the definition of Ψ_f are $C^p_1, \ldots, C^p_p, F^b$.

$C^p_1, \ldots, C^p_p$ are p-ary relations with only one tuple which represent the content of the p cells of the TM tape of $\mathcal{M}_f$ which might have been used, *before* steps $1, \ldots, p$, respectively, in a computation of the machine which reaches the state q_T after $(p-1)$ steps. The tuple also represents the current state of $\mathcal{M}_f$ and the current position of its TM tape head.

F^b is a b-ary relation which encodes the transition function of $\mathcal{M}_f$, restricted to the part of the computation defined by the states q_I and q_T as beginning and ending states, respectively. If we denote by c_q the number of states in $\mathcal{M}_f$, then it is $b = c_q \cdot 2 + 3$. There will be one tuple in F for every tuple in the transition function of $\mathcal{M}_f$, restricted as we explained above, and the tuple will have the following structure:

a) The first component will correspond to the old symbol in the current tape cell, from the alphabet $\{1, -, *\}$.

b) The sub-tuple beginning with the second component and ending with the $(c_q + 1)$-th component, will encode the current state in unary notation.

c) The $(c_q + 2)$-th component will represent the direction for the movement of the tape head as follows: 1 means to the left, * to the right and $-$ means no movement.

d) The $(c_q + 3)$-th component will define the symbol to be printed in the current tape cell, from the alphabet $\{1, -, *\}$.

e) The sub-tuple beginning with the $(c_q + 4)$-th component and ending with the $(2c_q + 3)$-th component, will encode the new state in unary notation. The encoding of both states will be filled with the symbol $-$ to the left, as needed.

The formula Ψ_f will say that there exists a GM_u machine ($\mathcal{M}_f$) which on input $I_l \in \mathcal{E}_{\sigma',fin}$ generates a relation of some arity $t \geq 0$, and, when the formula is interpreted by I_l, it will generate that t-ary relation. Actually, Ψ_f refers to a TM but, as we said before, that TM will represent the part of pure TM computation of $\mathcal{M}_f$. We don't need to represent the part of the computation of $\mathcal{M}_f$ where it builds the encoding of I_l in the TM tape, because in Ψ_f we check that the tape does contain a right encoding of I_l, as we will see later. In fact it checks that it is a *canonical encoding* for I_l, since in the formula we use the different strings which encode different contents of the relations in I_l in lexicographical order, and we proceed in the same way in $\mathcal{M}_f$. We don't need either to represent the part of the computation of $\mathcal{M}_f$ where it decodes the resulting relation from the TM tape, because Ψ_f will do that decoding.

Now we define Ψ_f:

$$\Psi_f \equiv \exists F^b(\varphi_F \wedge \bigvee_{I_l \in \mathcal{E}_{\sigma',fin}} \bigvee_{n \geq 4} \exists x_1 \ldots x_n(x_1 \neq \cdots \neq x_n \wedge$$

$$\forall x_{n+1}(x_{n+1} = x_1 \vee \cdots \vee x_{n+1} = x_n) \wedge$$

$$\psi_{n,I_l}(x_1, \ldots, x_n) \wedge * \neq 1 \neq - \neq @ \wedge$$

$$\bigvee_{p \in \omega} \exists C_1^p \ldots C_p^p(\gamma_{I_l,C_1^p} \wedge \alpha_{C_{1,p}} \wedge$$

$$((\bigvee_{t \geq 1} \bigvee_{(i_1,\ldots,i_t) \in \{1,\ldots,n\}^t} \phi^c_{p,(i_1,\ldots,i_t)}(x_{i_1}, \ldots, x_{i_t})) \vee \phi_{p,0}) \cdots)$$

The actual meaning of the expression $I_l \in \mathcal{E}_{\sigma',fin}$ in the infinitary disjunction is taking every possible string of length $(n^{r_1} \cdot r_1 + \cdots + n^{r_k} \cdot r_k + k)$, for every $n \geq 4$ (this is due to the second infinitary disjunction), from the alphabet $\{1, \ldots, n, *\}$, with the same restrictions as in the Proof of Theorem 3.1. And we take these strings in lexicographical order.

Next, we will consider the different sub-formulae which we used in defining Ψ_f.

- φ_F means *F is well formed for $\mathcal{M}_f$*, in the sense that it encodes the transition function of $\mathcal{M}_f$, restricted from the state q_I to the state q_T, as we indicated above.

- $\psi_{n,I_l}(x_1,\ldots,x_n)$ means *the σ'-structure is I_l and its size is n*. There is one formula of this kind for every structure $I_l \in \mathcal{E}_{\sigma',fin}$. These formulae are built by using the strings which we mentioned above, as encodings of the sub-indices for the variables $\{x_1,\ldots,x_n\}$, and then by building the atomic and the negated atomic formulae by using those sub-indices for the free variables for every relation in the structure. So that each one of these strings is a canonical representation of one structure in $\mathcal{E}_{\sigma',fin}$. This is the same construction which we have used in the Proof of Theorem 3.1.

- γ_{I_l,C_1^p} means *the state encoded in C_1^p is q_I and the input structure encoded in C_1^p is I_l*. There is one formula of this kind for every structure $I_l \in \mathcal{E}_{\sigma',fin}$, and for every possible length p of the computation of $\mathcal{M}_f$ on I_l. To build this kind of formulae we first define p variables, which will represent every cell in the tape. Then we consider every possible position of the tape head (that is, the second symbol @), and we use the string which represents I_l to ask for the corresponding amount of variables which represent consecutive cells in the tape to be assigned the constant 1, for every component of every tuple in every relation of the structure encoded in C_1^p, depending on the corresponding integer in the string.

- $\alpha_{C_{1,p}}$ means *$C_1^p,\ldots,C_p^p$ are well formed*. That is, each one of this relations has exactly one tuple which is a string over the alphabet $\{1,-,*,@\}$ and which has exactly two occurrences of the symbol @, separated by c_q symbols which are either 1 or $-$, and where all the occurrences of $-$ precede any occurrence of 1.

- $$\phi^c_{p,(i_1,\ldots,i_t)}(x_{i_1},\ldots,x_{i_t}) \equiv \exists^c z_1 \ldots z_t(\neg\bar{\delta}_{l,t}, \delta_{l,(i_1,\ldots,i_t)}(z_1,\ldots,z_t,x_{i_1},\ldots,x_{i_t}))$$

  Note that the variables $\{x_{i_1},\ldots,x_{i_t}\}$ are bounded in Ψ_f, and the variables $\{z_1,\ldots,z_t\}$ will be free in Ψ_f for I_l iff $I_l \models \bar{\delta}_{l,t}$.

  Where:

  - $$\delta_{l,(i_1,\ldots,i_t)}(z_1,\ldots,z_t,x_{i_1},\ldots,x_{i_t}) \equiv z_1 = x_{i_1} \wedge \ldots \wedge z_t = x_{i_t} \wedge \bigwedge_{1\leq j<p} \beta_{p,j} \wedge \pi_p \wedge \rho_{p,t} \wedge \tau_{n,p,(i_1,\ldots,i_t)}$$

  - $\bar{\delta}_{l,t}$ is a sentence, which is a sub-formula of $\delta_{l,(i_1,\ldots,i_t)}$:

    $$\bar{\delta}_{l,t} \equiv \bigwedge_{1\leq j<p} \beta_{p,j} \wedge \pi_p \wedge \rho_{p,t}$$

  - $\beta_{p,j}$ means *the transition from C_j^p to C_{j+1}^p corresponds to F*. That is, the content of the TM tape before step j in the computation of $\mathcal{M}_f$ is changed to the content before step $(j+1)$ according to the transition function encoded in F. And it includes also the checking of the possible change in the state and in the position of the tape head. There is one formula of this kind for every possible length p of a computation of $\mathcal{M}_f$ and for every step j in a computation. To build this kind of formulae we define p variables for C_j^p, another p variables for C_{j+1}^p, and b variables for F. We must consider every possible position of the pair of symbols @ in C_j^p, and there must be a tuple in F which represents a transition which allows the changes in the current cell symbol, state and tape head position, from the configuration represented by C_j^p to the configuration represented by C_{j+1}^p.

- π_p means *the state encoded in* C_p^p *is* q_T. There is one formula of this kind for every possible length p of a computation of $\mathcal{M}_f$. Its construction is trivial.

- $\rho_{p,t}$ means *the resulting relation encoded in* C_p^p *is of arity* t. There is one formula of this kind for every possible length p of a computation of $\mathcal{M}_f$, and for every possible arity t of the resulting relation. To build this kind of formulae we define p variables for C_p^p. Then we must consider every possible position of the pair of symbols @ which will indicate the beginning cell of the arity of the output relation. Finally, we just need to check that there are t consecutive symbols 1, and then a string ******.

- $\tau_{n,p,(i_1,\ldots,i_t)}$ means *the* t*-tuple* $(i_1,\ldots,i_t)$ *belongs to the resulting relation in* C_p^p. There is one formula of this kind for every size n of a σ'-structure, for every possible length p of a computation of $\mathcal{M}_f$, and for every possible t-tuple $(i_1,\ldots,i_t)$ of integers from $\{1,\ldots,n\}$. To build this kind of formulae we define p variables for C_p^p. Then we must consider every possible position of the pair of symbols @, and every possible subset of t-tuples of integers from $\{1,\ldots,n\}$. We must say that the output relation in C_p^p has that subset of tuples, and then the tuple $(i_1,\ldots,i_t)$. We must consider the separator *, and we must stop when trying to use the $(p+1)$-th variable.

Note that if $\bar{\delta}_{l,t}$ is FALSE then the variables $\{z_1,\ldots,z_t\}$ become bounded in $\phi^c_{p,(i_1,\ldots,i_t)}(x_{i_1},\ldots,x_{i_t})$ for I_l and, hence, also in Ψ_f for I_l. Further, the formula $\phi^c_{p,(i_1,\ldots,i_t)}(x_{i_1},\ldots,x_{i_t})$ is FALSE for every possible valuation of $\{x_{i_1},\ldots,x_{i_t}\}$ on the domain of I_l, because some of the conjuncts are FALSE. So, it will not provide a value TRUE for the second disjunction in Ψ_f.

As to the case of $t=0$, we have the sub-formula:

$$\phi_{p,0} \equiv \bigwedge_{1\le j<p} \beta_{p,j} \wedge \pi_p \wedge \lambda_p$$

Where $\beta_{p,j}$ and π_p are as before, and λ_p means *the arity of the resulting relation in* C_p^p *is* 0 *and its value is TRUE.* There is one formula of this kind for every possible length p of a computation of $\mathcal{M}_f$. To build this kind of formulae we define p variables for C_p^p. Then we must consider every possible position of the pair of symbols @, which will indicate the beginning of the arity of the resulting relation, and we must check that we have the string ******, which indicates arity 0, and then the right configuration for the value TRUE of a 0-ary relation according what we stated at the beginning of the present Proof.

It is clear that all this sub-formulae can be expressed in $\mathcal{L}_{\omega_1\omega}$, with the exception of the sub-formula $\phi^c_{p,(i_1,\ldots,i_t)}$, which can be expressed in $\mathcal{L}^c_{\omega_1\omega}$.

Roughly, the formula Ψ_f says that there is a TM with transition function F such that, for one σ'-structure, I_l, of some size $n \geq 0$, for which the constants $\{1, -, *, @\}$ have different interpretations, and for some finite number p of steps in the computation of the machine on I_l, there is a sequence of p configurations of that machine, with up to p used cells in its tape, $C_1, \ldots, C_p$, such that all of them are well formed, and C_1 has a canonical encoding of I_l, and for some integer $t \geq 0$, and for every possible t-tuple of integers $(i_1, \ldots, i_t) \in \{1, \ldots, n\}^t$, the t-tuple of elements of $I_l \quad (x_{i_1}, \ldots, x_{i_t})$, according to the canonical encoding of I_l for the indices of the variables, will be in the resulting t-ary relation if and only if the sequence of the machine configurations $C_1, \ldots, C_p$ represent a computation of the TM on the input structure I_l, according to the transition function F, and in C_1 the distinguished state q_I is the encoded state and in C_p the distinguished state q_T is the encoded state, and the tuple $(i_1, \ldots, i_t)$ is present in the resulting relation of the computation of the machine, which is encoded in C_p.

Then, as there is a TM, according to Ψ_f, which computes f on an input structure I_l encoded in its tape in a canonical form, and which writes an encoding of the output relation in its tape, there is also a GM_u machine which works as in the Proof of Theorem 3.1, which first writes a canonical encoding of the input structure in its TM tape, then proceeds to work as a pure TM, and finally it decodes the output relation from the TM tape. And the "pure" TM part of that GM_u machine is the one which Ψ_f says to exist.

Note that if the resulting relation for an input structure I_l is 0-ary, the formula Ψ_f becomes a sentence for I_l, which will be TRUE or FALSE according to the value of the resulting relation. The TRUE value would be a consequence of the sub-sentence $\phi_{p,0}$ being TRUE for I_l, while the FALSE value would result from the fact that no disjunct would be true on that structure. So, in the case of numerical queries the FALSE value will indicate that the query f is not defined for I_l. That is, $\mathcal{L}^c_{\omega_1\omega}$ can express the *whole* class $\mathcal{NQ}$, including the *partial* queries.

On the other hand, this is not the case for the rest of the class $\mathcal{CQ}$, since a result of a 0-ary relation with value FALSE is quite possible, and this would imply Ψ_f becoming a sentence for that input structure, but with the value FALSE. Note that a result of an empty relation with arity $t > 0$ is also possible, and this case would make Ψ_f a formula with t free variables for that input structure, and the t-ary relation $\Psi_f^{I_l}$ would be empty. So, $\mathcal{L}^c_{\omega_1\omega}$ can express $total(\mathcal{CQ} - \mathcal{NQ})$, that is, it cannot express partial queries which are not numerical. ■

Remark 4.1:

A much simpler proof for Theorem 4.1 was suggested by Tapani Hyttinen, showing that every total and isomorphism preserving global predicate f, being either typed or untyped, and being also either recursive or non recursive, can be expressed in the logic $\mathcal{L}^c_{\omega_1\omega}$. Next we give a sketch of this proof:

- Fix a signature σ. For every σ-structure I of size n, let Δ_I be the *diagram* of I, which we build over some bijective valuation of the variables $\{x_1, \ldots, x_n\}$ on $dom(I)$. Let $\varphi_{f,I}(z_1, \ldots, z_r)$ be the FO formula which defines the relation $f(I)$ on the structure I, and let r be the arity of the relation $f(I)$. Then, the following $\mathcal{L}^c_{\omega_1\omega}$ formula Ψ_f expresses the global predicate f:

$$\Psi_f \equiv \bigvee_{n\in\omega} \quad \bigvee_{I\in\mathcal{E}_{\sigma,fin}} \Delta_I \wedge \exists^c z_1 \ldots z_r(\neg\Delta_I, \varphi_{f,I}(z_1, \ldots, z_r))$$

However, for the definition of the fragment $\mathcal{L}^c_*$ of the logic $\mathcal{L}^c_{\omega_1\omega}$ which we claim that captures exactly the class $total(\mathcal{CQ})$, and mainly for the proof of the claim, we need Ψ_f to be of the form which we used in the proof of Theorem 4.1, so we can have a recursive function which builds the disjunct for every possible set of parameters for the different infinitary disjunctions. See Definition 5.1 and Theorem 5.1 in the next Section.

Note that, as we saw in the Proof of Theorem 4.1, regarding the class $\mathcal{NQ}$ we can express with the logic $\mathcal{L}^c_{\omega_1\omega}$ the *whole* class, including the *partial* queries. On the other hand, logicians in this field are used to consider only total queries, unless they work with three-valued logics.

However, we should point out that this is just a matter of definitions. With the logic $\mathcal{L}^c_{\omega_1\omega}$ we can express also the partial numerical queries because of the way in which we defined them in Section 2. But we could also consider partial queries in the subclass of typed queries in other logics. For instance, let's consider the logic FO with the addition of *partial fixed point*. The iteration defined by this quantifier is such that it doesn't necessarily stop when the formula is interpreted on a given structure. And we could then consider that the query which the formula expresses is a partial one, and that it is undefined for that structure. Though this is not the usual way in which it is considered.

On the other hand, if we consider the logic $\mathcal{L}_{\omega_1\omega}$, we could think of queries expressed by pairs of formulae in that logic.

Definition 4.4: A *partial query* q of finite relational signature σ in $typed(\mathcal{CQ})$ is *expressible* in the logic $\mathcal{L}_{\omega_1\omega}$, if there is a pair of $\mathcal{L}_{\omega_1\omega}$ σ-formulae $\langle \Psi_1, \Psi_2 \rangle$, where Ψ_2 is a sentence and Ψ_1 is an arbitrary formula, such that for every finite σ-structure I:

- $q(I)$ is defined iff $I \models \Psi_2$.
- If $q(I)$ is defined, then $q(I) = \Psi_1^I$, that is, it is the relation induced in I when the formula Ψ_1 is interpreted by I.

We could define expressibility of partial untyped queries in the logic $\mathcal{L}^c_{\omega_1\omega}$ in the same way.

Section 5: An Undecidable Characterization of the Total Computable Queries.

According Theorem 4.1, the logic $\mathcal{L}^c_{\omega_1\omega}$ strictly includes $total(\mathcal{CQ})$, so it would be very interesting to characterize the fragment of the logic which captures *exactly* that class. As to the logic $\mathcal{L}_{\omega_1\omega}$, no fragment that captures exactly $total\text{-}typed(\mathcal{CQ})$ is known either. So, in this Section we define such fragments for both of these logics.

Unfortunately, our characterization defines a formalism which is clearly undecidable in both cases, but we hope it can still be of interest, at least by giving some intuition for the search of a decidable formalism, if such a formalism exists. We think that for the case of $total(\mathcal{CQ})$ it most probably doesn't exist, because it is closely related to the class of Turing Machines which compute total functions, which is well known to be undecidable (see [DW,83], among others).

Definition 5.1:

Let's denote by $\mathcal{L}$ the sub-logic of the logic $\mathcal{L}^c_{\omega_1\omega}$ defined as the set of formulae Ψ of signature σ, for every finite relational signature σ, such that the following holds:

a) Ψ is of the form

$$\bigvee_{p_1 \in P_1} \cdots \bigvee_{p_h \in P_h} \alpha_{p_1,\ldots,p_h}$$

where

- For every $1 \leq i \leq h$, the set P_i is recursive.
- For every $1 \leq i \leq h$, every element in the set P_i can be encoded with a finite string over a finite alphabet.
- For every h-tuple $(p_1, \ldots, p_h) \in P_1 \times \cdots \times P_h$, the formula $\alpha_{p_1,\ldots,p_h}$ is in FO^c (see the remark after Definition 4.3 in Section 4).

b) For every σ-structure I, the set defined in the following way must be finite:

$$\bigcup_{(p_1,\ldots,p_h)\in P_1\times\cdots\times P_h} \text{FREEVAR}(\hat{\alpha}_{I,p_1,\ldots,p_h})$$

Where FREEVAR(φ) denotes the set of free variables in the formula φ, and $\hat{\alpha}$ is as in Definition 4.3.

That is, for every σ-structure I the resulting relation of the query expressed by Ψ must be of a finite arity.

c) There exists a TM M_Ψ which is total, and which with input $(p_1, \ldots, p_h) \in P_1 \times \cdots \times P_h$ and an encoding of a σ-structure I writes in its output tape the arity of the formula $\hat{\Psi}_I$ and the formula $\hat{\alpha}_{I,p_1,\ldots,p_h}$. Further, M_Ψ must preserve isomorphisms, so that the output does not depend on the encoding of the structure I. If any of the parameters happens to be inconsistent with the structure I, it writes a distinguished string in the output tape to indicate the special situation. On the other hand, if the number of free variables in $\hat{\alpha}_{I,p_1,\ldots,p_h}$ is greater than the computed arity for $\hat{\Psi}_I$, the machine M_Ψ halts in an error state.

Thus M_Ψ builds the formula in FO corresponding to $\alpha_{p_1,\ldots,p_h}$ for the structure I according to the Definition 4.3, and it also computes the arity of the output relation of the query expressed by Ψ on the structure I.

Now we define $\mathcal{L}^c_*$ as the fragment of $\mathcal{L}$ which is closed under complement, that is, for every $\Psi \in \mathcal{L}^c_*$ there is a formula *equivalent* to $\neg\Psi$ in $\mathcal{L}^c_*$.

Note that the specification of the TM machine M_Ψ is a finite representation of the infinitary formula Ψ. This approach is quite similar to the concept of *uniformity* for countably infinite families of boolean circuits (see [BDG,90] for the subject). The main difference here is that we are dealing with a single circuit, and that this circuit is not finite. The circuit which represents the infinitary formula is of countably infinite fan-in.

Definition 5.2: For every query q of signature σ in $total(\mathcal{CQ})$ and for every σ-structure I, we define the *complement query* $\bar{q}$ as follows:

$\bar{q}(I) = (dom(I))^r - q(I)$

Where $q(I)$ is of arity r.

Remark 5.1: Note that there are two facts which are straightforward, regarding complement queries:

- $q \in total(\mathcal{CQ}) \Longleftrightarrow \bar{q} \in total(\mathcal{CQ})$.
- For every $\Psi \in \mathcal{L}^c_*$, Ψ expresses q iff $\neg\Psi$ expresses $\bar{q}$.

Theorem 5.1: *Let $\mathcal{L}^c_*$ be as in Definition 5.1, then it captures exactly the class $total(\mathcal{CQ})$.*

Proof:

a) $total(\mathcal{CQ}) \subseteq \mathcal{L}^c_*$: Let $f \in total(\mathcal{CQ})$. Then, by Theorem 3.1 there exists a GM_u machine $\mathcal{M}_f$ which computes f. And this machine works according to the normal form we defined in that Proof. Then, there exists also a TM machine M_f which is the "pure" TM part of the computation of $\mathcal{M}_f$.

Now, by Theorem 4.1, there exists a formula Ψ_f which expresses f and which is of the form which is shown in the Proof of the Theorem. It is clear that we can put together the three infinitary disjunctions (for the parameters n, p and t, respectively, where all of them are defined in a recursive set, namely ω) and we get in this way the required form for the formula, according to Definition 5.1. Considering that formula, it is also clear that there exists a TM M_Ψ which on input $(n, p, t) \in \omega^3$, and a canonical encoding of a σ-structure I, as the one we used in the Proof of Theorem 3.1, writes the FO formula $\hat{\alpha}_{I,n,p,t}$ in its output tape, or a distinguished string to indicate that $n \neq |dom(I)|$, if this is the case. To write the FO formula, it first writes the corresponding FO^c formula in its work tape, and then, for every sub-formula of the form $\exists^c x(\delta, \varphi(x))$, M_Ψ evaluates δ on I and then rewrites the sub-formula according to Definition 4.3. Note that the requirement of a canonical encoding for the input of the structure enforces the preservation of isomorphisms. Note also that for the evaluation of the different sentences on I we can use the fact that the transition function which must be encoded in the relation F and which must be checked by the sub-formula φ_F in Ψ_f, is the transition function of the TM machine M_f, which exists as we saw above. This fact will also allow M_Ψ to evaluate on I the sub-formulae $\beta_{p,j}$ of Ψ_f.

The machine M_Ψ has to compute also the arity of the resulting relation $f(I)$. For this matter, it generates the sub-formula $\hat{\alpha}_{I,n,p_i,t_j}$ for every $(p_i, t_j) \in \omega^2$ and then evaluates it on I. It keeps on doing so until it gets the value TRUE for the sub-formula $\bar{\delta}_{l,t}$ in some $\hat{\alpha}_{I,n,p_i,t_j}$. This will mean that the arity of $f(I)$ is t_j. And then it can write t_j in its output tape. In the iteration M_Ψ increments every one of the two variable parameters by one, starting from one, and alternating the parameter being modified in every step in such a way that every possible pair can be reached in finite time. One way of achieving this is as follows:

for $p_i = 1$ to ω
  for $t_j = 1$ to $p_i + 1$
  for $k = 1$ to $p_j - 2$

Where k should be considered as a value for p_i, but without updating the current content of p_i. Of course we could consider in this case that according to the meaning of the parameters it should be always $p_i > t_j$, but we are interested in showing a general schema which does not depend on the meaning of the parameters, and which will be used below in this Proof.

This iteration will always stop because the query f is total, and because, as we stated above, M_f exists. Finally, note that by Remark 5.1 the complement query $\bar{f}$ is also in $\mathcal{L}^c_*$.

b) $\mathcal{L}^c_* \subseteq total(\mathcal{CQ})$: Let Ψ be a formula of signature σ in $\mathcal{L}^c_*$. To show that Ψ expresses a query in $total(\mathcal{CQ})$, we will show that there exists a GM_u machine $\mathcal{M}_\Psi$ which with a σ-structure I as input in its relational store, computes the query expressed by Ψ.

As $\Psi \in \mathcal{L}^c_*$ there exists a TM machine M_Ψ as in Definition 5.1. And as $\mathcal{L}^c_*$ is closed under complement, $\neg\Psi$ is also in the fragment and, hence, there is a machine $M_{\neg\Psi}$. By Remark 5.1 $\neg\Psi$ expresses the complement query of the query expressed by Ψ. The GM_u machine $\mathcal{M}_\Psi$ will make use of both machines, M_Ψ and $M_{\neg\Psi}$. It will generate a canonical encoding of the input structure I, as in the Proof of Theorem 3.1, in its TM tape, and it will also generate every h-tuple $(p_1, \ldots, p_h) \in P_1 \times \cdots \times P_h$ in its TM tape, for Ψ, and also every tuple corresponding to the parameters in the formula equivalent to $\neg\Psi$ in $\mathcal{L}^c_*$. The generation of the two sets of tuples of parameters will be carried on in such a way that every tuple can be generated in finite time, which can be done in a similar way as we did above for (p_i, t_j).

Before the generation of the tuples for the parameters, $\mathcal{M}_\Psi$ will compute and write the arity of Ψ on the σ-structure I in its TM tape, running the machine M_Ψ for this purpose.

Then, for every pair of parameters, $\mathcal{M}_\Psi$ will first run the two machines M_Ψ and $M_{\neg\Psi}$, and then it will evaluate the output formula from M_Ψ on the structure I and the output formula from $M_{\neg\Psi}$ also on the structure I. $\mathcal{M}_\Psi$ will update one auxiliary relation on its relational store for each one of the two machines. The arity of every auxiliary relation will be given by the formula which is being evaluated in both cases. As a new formula is evaluated, the arity for the corresponding auxiliary relation can change, but only being incremented, because the set of formulae written by each machine is actually a disjunction which is a sub-formula of Ψ or $\neg\Psi$, respectively.

Then the set of free variables can only either remain the same or be incremented.

As the arity of the relation induced in I by Ψ is already computed, and as it must be the same as for $\neg\Psi$, both auxiliary relations will eventually be of that arity, say r. At this point we begin to check, after every further evaluation of a new sub-formula, whether the union of the two auxiliary relations is $(dom(I))^r$. We should eventually reach this point because of the definition of the complement of a query.

Then the result of computing the query expressed by Ψ on the structure I is the auxiliary relation which we were using to keep the partial results of the evaluation of Ψ. ■

Regarding the logic $\mathcal{L}_{\omega_1\omega}$ and the class *total-typed*$(\mathcal{CQ})$, we can also define an analogous fragment. Let's define the fragment $\mathcal{L}_*$ of the logic $\mathcal{L}_{\omega_1\omega}$ in almost the same way as in Definition 5.1. Just consider that for every σ-structure I the arity of the relation induced by a given $\mathcal{L}_{\omega_1\omega}$ formula Ψ of the same signature when interpreted by I, is the same. Besides, the corresponding formula $\hat{\Psi}_I$ by Definition 4.3 is the formula Ψ itself. So, for the TM M_Ψ we don't need the structure I as input, and it will just build the FO sub-formula $\alpha_{p_1,\ldots,p_h}$, for every input $(p_1,\ldots,p_h) \in P_1 \times \cdots \times P_h$.

Then the following Corollary is straightforward.

Corollary 5.1: *The fragment $\mathcal{L}_*$ of the logic $\mathcal{L}_{\omega_1\omega}$ captures exactly the class total-typed$(\mathcal{CQ})$.*

Proof Sketch:

Regarding Theorem 3.1, it is clear that we can get the same kind of Normal Form for the class of machines GM^n, because we have used more than $O(n)$ variables only in the formula $\Psi'(z_1,\ldots,z_s)$, with s being the arity of the resulting relation. But now, as we are dealing with typed queries, the arity is fixed for the whole class of σ-structures, so s is $O(n)$.

As to the Theorem 4.1, we can then restrict $\mathcal{M}_f$ to be a GM^n machine. Now, regarding the formula Ψ_f, we can delete the infinitary disjunction which makes the arity of the query range over ω, as it is fixed. For the formula $\phi^c_{p,(i_1,\ldots,i_t)}$, we don't need to have a variable number of free variables, for the same reason, so we can restrict it to be a FO formula, as follows (where t is now a constant):

$$\phi_{p,(i_1,\ldots,i_t)}(x_{i_1},\ldots,x_{i_t}) \equiv \delta_{l,(i_1,\ldots,i_t)}(z_1,\ldots,z_t,x_{i_1},\ldots,x_{i_t})$$

Where $\delta_{l,(i_1,\ldots,i_t)}$ remains as in Theorem 4.1. Note that now every disjunct in Ψ_f has the same set of free variables, namely $\{z_1,\ldots,z_t\}$.

Finally, the modification of the Proof of Theorem 5.1 is straightforward. We must just restrict the machine $\mathcal{M}_f$ to be a GM^n machine, as we did above, and we must consider the remarks we made before the present Corollary. Note that Definition 5.2

is still valid for typed queries, and that Remark 5.1 remains true for the fragment $\mathcal{L}_*$.
■

Acknowledgements

My deep gratitude to Lauri Hella and Jouko Väänänen for letting me visit the Department of Mathematics of the University of Helsinki, as well as for their partial funding. Also warm thanks to Lauri Hella for the stimulating discussions we had, and mainly for his questions, which helped me to define a direction to follow in my research. He also spent a lot of time revising my work in many opportunities and he helped me with his deep knowledge. I would also like to thank Tapani Hyttinen for his important observation regarding Theorem 4.1.

References

[APV,94] Abiteboul, S., Papadimitriou, C. and Vianu, V., "The Power of the Reflective Relational Machine", in LICS, 1994.

[AV,91] Abiteboul, S. and Vianu, V., "Generic Computation and its Complexity", STOC 1991.

[AVV,92a] Abiteboul, S., Vardi, M. and Vianu, V., "Fixpoint Logics, Relational Machines, and Computational Complexity", preliminary version, 1992.

[AVV,92b] Abiteboul, S., Vardi, M. and Vianu, V., "Computing with Infinitary Logic", Research Report RJ 8925, 1992.

[BDG,90] Balcázar, J., Díaz, J., Gabarró, J., "Structural Complexity II", Springer Verlag, 1990.

[CH,80] Chandra, A. and Harel, D., "Computable Queries for Relational Data Bases", Journal of Computer and System Sciences 21, 1980.

[DLW,95] Dawar, A., Lindell, S., Weinstein, S., "Infinitary Logic and Inductive Definability over Finite Structures", Information and Computation, 119, 1995.

[DW,83] Davis, M. and Weyuker, E., "Computability, Complexity and Languages", corrected printing, Academic Press, 1983.

[Ebi,85] Ebbinghaus, H., "Extended Logics: The General Framework", in Model Theoretic Logics, ed. Barwise and Fefferman, Springer-Verlag, 1985.

[EF,95] Ebbinghaus, H., Flum, J., "Finite Model Theory", Springer Verlag, 1995.

[Flu,85] Flum, J., "Characterizing Logics", in Model Theoretic Logics, ed. Barwise and Fefferman, Springer Verlag, 1985.

[GG,95] Grädel, E., Gurevich, Y., "Meta Finite Model Theory", to appear in Information and Computation. Preliminary version in Logic and Computational Complexity, International Workshop LCC'94, USA, 1995.

[Gur,84] Gurevich, Y., "Toward Logic Tailored for Computational Complexity", in Computation and Proof Theory, Lecture Notes in Mathematics 1104, 1984.

[Tur,95] Turull Torres, J.M.,"Query Completeness, Distinguishability and Relational Machines", to appear in the Proceddings of the X Latin American Symposium on Mathematical Logic, Bogotá, 1995.

[Tys,96] Tyszkiewicz, J., "Queries and Algorithms Computable by Polynomial Time Existential Reflective Machines", preliminary report, October 1996, to appear in PODS 1997.