ON THE COMPLEXITY OF SEARCH QUERIES

Nicola Leone[1], Luigi Palopoli[2], Domenico Saccà[2]

[1] Information Systems Dept.
Technical University of Vienna
1040 Vienna, Austria
leone@dbai.tuwien.ac.at
[2] DEIS Dept.,
Universitá della Calabria,
I-87030 Rende (CS), Italy,
{palopoli,sacca}@unical.it

Abstract. This paper presents the definitions of several complexity classes of non-deterministic search queries. Classes are defined via two generalizations of the query output tuple problem (used for deterministic queries) to the context of non-deterministic queries. Several taxonomic results are then established. Furthermore, non-deterministic queries are shown not to be in general refinable (i.e., they cannot be replaced by) deterministic queries unless they loose "genericity" (i.e., the independence from data encoding). In other words, either the genericity is to be withdrawn or any 'meaningful' query language must be non-deterministic.
An other important issue is to recognize which queries classes can be efficiently implemented, i.e., one of solutions can be computed using a polynomial time single-valued function. We analyze two classes enjoying this property: the well-known class NQPTIME and a new class, NQPF. NQPTIME is shown to have a strange behavior: finding a solution of a query in the class can be done in polynomial time but testing a possible solution is not. NQPF does not suffer from this anomaly and, besides, it is captured by a query language.

Introduction

In complexity theory, problems can be roughly classified either as *decision* problems or as *search* problems. From the perspective of automatic computation, search problems are certainly more interesting than decision problems, in that one is usually more interested in computing a solution than just knowing whether a solution with a certain structure exists. Nevertheless, classic complexity theory focused on studying the properties of classes of decision problems (alias, boolean functions or languages) in the place of studying classes of search problems (alias, functions), the underlying assumption being that properties of classes of functions could be obtained fairly easily by extending corresponding results on languages. However, as shown by recent papers [13, 8, 14], this is not completely true. In moving from language complexity to function complexity, new ad hoc

formal analysis tools are needed to deal with non-determinism: indeed functions are in general multivalued [8].

A different framework would seem to arise for database query theory. Here the theory started with search queries which were classified using the so called query output tuple problem (QOT, for short), whereby the complexity of a query is defined as the complexity of establishing if a given tuple belongs to the result relation, for a given input database. The classical QOT works fine for deterministic queries, since the entire answer query answer can be contructed tuple after tuple, but fails for non-deterministic ones to measure the complexity of finding all the tuples of an answer as two tuples may belong to two different results. As a consequence, non-deterministic queries have not been the subject of a systematic study. Some interesting exceptions are represented by some recent papers [3, 2, 9].

The aim of this paper is to supply a systematic study of structural properties of complexity classes of search queries. To pinpoint the complexity of non-deterministic search queries, we define two extension of the QOT: the QOS (query output subset) problem that is based on the cost of testing whether a relation is a subset of a query result, and the QOR (query output relation) problem that is based on the cost of testing whether a relation is exactly a query result. Both of them can be used to define query complexity classes in this more general framework. A first contribution of this paper is to provide taxonomic results for various classes of queries thus defined.

An important notion in the evaluation of function complexity is the function refinement: given a multivalued function f, a function f' refines f if f' has the same domain as f and, for each input x, $f'(x)$ is a subset of $f(x)$. Of particular practical interest are single-valued functions refining multivalued ones. Indeed, when asking for solving the search problem defined by a multivalued function (e.g., find a minimal spanning tree of an input graph), we are usually interested in obtaining *one* of its solutions. A simple, yet important result is that a non-deterministic query cannot be in general refined by a deterministic one because of the genericity requirement assumed for queries. This means that non-determinism in the queries cannot be given up as long as they are required to be generic, i.e., independent from the encoding of data. In particular, deterministic queries cannot express most of the search problems. Our belief is that either genericity requirement is eventually withdrawn, or any "practical general-purpose" query language must be non-deterministic.

The fact that a query language must be non-deterministic is not a negative result 'in se'. Coexisting with non-determinism is not unusual in computer science, provides a 'declarative' style to express problems and does not refrain from effectively solving them. Indeed a resolution of a problem (i.e., a non-deterministic query) can be carried out by selecting and computing a single-valued function refining the query, i.e., returning one of the possible results. (This implementation is typically done by the database system in a way transparent to the query user.) Thus, while genericity and non-determinism are necessary in the formulation of a query, they can be eventually removed during its execution.

An important question is then: which classes of non-deterministic queries can be refined by a polynomial-time single-valued function, i.e., can be executed in polynomial-time?

A class enjoying this property and already known in the literature [3] is NQPTIME, that is, the class of all queries computed by polynomial-time transducers such that for each input, each branch of the transducer's computation halts into an accepting state. As shown in the paper, this class suffers from a severe drawback: the QOR is not in P unless P=NP. Therefore, while finding a result is done in polynomial-time, testing whether a given relation is a result is not — the typical situation is the reverse: the hard part is finding a result rather than testing it! We then define a new class NQPF as the class of all queries whose results are both computable and verifiable in polynomial time. As such, queries in this class have a very nice characterization: both the QOR and the QOS for them are in P. A scheme of language capturing NQPF is finally presented.

The rest of the paper is organized as follows. In the next section we recall some preliminary concepts about language and function complexity as well as about database queries . Section defines non-deterministic query classes and present some taxonomic characterization of them. In Section we discuss query refinement. More in details, in Subsection we show that non-deterministic queries are not in general refinable by deterministic ones, whereas in Subsection we discuss classes of non-deterministic queries refined by polynomial-time single-valued functions. In the latter subsection we also introduce the class NQPF and compare it with the class NQPTIME. Finally, in Section , we present a scheme of a query language capturing NQPF.

Preliminaries

Complexity

In classifying query classes, we shall make use of two basic computational models: acceptors and transducers that may be either *non-deterministic* or *deterministic*. Given a finite *alphabet* Σ with at least two elements, an *acceptor* is a (possibly, non-deterministic) Turing machine A on Σ with two types of final states: *accepting* and *rejecting*. For any string $x \in \Sigma^*$ given on the input tape of an acceptor A, if there exists at least one computation of A that terminates into an accepting state, then x is said to be *accepted* by A. A is said to *recognize* the language consisting of all strings accepted by A. A *transducer* is a Turing machine T which, in addition to possibly accept a string x, writes a string y on the output tape before entering an accepting state. For each x accepted by T, we denote by $T(x)$ the set of all strings that are written by T in all possible accepting computations.

Throughout the paper, we shall be mainly referring to the following classes of languages: deterministic and non-deterministic Logspace (denoted by L and NL, resp.), deterministic and non-deterministic polynomial time (denoted by P and NP, resp.), the complement of NP (coNP), and polynomial space (PSPACE).

The reader can refer to [10, 11] for excellent sources of information on this subject.

Next, we recall the definition of several complexity classes of functions. The source for this material is mainly [13, 14]. Let $f : \Sigma^* \mapsto \Sigma^*$ be a partial multi-valued function. Let $f(x)$ stand for the set of possible outcomes (*results*) of the function f when applied to the input string x. Thus, we write $y \in f(x)$, if y is a value of f on the input string x. Define $dom(f) = \{x \mid \exists y(y \in f(x))\}$ and $graph(f) = \{\langle x, y\rangle \mid x \in dom(f),\ y \in f(x)\}$. If $x \notin dom(f)$, we will say that f is undefined at x. The function f is *total* if $dom(f) = \Sigma^*$. Any non-deterministic transducer T *computes* the partial multi-valued function f such that for each x, $f(x) = T(x)$ if x is accepted by T or otherwise f is undefined at x.

PF is the class of all single-valued functions computed by deterministic polynomial-time bounded transducers. The class NPMV is defined as the set of all multivalued functions computed by non-deterministic polynomial-time bounded transducers. It is known that a function is in NPMV if and only if it is both polynomially balanced (i.e., for each x, the size of each result in $f(x)$ is polynomially bounded in the size of x) and $graph(f)$ is in NP. By analogy, the class coNPMV is defined as the class of partial multivalued functions g which are polynomially-length bounded and such that $graph(g)$ is in *co*NP [8]. Given a class of functions FC, the class FC^T denotes the set of all total functions in FC.

Relational databases and queries

We are given a *universe* U (that is a countable set of constant symbols) and a countable set S of relation symbols with given finite arities. Let r be any relation symbol in S, say with arity k: a *tuple on* r is any element of U^k, a *relation on* r is any finite set R of tuples on r, and the set of all relations on r is denoted by $inst(r)$. A (*relational*) *database scheme* $\mathbf{d}$ is a tuple $\langle r_1, \ldots, r_m\rangle$ of different relation symbols. A (*relational*) *database* D on $\mathbf{d}$ is a tuple $\langle R_1, \ldots, R_m\rangle$, where for each i, $1 \leq i \leq m$, $R_i \in inst(r_i)$. The set of all databases on $\mathbf{d}$ is denoted by $inst(\mathbf{d})$.

A (*database non-deterministic*) *query* $Q =< \mathbf{d}, g >$, where $\mathbf{d}$ is a database scheme and g is a relation symbol not in $\mathbf{d}$, is a partial recursive (i.e., computable), generic multivalued function from $inst(\mathbf{d})$ to $inst(g)$; for each $D \in inst(\mathbf{d})$, the set of results $Q(D)$ is the *answer* of the query on D. Genericity for Q means that for any D on $\mathbf{d}$ and for any isomorphism ρ on U, $Q(\rho(D)) = \rho(Q(D))$ [5, 15]. Informally speaking, the answer of a query does not depend on the internal representation of the constants in the database. An important consequence of genericity is that Q is a polynomially balanced function. To see a query as a function, we assume a standard encoding of both the input database and the answer relations in the form of strings. Hence, in what follows, we shall freely talk about classes of queries being subsets of classes of functions.

A query $Q =< \mathbf{d}, g >$ will be said *deterministic* if it is single valued; if g has arity 0, the query Q is said to be *boolean* or *bound*.

We denote by **NQ** the set of all (non-deterministic) queries and by **Q** the set of all deterministic queries; moreover, $\mathbf{NQ}^T$ and $\mathbf{Q}^T$ denote their restrictions to total queries. Obviously $\mathbf{Q} \subset \mathbf{NQ}$ and $\mathbf{Q}^T \subset \mathbf{NQ}^T$.

Throughout the paper, we assume that an input database D_G with schema (e, v) is given representing graphs in a monadic relation v and a binary relation e, encoding vertices and edges in the obvious way.

Non-deterministic query classes

Classical complexity theory classifies languages on the basis of how difficult is to decide that a given input string belongs to the language. Deterministic search queries have been classified in a similar fashion, defining a recognition problem associated to them: given a deterministic query Q and a database D over a fixed scheme $\mathbf{d}$, the *query output tuple recognition problem* (QOT) for $Q(D)$ amounts to determining whether a given tuple t belongs to the unique result in $Q(D)$. The QOT is an appropriate tool for defining deterministic query complexity classes since the overall result relation can be constructed by iteratively checking the QOT on all tuples (whose number is polynomially bounded).

However, the QOT is not appropriate for classifying non-deterministic queries as two tuples may belong to two different results returned by the query on the same input and, as a consequence, does not allow to construct any of the query result relations.

A natural generalization of the QOT to non-deterministic queries is the *Query Output Subset* problem (QOS for short). Given a non-deterministic query Q, a database $D \in inst(\mathbf{d})$, and a relation S, the QOS for Q amounts to determining whether there exists $R \in q(D)$ such that $S \subseteq R$, i.e., whether S is a subset of a possible result of evaluating Q over D. In fact, our first result shows that the QOS coincides with the QOT over deterministic total queries and, as for the QOT, the QOS allows to construct answer relations of non-deterministic queries , as follows. Begin with $S = \emptyset$. At each step, invoke the QOS for $S' = S \cup \{t\}$, for each tuple t of appropriate arity, and set S to S' if the answer is positive. This process allows us to incrementally construct results of the queries. In case of non-deterministic queries, possible relations that are included in other results are not generated by this process — we shall discuss this anomaly below in this section.

Proposition 1. *Let Q be a deterministic query and C be a complexity class closed under conjunction. Then,*

1. *if the QOS for Q is in C then the QOT for Q is in C, and*
2. *if Q is total and the QOT for Q is in C then the QOS for Q is in C.* ∎

Another possibility for classifying non-deterministic queries is to generalize, to the framework of database queries, the so called *graph-recognition problem*, which is used for classifying the complexity of multivalued functions[13].

This corresponds to adopt the *Query Output Relation* problem (QOR for short), which, given a non-deterministic query Q and a database $D \in inst(\mathbf{d})$, and a relation R, amounts to determining whether $R \in Q(D)$.

Both QOR and QOS can be used to define query complexity classes, as follows. Let C be a complexity class of languages. Then define the query class $NQ^{\underline{s}}C$ (resp., $NQ^{\underline{r}}C$) as the set of queries $Q \in \mathbf{NQ}$ such that the QOS (resp., QOR) for Q can be solved within C.

Observe that $NQ^{\underline{r}}$NP and $NQ^{\underline{r}}co$NP correspond to the generic functions of the class NPMV and coNPMV, respectively; moreover, $NQ^{\underline{r}}$P, corresponds to the generic functions in NPMV_g, that is the set of all functions in NPMV whose graph is verifiable in polynomial time — this class is also called FNP in the literature.

Our next concern is to find relationships among classes of queries defined by means of the QOS and the QOR, respectively — such relationships for the case of deterministic classes have been studied in [12]. To this end, let us first give the following definitions. A query Q is *maximal* if, for any input database D, and for each pair of results R_1, R_2, $R_1 \subset R_2$ never holds. Given a class QC of queries, the set of all maximal queries in QC is denoted by $(QC)_{MAX}$.

Proposition 2. *Let $C \supseteq$ L be a complexity class of languages closed under complementation. Then, $(NQ^{\underline{s}}C)_{MAX} \subseteq NQ^{\underline{r}}C$.*

Proof. Let $Q \in (NQ^{\underline{s}}C)_{MAX}$. We have to show that, given a relation R, the QOR for Q over R can be solved in C. We proceed as follows. First, the QOS is invoked over R. If the answer is negative, then we stop, since also the answer to the QOR over R will be negative. Otherwise, from the active domain we take each tuple $t \notin R$ one at a time and invoke the complementary problem of the QOS over $R \cup \{t\}$. This test can be done for we are assuming C closed under complementation; moreover, as $C \supseteq$ L, the iteration over all tuple can be done as well. Since Q is maximal, we get at least one negative answer if and only if the the answer to the original QOR is negative. ∎

The proposition above holds in particular for the classes L, NL, P, PSPACE. To see that maximality is necessary for Proposition 2, take a non-maximal query Q in $NQ^{\underline{s}}$P. Then every maximal result is recognized in Ptime. We may choose Q in such a way that every subrelation of a result is itself a result iff some PSPACE-complete condition is satisfied.

Theorem 3. *The following relationships hold:*

1. $(NQ^{\underline{r}}\text{NP})_{MAX} \subseteq (NQ^{\underline{s}}\text{NP})_{MAX}$*; the containment is strict unless* NP = *co*NP*;*
2. $(NQ^{\underline{s}}co\text{NP})_{MAX} \not\subseteq NQ^{\underline{r}}co\text{NP}$*, unless* NP = *co*NP*;*
3. $(NQ^{\underline{r}}co\text{NP})_{MAX} \not\subseteq (NQ^{\underline{s}}co\text{NP})_{MAX}$*, unless* NP = *co*NP*;*
4. $(NQ^{\underline{s}}co\text{NP})_{MAX} \not\subseteq NQ^{\underline{r}}\text{NP}$*, unless* NP = *co*NP*;*

5. *for each complexity class* $C \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$, $(NQ^{\underline{s}}C)_{MAX} \subseteq (NQ^{\underline{r}}C)_{MAX}$ *and the containment is strict unless* $C = \mathrm{NP}$*;*
6. $NQ^{\underline{s}}PSPACE = NQ^{\underline{r}}PSPACE$.

Proof. (Sketch)

1. Let $Q \in (NQ^{\underline{r}}\mathrm{NP})_{MAX}$. Given a set S of tuples, we can decide that S is a subset of a possible result of Q (on the given input database) as follows. Guess a relation R and its polynomial certificate $C(R)$, and check (in polynomial time) that (i) $C(R)$ certificates that R is a result relation of Q, and (ii) $S \subseteq R$.

   To prove strictness, consider the query *AllK* that, given an input graph G, returns the union of all the kernels of G; if G has no kernel, then the empty set is returned. *AllK* is in $(NQ^{\underline{s}}\mathrm{NP})_{MAX}$. Indeed, to recognize that a set of nodes S (represented by a set of tuples) is a subset of the result relation, it is sufficient to guess sets of nodes $N_1, ..., N_c$ ($c \leq |S|$) of nodes and verify (in polynomial time) that (i) N_i is a kernel ($1 \leq i \leq c$), and (ii) $S \subseteq \cup_{1 \leq i \leq c} N_i$. Now, if *AllK* were in $(NQ^{\underline{r}}\mathrm{NP})_{MAX}$, then the *co*NP-complete problem of deciding whether a nonempty graph G *has not* a kernel would be in NP (as G has not a kernel iff the empty relation is the result of *AllK*). Therefore, $(NQ^{\underline{r}}\mathrm{NP})_{MAX} = (NQ^{\underline{s}}\mathrm{NP})_{MAX}$ would imply NP = *co*NP.
2. The query *InK* that, given an input graph G, returns the intersection of all the kernels of G is in $(NQ^{\underline{s}}co\mathrm{NP})_{MAX}$ (if G has no kernel, then the set of all nodes is returned). Indeed, we can recognize that set S is not in the intersection of the kernels as follows: guess a set N of nodes, check that N is a kernel and $S \not\subseteq N$. *InK* is not in $NQ^{\underline{r}}co\mathrm{NP}$, unless NP = *co*NP. Indeed, if *InK* were in $NQ^{\underline{r}}co\mathrm{NP}$, then we could decide if a nonempty graph G *has* a kernel (NP-complete problem) in *co*NP. To show this, we build a new graph $G' = \langle e', v' \rangle$, where $v' = v \cup \{n\}$ ($n \notin v$) and $e' = e \cup \{(n, x) \mid x \in v\} \cup \{(x, n) \mid x \in v\}$. Clearly, $\{n\}$ is a kernel for G' and n does not belong to any other kernel of G'. Moreover, a subset M of nodes in v is a kernel of G iff it is a kernel of G'. Therefore, checking the existence of a kernel of the input graph G is equivalent to verifying that the empty set is the result of *InK* on G'.
3. Consider the query *Ex1K* that, given an input graph G, returns the unique kernel of G, and is undefined on graphs which do not have a unique kernel. *Ex1K* is in $(NQ^{\underline{r}}co\mathrm{NP})_{MAX}$. Indeed, we can check in NP that a given set N of nodes *is not* the unique kernel of G as follows. Guess a set M of nodes, then check (in polynomial time) that: (1) either M is a kernel of G different from N, or (2) N is not a kernel. However, *Ex1K* is not in $(NQ^{\underline{s}}co\mathrm{NP})_{MAX}$ (unless D^P = *co*NP). If *Ex1K* were in $(NQ^{\underline{s}}co\mathrm{NP})_{MAX}$, then the D^P-complete problem of deciding whether a graph G has a unique kernel would be in *co*NP. To show this, consider the graph $G' = \langle e, v \cup \{n\} \rangle$, where n is a node which is not in v. Since n is an isolated node, a subset M of nodes in v is a kernel of G iff $M \cup \{n\}$ is a kernel of G'. Therefore,

checking that G has a unique kernel (D^P-complete problem) is equivalent to verifying that $\{n\}$ is a subset of the result of *Ex1K* on G'.
4. It can be shown using an argument similar to the one in the proof of Point 2 above.
5. The containment follows from Proposition 2. To see that the containment is 'very likely' strict, consider the query *HC* returning a Hamiltonian circuit. The QOR for this query is in L but deciding whether the empty set is a subset of some solution (and, therefore, a Hamiltonian circuit does exists) is NP-complete.
6. $(NQ^{\underline{s}}\text{PSPACE})_{MAX} \subseteq (NQ^{\underline{r}}\text{PSPACE})_{MAX}$ follows from Proposition 2. It is now easy to see that the maximality restriction is not necessary for PSPACE. By applying an argument similar to the one in the proof of part (1) above, we derive that $NQ^{\underline{r}}\text{NPSPACE} \subseteq NQ^{\underline{s}}\text{NPSPACE}$. But, as PSPACE = NPSPACE, we obtain $NQ^{\underline{r}}\text{PSPACE} \subseteq NQ^{\underline{s}}\text{PSPACE}$. ∎

Example 1. Examples of queries belonging to query complexity classes we have defined are reported in the proofs of results previously presented. Some further examples are given next.

1. The query *Ker* returning a kernel of the input graph G is in $NQ^{\underline{r}}\text{P}$. *Ker* is also in $NQ^{\underline{s}}\text{NP}$, but not in $NQ^{\underline{s}}\text{P}$, unless P = NP.
2. The query *NodK* that, given an input graph G, returns a vertex belonging to some kernel of G is in $NQ^{\underline{r}}\text{NP}$ but not in $NQ^{\underline{r}}\text{P}$ (unless P = NP).

Query refinement

Non-deterministic queries cannot be refined

Given a multivalued function f, it is often interesting to establish whether there exists a single-valued function f' which refines f, i.e., for each input x in $dom(f)$, f' returns one of the possible results in $f(x)$. Indeed, when asking for solving the search problem defined by a multivalued function (e.g., find a minimal spanning tree of an input graph), we are usually interested in obtaining *one* of its solutions. Therefore, in the actual computation, the multivalued function is eventually replaced by a refining, single-valued one (e.g., to actually compute a minimal spanning tree, we may use Prim's or Kruskal's classical algorithm). We can then say that, as far as function computation is concerned, non-determinism can be subsumed by determinism. We next show that this is not the case for query languages: a non-deterministic query cannot be in general refined by a deterministic one. In other words, a search problem cannot be defined by a single-valued, generic function. For instance, finding a spanning tree cannot be formulated as a deterministic query since there is no way to single out one solution without fixing some ordering and, then, loosing genericity — for the Prim's or Kruskal's algorithm such a characterization remains obscure (and hardwired in the actual writing of the algorithm) even after introducing some ordering!

Let us first of all recall the formal definition of refinement [13]. Given two partial multivalued functions f and g, define g to be a *refinement* of f if $dom(g) = dom(f)$ and $graph(g) \subseteq graph(f)$. Let F and G be two classes of partial multivalued functions. Let f be a partial multivalued function. Following [13], we define $f \in_c G$ if G contains a refinement of f. Moreover, we define $F \subseteq_c G$ if, for all $f \in F$, $f \in_c G$.

In the previous sections, we have defined the notion of maximal queries, those never returning pairs of results included either ways into one another. Refinement allows us to introduce a further stronger concept of 'regularity' of a query, which is given next. We say that a query Q is *reduced* if there exists no other query $Q' \neq Q$ such that Q' refines Q. Informally, a non-reduced query actually consists of the combination of two or more queries, possibly unrelated to each other, and, therefore, it may not correspond to a well-stated problem. Given a query class QC, $red(QC)$ is the set of all reduced queries occurring in QC; QC is *reduced* if $QC = red(QC)$.

Proposition 4. *If Q is a reduced query then it is maximal.* ∎

In general, the converse does not hold. Deterministic queries are obviously reduced. We are now ready to show that non-deterministic queries cannot be refined by deterministic ones; this holds also for total or reduced queries.

Proposition 5. $\mathbf{NQ} \not\subseteq_c \mathbf{Q}$, $red(\mathbf{NQ}) \not\subseteq_c \mathbf{Q}$, $\mathbf{NQ}^T \not\subseteq_c \mathbf{Q}^T$ *and* $red(\mathbf{NQ}^T) \not\subseteq_c \mathbf{Q}^T$.

Proof. (Sketch) It is sufficient to prove $red(\mathbf{NQ}^T) \not\subseteq_c \mathbf{Q}^T$. Consider the following query *OneV* that, given an input graph, returns a set containing one single vertex of the graph (or the empty set if the graph is empty). *OneV* is obviously in $red(\mathbf{NQ}^T)$. Now, consider any single-valued refinement f of *OneV*. We show next that f is not a query. Consider a specific input graph G with vertices $\{v_1, \ldots, v_n\}$. The function f maps G into a set containing just one of its vertices, say v_i. Then, consider the isomorphism ρ that maps v_i to v_j $(i \neq j)$ and v_j to v_i and is the identity elsewhere. Then, $\rho(f(G)) = \{v_j\} \neq \{v_i\} = f(\rho(G))$. ∎

The above statement rephrases the fact that there exists no generic choice function. We use this well known fact to conclude that deterministic database query language cannot replace non-deterministic ones. The following results illustrate the fact that the statement of the proposition above carries over to query classes.

Proposition 6. *For each complexity class* $C \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}, co\mathrm{NP}, PSPACE\}$, $NQ^{\underline{s}}C \not\subseteq_c Q^{\underline{s}}C$. ∎

Polynomial-time refinements

Consider any non-deterministic query Q. Since Q is a multivalued function, the actual computation of Q consists in selecting a single-valued function f refining

Q and, then, in computing f — indeed, the selection and computation of f is typically provided by the underlying database system in a way transparent to the user. Therefore, an important practical question is whether a result of a given query can be efficiently computed by means of a polynomial-time, single-valued function — the function need not be generic since it is only used for implementation. To answer this question, we next evaluate which query classes are refined by PF.

Theorem 7.

1. *For each complexity class* $C \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}, co\mathrm{NP}, \mathrm{PSPACE}\}$, $NQ^{\underline{r}}C \not\subseteq_c \mathrm{PF}$, *unless* $\mathrm{P} = \mathrm{NP}$.
2. *For each complexity class* $C \in \{\mathrm{NP}, co\mathrm{NP}, \mathrm{PSPACE}\}$, $(NQ^{\underline{s}}C)_{MAX} \not\subseteq_c \mathrm{PF}$, *unless* $\mathrm{P} = \mathrm{NP}$.
3. *For each complexity class* $C \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$, $(NQ^{\underline{s}}C)_{MAX} \subseteq_c \mathrm{PF}$.

Proof. (Sketch)

1. Consider the query *HC* which returns an Hamiltonian circuit of the input graph (if any) and is undefined on graphs having no Hamiltonian circuit. *HC* is in $NQ^{\underline{r}}\mathrm{L}$. However, if *HC* would have a refining function in PF then P would coincide with NP, as we would solve the NP-complete problem of checking if a graph has an Hamiltonian circuit in polynomial time.
2. To prove the result for NP (resp., coNP) it is sufficient to consider the query *AllK* (resp., *InK*), as defined in the proof of Theorem 3. The result for PSPACE then follows.
3. To show this, it is sufficient to consider the construction process of a query result relation through its QOS, as illustrated in Section . ∎

Being refined by PF, $(NQ^{\underline{s}}\mathrm{P})_{MAX}$ turns out to be the largest desirable class among those so far analyzed. But which is the largest, 'natural' query class that is refined by PF? Thus, what is the most appropriate non-deterministic extension of QPF, the class of all deterministic polynomial-time queries? A possible candidate is the class $NQ^{\underline{s}}P \cap NQ^{\underline{r}}P$ that is the set of all queries for which both the QOS and the QOR are solvable in polynomial time. From now on, we shall denote this class by NQPF; obviously QPF, known in the literature as QPTIME, is the deterministic fragment of NQPF.

Proposition 8.

1. $\mathrm{NQPF} \subseteq_c \mathrm{PF}$;
2. $(\mathrm{NQPF})_{MAX} = (NQ^{\underline{s}}P)_{MAX}$. ∎

Thus NQPF is indeed refined by PF and $(NQ^{\underline{s}}P)_{MAX}$ includes all maximal queries in NQPF. In Section we present a language that captures NQPF; some hints on the proof of the above theorem can be taken from the structure of

this language. Observe that any partial query in NQPF can be made total by a suitable encoding of undefinedness.

There is another class that is refined by PF, namely the class NQPTIME of [2, 3], that is the class of all queries computed by polynomial-time transducers such that for each input, each branch of the transducer's computation halts into an accepting state. Note that the definition of NQPTIME has been originally given as 'the class of all non-deterministic database transformations which can be computed by a non-deterministic Turing machine in polynomial time' [3]. Thus, since a non-deterministic database transformation corresponds to a total query, the existence of an acceptance state for every branch was not explicitly required. However, this requirement was then implicitly enforced in characterizing query languages capturing NQPTIME (e.g., inflationary fixpoint augmented with a non deterministic construct, the witness). From then on, the above requirement is to be 'de facto' added to the definition of the class; otherwise NQPTIME would coincide with the larger class $(NQ^{\underline{r}}\text{NP})^T$ (see, Theorem 12 below) and the related expressiveness results would be wrong.

Theorem 9. NQPTIME $\subseteq_c$ PF *and* $red(\text{NQPF}^T) = red(\text{NQPTIME})$. ∎

As shown next, a severe drawback of a query in NQPTIME is that finding a result is done in polynomial-time but testing whether a given relation is a result is not (unless P=NP). This contrasts with the typical situation in classical NP-hard search problems: the hard part is finding a result rather than testing it.

Theorem 10. NQPTIME $\not\subseteq NQ^{\underline{r}}\text{P}$ *unless* $\text{P} = \text{NP}$.

Proof. (Sketch) Consider the following query *01K* that, given a graph G, returns $\{0\}$ if G has no kernel, $\{1\}$ if every subset of vertices of G is a kernel, both $\{0\}$ and $\{1\}$ otherwise. Clearly, *01K* is in NQPTIME: the transducer first generates a subset of vertices of G and then outputs $\{1\}$ or $\{0\}$ according to whether this subset is a kernel or not (this computation is obviously polynomial-time). If *01K* were in $NQ^{\underline{r}}\text{P}$ then we could check in polynomial time if a graph has a kernel – as the graph has a kernel iff $\{1\}$ is a result of *01K* – and, therefore, P would coincide with NP. ∎

By removing the above anomaly, we get a fragment of NQPTIME that includes all total maximal queries of NQPF.

Proposition 11. $(\text{NQPF}^T)_{MAX} = (\text{NQPTIME} \cap NQ^{\underline{s}}\text{P})_{MAX}$.

Let us now show that the class NQPTIME is almost surely separated from the class $(NQ^{\underline{r}}\text{NP})^T$. This confirms that the requirement about transducer's acceptance over every computation branch (cited above while discussing the definition of NQPTIME) is mandatory.

Theorem 12. NQPTIME $\subseteq (NQ^{\underline{r}}\text{NP})^T$*; the containment is strict unless* $\text{P} = \text{NP} \cap co\text{NP}$.

Proof. (Sketch) By definition, NQPTIME $\subseteq (NQ^{\underline{r}}\text{NP})^T$. To show the strictness, we prove the following claim.

Claim 13 $(NQ^{\underline{r}}\text{NP})^T \subseteq_c \text{PF}$ *implies* $\text{NPMV}^T \subseteq_c \text{PF}$.

Proof. (Sketch) Assume by contradiction that there exists f in NPMV^T such that $f \not\in_c \text{PF}$. Therefore, there exists at least one input x such that no element of $f(x)$ can be computed in PF. Now, consider the following function g:

$g(x') = f(x')$ if there exists an isomorphism ρ s.t. $\rho(x) = x'$

$g(x') = 0$ otherwise.

Clearly, g is a query. Moreover, g cannot be refined in PF otherwise an element of $f(x)$ would be computable in PF (a contradiction). This concludes claim's proof.

We can then resume the proof of Theorem 12. We assume by contradiction that $(NQ^{\underline{r}}\text{NP})^T \subseteq$ NQPTIME. Then, from Theorem 9, it follows that $(NQ^{\underline{r}}\text{NP})^T \subseteq_c$ PF. It then follows from the above claim and results in [7] that $\text{P} = \text{NP} \cap co\text{NP}$. This closes the proof. ∎

To conclude with the taxonomic characterization of NQPTIME, we next relate this class with the classes of total queries defined by a QOR in *co*NP.

Theorem 14.

1. NQPTIME $\not\subseteq (NQ^{\underline{r}}co\text{NP})^T$ *unless* NP $\subseteq co$NP.
2. $(NQ^{\underline{r}}co\text{NP})^T \not\subseteq$ NQPTIME *unless* $\text{P} = \text{P}^{NP[O(log\ n)]}$. ∎

(Skecth)
Point 1. Consider again the NQPTIME query *01K* described in the proof of Theorem 10. If *01K* were in $(NQ^{\underline{r}}co\text{NP})^T$ then we could check in *co*NP if a graph has a kernel – as the graph has a kernel iff $\{1\}$ is a result of *01K* – and, therefore, every problem in NP would be solvable in *co*NP.
Point 2. Consider the query *MinMod* that, given a positive CNF T theory [3], returns a minimal model of T. *MinMod* is total since positive CNF theories always have at least one minimal model. Moreover, it is in $NQ^{\underline{r}}co$NP, since to check that a given set of literals is a model of an input positive CNF theory is in *co*NP. Now, assume by contradiction that *MinMod* is in NQPTIME. Then, since NQPTIME $\subseteq_c$ PF by Theorem 9, it exists a single-valued function $f \in$ PF that returns a minimal model of the input theory. This latter task was proved hard for $\text{P}^{NP[O(log\ n)]}$ in [4]. This concludes the proof. ∎

A language for NQPF

We have shown above that NQPF has some desirable properties. However, something important is still missing: we also need to show that this class can be

[3] A CNF theory is positive when it does not include any clause where no positive literal occurs.

captured by some query language. Such a language is introduced next.

Our language is first order + inflationary fixpoint hosted in a prefixed 'while' framework with a fixed control structure and the following operators:

1. *choice_tuple*(G) non deterministically selects a tuple for the result relation G;
2. *retract_tuple*(G) removes the last chosen tuple for G;
3. *exists_choice*(G) returns 'false' iff all possible tuples have been already chosen for G;
4. *choice_order*(O) non deterministically selects an order relation O for the active domain;
5. *choice_bool*() non deterministically returns 'true' or 'false';
6. $verify\_rel(G, O)$ (resp., $verify\_sub(G, O)$) fires the fixpoint query implementing the QOR (resp., QOS) for G, given the order relation O.

Observe that, because of the given order, the fixpoint queries can perform any polynomial time test. Instead of first order + non-inflationary fixpoint we may use any other query language with the same expressive power [1] (see also e.g., [6]).

Any query $Q =< \mathbf{d}, g >$ in NQPF on a database D is written in our language as depicted in Figure 1. It turns out that the user only needs to supply the definition of the two subqueries for QOS and QOR.

The language is rather 'rudimental' and can be refined and improved. Nevertheless, our belief is that any other language will maintain a procedural flavor that is needed to perform the QOS test. Removing procedurality (e.g., choosing more than one tuple at a time) would increase the expressive power up to NQPTIME bringing back the anomaly that a solution cannot be tested in polynomial time.

Conclusion

In this paper we defined several classes of search queries and studied their computational characterization. A comparative analysis of defined classes has been carried out along two main classification criteria, namely, standard set inclusion and refinability. Taxonomic results we provided establishe the relative generality of defined classes.

Then, efficient implementability, i.e., refinability through polynomial time functions, has been studied. In particular, the structural complexity characterization of two classes featuring this desirable property, namely, the well-known class NQPTIME and the new class NQPF (here defined), has been analyzed. Finally, a query language capturing NQPTIME has been presented.

Acknowledgements The work was partially supported by *FWF (Austrian Science Funds)* under the project P11580-MAT "A Query System for Disjunctive Deductive Databases" and by the Italian MURST under the project for

```
var verified_sub, verified_rel : bool;
var G : relation on g, O: order relation;
function continue() : bool begin
    if verified_rel then
        return choice_bool()
    else
        return verified_sub
end;
begin
    choice_order(O); G := ∅;
    verified_sub := verify_sub(G, O);
    verified_rel := verify_rel(G, O);
    while continue() and exists_choice(G) do begin
        verified_sub := false;
        while not verified_sub and exists_choice(G) do begin
            choice_tuple(G); verified_sub := verify_sub(G, O);
            if not verified_sub then
                retract_tuple(G)
        end;
        if verified_sub then
            verified_rel := verify_rel(G, O);
    end
    if verified_rel then
        output G
    else
        notify undefinedness
end.
```

Fig. 1. *Definition of a query in* NQPF

Italy-Austria cooperation *Advanced Formalisms and Systems for Nonmonotonic Reasoning*.

References

1. S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1994.
2. S. Abiteboul, E. Simon and V. Vianu. Non-deterministic languages to express deterministic transformations. *ACM PODS '90*, 218-229.
3. S. Abiteboul and V. Vianu. Non-determinism in logic-based languages, *Annals on Mathematics and AI*, Vol. 3, No. II-IV, 1991.
4. M. Cadoli. On the complexity of model finding for non-monotonic propositional logics. *Proc. 4th Italian Conf. on Theoretical Computer Science*, 125-139, World Scientific Pub., 1992.
5. A. Chandra and D. Harel. Computable queries for relational databases. *JCSS*, 21, 2, 1980, 156-178.

6. E. Dantsin, T. Eiter, G. Gottlob and A. Voronkov. Complexity and expressive power of Logic Programming, *Proc. 12th IEEE Conf. on Computational Complexity*, Ulm, 1997.
7. S. Fenner, L. Fortnow, A. Naik and J. Rogers, On inverting onto functions, *Proc. IEEE Conf. on Computational Complexity*, 1996.
8. S. Fenner, F. Green, S. Homer, A. Selman, T. Thierauf, H. Vollmer, Complements of multivalued functions, *Proc. IEEE Conf. on Computational Complexity*, 1996, 260–269.
9. S. Grumbach and Z. Lacroix, On non-determinism in machine and languages, *Annals of Mathematics and AI*, 1997.
10. D.S.Johnson. A Catalog of Complexity Classes, In van Leeuwen, ed., *Handbook of Theoretical Computer Science*, pp.67–162, Elsevier 1990.
11. Papadimitriou C.H., *Computational Complexity*, Addison Wesley, 1994.
12. D. Saccá, The expressive powers of stable models for bound and unbound Datalog queries, *JCSS 54*(3): pp. 441–464, 1997.
13. A. Selman, A taxonomy of complexity classes of functions, *JCSS*, 48, 357–381, 1994.
14. A. Selman, Much ado about functions, *Proc. Conf. on Structures in Complexity Theory*, IEEE, 1996, 198–212.
15. Vardi M.Y., "The Complexity of Relational Query Languages", *Proc. ACM Symp. on Theory of Computing*, 1982, pp. 137-146.

This article was processed using the LaTeX macro package with LLNCS style