Baltzer Journals

On Non-Determinism in Machines and Languages*

Stéphane Grumbach Zoé Lacroix
I.N.R.I.A.
Rocquencourt BP 105
78153 Le Chesnay, France
Stephane.Grumbach@inria.fr
Zoe.Lacroix@inria.fr

Non-deterministic computation is not really computation, but the difference with real computation is blurred. We study in detail various levels of non-determinism in computations on non-deterministic Turing machines with polynomial bounds on the resources. Meanwhile, we consider numerous query languages, implicit logic, choice logic, order invariant logic, and restrictions of second-order logic, and establish correspondences between all these formalisms for both deterministic and non-deterministic queries. To the degrees of non-determinism in the computations, correspond levels of non-determinism in the logical definitions. Our main contribution is to characterize various complexity classes between Ptime and Pspace, by several logical means, thus translating open questions in complexity theory to open questions in logic related to the use of the non-determinism.

1 Introduction

Non-determinism is a fundamental concept in various areas of computer science. It is central in complexity theory, where the main open problem is related to the relationship between deterministic and non-deterministic computation ($P \stackrel{?}{=} NP$). It is very important in artificial intelligence and combinatorial optimization for problems related to the search of a solution satisfying some properties in a large space of candidates.

There is a large class of problems, for which it is "easy" to verify that a solution satisfies the problem, while it is "difficult" to find such a solution. This is the case for instance of NP-complete problems. It is difficult to tell if a graph admits an Hamiltonian path, but if such a path exists, it is easy to check that it has the desired property, and so to answer positively the problem. A non-deterministic algorithm can be easily defined to solve such a problem. It guesses the path and then checks the property. This kind of problems have motivated the investigation of non-deterministic computation models. Non-deterministic Turing machines allow multiple choices of transitions for a given configuration, and a non-deterministic machine is said to solve a problem if there is one accepting computation.

The flaw of the concept of non-deterministic computation devices, is that it does

*Work supported in part by the EC-NSF cooperation grant *Deus Ex Machina*.

not reflect a real notion of computation. Only the existence of an accepting branch of computation allows to conclude. Nevertheless, the distinction with real computation on deterministic devices is blurred. We will see that a similar weakness arises in the context of non-deterministic definitions in logical languages — both problems being naturally related.

Logic is of course another field where the non-determinism is crucial — the notion of proof is of a fundamentally non-deterministic nature. In the present work, we extend further the inherent non-determinism of logic by considering various formalisms with non-deterministic semantics. The non-determinism has been introduced explicitly in logic, strongly motivated by applications in artificial intelligence. It has also been studied in the context of database query languages [ASV90], where the motivation is twofold. First, for the expression of non-deterministic queries in practical applications, such as find an adviser for each master student. Then, more interestingly, as for Turing computation, non-determinism plays a crucial role in the expressive power of the languages. In this paper, we compare various means to define non-deterministic constructs, both in computing devices and in logical languages. Our goal is to establish correspondences between different formalisms, and to better understand the expressive power of the constructs.

A deterministic query is a mapping from databases to relations. A non-deterministic query is a relation which associates possibly several distinct relations to each database. We define the class $\mathcal{NQ}$ of all computable non-deterministic queries. Our purpose is to characterize subclasses of $\mathcal{NQ}$ with query languages on one side and classes of queries computable on machines of various complexity classes on the other side. Non-deterministic queries correspond to multivalued functions in complexity theory [Sel94].

Classes of deterministic queries computable on Turing machines with bounded resources are defined with respect to the recognition problem as usual [Kan90]. In the case of non-deterministic queries, we followed the definition of [ASV90], and considered the computation problem, which is more appropriate in the later case. We consider classical complexity classes in the polynomial range, PTIME, NP, and their complement and intersections, NP$\cap$co-NP, etc. We also consider less usual classes, such as in particular the class UP of unambiguous computations [Val76], the class DP, of problems whose computation requires in parallel a branch in NP and a branch in co-NP [PY82], and the class P^{NP} of computations in P with oracles in NP.

We first remark that a deterministic query may be computed with non-deterministic computation devices. We distinguish various levels of non-determinism in the computation. The local non-determinism at the level of the transition function, and the global non-determinism at the level of the computation (one or more accepting computation branches). Unambiguous Turing machines are locally, but not globally non-deterministic. Other more subtle distinctions are made, such as all accepting computations lead to the same output, or all computations lead to the same outcome.

We consider various techniques to define deterministic queries from non-deterministic ones. First the well known semantics, *Possible* and *Certain*, taking respectively the union and the intersection of the possible outcomes of a query. These techniques were introduced in the context of query languages in [ASV90] and further investigated in [AV91b]. We then introduce the semantics *Possible-is-Certain*, which is shown to characterize the functional fragment of non-deterministic languages. Finally, we define three other semantics, *Disjoint*, *Co-Disjoint*, and *Unique*, which reflect various techniques to control the non-determinism.

We apply these functionalization techniques to classes of bounded complexity queries as well as to non-deterministic query languages, which are extensions of first-order logic, FO, with choice constructs: FO+W_1, with first-order choice (choice of elements), and FO+W_2, with second-order choice (choice of relations). We prove that FO+W_2 captures the class N$\mathcal{Q}$NP^T of total non-deterministic queries computable in polynomial time. We establish numerous equivalence results between the deterministic queries defined by one of the functionalization techniques from queries in FO+W_1 or FO+W_2, and deterministic languages, such as implicit logics: IMP [Kol90], LIMP and GIMP [GLL95], restrictions of second-order logic: Σ^1_1, Π^1_1, UΣ^1_1, and UΠ^1_1 [Kol90], and order invariant versions of fixpoint logics with an order relation on the domain: FO+IFP [Imm86, Var82], etc.

Finally, we prove equivalences between the languages obtained, and complexity classes of queries. Our main results translate open questions in complexity theory to questions in logic. Such characterizations follow the trend of descriptive complexity theory, initiated by Fagin [Fag74], with the characterization of NP by existential second-order logic. It is interesting to note that descriptive complexity was initiated with a non-deterministic complexity class. Numerous characterization results are now known for complexity classes above NP, or for restricted classes of inputs, such as ordered domains [Imm86, Var82, Fag93]. The approach followed here emphasizes a specific viewpoint, namely the use of non-determinism in logical formalisms. Results in the same spirit concerning the classes NP, co-NP, and DP have been obtained in the context of logic programming with rule languages with the inflationary semantics [AV91b], and with the stratified and the well-founded semantics [SZ90, GPSZ91, GSZ95].

We devote a particular attention to the class N$\mathcal{Q}$NP^T of total non-deterministic queries computable in NP, which is closed under complement. We prove that the functional fragment (subset of deterministic queries) of N$\mathcal{Q}$NP^T coincides with the set of total deterministic queries computable both in NP and in co-NP, $\mathcal{Q}$NP$\cap$co-NP, and most probably not with $\mathcal{Q}$NP unless NP = co-NP. Deterministic queries of N$\mathcal{Q}$NP, are computable by non-deterministic Turing machines such that for each input, all accepting computations yield the same output.

The functionalization semantics on the class N$\mathcal{Q}$NP^T, *Possible*, *Certain*, *Possible-is-Certain*, *Disjoint*, *Co-Disjoint*, and *Unique*, lead respectively to the classes of deterministic queries: $\mathcal{Q}$NP, $\mathcal{Q}$co-NP, $\mathcal{Q}$NP $\cap$ co-NP, $\mathcal{Q}$UP, $\mathcal{Q}$co-UP, and $\mathcal{Q}$UP$\cap$co-UP. The functionalization principles correspond to specific restrictions on the non-determinism allowed in the computation of non-deterministic machines.

The main contribution of the paper is twofold. First, we prove that the functional fragment of classes of non-deterministic queries coincides with implicit queries. For instance, the functional fragment of N$\mathcal{Q}$NP^T coincides with GIMP. Second, we define sub-classes of the functional fragments characterizing unambiguous queries.

The paper is organized as follows. In the next section, we recall classical concepts of complexity theory. Section 3 is devoted to deterministic and non-deterministic queries, and transformations from the later to the former. In Section 4, we introduce classes of queries based on restricted Turing machines. Section 5 is devoted to deterministic query languages, while non-deterministic query languages are presented in Section 6. In Section 7, we draw some conclusions.

2 Machines, problems and complexity classes

In this section, we recall the definitions of Turing machines, problems, and various complexity classes needed in the following sections.

2.1 Non-deterministic machines

The only computational device considered throughout this paper is the Turing machine (TM). We distinguish various classes of Turing machines which are classical [Boa90], and also introduce new distinctions based on the use of the non-determinism by the machines. For simplicity, we assume that the machines admit the following common description. Each machine has three tapes, a semi-infinite read-only tape for the input, a read-write work-tape and a semi-infinite write-only tape for the output. The notions of states, tape alphabet, and transition function are standard. A machine is *deterministic* if its transition function is deterministic (given the symbols read by the head on the tapes and the state, there is at most one possible move). A machine is *non-deterministic* if its "transition function" allows several possible choices for the next move.

In a few cases we will need more complex machines with oracles. An *oracle Turing machine* has an additional oracle tape, with two alternating mode, write-only and read-only. In the write-only mode, the machine can trigger an oracle on the string written on the oracle tape which is solved for free. The answer of the oracle is automatically written on the oracle tape, and the mode is changed to read-only.

We consider two different types of machines: the *boolean* and the *constructive* machines. We say that a machine is *boolean* (also called an *acceptor*) if it solves a decision problem (the output-tape is ignored, and only the state accepting or not is relevant). In this case the output is said to be boolean. It is TRUE if there is an accepting computation. A machine is *constructive* if it computes a relation, mapping input strings to output strings, obtained by a valid computation of the machine starting in the initial state with the input string on the input-tape, and reaching an accepting state with the output string on the output-tape. (A constructive machine is also called a *transducer*.) In this case, an output is the content of the output tape for an accepting state.

A non-deterministic machine may lead to several distinct computations on a given input. Consider the *computation tree* in Figure 1, whose nodes are the configuration of the machine, the initial configuration with the input at the root, and whose edges denote possible transitions from one configuration to another. An *accepting branch* in the tree is a path from the root to a leaf with accepting state. If the machine is deterministic, the computation tree consists of a single branch.

We can now distinguish between various levels of non-determinism, and introduce the notions of *local* and *global* non-determinism.

Definition 2.1
A machine is *locally non-deterministic* if its transition function is non-deterministic. A machine is *globally non-deterministic* if there exists an input for which there are at least two distinct accepting computations.

To be *globally* non-deterministic, a machine has to be *locally* non-deterministic. On the other hand, a machine may be *locally* non-deterministic without being *globally* non-

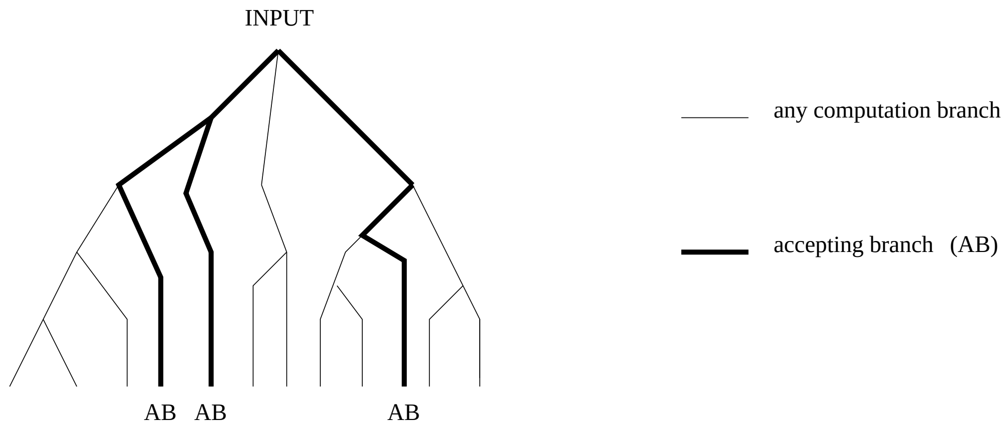

Figure 1: Computation tree of a non-deterministic machine

deterministic. We next consider Turing machines which are locally but not globally non-deterministic. These machines coincide with the *unambiguous Turing machines* introduced by Valiant in [Val76], and the categorical machines defined in [HH88].

Definition 2.2
An *unambiguous Turing machine* (UTM) is a non-deterministic Turing machine which, for each possible input, has at most one accepting computation.

It was shown that the class of non-deterministic machines which are not UTMs is r.e. complete [HH88].

The resources considered are standard, namely the time and space used by the machines. For the kind of problems in which we are interested, we may assume that the computation trees are finite (in other words, the machines considered in the paper halt on all inputs). The *time* used by a (non-)deterministic Turing machine is the depth of the computation tree and the *space* is the maximum, over all computations, of the number of work-tape cells in use. The size of an input I of a machine is the length of the word written on the input-tape of the machine. The resources are bounded in the size of the input. We focus mainly in fact on polynomial time machines. We next consider problems.

2.2 Problems

A *decision problem* is a function mapping strings to TRUE or FALSE. Decision problems are *computable*, if there exists a machine which halts in an accepting state on an *encoding* of the input iff the answer to the problem is TRUE on this input. Most of the problems we consider are problems on graphs and their usual encoding is the standard encoding of relations modulo some order on the constants [AHV94]. Example 2.1 illustrates the standard encoding. The input of a machine is a word while the input of a graph problem is a graph. In the following we often identify the input $\mathbf{I}$ of a problem with its encoding I, input of the machine (see Figure 2).

Another class of problems of interest in this paper is constituted by the *constructive*

$$
\begin{array}{lccc}
 & \text{Problem} & & \text{Computation} \\
\text{INPUT} \quad : & \mathbf{I} & \xrightarrow{encod} & I \\
 & \downarrow & & \downarrow \\
\text{OUTPUT} : & \mathbf{O} & \xleftarrow{decod} & O
\end{array}
$$

Figure 2: Computation of a constructive problem

problems (also called search problems). A *constructive problem* is a mapping from inputs to outputs different from TRUE or FALSE, such as integers, relations, etc. A constructive problem is computable if there exist a machine and an encoding function such that the machine halts in an accepting state on the encoded input iff the output of the machine is an encoding of the answer of the problem. In particular, for any possible answer of the constructive problem, there exists an accepting computation of the machine.

Example 2.1
Let **dom**$=\{a, b, c, \ldots\}$ be an ordered set of constants and P be a boolean graph problem whose input consists in two finite binary graphs G_1 and G_2. An input $\mathbf{I}$ of P is two finite subsets of **dom**$\times$**dom**. All encodings of inputs of P use the alphabet $\{0, 1, [,], \sharp\} \cup \{G_1, G_2\}$. The encoding of the input $\mathbf{I}$ defined by $G_1 = \{[a, b], [b, b], [b, c]\}$ and $G_2 = \{[a, c]\}$ is then the word $G_1[0\sharp1][1\sharp1][1\sharp10]G_2[0\sharp10]$.

Some standard assumptions are made on the encoding schemes used in practice [GJ79]. In particular, the encoding I of an instance $\mathbf{I}$ must be concise and not "padded" with unnecessary information. Constants occurring in $\mathbf{I}$ must be represented in some finite alphabet. Without loss of generality, we make an additional assumption, which restrict unnecessary non-deterministic behavior of the machines. We assume that the machines and encoding are "reasonable[1]", and satisfy the following property (see Figure 2):

> Given a problem $\mathbf{P}$, mapping an input $\mathbf{I}$ to an output $\mathbf{O}$. Let M be a machine solving $\mathbf{P}$, and *encod* some arbitrary 1-1 encoding. Then, M on $encod(\mathbf{I})$ outputs $encod(\mathbf{O})$, for the same encoding.

A "reasonable" machine is such that if two different accepting branches respectively lead to two different strings on the output-tape, O and O' $(O \neq O')$, then $\mathbf{O}$ and $\mathbf{O'}$ such that $O = encod(\mathbf{O})$ and $O' = encod(\mathbf{O'})$ satisfy $\mathbf{O} \neq \mathbf{O'}$. This assumption is made in order not to consider non-determinism at the level of the encoding of the output. It is always possible to ensure such a property in the complexity classes considered in the present paper. It suffices to reorganize the output string before writing it on the output-tape. In the case of relational queries, this consists in sorting the tuples in an order which is uniquely defined by the encoding of the input. Logarithmic space is enough to perform this task.

We associate a *decision* problem P_d to a *constructive* problem P in the following way.

[1]An encoding is "reasonable" if for each input $\mathbf{I}$, its encoding I does not contain more information than $\mathbf{I}$ and is not too long.

On each input **I** which is the encoding of a pair $\langle \mathbf{I}', \mathbf{O}' \rangle$, P_d is TRUE iff **O'** is a possible answer of P on input **I'**.

2.3 Complexity classes for non-deterministic computations

A complexity class $\mathcal{C}$ consists of the decision problems solvable by a Turing machine with bounded (time or space) resources (see [Joh90] for a general introduction). Complexity classes can be alternatively defined in terms of languages. We consider in the following the classes, P, NP, co-NP, UP, etc. and present examples of problems in each class when possible.

P is the class of decision problems solvable by a deterministic Turing machine in polynomial time. NP is the class of decision problems solvable by a non-deterministic Turing machine in polynomial time.

Example 2.2
An example of a problem in NP is HAMILTONICITY.

$$\begin{cases} \text{INPUT} & : \text{A graph } G \\ \text{OUTPUT} & : \text{TRUE if } G \text{ admits a Hamiltonian circuit} \end{cases}$$

The closure under complementation of the class NP remains open. The class co-NP consists of decision problems whose complement is solvable in NP.

Example 2.3
An example of a problem in co-NP is TAUTOLOGY.

$$\begin{cases} \text{INPUT} & : \text{A propositional sentence } \theta \\ \text{OUTPUT} & : \text{TRUE if } \theta \text{ is valid} \end{cases}$$

NP $\cap$ co-NP is the class of decision problems solvable in both NP and co-NP. By definition, P $\subseteq$ NP $\cap$ co-NP $\subseteq$ NP as well as $\subseteq$ co-NP, but the strictness of these inclusions remains open.

Example 2.4
An example of a problem in NP $\cap$ co-NP is PRIMALITY.

$$\begin{cases} \text{INPUT} & : \text{An integer } n \\ \text{OUTPUT} & : \text{TRUE if } n \text{ is prime} \end{cases}$$

Other alternatives based on the previous resource bounds are possible. It is natural to consider decision problems whose computation uses in parallel a branch in NP and a branch in co-NP. These compounded decision problems are in DP, a class introduced and studied by Papadimitriou and Yannakakis in [PY82]. DP consists of all decision problems that can be solved by X $\cap$ Y where X $\in$ NP and Y $\in$ co-NP.

It is clear that NP $\cup$ co-NP $\subseteq$ DP $\subseteq$ P^{NP}, where P^{NP} is the class of problems computed in deterministic polynomial time with oracles in NP. DP=NP$\cup$co-NP iff NP=co-NP [PY84]. The "unique" version of any problem in NP is a problem in DP.

Example 2.5
An example of a problem in DP is UNIQUE HAMILTONICITY.

$$\begin{cases} \text{INPUT} & : \text{A graph } G \\ \text{OUTPUT} & : \text{TRUE if } G \text{ admits a unique Hamiltonian circuit} \end{cases}$$

We have recalled in Definition 2.2 the notion of *unambiguous* computation. The class UP of decision problems solvable by unambiguous Turing machines in polynomial time has been introduced in [Val76]. Except for the deterministic machines, no example of a non-deterministic machine, such that for each input there is at most one accepted output, is known. In other words, all known problems in UP are already known to be in P. The existence of problems in UP such that there is no constructive proof that they are in UP (that is there is no way to describe the unambiguous TM which solves the problem) is related to the existence of a complete problem in UP. It has been shown in [HH88] that if UP has no complete problem, then there exist problems in UP for which there is no constructive proof. It is easy to verify (see Figure 3) that P $\subseteq$ UP $\subseteq$ NP. co-UP is the class of decision problems whose complement is computable in UP, and UP $\cap$ co-UP, the class of decision problems solvable in both UP and co-UP. Blass and Gurevich [BG82] introduced the class US whose resemblance to UP is confusing. The class US consists of sets L that can be represented in the form $L = \{x \mid \exists! y\ R(x,y)\}$ where R is a polynomially bounded relation computable in polynomial time. US consists then of all "unique" version of problems in NP (UNIQUE HAMILTONICITY $\in$ US) and US $\subseteq$ DP. We will not consider US in the following, but it is worth mentioning.

Remark that the difference between US and UP is subtle. US is not defined relatively to a machine. It is not surprising that UP $\subseteq$ US. The converse is open.

Proposition 2.1
UP $\subseteq$ US

Proof : Let P be a decision problem in UP. It suffices to consider M_P an UTM which solves P and R_P the relation defined on each accepted input I by the description of the computation of M_P on I. Since M_P is a non-deterministic Turing machine in polynomial time, R_P is computable in polynomial time. Moreover, the computation on I is described by $R(I,G)$ where G encodes the guesses. For each I, G is unique since M_P is unambiguous. □

Moreover UNIQUE SATISFIABILITY (Example 2.6) which is co-NP hard is complete for the class US. It follows that co-NP $\subseteq$ US and that NP $\subseteq$ US implies US = DP [BG82]. Similarly, if US $\subseteq$ NP then NP = co-NP = DP = P^{NP}.

Example 2.6
An example of a decision problem in US is UNIQUE SATISFIABILITY.

$$\begin{cases} \text{INPUT} & : \text{A propositional sentence } \theta \\ \text{OUTPUT} & : \text{TRUE if } \theta \text{ admits a unique model} \end{cases}$$

PSPACE is the class of decision problems solvable by a deterministic Turing machine in polynomial space.

Figure 3, and more completely Figure 7 at the end of the paper, illustrate the relationships between these classes.

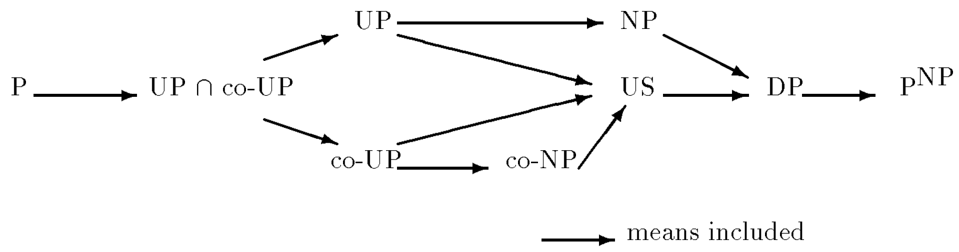

Figure 3: Non-deterministic polynomial classes

The previous classes are defined relatively to decision problems. We now consider complexity classes related to constructive problems. If $\mathcal{C}$ is a complexity class as defined previously, then f-$\mathcal{C}$ is the class of partial functions computable by a Turing machine whose resources (time or space) have the same bounds as the machines considered to define $\mathcal{C}$. (f-$\mathcal{C}$ is denoted by F$\mathcal{C}$ in [Joh90].) f-P is the class of constructive problems computable by a Turing machine in deterministic polynomial time while f-NP is the class of constructive problems computable by a Turing machine in non-deterministic polynomial time. Note that since P is a deterministic class, P is a class of functions. f-P is also called FP in [Joh90] and PF in [Sel94]. A function of f-NP is not really a function, since it can have several different outputs on a single input. f-NP is the class of multivalued functions computable in non-deterministic polynomial time (denoted by NPMV in [Sel94]).

Note that if P is a constructive problem of f-$\mathcal{C}$, then its associated problem P_d is in $\mathcal{C}$. f-US (also denoted by NPSV in [Sel94]) and f-UP are important in cryptography and cryptoanalysis [GS88].

Proposition 2.2
P=UP iff f-P=f-UP.

Proof : Suppose that f-P=f-UP. Let P be a decision problem in UP resolved by an UTM M. Let f be the function which associates to m the unique accepting computation of M on the input m, if M accepts m. Let M_f be a machine which on each input m, (1) uses the transitions of M on m and writes on its output-tape each step of the computation of M on m; then (2) accepts if M accepts m. M_f is an UTM since M is an UTM. f is then a partial function computable in unambiguous polynomial time. By hypothesis f is in f-P and there is a deterministic machine M'_f which associates to each input m the accepting computation of M on m. Finally, the machine which, on an input m, (1) uses the transitions of M'_f and (2) accepts when M'_f accepts, solves P in deterministic polynomial time. The converse is immediate. □

The class f-P$^{\mathrm{NP}}$ of constructive problems NP-easy is also of interest. A constructive problem is NP-easy if there exists a decision problem of NP which reduces to it. In other words, an NP-easy problem is computable with an oracle in NP [Joh90]. It is easy to see that if P=NP then f-P$^{\mathrm{NP}}$=f-P.

3 Queries

In this section, we introduce the notion of non-deterministic query and present various methods to define deterministic queries from non-deterministic ones.

Let **dom** be an infinite set of constants containing two special constants $\perp$, $\top$. Let $P_1, \ldots, P_s$ be relation symbols, where each P_i is of fixed arity r_i, and $\mathbf{P} = \{P_1, \ldots, P_s\}$ be a database schema. An instance over $P_i \in \mathbf{P}$ is a finite set of tuples of arity r_i over **dom**. An instance (or structure) **I** over a database schema **P** is a mapping from **P** such that for each $P_i \in \mathbf{P}$, $\mathbf{I}[P_i]$ is an instance over P_i. Finally, the set of all instances over a schema **P** (resp. a relation R) is denoted by inst($\mathbf{P}$) (resp. inst(R)). The active domain of a database instance **I**, denoted adom($\mathbf{I}$), is the set of all constants occurring in **I**.

A *deterministic query* Q of arity k $(1 \leq k)$ is a recursive mapping which associates to each $\mathbf{I} \in \mathrm{inst}(\mathbf{P})$ a relation $\mathbf{I}[R]$, where R is a predicate symbol of arity k which is not in **P**. $\mathbf{I}[R]$ constitutes the answer, denoted $Q(\mathbf{I})$, of Q on **I**. The natural way to extend the notion of deterministic query to the non-deterministic one is to enable different possible relations $\mathbf{I}[R]$ as possible answers to **I**. It follows that a query is no longer a mapping but a relation between database instances and relations.

Definition 3.1
Let **P** be a database schema, and R a predicate symbol of arity $k \geq 1$ which is not in **P**. A k-ary *non-deterministic* query Q is a *computable* and *generic* subset of $\mathrm{inst}(\mathbf{P}) \times \mathrm{inst}(R)$ such that for each $\mathbf{I} \in \mathrm{inst}(\mathbf{P})$, there exists at least one relation $\mathbf{I}[R]$ such that $(\mathbf{I},\mathbf{I}[R]) \in Q$. A boolean *non-deterministic* query is a subset of $\mathrm{inst}(\mathbf{P}) \times \{\perp, \top\}$ such that for each $\mathbf{I} \in \mathrm{inst}(\mathbf{P})$, $(\mathbf{I},\perp) \in Q$ or $(\mathbf{I},\top) \in Q$.

The semantics of $\perp$ and $\top$ are respectively $\emptyset$ and $\{[]\}$. The *genericity* criterion [CH80, AHV94] is classical. We next recall the definition.

Definition 3.2
Let **P** be a database schema, and R a predicate symbol. A subset Q of $\mathrm{inst}(\mathbf{P}) \times \mathrm{inst}(R)$ is *generic* iff for each bijection ρ over **dom**, $(\mathbf{I},\mathbf{I}[R]) \in Q$ iff $(\rho(\mathbf{I}),\rho(\mathbf{I}[R])) \in Q$.

In the next example, we illustrate a consequence of the genericity on non-deterministic queries.

Example 3.1
Let $\mathbf{P} = \{P\}$ where P is a unary relation symbol, and $\mathbf{I}[P] = \{a, b, c\}$. Let Q be a unary query such that $(\{a, b, c\}, \{a\}) \in Q$, then the genericity criterion implies that $(\{a, b, c\}, \{b\}) \in Q$ and $(\{a, b, c\}, \{c\}) \in Q$.

Extending the notions introduced in [CH82], we now define the class $\mathcal{NQ}$ of all computable non-deterministic queries.

Definition 3.3
$\mathcal{NQ}$ is the class of all computable non-deterministic queries.

We define the output (or answer) of a non-deterministic query as follows.

Definition 3.4
Let $\mathbf{P}$ be a database schema, R a predicate symbol of arity k and Q a k-ary query. An *input* of $Q \subseteq \text{inst}(\mathbf{P})\times\text{inst}(R)$ is a database instance $\mathbf{I} \in \text{inst}(\mathbf{P})$, an *output* (or *answer*) of Q on $\mathbf{I}$ is a relation $U \in \mathcal{P}(\text{adom}(\mathbf{I})^k)$ such that $(\mathbf{I},U) \in Q$.

Notation: For any set E, $\mathcal{P}(E)$ denotes the set of all subsets of E.

A query $Q \subseteq \text{inst}(\mathbf{P})\times\text{inst}(R)$ is *total* if for each input $\mathbf{I} \in \text{inst}(\mathbf{P})$, there is at least a relation $U \in \mathcal{P}(\text{adom}(\mathbf{I})^k)$ such that $(\mathbf{I},U) \in Q$. For any class $\mathcal{Q}$ of queries, $\mathcal{Q}^T$ is the class of all total queries of $\mathcal{Q}$.

We illustrate the previous definitions with an example of a non-deterministic query.

Example 3.2
The ORDER query is defined by:

$$\begin{cases} \text{INPUT} & : \text{a set } P \\ \text{OUTPUT} & : \text{a linear order on } P \end{cases}$$

It is important to note that even in the case of non-deterministic queries, the *output* consists by definition in one of the possible answers, in other words it is not the set of all possible answers to the query, which is called the *effect*. The *effect* of a query Q on a database instance $\mathbf{I}$ is the set of all possible outputs (answers) of Q on $\mathbf{I}$.

Definition 3.5
Let $\mathbf{P}$ be a database schema, R a predicate symbol of arity k and Q a k-ary query. The *effect* of $Q \subseteq \text{inst}(\mathbf{P})\times\text{inst}(R)$ on a database instance $\mathbf{I} \in \text{inst}(\mathbf{P})$ is the set:

$$\text{Effect}(Q,\mathbf{I}) = \{U \in \mathcal{P}(\text{adom}(\mathbf{I})^k) \ : \ (\mathbf{I},U) \in Q\}.$$

Example 3.3
Let $\mathbf{P}$=$\{P\}$ where P is a unary predicate symbol. Let $\mathbf{I}$ be the instance defined by $\mathbf{I}[P]$=$\{a,b,c\}$. The *effect* of the ORDER query on $\mathbf{I}$ is the set

$$\begin{aligned} \text{Effect}(\text{ORDER},\mathbf{I}) \quad = \quad & \{\{[a,b],[b,c],[a,c]\},\{[a,c],[c,b],[a,b]\},\{[b,a],[a,c],[b,c]\}, \\ & \{[b,c],[c,a],[b,a]\},\{[c,a],[a,b],[c,b]\},\{[c,b],[b,a],[c,a]\}\}. \end{aligned}$$

Note that two queries Q and Q' are *similar* if for each instance $\mathbf{I}$, Effect$(Q,\mathbf{I})$ = Effect$(Q',\mathbf{I})$. A query Q is deterministic iff for each instance $\mathbf{I}$, Effect$(Q,\mathbf{I})$ is a singleton. A total boolean non-deterministic query Q is such that on each instance $\mathbf{I}$, its effect is one out of the following sets: $\{\top\},\{\bot\},\{\top,\bot\}$.

In case of deterministic queries, there is a classical notion of complementation. The complement $\neg Q$ of a deterministic k-ary query Q on an instance $\mathbf{I}$, is the k-ary relation

$\neg Q(\mathbf{I})$ such that $\bar{a} \in \neg Q(\mathbf{I})$ iff $\bar{a} \notin Q(\mathbf{I})$. The notion of complementation for non-deterministic queries generalizes the deterministic case in the following way.

Definition 3.6
Let Q be a k-ary non-deterministic query. The complement $\neg Q$ of the query Q is defined by:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : \text{a relation } R \text{ such that } C(R) \text{ is a possible answer for } Q \text{ on } \mathbf{I}. \end{cases}$$

where $C(R)$ denotes the complement of the relation R.

Note that in case of non-deterministic queries, both a relation and its complement may be two possible answers of the same query. But it does not mean that $Q = \neg Q$. We say that $Q = \neg Q$ iff on each instance $\mathbf{I}$, $(\mathbf{I},R) \in Q(\mathbf{I})$ iff $(\mathbf{I},C(R)) \in Q(\mathbf{I})$.

A database query Q is *deterministic* if it is the *graph* of a recursive mapping. Deterministic queries constitute a subset of non-deterministic ones. We introduce the class $\mathcal{DQ} \subset \mathcal{NQ}$ of all computable deterministic queries which coincides with the queries of CQ in [CH82].

Definition 3.7
$\mathcal{DQ}$ is the class of all computable deterministic queries.

$\mathcal{NQ}$ and $\mathcal{DQ}$ are both closed under complementation. We next present a fundamental example of a deterministic query.

Example 3.4
The EVENNESS query is defined by:

$$\begin{cases} \text{INPUT} & : \text{a set } P \\ \text{OUTPUT} & : \text{true iff the cardinality of } P \text{ is even} \end{cases}$$

Note that the notion of query is defined independently of the medium, such as a machine or a language, in which the query is defined. We will see in the following that non-deterministic computations and definitions are fundamental to express deterministic queries.

In the sequel, we distinguish subclasses of $\mathcal{NQ}$. In sections 4, 5, and 6, we consider queries either defined by some formula in a query language or computed by some machine with bounded resources. The non-determinism both in the definitions and in the computations will result in particular in the existence of various distinct ways to get the same output. This can be roughly seen as a form of local non-determinism. We propose two techniques to capture deterministic queries considering a class $\mathcal{Q}$ of non-deterministic ones.

1. A class $\mathcal{Q}$ of non-deterministic queries may contain deterministic ones. These queries

are deterministic though computed or defined with non-deterministic tools. Capturing the deterministic fragment of $\mathcal{Q}$ consists in determining the *functional fragment* of $\mathcal{Q}$.

2. A deterministic query may be defined from a non-deterministic query. The technique consists in considering *deterministic semantics* on the non-deterministic tools (machine or definition) which characterize the latter.

In the following, we consider functionalization techniques. Let $\mathcal{Q}$ be a class of non-deterministic queries. We first consider the *functional fragment* of $\mathcal{Q}$ denoted by funct-$\mathcal{Q}$.

Definition 3.8
The *functional fragment* of a class $\mathcal{Q}$ of non-deterministic queries is the subset of $\mathcal{Q}$ consisting of all the deterministic queries in $\mathcal{Q}$.

By definition, funct-$\mathcal{NQ} = \mathcal{DQ}$.

We now introduce the *deterministic semantics*. The first deterministic semantics we consider, *Possible* and *Certain*, were introduced in [IL84] and first studied in the context of query languages in [ASV90]. They respectively consist in the union and the intersection of all possible outputs.

Definition 3.9
Let Q be a non-deterministic query in $\mathcal{NQ}$. The *Possible* semantics on Q defines the query $Possible(Q)$:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : \text{the relation } Possible(Q,\mathbf{I}) = \bigcup\{R | R \in \text{Effect}(Q,\mathbf{I})\} \end{cases}$$

For example, if $\text{Effect}(Q,\mathbf{I}) = \{\{1,4\},\{1,2\},\{1,2,3\}\}$, $Possible(Q,\mathbf{I}) = \{1,2,3,4\}$.

Definition 3.10
Let Q be a non-deterministic query in $\mathcal{NQ}$. The *Certain* semantics on Q defines the query $Certain(Q)$:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : \text{the relation } Certain(Q,\mathbf{I}) = \bigcap\{R | R \in \text{Effect}(Q,\mathbf{I})\} \end{cases}$$

For example, if $\text{Effect}(Q,\mathbf{I}) = \{\{1,4\},\{1,2\},\{1,2,3\}\}$, $Certain(Q,\mathbf{I}) = \{1\}$.

Note that these semantics can be seen as total functions from a class $\mathcal{Q} \subseteq \mathcal{NQ}$ to a class $\mathcal{Q}' \subseteq \mathcal{DQ}$. *Possible* and *Certain* are total since they associate a deterministic query Q' to any non-deterministic query $Q \in \mathcal{NQ}$. They also satisfy the following property.

Proposition 3.1
Let Q be a non-deterministic query in $\mathcal{NQ}$ and $\mathbf{I}$ an instance. The *Certain* semantics on Q over $\mathbf{I}$ defines the complement of the *Possible* semantics on $\neg Q$ over $\mathbf{I}$: $Certain(Q,\mathbf{I}) = C(Possible(\neg Q,\mathbf{I}))$.

Proof : It is easy to see that: $C(\mathit{Possible}(Q,\mathbf{I})) = C(\bigcup\{R|R \in \text{Effect}(Q,\mathbf{I})\}) = \bigcap\{C(R)|R \in \text{Effect}(Q,\mathbf{I})\} = \bigcap\{R|R \in \text{Effect}(\neg Q,\mathbf{I})\} = \mathit{Certain}(\neg Q,\mathbf{I})$. □

We next introduce another semantics on non-deterministic queries which is not defined on all queries in $\mathcal{NQ}$. *Possible-is-Certain* is a partial function from $\mathcal{NQ}$ to $\mathcal{DQ}$.

Definition 3.11
Let Q be a non-deterministic query in $\mathcal{NQ}$. The *Possible-is-Certain* semantics on Q defines the query *Possible-is-Certain*(Q):

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : R \text{ such that } \{R\} = \bigcup\{R|R \in \text{Effect}(Q,\mathbf{I})\} = \bigcap\{R|R \in \text{Effect}(Q,\mathbf{I})\}, \end{cases}$$

and is undefined otherwise.

The *Possible-is-Certain* semantics on a query Q over an instance $\mathbf{I}$ is defined iff *Possible*(Q) = *Certain*(Q). That is when on each instance $\mathbf{I}$, Effect(Q,$\mathbf{I}$) is a singleton.

For a class of non-deterministic queries $\mathcal{Q}$, we note by *Possible*$(\mathcal{Q})$, the class of queries obtained by applying the deterministic semantics on the queries of $\mathcal{Q}$. The same applies to the other deterministic semantics, with the restriction that some of them (e.g. *Possible-is-Certain*) are not defined for each $Q \in \mathcal{Q}$.

Let $\mathcal{Q}$ be any class of non-deterministic queries. The functional fragment funct-$\mathcal{Q}$ coincides with *Possible-is-Certain*$(\mathcal{Q})$.

Proposition 3.2
Let $\mathcal{Q}$ be any class of non-deterministic queries, funct-Q = *Possible-is-Certain*(Q).

We first prove a lemma which sheds some light on the *Possible-is-Certain* semantics, which does not really define a new query but is defined over deterministic queries. In other words, if the *Possible-is-Certain* semantics is defined on Q, then *Possible-is-Certain*(Q) = Q.

Lemma 3.3
Let $\mathcal{Q}$ be any class of non-deterministic queries. If $Q \in$ *Possible-is-Certain*$(\mathcal{Q})$ then Q = *Possible-is-Certain*(Q).

Proof : Let $Q \in$ *Possible-is-Certain*$(\mathcal{Q})$ then there is Q^* in $\mathcal{Q}$ such that Q = *Possible-is-Certain*(Q^*). That is on each instance $\mathbf{I}$, $Q(\mathbf{I}) = \bigcup\{R|R \in \text{Effect}(Q^*,\mathbf{I})\}$ where $\bigcup\{R|R \in \text{Effect}(Q^*,\mathbf{I})\} = \bigcap\{R|R \in \text{Effect}(Q^*,\mathbf{I})\}$. It follows that $Q(\mathbf{I}) = R$ where R is such that Effect(Q^*,$\mathbf{I}$)= $\{R\}$ and then Effect(Q^*,$\mathbf{I}$)= Effect(Q,$\mathbf{I}$). Since it is true that Effect(Q^*,$\mathbf{I}$) = Effect(Q,$\mathbf{I}$) on each instance $\mathbf{I}$, $Q = Q^*$. □

Proof of Proposition 3.2: Let Q be a deterministic query in $\mathcal{Q}$. Since Q is deterministic, on each instance $\mathbf{I}$, Effect(Q,$\mathbf{I}$) consists of a single relation $R^I = Q(\mathbf{I})$. It follows that $\bigcup\{R|R \in \text{Effect}(Q,\mathbf{I})\} = \bigcap\{R|R \in \text{Effect}(Q,\mathbf{I})\} = \{R^I\}$ and then Q=*Possible-*

is-Certain(Q). Finally, $Q \in$ *Possible-is-Certain*$(\mathcal{Q})$. Conversely, let $Q \in$ *Possible-is-Certain*$(\mathcal{Q})$. From Lemma 3.3, it follows that $Q =$ *Possible-is-Certain*(Q) and $Q \in$ funct-$\mathcal{Q}$. $\square$

A non-deterministic constructive machine presents two levels of non-determinism. First, it may have several different computations on a given input (local non-determinism). Second, it can have several different accepted computations on a given input (global non-determinism). We now introduce tools which distinguish between these different levels of non-determinism. To give a precise characterization of the queries which are computed by locally but not globally non-deterministic machines, we have to take into account the number of different ways to obtain an answer, and not anymore the answer only. This leads to the concept of *multi-effect*, which will be formally defined in the appropriate sections. Intuitively, if there are three distinct ways to get an output to a query Q on an instance $\mathbf{I}$, (e.g. three accepting branches in the computation tree), there should be three occurrences of it in the multi-effect. For instance, we have for the boolean query corresponding to the computation of Figure 1:

$$\text{Multi-Effect}(M_Q,\mathbf{I}) = \{\!\{\top, \top, \top\}\!\}.$$

We now introduce deterministic semantics whose definitions are related to the *multi-effect* of queries, that is to the effect, and for each output to the number of distinct ways to obtain each output. This will be formally defined with respect to a medium such as Turing machines or query languages. We ignore these mediums presently. We denote by D_Q the medium which computes or defines the query Q.

Definition 3.12
Let Q be a non-deterministic query. The *Disjoint* semantics on Q defines the query *Disjoint*(Q):

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT:} & \bigcup\{R | R \in \text{Effect}(Q, \mathbf{I})\} \text{ if for all } R, R' \in \text{Multi-Effect}(D_Q, \mathbf{I}),\ R \cap R' = \emptyset, \end{cases}$$

if there exists such a medium D_Q and is undefined otherwise.

For example, if Multi-Effect$(D_Q, \mathbf{I}) = \{\!\{\{1,4\},\{2,3\}\}\!\}$, *Disjoint*$(Q, \mathbf{I}) = \{1,2,3,4\}$. But, if Multi-Effect$(D_Q, \mathbf{I}) = \{\!\{\{1,4\},\{1,5\},\{2,3\}\}\!\}$, *Disjoint*$(Q, \mathbf{I})$ undefined.

Definition 3.13
Let Q be a non-deterministic query. The *Co-Disjoint* semantics on Q defines the query *Co-Disjoint*(Q):

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : \bigcap\{R | R \in \text{Effect}(Q, \mathbf{I})\}, \end{cases}$$

if there exists such a medium D_Q and $\forall R, R' \in$ Multi-Effect$(D_Q, \mathbf{I})$, $C(R) \cap C(R') = \emptyset$, and is undefined otherwise.

For example, if Multi-Effect$(D_Q, \mathbf{I}) = \{\!\{\{1,4\},\{1,2,3\}\}\!\}$, and $\mathbf{dom} = \{1,2,3,4\}$, *Co-Disjoint*$(Q, \mathbf{I}) = \{1\}$.

Proposition 3.4
Let Q be a non-deterministic query in $\mathcal{NQ}$. The *Co-Disjoint* semantics on Q is defined iff the *Disjoint* semantics is defined on $\neg Q$. Moreover, $\neg$*Disjoint*(Q) = *Co-Disjoint*$(\neg Q)$.

Proof : For each instance $\mathbf{I}$,

$$\begin{aligned} C(\mathit{Disjoint}(Q,\mathbf{I})) &= C(\bigcup\{R|R \in \text{Effect}(Q,\mathbf{I})\}) = \bigcap\{C(R)|R \in \text{Effect}(Q,\mathbf{I})\} \\ &= \bigcap\{R|R \in \text{Effect}(\neg Q,\mathbf{I})\} = \mathit{Co\text{-}Disjoint}(\neg Q,\mathbf{I}), \end{aligned}$$

and for all $R, R' \in$ Multi-Effect$(D_Q, \mathbf{I})$, $R \cap R' = \emptyset$ iff for all $R, R' \in$ Multi-Effect$(D_{\neg Q}, \mathbf{I})$, $C(R) \cap C(R') = \emptyset$. □

Finally, the *Unique* semantics exactly characterizes the local, but not global non-deterministic queries.

Definition 3.14
Let Q be a non-deterministic query. The *Unique* semantics on Q defines the query *Unique*(Q):

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \\ \text{OUTPUT} & : R \text{ such that } \{\!\{R\}\!\} = \text{Multi-Effect}(D_Q,\mathbf{I}), \end{cases}$$

if there exists such a medium D_Q and is undefined otherwise.

The *Unique* semantics is defined on Q over $\mathbf{I}$, if *Co-Disjoint*$(Q,\mathbf{I})$ and *Disjoint*$(Q,\mathbf{I})$ are both defined and *Co-Disjoint*$(Q,\mathbf{I})$ = *Disjoint*$(Q,\mathbf{I})$. It is worth to remark that when the *Unique* semantics is defined on Q over $\mathbf{I}$ then *Unique*$(Q) = Q$, moreover *Possible-is-Certain*(Q) = *Unique*(Q).

4 Bounded complexity queries

In this section, we introduce classes of queries computable on machines of various complexity classes. We first consider deterministic queries. Let $\mathcal{C}$ be a complexity class defined relatively to decision problems. We introduce the class $\mathcal{QC}$ of deterministic queries *solvable* in $\mathcal{C}$, following the convention of [CH82, KR90]. As usual, we define the *data complexity* of a query as the computational complexity of testing the membership of a tuple in the answer.

Definition 4.1
A k-ary query Q is in $\mathcal{QC}$ if the decision problem:

$$\begin{cases} \text{INPUT} & : \text{a database instance } \mathbf{I} \text{ and a tuple } \bar{a} \in \text{adom}(\mathbf{I})^k, \\ \text{OUTPUT} & : \textsc{true} \text{ if } \bar{a} \in Q(\mathbf{I}), \end{cases}$$

is solvable in $\mathcal{C}$ in the size of $\mathbf{I}$.

In the sequel, we consider the following classes of queries: $\mathcal{Q}$P, $\mathcal{Q}$NP, $\mathcal{Q}$co-NP,

$\mathcal{Q}$NP$\cap$co-NP, $\mathcal{Q}$UP, etc. The data complexity is defined with respect to the recognition problem (verify that a specific tuple is in the output), and not with respect to the computation problem (computation of the whole output). For various complexity classes (such as P), the two problems have equivalent complexity. This is not always the case though. In particular, for non-deterministic complexity classes, there is a gap of complexity between the decision and the constructive problems. Consider a query Q in $\mathcal{Q}$NP. A constructive machine with complexity P^{NP} can be designed. To compute Q, it enumerates all the tuples (with respect to the ordering given by the encoding) and for each one, checks the membership problem by an oracle (in NP). The use of non-deterministic machines permits the enumeration of all the permutations of a set of constants in polynomial time in the length of the set. The two problems, recognition and computation, coincide if P = NP.

Considering the decision problem:

$$\begin{cases} \text{INPUT} & : \text{a database instance } \mathbf{I} \text{ and a tuple } \bar{a} \in \mathrm{adom}(\mathbf{I})^k, \\ \text{OUTPUT} & : \text{TRUE if } \bar{a} \in R, \text{ where } (\mathbf{I},R) \in Q, \end{cases}$$

has no interest when the query is non-deterministic, since there may be several different relations R as outputs on each input $\mathbf{I}$. A way to generalize the notion of complexity of queries to non-deterministic ones, could be to consider the complexity of the following decision problem:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \text{ and a relation } R, \\ \text{OUTPUT} & : \text{TRUE if } (\mathbf{I},R) \in Q. \end{cases}$$

We do not choose this notion of complexity because it does not reflect the complexity of the computation of the query on an input $\mathbf{I}$. The decision problem associated with a non-deterministic query only consists in verifying that a relation is an output, supposing it has already been computed somehow.

We next focus on non-deterministic queries *computable* with bounded resources. For a complexity class f-$\mathcal{C}$ of constructive problems, We introduce the class N$\mathcal{QC}$ of *non-deterministic* queries *computable* in $\mathcal{C}$[2]. Note that a non-deterministic query can only be computed with a non-deterministic machine, and so the class $\mathcal{C}$ in N$\mathcal{QC}$ is necessarily non-deterministic. Unlike for deterministic queries, the definition of N$\mathcal{QC}$ is related to the constructive problem. This discrepancy appears when considering the functional fragment of non-deterministic languages (see Theorem 4.8).

Definition 4.2
A k-ary query Q is in N$\mathcal{QC}$ if the constructive problem:

$$\begin{cases} \text{INPUT} & : \text{a database instance } \mathbf{I} \\ \text{OUTPUT} & : \text{a possible answer for } Q(\mathbf{I}). \end{cases}$$

is computable in $\mathcal{C}$ in the size of $\mathbf{I}$.

A query Q is in N$\mathcal{QC}$, if there exists a Turing machine M_Q such that on each input $\mathbf{I}$,

[2] We will write $\mathcal{C}$ instead of f-$\mathcal{C}$.

- if M_Q reaches an accepting state then the word written on the output-tape is the encoding of a relation R such that $(\mathbf{I},R)\in Q$,
- and, if R is a relation such that $(\mathbf{I},R)\in Q$, then there exists a computation branch of M_Q over $\mathbf{I}$ such that the encoding of R is written on the output-tape when M_Q reaches its accepting state.

A machine which computes a boolean query in N$\mathcal{QC}$ is such that on each instance $\mathbf{I}$ when it reaches an accepting state it has on its output-tape the encoding of TRUE or FALSE.

Another possible notation of the class N$\mathcal{QC}$ is $\mathcal{Q}$f-$\mathcal{C}$, since the considered complexity is the one of the constructive problem. We prefer to denote the class of non-deterministic queries computable in the class $\mathcal{C}$ by N$\mathcal{QC}$, because the latter notation insists (with the N) on the fact that the queries are non-deterministic.

In the following, the total queries of N$\mathcal{QC}$ will be of particular interest. The complement of a non-deterministic query is introduced in Definition 3.6. We first show that for any reasonable complexity class $\mathcal{C}$, N$\mathcal{QC}$ is closed under complementation.

Proposition 4.1
If $\mathcal{C}$ is a complexity class closed under LOGSPACE reductions, then N$\mathcal{QC}$ is closed under complementation.

Proof : Let Q be a k-ary query in N$\mathcal{QC}$ computed by M_Q. If $\mathcal{C}$ is closed under LOGSPACE reductions, then it is easy to build the machine $M_{\neg Q}$ which computes $\neg Q$ in $\mathcal{C}$. Let $M_{\neg Q}$ be the machine whose transition function extends the transition function of M_Q with new transition rules having the following effect: when the machine reaches an accepting state of M_Q, it computes an enumeration (with respect to the order of the encoding) of all possible k tuples and writes on the output-tape the tuples which were not in the relation computed by M_Q. When the enumeration is done, it moves to the accepting state of $M_{\neg Q}$. $M_{\neg Q}$ is a non-deterministic Turing machine which computes $\neg Q$ that has the same bounded resources as M_Q. □

A boolean query Q is in $\mathcal{QC}$ if there is a machine such that on each input $\mathbf{I}$: $Q(\mathbf{I})$ = TRUE iff there is an accepting computation branch. A boolean query Q is in N$\mathcal{QC}$ if there is a machine such that on each input $\mathbf{I}$: $Q(\mathbf{I})$ = TRUE if there is an accepting computation branch such that TRUE is encoded on the output-tape, $Q(\mathbf{I})$ = FALSE if there is an accepting computation branch such that FALSE is encoded on the output-tape.

We now consider the correspondence between (i) inclusions existing between different complexity classes, and (ii) inclusions between classes of queries defined from them, in the spirit of the classical results of Chandra and Harel [CH82]. The inclusions between the complexity classes are summarized in Figure 7 at the end of the paper. We first recall the classical result [CH82]:

Theorem 4.2
If $\mathcal{C}$ and $\mathcal{C}'$ are complexity classes both closed under LOGSPACE reductions, then $\mathcal{C} \subset \mathcal{C}'$ iff $\mathcal{QC} \subset \mathcal{QC}'$.

The result extends to classes of non-deterministic queries.

Proposition 4.3
If $\mathcal{C}$ and $\mathcal{C}'$ are complexity classes both closed under LOGSPACE reductions, then f-$\mathcal{C} \subset$ f-$\mathcal{C}'$ iff N$\mathcal{QC} \subset$ N$\mathcal{QC}'$.

The proof is similar to the initial proof of [CH82].

Proof : Suppose f-$\mathcal{C} \not\subset$ f-$\mathcal{C}'$. There exists a partial function f such that $f \in$ f-$\mathcal{C}$ and $f \notin$ f-$\mathcal{C}'$. There exists a Turing machine M_f such that, on each input I, M_f writes $O = f(I)$ on its output-tape when it reaches an accepting state. The input I of M_f is a word over the alphabet $\{0,1\}^*$. Let $\mathbf{P}= \{P_1, P_2\}$ be a vocabulary of signature $\langle 1, 2\rangle$. An instance $\mathbf{I}$ of $\mathbf{P}$ represents $I = i_1 i_2 \ldots i_n$ if adom($\mathbf{I}$)=$\{a_1, \ldots, a_n\}$, $a_j \in P_1[\mathbf{I}]$ iff $i_j = 1$ and $P_2[\mathbf{I}]$ is a successor relation on adom($\mathbf{I}$) such that $(a, b) \in P_2$ iff there exists j such that $a = a_j$ and $b = a_{j+1}$. Let Q be the following query:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I}, \\ \text{OUTPUT} & : f(I) \text{ if } \mathbf{I} \text{ represents a word } I \text{ over } \{0,1\}^*, \text{ undefined, otherwise.} \end{cases}$$

$f(I)$ is a word, not a database, but can be represented by a relation storing the positions of the 1's in $f(I)$. For simplicity, we just write $f(I)$. Q is a query of N$\mathcal{QC}$. The Turing machine M_Q, on an input I, (1) verifies that I is the encoding of an instance $\mathbf{I}$; (2) verifies if $\mathbf{I}$ represents a word over $\{0,1\}^*$, and (3) extends the transitions of M_f. Since $\mathcal{C}$ is closed under LOGSPACE reductions, steps (1) and (2) are realized in the resources of $\mathcal{C}$ and Q is in N$\mathcal{QC}$. On the other hand, Q is not in N$\mathcal{QC}'$, otherwise there would exist a Turing machine which solves f with the resources of $\mathcal{C}'$ which contradicts $f \notin$ f-$\mathcal{C}'$. The converse is immediate. □

$\mathcal{QC}$ is related to the problem of verifying that a given tuple is in the answer of the query, while N$\mathcal{QC}$ is related to the problem of computing the query. The former problem is usually easier than the latter. The question whether all deterministic queries solvable in $\mathcal{C}$ are computable in $\mathcal{C}$ is expressed by $\mathcal{QC} \overset{?}{\subseteq}$ N$\mathcal{QC}$. It is easy to see that the answer is positive when $\mathcal{C}$ is P.

Proposition 4.4
$\mathcal{Q}$P=N$\mathcal{Q}$P^T.

Proof : Let Q be a query of arity k in $\mathcal{Q}$P. There exists a Turing machine M_Q which solves the decision problem:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I} \text{ and a } k\text{-tuple } \bar{a}, \\ \text{OUTPUT} & : \text{TRUE if } \bar{a} \in Q(\mathbf{I}). \end{cases}$$

in polynomial time. Let M be the constructive machine which (1) verifies that the input is the encoding of an instance $\mathbf{I}$; (2) enumerates all k-tuples (with respect to the linear order derived from the encoding); and (3) uses the transitions of M_Q over $\langle \mathbf{I}, \bar{a}\rangle$ for all

$\bar{a}$ in the enumeration and writes $\bar{a}$ on its output-tape when M_Q accepts. M computes Q in deterministic polynomial time. Conversely, let M be a machine that computes Q and M' defined as follows. On an input $\langle \mathbf{I}, \bar{a} \rangle$, M' (1) uses the transitions of M on $\mathbf{I}$ and (2) accepts if $\bar{a}$ is written on the output-tape of M. M' solves the decision problem associated to the query Q. $\square$

The question remains open for the class NP. It is easy to build a machine which computes a query of $\mathcal{Q}$NP from its associated decision problem as in proof of Proposition 4.4, but the resulting constructive machine is of complexity P^{NP}. It follows $\mathcal{Q}\mathrm{NP} \subseteq \mathrm{N}\mathcal{Q}\mathrm{P}^{\mathrm{NP}}$. We have seen that it is in general unlikely that $\mathcal{QC} \subseteq \mathrm{N}\mathcal{QC}$. On the other hand, the deterministic queries in N$\mathcal{QC}$ are in $\mathcal{QC}$, that is funct-N$\mathcal{QC} \subseteq \mathcal{QC}$. The relation between $\mathcal{QC}$ and N$\mathcal{QC}$ is explicited in the next proposition.

Proposition 4.5
$\mathcal{QC} \subseteq \mathrm{N}\mathcal{QC}$ iff $\mathcal{C} = \text{co-}\mathcal{C}$, for a class $\mathcal{C}$ closed under LOGSPACE reducibility.

Proof : Let $S \subseteq \{0,1\}^*$ be a language in $\mathcal{C}$. Like in proof of Proposition 4.3, any word $w = w_1 w_2 ... w_k$ in L can be encoded in LOGSPACE as an instance I_w with domain the segment of the first k integers, with a linear order on it, and a unary relation storing the positions of the 1's in the word w. Let Q_L be the query defined by:

$$\begin{cases} \text{INPUT} & : \text{an instance } \mathbf{I}, \\ \text{OUTPUT} & : \{[]\} \text{ if } \mathbf{I} \text{ represents a word } w \text{ of } L \subseteq \{0,1\}^*, \ \emptyset, \text{ otherwise.} \end{cases}$$

Q_L is a query of $\mathcal{QC}$. For a class $\mathcal{C}$ closed under LOGSPACE reducibility, L is a language of $\mathcal{C}$ iff Q_L is in $\mathcal{QC}$. If $\mathcal{QC} \subseteq \mathrm{N}\mathcal{QC}$, then the query Q_L is in N$\mathcal{QC}$, which is closed under complementation (Proposition 4.1). It follows that $\neg Q$ is in N$\mathcal{QC}$ and L is in co-$\mathcal{C}$. The converse is immediate. $\square$

Corollary 4.6
$\mathcal{Q}\mathrm{NP} \subseteq \mathrm{N}\mathcal{Q}\mathrm{NP}$ iff NP=co-NP.

As defined in Section 3, we denote the set of possible accepted outputs of a query Q by Effect$(Q, \mathbf{I})$. Recall that in Section 2.2 we made the assumption that the Turing machines we consider are such that for each input $\mathbf{I}$, if R_1 and R_2 are two distinct strings written on the output-tape of the machine in an accepting state, then R_1 and R_2 are the encoding of two different relations on $\mathbf{I}$. In Example 4.1 below, R_1, R_2 and R_3 are supposed to encode three different relations.

We now introduce formally the notion of Multi-Effect of a query Q, computed by a machine M_Q on an instance $\mathbf{I}$.

Definition 4.3
Let Q be a non-deterministic k-ary query computed by a machine M_Q and $\mathbf{I}$ an instance. The Multi-Effect of a query Q, computed by a machine M_Q on an instance $\mathbf{I}$ is the multiset Multi-Effect$(M_Q, \mathbf{I})$, constituted by all the accepted outputs such that each

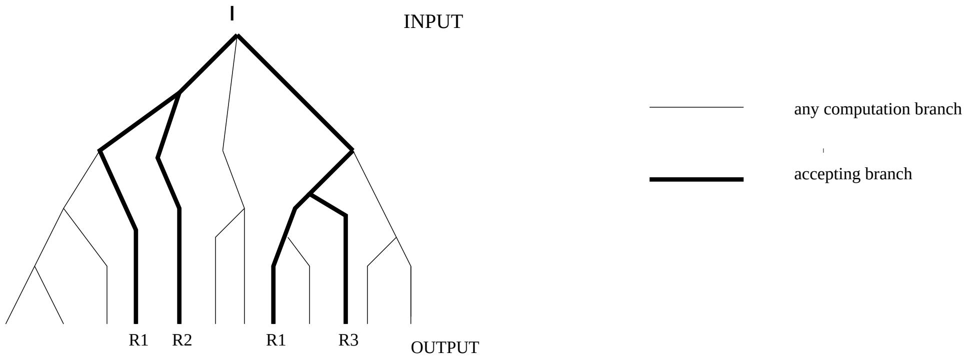

Figure 4: Computation tree of M_Q on $\mathbf{I}$

accepted output is repeated in Multi-Effect(M_Q, $\mathbf{I}$) as many times as M_Q has accepting computation branches which lead to this output on $\mathbf{I}$.

It follows that the multiset Multi-Effect(M_Q,$\mathbf{I}$) for a non-deterministic k-ary query Q, computed by a machine M_Q on an instance $\mathbf{I}$ has as many occurrences of outputs as M_Q has accepting branches on $\mathbf{I}$. The notion of effect and multi-effect of machines on an input are illustrated in the next example.

Example 4.1
Consider a query Q computable by the machine M_Q and $\mathbf{I}$ an instance, such that (see Figure 4) :

$$\text{Effect}(M_Q, \mathbf{I}) = \{R_1, R_2, R_3\}$$

$$\text{Multi-Effect}(M_Q, \mathbf{I}) = \{\!\{R_1, R_1, R_2, R_3\}\!\}$$

We consider in the following the classes of total queries N$\mathcal{Q}$NP^T and N$\mathcal{Q}$UP^T. We first recall that as a consequence of Proposition 4.1, the class N$\mathcal{Q}$NP^T is closed under complementation. The same holds for unambiguous Turing machines.

We now study classes of deterministic queries defined from the class N$\mathcal{Q}$NP^T using functionalization techniques. We first prove the following result.

Theorem 4.7
The following equivalences hold:

1. *Possible*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$NP.
2. *Certain*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$co-NP.

Proof : 1. *Possible*($\mathrm{N\mathcal{Q}NP}^T$) = $\mathcal{Q}$NP. Let $Q \in \mathrm{N\mathcal{Q}NP}^T$ be a non-deterministic k-ary query. Let M_Q be a non-deterministic TM which computes Q. On each input $\mathbf{I}$, M_Q halts in an accepting state with an output R written on the output tape. Let M^* be the following machine. M^* first checks if its input is the encoding of a structure $\mathbf{I}$, and of a k-ary tuple $\bar{a}$ (rejects otherwise). The computation of M^* consists then in first letting M_Q work on $\mathbf{I}$, and then verifying if $\bar{a}$ is in the relation written on the output-tape of M_Q. On each input $\langle \mathbf{I}, \bar{a}\rangle$, M^* accepts iff $\bar{a} \in R$ where $R \in \mathrm{Effect}(M_Q, \mathbf{I})$. It follows that M^* is a non-deterministic TM which computes *Possible*(Q) in polynomial time.

Conversely, let $Q \in \mathcal{Q}\mathrm{NP}$ be a deterministic k-ary query and M_Q a non-deterministic TM which computes Q. On each input $\langle \mathbf{I}, \bar{a}\rangle$, M_Q halts in an accepting state if $\bar{a} \in Q(\mathbf{I})$. Let M^* be the following machine. It first checks if its input is the encoding of a structure $\mathbf{I}$ (rejects otherwise). The computation of M^* consists in first guessing a k-ary tuple $\bar{a}$, then in letting M_Q work on $\mathbf{I}$ and $\bar{a}$. If M_Q accepts, then M^* accepts and writes $\bar{a}$ on its output-tape. Otherwise, M^* rejects. Let Q^* be the query computed by M^*. It is clear that *Possible*$(Q^*, \mathbf{I}) = \bigcup_{\bar{a} \in Q(\mathbf{I})} \{\bar{a}\}$. It follows that *Possible*$(Q^*) = Q$.

2. *Certain*($\mathrm{N\mathcal{Q}NP}^T$) = $\mathcal{Q}$co-NP follows now from Proposition 3.1. □

In the following, the classes of deterministic queries that we consider correspond to queries computable in polynomial time on non-deterministic Turing machines with very specific characteristics. The results shed some light on the relationships between the functionalization tools and the properties of the machines. We next consider the *functional fragment* of $\mathrm{N\mathcal{Q}NP}^T$ which is captured by the *Possible-is-Certain* semantics (see Proposition 3.2) and can be easily characterized.

Theorem 4.8
The functional fragment of $\mathrm{N\mathcal{Q}NP}^T$, funct-$\mathrm{N\mathcal{Q}NP}^T$, coincides with $\mathcal{Q}$NP∩co-NP.

Proof : We first prove that $\mathcal{Q}$NP∩co-NP $\subseteq$ funct-$\mathrm{N\mathcal{Q}NP}^T$. A k-ary query Q is in $\mathcal{Q}$NP∩co-NP iff it is both in $\mathcal{Q}$NP and in $\mathcal{Q}$co-NP. Let M_Q and $M_{\neg Q}$ be the boolean non-deterministic TMs which compute respectively Q and its complement, $\neg Q$, in polynomial time. Let M^* be the following constructive machine computing Q. It first checks if its input is the encoding of a structure $\mathbf{I}$ then it begins the computation, otherwise it rejects. The computation of M^* consists in first enumerating the sequence of all k-ary tuples $\bar{a}_1, \ldots, \bar{a}_n$, then, successively for each $\bar{a}_i$, in letting M_Q and $M_{\neg Q}$ work in parallel on $\mathbf{I}$ and $\bar{a}_i$. When M_Q accepts, $\bar{a}_i$ is written on the output-tape and $\bar{a}_{i+1}$ is considered, while when $M_{\neg Q}$ accepts, $\bar{a}_{i+1}$ is considered. Since on each input $\mathbf{I}$ and $\bar{a}_i$, one of the two machines will accept, it follows that for each input $\mathbf{I}$, there is a unique relation R in $\mathrm{Effect}(M^*, \mathbf{I})$, and therefore that Q is in funct-$\mathrm{N\mathcal{Q}NP}^T$.

Conversely, let Q be a deterministic k-ary query in funct-$\mathrm{N\mathcal{Q}NP}^T$ and $\mathbf{I}$ an instance. Q is computable in polynomial time with a non-deterministic Turing machine such that for each input, all accepting computations yield one and the same output relation. It follows that the *Possible-is-Certain* semantics is defined on Q and $\mathbf{I}$, since *Possible*(Q,$\mathbf{I}$) = *Certain*(Q,$\mathbf{I}$). Therefore, by Theorem 4.7, it follows that such a query Q is both in $\mathcal{Q}$NP and in $\mathcal{Q}$co-NP. □

Note that at first glance, we could have expected that the functional fragment of $\mathrm{N\mathcal{Q}NP}^T$ coincides with $\mathcal{Q}$NP. This discrepancy comes from the definitions of determin-

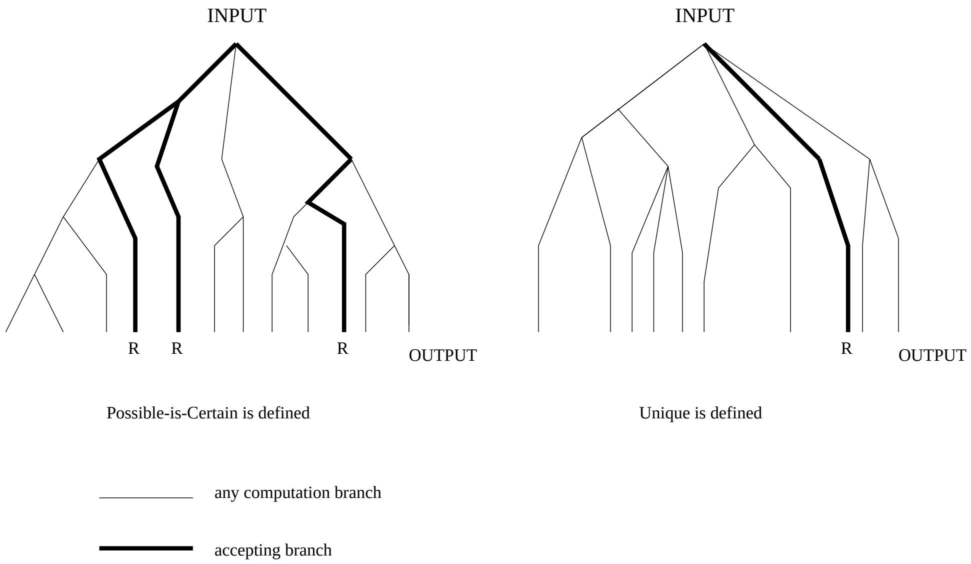

Figure 5: Computation trees on which *Possible-is-Certain* and *Unique* are defined

istic and non-deterministic classes of queries, the former being based on the recognition problem, and the latter on the computation problem.

The next result shows that it is very unlikely that the functional fragment of N$\mathcal{Q}$NP^T coincides with $\mathcal{Q}$NP. From Proposition 4.5 follows that the class of deterministic queries computable in non-deterministic polynomial time, $\mathcal{Q}$NP, is not included in the class of non-deterministic queries computable in non-deterministic polynomial time, N$\mathcal{Q}$NP, unless NP = co-NP.

The *Possible-is-Certain* semantics is not defined on all non-deterministic queries. It is easy to see that the *Possible-is-Certain* semantics is defined on a query Q computed by a machine M_Q iff on each instance $\mathbf{I}$, there exists a relation R such that Multi-Effect(M_Q, $\mathbf{I}$)=$\{\!\{R, R, \ldots, R\}\!\}$ (see Figure 5).

We consider now unambiguous computations. It is easy to see that N$\mathcal{Q}$UP is a subclass of N$\mathcal{Q}$NP constituted of deterministic queries. It follows from Theorem 4.8 that N$\mathcal{Q}$UP^T $\subseteq$ $\mathcal{Q}$NP$\cap$co-NP. More precisely, we prove that N$\mathcal{Q}$UP^T = $\mathcal{Q}$UP$\cap$co-UP.

Theorem 4.9
N$\mathcal{Q}$UP^T = funct-N$\mathcal{Q}$UP^T = $\mathcal{Q}$UP$\cap$co-UP.

Proof : The first equality is by definition since queries in N$\mathcal{Q}$UP^T are deterministic. Let $Q \in$ N$\mathcal{Q}$UP^T be a k-ary query. Let M_Q be an UTM which computes Q. On each input $\mathbf{I}$, it halts in an accepting state with an output R written on the output-tape. Let N_Q be the non-deterministic TM which first reads its input, and if it is the encoding of a structure $\mathbf{I}$ and a k-ary tuple $\bar{a}$, it begins the computation, otherwise it rejects. The computation of N_Q consists in first letting M_Q work on $\mathbf{I}$ and then checking if $\bar{a}$ is written on the output-tape of M_Q. N_Q is unambiguous since M_Q is an UTM and the final step is deterministic. N_Q is a boolean UTM which solves the decision problem associated to Q. Q is in $\mathcal{Q}$UP. Q is also in $\mathcal{Q}$co-UP, since N$\mathcal{Q}$UP^T is closed under complementation.

Conversely, let Q be a query in $\mathcal{Q}$UP$\cap$co-UP. Let N_Q and $N_{\neg Q}$ be UTMs which solve the decision problem associated respectively to Q and its complement. Let M_Q be the following machine. It first checks if its input is the encoding of a structure $\mathbf{I}$, then it begins the computation, otherwise it rejects. The computation of M_Q consists in first enumerating the ordered sequence $\bar{a}_1, \ldots, \bar{a}_n$ of all k-ary tuples (with respect to an order uniquely deduced from the encoding). Then for each $\bar{a}_i$, M_Q lets N_Q and $N_{\neg Q}$ work in parallel on $\langle \mathbf{I}, \bar{a}_i \rangle$. When N_Q accepts $\bar{a}_i$, then $\bar{a}_i$ is written on the output-tape and $\bar{a}_{i+1}$ is considered, while when $N_{\neg Q}$ accepts $\bar{a}_i$, then $\bar{a}_{i+1}$ is considered. Since on each input $\mathbf{I}$ and $\bar{a}_i$, one of the two machines will accept, and they both are UTMs, it follows that M_Q is an unambiguous non-deterministic machine which computes Q in polynomial time. So Q is in N$\mathcal{Q}$UP^T. □

Since the class N$\mathcal{Q}$UP^T is a class of deterministic queries, there is no interest to consider deterministic semantics applied on it.

We next consider the *disjoint* and *co-disjoint* semantics. Theorem 4.10 is the analogous to Theorem 4.7 for these semantics.

Theorem 4.10
The following equivalences hold:

1. *Disjoint*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$UP.
2. *Co-Disjoint*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$co-UP.

Proof: *Disjoint*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$UP. Let $Q \in$ N$\mathcal{Q}$NP^T be a non-deterministic k-ary query computable in polynomial time. Let M_Q be a non-deterministic TM which computes Q. On each input $\mathbf{I}$, it halts in an accepting state with an output R written on the output-tape. The *Disjoint* semantics is defined if the machine M_Q is such that for all $R, R' \in$ Multi-Effect(M_Q, I), $R \cap R' = \emptyset$. Let M^* be the following machine. It first checks if its input is the encoding of a structure $\mathbf{I}$ and a k-ary tuple $\bar{a}$, then it begins the computation, otherwise it rejects. The computation of M^* consists in first letting M_Q work on $\mathbf{I}$ and then verifying if $\bar{a}$ is in the relation written on the output-tape of M_Q. On each input $\langle \mathbf{I}, \bar{a} \rangle$, M^* accepts iff $\bar{a} \in R$ where $R \in$ Multi-Effect$(M_Q, \mathbf{I})$. Since the *Disjoint* semantics is defined, for each $\bar{a}$ accepted by M^*, there is a unique $R \in$ Multi-Effect$(M_Q, \mathbf{I})$ such that $\bar{a} \in R$. It follows that M^* is an unambiguous Turing machine and therefore *Disjoint*(Q) is in $\mathcal{Q}$UP.

Conversely, let $Q \in \mathcal{Q}$UP be a deterministic k-ary query, and M_Q an UTM which computes Q in polynomial time. On each input $\langle \mathbf{I}, \bar{a} \rangle$, M_Q halts in an accepting state if $\bar{a} \in Q(\mathbf{I})$. Let M^* be the following machine. It first checks if its input is the encoding of a structure $\mathbf{I}$, then it begins the computation, otherwise it rejects. The computation of M^* consists in first guessing a k-ary tuple $\bar{a}$, then in letting M_Q work on $\mathbf{I}$ and $\bar{a}$. If M_Q accepts then M^* accepts and writes $\bar{a}$ on its output-tape, otherwise, M^* rejects. Let Q^* be the query computed by M^*. Since M_Q is a UTM, it has at most one accepting branch on each input. Therefore, when M_{Q^*} is in an accepting state, Multi-Effect(M_{Q^*},$\mathbf{I}$)=$\{\!\{\{\bar{a}\}\}\!\}$ for a tuple of arity k in $\mathbf{I}$. It is then clear that *Disjoint*$(Q^*, \mathbf{I}) = \bigcup_{\bar{a}\in Q(\mathbf{I})}\{\bar{a}\}$. It follows that *Disjoint*$(Q^*) = Q$.

2. *Co-Disjoint*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$co-UP follows from Proposition 3.4. □

Finally, the deterministic queries defined by the *Unique* semantics on N$\mathcal{Q}$NP^T coincide with the deterministic queries in N$\mathcal{Q}$UP^T = $\mathcal{Q}$UP∩co-UP.

Theorem 4.11
Unique(N$\mathcal{Q}$NP^T) = N$\mathcal{Q}$UP^T = $\mathcal{Q}$UP∩co-UP

Proof: Let Q be a k-ary query in N$\mathcal{Q}$NP^T and $\mathbf{I}$ an instance. By definition, *Unique* is defined on Q and $\mathbf{I}$ iff *Disjoint*(Q,$\mathbf{I}$) and *Co-Disjoint*(Q,$\mathbf{I}$) are both defined and coincide. Let M_Q be a non-deterministic Turing machine which computes Q. Then *Disjoint*(Q,$\mathbf{I}$)=*Co-Disjoint*(Q,$\mathbf{I}$) and for all $R, R' \in$ Multi-Effect$(M_Q, \mathbf{I})$ $R \cap R' = \emptyset$ and $C(R) \cap C(R') = \emptyset$. It follows that M_Q is such that there is R such that Multi-Effect$(M_Q, \mathbf{I}) = \{\!\{R\}\!\}$. That is M_Q is an UTM. The converse is rather immediate. □

Theorem 4.11 gives a precise condition on the nature of the non-determinism of the machines which compute queries in N$\mathcal{Q}$NP^T on which the *Unique* semantics is defined. Indeed, the *Unique* semantics is defined on Q in N$\mathcal{Q}$NP^T iff there exists a locally but not globally non-deterministic constructive Turing machine which computes Q in polynomial time.

5 Deterministic languages

In this section, we consider various deterministic languages. A language $\mathcal{L}$ is called deterministic if its semantics is deterministic, that is every formula has a unique interpretation. We consider in the following first-order logic (FO), fixpoint logics (FO+IFP) [GS85, AV89], and fragments of second-order logic (SO), such as first-order implicit definition (IMP), existential second-order logic (Σ^1_1), universal second-order logic (Π^1_1), and generalized implicit definitions (GIMP) [GLL95]. Note that only FO, FO+IFP, Σ^1_1, Π^1_1 and SO are recursive sets of formulae.

A query Q of arity k is defined in a logic $\mathcal{L}$, if there is a formula $\varphi(\bar{x}) \in \mathcal{L}$ with k free variables such that on each $\mathbf{P}$-structure $\mathbf{I}$, and each k-tuple $\bar{a}$ in $\mathbf{I}$, $\bar{a} \in Q(\mathbf{I})$ iff $\mathbf{I} \models \varphi(\bar{a})$. In the following, we identify the logic $\mathcal{L}$ with the class of queries defined in $\mathcal{L}$.

Definition 5.1
Let $\mathbf{P}$ be a vocabulary,
- Σ^1_1 is the set of formulae of the form $\exists S\ \varphi(S, \bar{x})$ over $\mathbf{P} \cup \{S\}$ such that $\varphi(S, \bar{x}) \in$ FO.
- Π^1_1 is the set of formulae of the form $\forall S\ \varphi(S, \bar{x})$ over $\mathbf{P} \cup \{S\}$ such that $\varphi(S, \bar{x}) \in$ FO.
- Δ^1_1 is the subset of Σ^1_1 of formulae which are equivalent to a formula in Π^1_1.
- Fixpoint logic, FO+IFP, is the set of first-order formulae extended by formulae of the form $\mathrm{FP}_{S,\bar{x}}(\varphi(S, \bar{x}))$ over $\mathbf{P}\cup\{S\}$ such that $\varphi(S, \bar{x}) \in$ FO, and $\mathrm{FP}_{S,\bar{x}}$ is a fixpoint operator.
- IMP is the subset of Σ^1_1 consisting of formulae of the form $\exists U\ \Psi(U) \wedge \varphi(U, \bar{x})$, where

$\Psi(U)$ and $\varphi(U, \bar{x})$ are formulae in FO over $\mathbf{P} \cup \{U\}$, such that for each structure $\mathbf{I}$, there exists a unique relation R^I such that $I \models \Psi(R^I)$.

Σ_1^1, Π_1^1 and Δ_1^1 respectively coincide with $\mathcal{Q}$NP, $\mathcal{Q}$co-NP and $\mathcal{Q}$NP$\cap$co-NP [Fag74]. It is well known that on the class of all structures, IMP = FO [Bet53]. Gurevich has proven that on the class of finite structures, IMP strictly contains FO [Gur84]. The result was extended by showing that IMP strictly contains fixpoint logic FO+IFP [DHK95]. Over the class of finite structures, IMP is a complete co-recursively enumerable set of formulae. The problem whether or not $\exists U\ \Psi(U) \wedge \varphi(U, \bar{x})$ over $\mathbf{P} \cup \{U\}$, where $U \notin \mathbf{P}$ of arity k, is in IMP is not decidable [Kol90]. IMP is clearly co-r.e. since for each sentence $\exists U\ \Psi(U) \wedge \varphi(U, \bar{x})$ we can enumerate (up to isomorphism) all finite $\mathbf{P}$-structures $\mathbf{I}$, and check if there is no R^I such that $\mathbf{I} \models \Psi(R^I)$ or if there are at least two different relations which satisfy $\Psi(U)$ on $\mathbf{I}$.

Example 5.1
Let $\mathbf{P}$ be a vocabulary containing only one binary relation symbol $<$. The unary query Q consisting of the set of even elements with respect to $<$, if $<$ is a linear order on the structure, and of the empty set otherwise, is implicitly defined by $\exists S\ \Psi(S) \wedge Sx$ where S is a new unary relation symbol, and

$$\Psi(S) \equiv [\theta(<) \rightarrow \phi(S)] \wedge [\neg\theta(<) \rightarrow \forall x \neg Sx],$$

where $\theta(<)$ asserts that $<$ is a linear order and

$$\phi(S) \equiv \forall x[(\forall y\ \neg y < x) \rightarrow \neg Sx] \wedge \forall x \forall y\ [\neg Sx \leftrightarrow (\neg(\exists z\ x < z < y) \rightarrow Sy)]$$

expresses that S is the set of even elements with respect to the linear order $<$. More succinctly, if the domain is $\{1, \ldots, n\}$, $\phi(S)$ says $\neg S(1) \wedge \forall x[\neg S(x) \leftrightarrow S(x+1)]$. Q is not explicit since there is no first-order formula that defines the set of even elements on finite total orders (proof by 0/1 laws [Fag76], or Ehrenfeucht-Fraïssé games [Fra54, Ehr61]).

GIMP is a generalization of IMP introduced in [GLL95] to relax the strong uniqueness condition. Instead of requiring sentences to admit a unique solution, the uniqueness of the solution is ensured only "modulo" another formula. More precisely, given a sentence $\psi(S)$ and a formula $\varphi(S, \bar{x})$, it is required that on each structure $\mathbf{I}$, $\psi(S)$ always has at least one solution S^I, and that $\{\bar{x} | \varphi(S^I, \bar{x})\}$ is uniquely defined independently of the choice of S^I. In a sense, S is being used as a (non-deterministic) "working area" for the purpose of defining a deterministic query.

Definition 5.2
GIMP is the set of formulae $\varphi(S, \bar{x})$ mod $\psi(S)$, where $S \notin \mathbf{P}$, $\psi(S)$ is a sentence such that for each structure $\mathbf{I}$, $\mathbf{I} \models \exists S \psi(S)$, and $\varphi(S, \bar{x}) \in$ FO is such that $\mathbf{I} \models \forall S \forall T[(\psi(S) \wedge \psi(T)) \rightarrow \forall \bar{x}(\varphi(S, \bar{x}) \leftrightarrow \varphi(T, \bar{x}))]$.

Example 5.2
The Evenness query is definable in GIMP. Let S be a relational symbol of arity 3 that is not in the vocabulary. The formula $\psi(S)$ will define the Cartesian product of a (strict)

linear order and the set of even numbered elements with respect to it.

$$\begin{aligned}
\psi(S) \quad \equiv \quad & \forall x \forall y \, [\exists u Sxyu \vee x = y \vee \exists u Syxu] & (1)\\
& \wedge \forall x \forall y \forall u \, [Sxyu \rightarrow (\forall v \neg Syxv \wedge x \neq y)] & (2)\\
& \wedge \forall x \forall y \forall z \forall u \forall v \, [(Sxyu \wedge Syzv) \rightarrow \exists w Sxzw] & (3)\\
& \wedge \forall x \, [x = Min \rightarrow \forall y \forall z \neg Syzx] & (4)\\
& \wedge \forall x \forall y \forall u \forall v \, [(Sxyu \wedge v = Suc(u)) \rightarrow \forall z \forall t \neg Sztv] & (5)\\
& \wedge \forall x \forall y \forall u \forall v \, [(\neg Sxyu \wedge \exists w Sxyw \wedge v = Suc(u)) \rightarrow Sxyv] & (6)
\end{aligned}$$

where $x = Min \equiv \forall y \forall u \; \neg Syxu$ and

$$\begin{aligned}
y = Suc(x) \quad \equiv \quad & \exists u \; Sxyu \wedge \forall z \forall u \, [Sxzu \rightarrow (\exists v Syzv \vee z = y)] \\
& \wedge \forall z \forall u \, [Szyu \rightarrow (\exists v Szxv \vee z = x)].
\end{aligned}$$

The first three expressions say that the projection on the first two components is a linear order: (1) says that $<$ is total; (2) says that $<$ is anti-symmetric; and (3) says that $<$ is transitive. The next three expressions define the set of $EVEN$ elements with respect to the order induced on the domain $\{1, \ldots, n\}$ by the two first components: (4) says $1 \notin EVEN$; (5) says $x \in EVEN \rightarrow x + 1 \notin EVEN$; and (6) says roughly that $x \notin EVEN \rightarrow x + 1 \in EVEN$. Finally,

$$\forall x [\forall y \forall u \neg Sxyu \rightarrow \exists y \exists z Syzx] \text{ mod } \psi(S)$$

expresses the fact that the maximal element is in $EVEN$, and so is a definition of EVENNESS in GIMP.

GIMP is obviously included in Σ_1^1 and closed under complementation. Moreover, GIMP is equivalent to Δ_1^1 [GLL95].

Proposition 5.1
GIMP $= \Delta_1^1$.

Like IMP or Δ_1^1, GIMP is not a recursively enumerable set of formulae. The question whether or not $\varphi(S, \bar{x})$ mod $\psi(S)$ is a correct definition in GIMP is clearly co-recursively enumerable. It is co-recursively enumerable complete since the validity problem of first-order sentences over finite structures can be reduced to it. For each first-order sentence θ over a vocabulary $\mathbf{P}$ containing some binary relation symbol, we consider $\neg Sx$ mod $(\theta \wedge \forall x Sx) \vee (\neg\theta \wedge \exists x Sx)$ which is a correct implicit definition in GIMP on the class of finite structures (of cardinality at least 2) iff θ is valid. Trakhtenbrot's theorem asserts that the set of first-order sentences that are finitely valid is a complete co-recursively enumerable set [Tra50].

The question whether there exists a recursively enumerable logic for GIMP is related to the existence of a complete problem for NP $\cap$ co-NP. Gurevich [Gur88] showed that if there exists a logic that captures NP $\cap$ co-NP then NP $\cap$ co-NP has a complete problem with respect to polynomial time reducibility. Since our correspondence is constructive, a recursively enumerable logic for GIMP would testify to the existence of a complete problem for NP $\cap$ co-NP. Gurevich conjectures that if that were the case, something drastic would happen (such as NP $\cap$ co-NP $=$ P).

The generalized implicit definitions of GIMP proposed in Definition 5.2 provide a logical characterization of NP $\cap$ co-NP over finite structures. The language LIMP, introduced in [GLL95], constitutes a restriction of GIMP but a generalization of IMP, where the choice consists of a linear order $\prec$ over the domain of the structure and a unique relation S on $\prec$. More formally, LIMP is defined as follows.

Definition 5.3
LIMP is the set of formulae $\varphi(\prec, S, \bar{x})$ mod $\psi(\prec, S)$, where $\psi(\prec, S)$ is a FO-sentence such that for each structure $\mathbf{I}$ and for each total linear order $\prec$ on $\mathbf{I}$ there exists a unique $S_\prec$ such that $\mathbf{I} \models \psi(\prec, S_\prec)$, and $\varphi(\prec, S, \bar{x}) \in$ FO, satisfying the following property: for each finite $\mathbf{P}$-structure $\mathbf{I}$, and for each $(<, S_<)$ and $(<', S_{<'})$ satisfying $\psi(\prec, S)$ on $\mathbf{I}$, the following holds

$$\mathbf{I} \models \forall \bar{x} \varphi(<, S_<, \bar{x}) \leftrightarrow \varphi(<', S_{<'}, \bar{x}).$$

It is easy to see that LIMP $\subseteq$ GIMP by encoding together the order $\prec$ and the relation S into a new relation symbol U. The Evenness query admits the following definition in LIMP since its non-deterministic part consists of the choice of a linear order $\prec$ and the set of the even elements with respect to $\prec$ (that is unique).

Example 5.3
The Evenness query has a definition in LIMP. Let S be a relational symbol of arity 1 that is not in the vocabulary. The formula $\psi(\prec, S)$ defines the set S of even elements with respect to the chosen order $\prec$.

$$\begin{aligned} \psi(\prec, S) \quad \equiv \quad & \forall x \ [(\forall y \neg y \prec x) \rightarrow \neg Sx] \\ & \wedge \forall x \ \forall y \ \ [(x \prec y \wedge \neg \exists z \ x \prec z \prec y) \rightarrow (Sx \leftrightarrow \neg Sy)] \end{aligned}$$

Then $\forall x[(\forall y \ \neg x \prec y) \rightarrow Sx]$ mod $\psi(\prec, S)$ is a definition of Evenness in LIMP.

Since a definition in LIMP consists of the non-deterministic choice of a linear order, followed by an order-invariant part, it is natural to consider a one-one mapping from the class of queries $\mathcal{Q}$ over finite structures to the class of queries $\mathcal{Q}^*$ over finite ordered structures that are *order-invariant*. For each finite $\mathbf{P}$-structure $\mathbf{I}$ of size n, let $\mathbf{I}_1^*, \ldots, \mathbf{I}_{n!}^*$ be the $n!$ extensions of $\mathbf{I}$ over $\mathbf{P} \cup \{<\}$. For each finite $\mathbf{P} \cup \{<\}$-structure $\mathbf{I}^*$, let $\mathbf{I}$ be its restriction to $\mathbf{P}$, i.e. the finite $\mathbf{P}$-structure $\mathbf{I}$ with the same domain I and the same interpretation of the symbols in $\mathbf{P}$.

For any k-ary query Q over the class of finite $\mathbf{P}$-structures, there exists a k-ary query Q^* over finite $\mathbf{P} \cup \{<\}$-structures where $<$ is interpreted by a linear order, such that, for each finite $\mathbf{P}$-structure $\mathbf{I}$ of size n, $Q(\mathbf{I}) = Q^*(\mathbf{I}_1^*) = \cdots = Q^*(\mathbf{I}_{n!}^*)$. It is clear that such a Q^* is order-invariant. Conversely, for each query Q^* on finite $\mathbf{P} \cup \{<\}$-structures that is *order-invariant* there is a query Q over finite $\mathbf{P}$-structures such that for all finite $\mathbf{P} \cup \{<\}$-structures $\mathbf{I}^*$, $Q(\mathbf{I}) = Q^*(\mathbf{I}^*)$. We can prove the following.

Proposition 5.2
LIMP $= Inv_<(\text{IMP}+<)$.

Proof : Let $\Phi : \varphi(\prec, S, \bar{x})$ mod $\psi(\prec, S)$ be a definition in LIMP. For each $\mathbf{P}$-structure

$\mathbf{I}$, and for each linear order $<$ on $\mathbf{I}$, let $\mathbf{I}^*$ be the resulting extension of $\mathbf{I}$ over $\mathbf{P} \cup \{<\}$. Φ on $\mathbf{I}^*$ defines the set $\{\bar{a} \in I^k \mid \mathbf{I}^* \models \exists S\ \psi(<, S) \wedge \varphi(<, S, \bar{a})\}$. By definition, there exists a unique S^{I^*} such that $\mathbf{I}^* \models \psi(<, S^{I^*})$. It follows that $\exists S\ \psi(<, S) \wedge \varphi(<, S, \bar{x})$ is a formula of IMP + $<$. Moreover, since for each $(<, S_<)$ and $(<', S_{<'})$ satisfying $\psi(\prec, S)$ on $\mathbf{I}$, we have $\mathbf{I} \models \forall \bar{x} \varphi(<, S_<, \bar{x}) \leftrightarrow \varphi(<', S_{<'}, \bar{x})$, it follows that $\{\bar{a} \in I^k \mid \mathbf{I}^* \models \exists S\ \psi(<, S) \wedge \varphi(<, S, \bar{a})\}$ does not depend on the chosen order on $\mathbf{I}$. Therefore, $\exists S\ \psi(<, S) \wedge \varphi(<, S, \bar{x})$ is a definition in $Inv_<$(IMP+ $<$).

Conversely, Let $\exists U\ \Psi(<, U) \wedge \varphi(<, U, \bar{x})$ be a definition in IMP + $<$ such that for each $\mathbf{P}$-structure $\mathbf{I}$, for each two extensions $\mathbf{I}_1^*$ and $\mathbf{I}_2^*$ of $\mathbf{I}$ to $\mathbf{P} \cup \{<\}$, then $\{\bar{a} \in I^k \mid \mathbf{I}_1^* \models \exists U\ \psi(<_1, U) \wedge \varphi(<_1, U, \bar{a})\} = \{\bar{a} \in I^k \mid \mathbf{I}_2^* \models \exists U\ \psi(<_2, U) \wedge \varphi(<_2, U, \bar{a})\}$. Finally, $\varphi(<, U, \bar{x})$ mod $\Psi(<, U)$ is a definition in LIMP. $\square$

An interesting example of an order invariant query that is not first-order definable without the order relation is presented in ([AHV94], Exercise 17.27, page 462). The presence of an order has an important impact on the definability because it rigidifies finite structures. It is not surprising that the query used by Dawar, Hella and Kolaitis in [DHK95] to separate FO+IFP from IMP is an order on a first-order definable class of rigid finite structures. It is well known that the rigidity given by an order on a structure enables the simulation of a computation. The simulation of the computation of a non-deterministic polynomial time Turing machine can be done with an existential second-order formula, since a linear order can be defined with such a formula [Fag74]. When trying to simulate computation of a deterministic polynomial time Turing machine with fixpoint logic, Immerman and Vardi introduced an order on the structure [Imm86, Var82]. It followed that PTIME coincides with $Inv_<$(FO+IFP+ $<$).

The decision problem of whether a first-order formula $\varphi(\prec, S, \bar{x})$ mod $\psi(\prec, S)$, defines a semantically correct query in LIMP over finite $\mathbf{P}$-structures is undecidable. This is proved by an easy reduction from the validity of first-order sentences on finite structures. Let α be a first-order sentence over $\mathbf{P}$ (containing some binary relation symbol), then consider $\varphi(\prec, x) \equiv (\neg\alpha \wedge \forall y \neg y \prec x) \vee (\alpha \wedge x = x)$. The expression $\varphi(\prec, x)$ mod $\forall x\ Sx$ is in LIMP iff α is true on every finite $\mathbf{P}$-structure. Furthermore, there is a first-order $\mathbf{P} \cup \{<\}$-sentence φ such that the decision problem whether φ is order invariant on a given $\mathbf{P}$-structure $\mathbf{I}$ is co-NP complete [Gur88].

We next consider additional restrictions of SO, inspired by complexity theory, and more precisely by the *unambiguous* computations introduced by Valiant [Val76]. The corresponding logic $\mathrm{U}\Sigma_1^1$, a subclass of Σ_1^1, was first introduced by Kolaitis [Kol90].

Definition 5.4

- $\mathrm{U}\Sigma_1^1$ is the subset of Σ_1^1 consisting of formulae of the form $\exists U \varphi(U, \bar{x})$, over $\mathbf{P}$ where $\varphi(U, \bar{x})$ in FO and such that for each $\mathbf{P}$-structure $\mathbf{I}$, $\mathbf{I} \models \forall \bar{x}[\exists U \varphi(U, \bar{x}) \rightarrow \exists! U \varphi(U, \bar{x})]$. [3]
- $\mathrm{U}\Pi_1^1$ is the subset of Π_1^1 consisting of formulae of the form $\forall U \varphi(U, \bar{x})$ over $\mathbf{P}$, such that $\exists U \neg\varphi(U, \bar{x})$ is in $\mathrm{U}\Sigma_1^1$.
- $\mathrm{U}\Delta_1^1$ is the set of formulae of the form $\exists U \varphi(U, \bar{x}) \in \mathrm{U}\Sigma_1^1$ which are equivalent to a

[3] $\exists! U$ means "there is exactly one relation U".

formula in $\mathrm{U}\Pi^1_1$.

$\mathrm{U}\Sigma^1_1$ is a co-recursively enumerable set. The condition $\forall\bar{x}[\exists S\varphi(S,\bar{x}) \rightarrow \exists! S\varphi(S,\bar{x})]$ is expressible in Π^1_1 : $\forall\bar{x}\forall S\forall T((\varphi(S,\bar{x}) \wedge \varphi(T,\bar{x})) \rightarrow \forall\bar{y}(S\bar{y} \leftrightarrow T\bar{y}))$. In other words, a sentence $\exists S\varphi(S,\bar{x}) \notin \mathrm{U}\Sigma^1_1$ iff $\exists\bar{x}\exists S\exists T\varphi(S,\bar{x})\wedge\varphi(T,\bar{x})\wedge\exists\bar{y}(S\bar{y}\wedge\neg T\bar{y})$ is finitely satisfiable. We now prove a proposition in the spirit of Proposition 5.2 for IMP whose statement first appeared in [Kol90].

Proposition 5.3
IMP $= \mathrm{U}\Delta^1_1$

Proof : Let $\exists U\ \Psi(U)\wedge\varphi(U,\bar{x})$ be in IMP. $\exists U\ \Psi(U)\wedge\varphi(U,\bar{x})$ is obviously in $\mathrm{U}\Sigma^1_1$. Since $\exists U\ \Psi(U)$ is a valid sentence, $\forall U\ \Psi(U) \rightarrow \varphi(U,\bar{x})$ is equivalent to $\exists U\ \Psi(U) \wedge \varphi(U,\bar{x})$. Moreover, since $\exists! U\Psi(U)$ is valid, $\exists U\ \Psi(U) \wedge \neg\varphi(U,\bar{x})$ is in $\mathrm{U}\Sigma^1_1$ and it follows that $\forall U\ \Psi(U) \rightarrow \varphi(U,\bar{x})$ is in $\mathrm{U}\Pi^1_1$.

Conversely, let $\exists U\ \varphi(U,\bar{x})$ be in $\mathrm{U}\Delta^1_1$ and let $\forall T\ \neg\psi(T,\bar{x})$ be an equivalent expression in $\mathrm{U}\Pi^1_1$. We can assume without loss of generality that U and T are of the same arity. Then $\forall\bar{x}\exists U\ (\varphi(U,\bar{x}) \vee \psi(U,\bar{x}))$ is valid. A Skolemization leads to the valid sentence $\exists S\forall\bar{x}\ (\varphi'(S,\bar{x}) \vee \psi'(S,\bar{x}))$. Let $\Psi(S) \equiv \forall\bar{x}\ (\varphi'(S,\bar{x}) \vee \psi'(S,\bar{x}))$. Then $\exists S\ \Psi(S) \wedge \varphi'(S,\bar{x})$ is an equivalent definition in IMP. □

On the class of finite ordered structures $\mathcal{O}$, UP is characterized by the logic $\mathrm{U}\Sigma^1_1$. Notice that a built-in order is required to get the correspondence, because it is not possible to postulate a unique order in general.

6 Non-deterministic languages

A language $\mathcal{L}$ admits a *non-deterministic* semantics if for some formula $\varphi(\bar{x}) \in \mathcal{L}$ and some structure $\mathbf{I}$, φ defines several distinct relations on $\mathbf{I}$. Non-deterministic languages can be defined with a *choice construct*, which chooses among tuples or relations. We study languages which are extensions of first-order logic with a *choice construct*, and consider two such constructs: W_1 on tuples and W_2 on relations. We then investigate the expressive power of FO extended with these first and second-order constructs. In the following, $\mathcal{L} + W$ denotes a non-deterministic language with the construct W. Let $\varphi(\bar{x})$ be a formula with k free variables, defining a k-ary non-deterministic query Q, and $\mathbf{I}$ an instance. As in Section 3, we denote the set of all possible answers of Q on $\mathbf{I}$ by Effect(Q,$\mathbf{I}$). If Q is defined by φ, then on each instance $\mathbf{I}$, Effect(φ,$\mathbf{I}$) = Effect(Q,$\mathbf{I}$). In the following we use both notations. The multi-effect of a query defined by a formula φ, Multi-Effect(φ,$\mathbf{I}$) is the multiset of all possible answers R with as many occurrences of R as there are distinct sequences of choices which lead to R (see Example 6.2).

Definition 6.1
Let Q be a non-deterministic k-ary query defined by φ and $\mathbf{I}$ an instance. The *multi-effect* of a query Q, defined by φ on an instance $\mathbf{I}$ is the multiset Multi-Effect(φ,$\mathbf{I}$), constituted by all the outputs such that each output is repeated in Multi-Effect(φ,$\mathbf{I}$) as many times

as φ has distinct sequences of choices which lead to this output on $\mathbf{I}$.

We first consider the extension of first-order logic with a first-order choice construct, W_1. It was introduced by Abiteboul, Simon and Vianu in [ASV90] as the *witness* operator. The non-deterministic semantics is defined as follows. $W_1\bar{x}\varphi(\bar{x})$ indicates that *one* witness $\bar{x}$ which satisfies $\varphi(\bar{x})$ is chosen. More generally, $W_1\bar{x}\varphi(\bar{x},\bar{y})$ indicates that *one* witness $\bar{x}_{\bar{y}}$ is chosen for each $\bar{y}$ satisfying $\exists\bar{x}\varphi(\bar{x},\bar{y})$. In the context of databases, the result of $W_1\bar{x}\varphi(\bar{x},\bar{y})$ is any maximal subset of $\varphi(\bar{x},\bar{y})$ which satisfies the functional dependency $\bar{y}\rightarrow\bar{x}$ (for a presentation of functional dependency, see [AHV94]).

Example 6.1
Let P be a binary relation and $\mathbf{I}$ a structure with $\mathbf{I}[P]=\{[0,1],[2,1],[2,3]\}$.

$$\text{Effect}(W_1yPxy,\mathbf{I}) = \{\{[0,1],[2,1]\},\{[0,1],[2,3]\}\}$$

To be able to keep track of a given choice, affectation rules were introduced in [ASV90]. They are fundamental to get a satisfying expressive power. The syntax of FO+W_1 is defined as follows.

Definition 6.2
Let $\mathbf{P}$ be a vocabulary, and $\mathcal{S}=\{S_1,S_2,\ldots,S_n,\ldots\}$ be an infinite set of relation variables of fixed arity such that $\mathbf{P}\cap\mathcal{S}=\emptyset$. The set of W_1-formulae is the logic closed under first-order constructs and the W_1 operator applied to first-order variables. A W_1-rule is an expression $S\bar{x} := \varphi(\bar{x})$ where $\varphi(\bar{x})$ is a W_1-formula and $S\in\mathcal{S}$. FO+W_1 is the set of conjunctions of W_1-rules.

The effect and multi-effect of a query are illustrated in the next example.

Example 6.2
Let $\mathbf{P}$=$\{P_1,P_2\}$ where P_1 and P_2 are two unary predicates. Let $\mathbf{I}$ be an instance such that $\mathbf{I}[P_1]$=$\{a,b,c\}$ and $\mathbf{I}[P_2]$=$\{a,d\}$. Let Q be the query defined as follows:

$$\begin{cases}\text{INPUT} & \text{: an instance } \mathbf{I}\\ \text{OUTPUT} & \text{: one element in } \mathbf{I}[P_1] \text{ or in } \mathbf{I}[P_2]\end{cases}$$

A definition of Q in FO+W_1 is $\varphi(x)\equiv W_1x\ P_1(x)\vee P_2(x)$. On the instance $\mathbf{I}$, Effect(φ,$\mathbf{I}$) = $\{\{a\},\{b\},\{c\},\{d\}\}$. The possible answer $\{a\}$ may be obtained by two different ways: first by choosing a in $\mathbf{I}[P_1]$, second by choosing a in $\mathbf{I}[P_2]$. Thus Multi-Effect(φ,$\mathbf{I}$) = $\{\!\{\{a\},\{a\},\{b\},\{c\},\{d\}\}\!\}$.

The notion of satisfaction extends the classical first-order satisfaction, with the interpretation of terms with the operator W_1. The satisfaction can be easily defined modulo a given sequence of choices to interpret the W_1's. The satisfaction, $\models_{W_1}$, is recursively defined as follows. On an instance $\mathbf{I}$, a W_1-formula $W_1x\varphi(x,y)$ is interpreted by a sub-relation R of $\{[a,b]\mid\mathbf{I}\models_{W_1}\varphi(a,b)\}$ such that for all a, a' and b, if $[a,b]$ and $[a',b]$ are both in R then $a=a'$.

Note that in general, $W_1xW_1y\varphi(x,y)$, $W_1yW_1x\varphi(x,y)$, and $W_1xy\varphi(x,y)$ are not equivalent, as shown by the following example.

Example 6.3
Let P be a binary relation and $\mathbf{I}$ a structure with $P^I = \{[0,1],[2,1],[2,3]\}$.
Effect$(W_1 yPxy, \mathbf{I}) = \{\{[0,1],[2,1]\},\{[0,1],[2,3]\}\}$,
Effect$(W_1 xW_1 yPxy, \mathbf{I}) = \{\{[0,1]\},\{[2,1]\},\{[0,1],[2,3]\}\}$,
Effect$(W_1 xPxy, \mathbf{I}) = \{\{[0,1],[2,3]\},\{[2,1],[2,3]\}\}$,
Effect$(W_1 yW_1 xPxy, \mathbf{I}) = \{\{[0,1],[2,3]\},\{[2,1]\},\{[2,3]\}\}$,
Effect$(W_1 xyPxy, \mathbf{I}) = \{\{[0,1]\},\{[2,1]\},\{[2,3]\}\}$.

The relations defined by the choice of tuples are subsets of explicit (first-order definable) relations. It follows that these relations are computable in polynomial time in the size of the instance. For instance, let $W_1\bar{y}\varphi(\bar{x},\bar{y})$ be a formula in FO+W_1. To evaluate $W_1\bar{y}\varphi(\bar{x},\bar{y})$ in polynomial time, it suffices for instance to first compute all the tuples $(\bar{a},\bar{b})$ which satisfy φ, and second, for each $\bar{a}$ such that $(\bar{a},\bar{b})$ is computed, to select $(\bar{a},\bar{b})$ and erase all tuples where $\bar{a}$ appears at the first position. It follows in this case that the chosen tuples $(\bar{a},\bar{b})$ are the first tuples (with respect to the order given by the encoding) with $\bar{a}$ at the first position in the enumeration of tuples satisfying φ.

The choice construct W_1 enables the definition of an arbitrary k-ary relation (the expressions 7 and 8 in Example 6.4 below define any subset T of $P \times P$). The affectation rule keeps track of the chosen relation. It is then possible to test if the chosen relation satisfies a given property (the expression 9 tests if T is a linear order in Example 6.4) and defines the output with respect to the relation which satisfies this property. When the chosen relation T does not satisfy the property, the output of the query is the empty set.

Example 6.4
Let P be a unary relation symbol. The following program defines an order over the domain or the empty set.

$$\begin{aligned}
S(x,y,z) &:= W_1 z \; Px \wedge Py \wedge x \neq y \wedge (z=0 \vee z=1) && (7)\\
T(x,y) &:= S(x,y,1) && (8)\\
linear &:= \forall x \forall y \; (T(x,y) \vee T(y,x)) \wedge \forall x \forall y \forall z \; (T(x,y) \wedge T(y,z) \rightarrow T(x,z)) \\
&\quad\; \wedge \forall x \forall y \; (\neg T(x,y) \vee \neg T(y,x)) && (9)\\
x < y &:= linear \wedge T(x,y) && (10)
\end{aligned}$$

Rule (7) defines a partition S of $P \times P$, then rule (8) defines T as a subset of $P \times P$. Rule (9) verifies that T is a total linear order on P. Finally rule (10) defines $<$ as a linear order when T is a linear order, and the empty set, otherwise.

The choice construct W_1 does not seem to suffice to express all relations which satisfy a given property (such as linear orders) and only these relations. In Example 6.4, the query define both order relations and the empty relation. We conjecture that the ORDER query, introduced in Example 3.2, is not definable in FO+W_1. Nevertheless some interesting queries such as the EVENNESS query, also defined in Section 3, is defined with a W_1-formula under the deterministic semantics *Possible* (see Example 6.6).

We next consider first-order logic with the second-order choice construct, W_2, for relations. The non-deterministic semantics of $W_2 S \; \Psi(S)$ consists of any relation S which satisfies the sentence $\Psi(S)$. In contrast to the first-order choice construct, the use of W_2 enables to choose among relations which satisfy a given property.

Definition 6.3
Let $\mathbf{P}$ be a vocabulary, and $\mathcal{S} = \{S_1, S_2, \ldots, S_n, \ldots\}$ be an infinite set of relation variables of fixed arity such that $\mathbf{P} \cap \mathcal{S} = \emptyset$. The set of FO+$W_2$ formulae is the logic closed under first-order constructs and W_2 operators applied on relational variables. A W_2-rule is an expression $S(\bar{x}) := \varphi(\bar{x})$ where $\varphi(\bar{x})$ is a W_2-formula and $S \in \mathcal{S}$. FO+W_2 is the set of programs, that is conjunctions of first-order W_2-rules satisfying the following criterion: for each expression $W_2 S \Psi(S)$, and each finite $\mathbf{P}$-structure $\mathbf{I}$ there exists at least one relation R^I such that $\mathbf{I} \models \Psi(R^I)$.

Note that at least one choice of relation must be possible. Not all first-order formulae $\Psi(S)$ can therefore be used in an W_2-formula. $W_2 S \Psi(S)$ is a formula of FO+W_2 iff the second-order formula $\exists S \Psi(S)$ is finitely satisfiable. This strong condition ensures that FO+W_2 is a class of total non-deterministic queries.

In contrast to the first-order choice construct, W_1, which chooses tuples among explicit relations, the second-order choice construct, W_2, is related to the notion of implicit definition. In $W_2 S \Psi(S)$, the choice consists of a relation which satisfy a certain property expressed by the first-order sentence Ψ. Their computation might be of exponential complexity. FO+W_2 queries such as "choose a non-empty subset of even cardinality of the set P" can be defined as shown in the following example.

Example 6.5
Let P be a unary relation symbol. The following program defines a subset of P of even cardinality.

$$\begin{aligned}
S \quad &:= \quad W_2 S\ [\forall x \forall y\ (S(x,y) \rightarrow (P(x) \wedge P(y) \wedge x \neq y)) && (11)\\
&\wedge\ \forall x \forall y \forall z\ (S(x,y) \wedge S(x,z) \rightarrow y = z) && (12)\\
&\wedge\ \forall x \forall y \forall z\ (S(x,y) \wedge S(z,y) \rightarrow x = z)] && (13)\\
&\wedge\ \forall x \forall y \forall z\ S(x,y) \rightarrow (\neg S(z,x) \wedge \neg S(y,z))] && (14)\\
U(x) \quad &:= \quad \exists y (Sxy \vee Syx) && (15)
\end{aligned}$$

Expression (11) defines S as a subset of $P \times P$ and expressions (12) and (13) establish that S is a one-one function between its two disjoint (14) components. Then U is the unions of the two components of S (15).

Programs of FO+W_2 admit a normal form with a unique occurrence of the W_2 operator, as follows:

$$\begin{cases} S := W_2 S \Psi(S) \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

In suffices to encode all choices of relations in a single one. In Example 6.5, two relations are successively chosen: a unary relation S and a binary relation T. They can be replaced by a Cartesian product of S and T.

On a structure $\mathbf{I}$, the non-deterministic semantics of such a definition is then defined as follows:

1. choice of a relation R^I such that $\mathbf{I} \models \Psi(R^I)$,
2. affectation of R^I to S (note that from now on, S is known and fixed),
3. $Q(\mathbf{I})=\{\bar{a} \mid \mathbf{I} \models \varphi(S, \bar{a})\}$.

It is easy to see that any definition of FO+W_1 is equivalent to a definition in FO+W_2. A W_1-formula of the form $W_1\bar{x}\varphi(\bar{x}, \bar{y})$ is equivalent to the W_2-rule: $S := W_2S[\forall\bar{x}\forall\bar{y}(S\bar{x}\bar{y} \rightarrow \varphi(\bar{x}, \bar{y})) \wedge \forall\bar{x}\forall\bar{y}\forall\bar{z}((S\bar{x}\bar{y} \wedge S\bar{z}\bar{y}) \rightarrow \bar{x} = \bar{z})]$. We conjecture that the inclusion is strict as we will see in the sequel.

We are now interested in the impact of deterministic semantics on non-deterministic languages. FO+W_1 with deterministic semantics expresses the same queries as fragments of second-order logic. The EVENNESS query, defined in Example 3.4, is defined by a W_1-formula with the deterministic semantics *Possible* as described in the next example.

Example 6.6
Let P be a unary relation symbol. Let Q be the boolean query defined by:

$$\begin{aligned} S(x,z) &:= W_1 z\ Px \wedge (z = 0 \vee z = 1) && (16)\\ T(x,y) &:= W_1 x[W_1 y(S(x,1) \wedge S(y,0)] && (17)\\ Q &:= \forall x \exists y\ (T(x,y) \vee T(y,x)) && (18) \end{aligned}$$

Expression (16) affects to S a partition of P. T is a partial one-one mapping between the two components of P defined by S (17). Finally, the output of the query is TRUE if T is a total one-one function, and FALSE, otherwise. Recall that TRUE is interpreted by $\{[]\}$ and FALSE by $\emptyset$. It follows that *Possible*$(Q)=\{[]\}$ if P has an even cardinality, and *Possible*$(Q)=\emptyset$, otherwise. EVENNESS=*Possible*(Q).

In [ASV90] it is shown that *Possible*(FO+W_1) coincides with existential second-order logic, Σ^1_1, and *Certain*(FO+W_1) coincides with universal second-order logic, Π^1_1.

Theorem 6.1

1. *Possible*(FO+W_1) $= \Sigma^1_1$
2. *Certain*(FO+W_1) $= \Pi^1_1$

It follows from the previous theorem, that *Possible-is-Certain*(FO+W_1) is included in Δ^1_1. In Proposition 3.2, we proved that *Possible-is-Certain*(FO+W_1) = funct-(FO+W_1). It follows that a deterministic query defined in FO+W_1 is such that **all** interpretations of all W_1-formulae lead to **the same** output. A lower complexity bound can thus be obtained. funct-(FO+W_1) is included in $\mathcal{Q}$LOGSPACE. The proof is standard for FO [AHV94]. The W_1 expressions can be systematically computed by taking the least witness for the order over the domain deduced from the input-tape. This is easily done in LOGSPACE. Over the class of ordered inputs, funct-(FO+W_1) collapses to FO. Indeed, the least witness can be expressed by a first-order formula. Abiteboul, Simon and Vianu conjectured that funct-(FO+W_1) = FO in the general case. On the other hand, they proved that the functional fragment of the logic extended with an inflationary fixpoint operator, funct-(FO+IFP+W_1), coincides with the queries in $\mathcal{Q}$P [ASV90]. The fixpoint combined with a choice operator allows the definition of an order relation over the domain. The rest of the proof is standard [Var82, Imm86].

The semantics *Disjoint*, *Co-Disjoint* and *Unique* on FO+W_1 also capture fragments of second-order logic.

Theorem 6.2

1. *Disjoint*(FO+W_1) = UΣ^1_1
2. *Co-Disjoint*(FO+W_1) = UΠ^1_1

Proof : 1. *Disjoint*(FO+W_1) = UΣ^1_1. Let $\Psi(\bar{x})$ be a formula in FO+W_1 defining a query Q, and let Q^* be the corresponding query defined with the *Disjoint* semantics. $Q^*(\mathbf{I})$=*Disjoint*$(Q,\mathbf{I}) = \bigcup\{R|R \in \text{Effect}(\Psi, I)\}$ if for all $R, R' \in$ Multi-Effect$(\Psi,\mathbf{I})$, $R \cap R' = \emptyset$, and is undefined otherwise. By definition, if $\bar{a} \in$ *Disjoint*$(Q,\mathbf{I})$, there exists a relation R such that $R \in$ Effect$(\Psi,\mathbf{I})$ and $\bar{a} \in R$. $\Psi(\bar{x})$ is a conjunctions of W_1-rules, that is $\Psi(\bar{x}) \equiv \bigwedge_{i=1}^{i=n} S_i(\bar{x}) := \varphi_i(\bar{x})$ where S_n is $Q(\mathbf{I})$. Consider the following translation of $\Psi(\bar{x})$:

1. Replace recursively each expression $W_1\bar{u}\ \theta(\bar{u},\bar{v})$ in $\varphi_i(\bar{x})$, where W_1 does not appear in θ, by some new predicate symbol $S_\theta(\bar{u},\bar{v})$ where $S_\theta \in \mathcal{S} - \{S_1, ..., S_n\}$.
2. Let $\phi_i(\bar{x})$ be the resulting formula.

Then let $S_1, \ldots, S_n$ be symbols in $\mathcal{S}$, and $\Omega(\bar{x})$ be the following formula:

$$\begin{aligned}\Omega(\bar{x}) \quad \equiv \quad & \phi_n(\bar{x}) \wedge \bigwedge_{i=1}^{i=n-1} \exists S_i\ [\forall \bar{x}(S_i(\bar{x}) \leftrightarrow \phi_i(\bar{x})) \\ & \wedge \bigwedge_{\theta \in \Theta} \exists S_\theta \forall \bar{u} \forall \bar{v}(S_\theta(\bar{u},\bar{v}) \rightarrow \theta(\bar{u},\bar{v})) \wedge \forall \bar{u} \forall \bar{v} \forall \bar{w}((S_\theta(\bar{u},\bar{v}) \wedge S_\theta(\bar{w},\bar{v})) \rightarrow \bar{u} = \bar{w})]\end{aligned}$$

$\Omega(\bar{x})$ is a definition of Q^* in UΣ^1_1. Ω is obviously in Σ^1_1. The reason why Ω is in UΣ^1_1 is the following. The S_θ corresponds to the relation constituted by the non-deterministic choices of tuples $\bar{u}$ such that $\theta(\bar{u},\bar{v})$. In a sense, the resulting choice of tuples such that $W_1\bar{u}\ \theta(\bar{u},\bar{v})$ is affected to S_θ. To each sequence of choices in φ corresponds an effect, that is a relation in Multi-Effect(φ,$\mathbf{I}$). Since *Disjoint* is defined on φ and $\mathbf{I}$, the relations in Multi-Effect(φ,$\mathbf{I}$) are pairwise disjoint. If there is an $\bar{a} \in \mathbf{I}^k$ such that there are two different sequences of relations $(S_i^I)_{i \leq n-1}, (S_\theta^I)_{\theta\in\Theta}$ and $({S'}_i^I)_{i \leq n-1}, ({S'}_\theta^I)_{\theta\in\Theta}$ such that $\mathbf{I} \models \Omega(\bar{a})$, then there are two different relations R and R' in Multi-Effect(φ,$\mathbf{I}$) such that $\bar{a} \in R$ and $\bar{a} \in R'$, which is a contradiction. Thus,

$$\mathbf{I} \quad \models \quad \forall \bar{x}[\Omega(\bar{x}) \rightarrow$$

$$\phi_n(\bar{x}) \wedge \bigwedge_{i=1}^{i=n-1} \exists! S_i \; [\forall \bar{x}(S_i(\bar{x}) \leftrightarrow \phi_i(\bar{x}))$$
$$\wedge \bigwedge_{\theta} \exists! S_\theta \forall \bar{u} \forall \bar{v}(S_\theta(\bar{u},\bar{v}) \rightarrow \theta(\bar{u},\bar{v})) \wedge \forall \bar{u} \forall \bar{v} \forall \bar{w}((S_\theta(\bar{u},\bar{v}) \wedge S_\theta(\bar{w},\bar{v})) \rightarrow \bar{u} = \bar{w})]$$

Conversely, let $\exists S \; \varphi(S,\bar{x})$ be a definition in $\mathrm{U}\Sigma_1^1$. Then the formula $\psi(\bar{x})$

$$\begin{cases} T(\bar{y},z) & := & W_1 z \; (\bar{y} = \bar{y} \wedge (z = 0 \vee z = 1)) \\ S(\bar{y}) & := & T(\bar{y},0) \\ U(\bar{x}) & := & \varphi(S,\bar{x}) \end{cases}$$

is a definition in FO + W_1 such that for each structure $\mathbf{I}$, *Disjoint*$(Q,\mathbf{I})=\{\bar{a} \mid \mathbf{I} \models \exists S \; \varphi(S,\bar{a})\}$.

2. *Co-Disjoint*(FO+W_1) $= \mathrm{U}\Pi_1^1$. Let $\varphi(\bar{x})$ be a definition of a query Q in FO+W_1. The semantics *Co-Disjoint* is defined on Q by *Co-Disjoint*$(Q,\mathbf{I})=\bigcap\{R \mid R \in$ *Effect*$(\varphi,\mathbf{I})\}$ if for all $R, R' \in$ *Multi-Effect*(φ, I), $C(R) \cap C(R') = \emptyset$, undefined otherwise. By definition, $\bar{a} \in$ *Co-Disjoint*$(Q,\mathbf{I})$ iff $\bar{a} \notin$ *Disjoint*$(\neg Q,\mathbf{I})$. Let $\exists S \; \phi(S,\bar{x})$ be a definition in $\mathrm{U}\Sigma_1^1$ such that *Disjoint*$(\neg Q,\mathbf{I}) = \{\bar{a} \mid \mathbf{I} \models \exists S \; \phi(S,\bar{a})\}$. Then, *Co-Disjoint*$(Q,\mathbf{I})$ is equivalent to $\forall S \; \neg\phi(S,\bar{x})$ which is in $\mathrm{U}\Pi_1^1$, since $\exists S \; \phi(S,\bar{x})$ is in $\mathrm{U}\Sigma_1^1$.

Conversely, let $\forall S \; \phi(S,\bar{x})$ be a formula in $\mathrm{U}\Pi_1^1$ defining the query Q. Then the formula $\exists S \; \neg\phi(S,\bar{x})$ is in $\mathrm{U}\Sigma_1^1$. It follows from the previous result that there is a query Q^* defined by $\varphi(\bar{x})$ in FO + W_1 such that $\neg Q=$ *Disjoint*$(Q^*,\mathbf{I})$. Since for each tuple $\bar{a}$, $\bar{a} \in$ *Co-Disjoint*$(Q^*,\mathbf{I})$ iff $\bar{a} \notin$ *Disjoint*$(\neg Q^*,\mathbf{I})$, $Q(\mathbf{I}) =$ *Co-Disjoint*$(\neg Q^*,\mathbf{I})$. □

We next consider deterministic semantics over FO+W_2. Again, similar fragments of second-order logic can be captured. The proofs of Theorem 6.3 and Theorem 6.4 give a precise correspondence between the deterministic semantics defined in Section 3 and the usual deterministic semantics recalled in Section 5. The following table gives an intuition of these correspondences. For each formula in FO+W_2 on which the semantics is defined, it gives the translation and the resulting deterministic logic.

deterministic semantics	translation	deterministic logic
Possible	replace $W_2 S$ by $\exists S$	Σ_1^1
Certain	replace $W_2 S$ by $\forall S$	Π_1^1
Possible-is-Certain	replace $W_2 S$ by $\exists S$ or by $\forall S$	Δ_1^1

The idea is the following. The non-deterministic semantics of a definition in FO+W_2 in normal form, consists in first choosing a relation R^I which satisfies $\Psi(R^I)$ and affecting it and then verifying a first-order formula $\varphi(R^I,\bar{x})$. The *Possible* semantics applied on this definition consists in considering all $\bar{a}$ which satisfies $\varphi(S,\bar{a})$ for at least one relation which satisfies $\Psi(S)$. That is the tuples $\bar{a}$ no matter what the chosen relation is. In other words, the tuples for which there exists a relation R^I such that $\Psi(R^I)$ and $\varphi(R^I,\bar{x})$.

Theorem 6.3

1. *Possible*(FO+W_2) $= \Sigma_1^1$

2. $Certain(\text{FO}+W_2) = \Pi^1_1$

3. $Possible\text{-}is\text{-}Certain(\text{FO}+W_2) = \Delta^1_1 = \text{GIMP}$

Proof : 1. $Possible(\text{FO}+W_2) = \Sigma^1_1$ follows from the fact that the *Possible* semantics on FO+W_2 can be expressed with the following syntactic translation: replace each definition

$$\begin{cases} S := W_2 S\ \Psi(S) \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

by $\exists S\ \Psi(S) \wedge \varphi(S, \bar{x})$, which is a definition in Σ^1_1. Conversely, let $\exists S\ \varphi(S, \bar{x})$ be a definition in Σ^1_1, then it has the same output as the *Possible* semantics on

$$\begin{cases} S := W_2 S\ \forall \bar{x} S\bar{x} \leftrightarrow S\bar{x} \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

($\forall \bar{x} S\bar{x} \rightarrow S\bar{x}$ is a tautology, that is any set S may be chosen).

2. $Certain(\text{FO}+W_2)=\Pi^1_1$. A query Q is defined in $Certain(\text{FO}+W_2)$ iff its complement $\neg Q$ is defined in $Possible(\text{FO}+W_2)$. Then if

$$\begin{cases} S := W_2 S\ \Psi(S) \\ Q^*(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

is a definition of Q^* such that $Q = Certain(Q^*)$, then

$$\begin{cases} S := W_2 S\ \Psi(S) \\ \neg Q^*(\bar{x}) := \neg\varphi(S, \bar{x}) \end{cases}$$

is a definition of $\neg Q^*$ such that $\neg Q = Possible(\neg Q^*)$. Therefore $\exists S\ \Psi(S) \wedge \neg\varphi(S, \bar{x})$ is a definition of $\neg Q$. It follows that $\forall S\ \Psi(S) \rightarrow \varphi(S, \bar{x})$ is a definition of Q.

Conversely, let Q be defined by $\forall S\ \varphi(S, \bar{x})$, then Q^* defined by

$$\begin{cases} S := W_2 S\ \forall \bar{x} S\bar{x} \leftrightarrow S\bar{x} \\ Q^*(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

is such that $Q = Certain(Q^*)$.

3. $Possible\text{-}is\text{-}Certain(\text{FO}+W_2) = \Delta^1_1$ =GIMP. The translation is the following: to any

$$\begin{cases} S := W_2 S\ \Psi(S) \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

definition in FO+W_2, the *Possible-is-Certain* semantics coincides with $\varphi(S, \bar{x})\ mod\ \Psi(S)$ in GIMP. □

We can now consider the relationships between FO+W_1 and FO+W_2 in more details. It is clear that FO+W_1 = FO+W_2 implies funct-FO+W_1 = funct-FO+W_2. We have seen that funct-FO+$W_1 \subseteq \mathcal{Q}$LOGSPACE and funct-FO+$W_2$ = $\mathcal{Q}$NP$\cap$co-NP. So, by Theorem 4.2, FO+W_1 = FO+W_2 implies LOGSPACE = NP$\cap$co-NP.

Remark that Theorem 6.3 gives a precise characterization of the functional fragment of FO+W_2 by the implicit logic GIMP. These results show the relationships between determinism and implicitness of queries. The subset of the functional fragment of FO+W_2 defined by the *Unique* semantics is also characterized by an implicit language as it is shown in the following.

Theorem 6.4

1. *Disjoint*(FO+W_2) = $\mathrm{U}\Sigma_1^1$
2. *Co-Disjoint*(FO+W_2) = $\mathrm{U}\Pi_1^1$
3. *Unique*(FO+W_2) = $\mathrm{U}\Delta_1^1$ = IMP

Proof : The proof of Theorem 6.4 is a consequence of the translations given in the proof of Theorem 6.3. It suffices to verify the restricted conditions.

1. *Disjoint*(FO+W_2) = $\mathrm{U}\Sigma_1^1$. Let $\varphi(\bar{x})$ be a definition of Q in FO+W_2. The semantics *Disjoint* is defined on $\varphi(\bar{x})$ by $Disjoint(\varphi, I) = \bigcup\{R | R \in \text{ Effect}(\varphi, I)\}$ if for all $R, R' \in$ Multi-Effect(φ, I), $R \cap R' = \emptyset$, undefined otherwise. Let $\varphi(\bar{x})$ be

$$\begin{cases} S := W_2 S \ \Psi(S) \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

We verify that $\exists S \ \Psi(S) \wedge \varphi(S, \bar{x})$ is in $\mathrm{U}\Sigma_1^1$. Indeed, if it was not in $\mathrm{U}\Sigma_1^1$, then there would exist a structure **I** and a tuple $\bar{a}$ such that there exist two different relations R and R' such that $I \models \Psi(R) \wedge \Psi(R')$ and $I \models \varphi(R, \bar{a}) \wedge \varphi(R', \bar{a})$ which contradicts the definition of the *Disjoint* semantics on Q defined by φ.

Conversely, let $\exists S \ \varphi(S, \bar{x})$ be a definition of Q in $\mathrm{U}\Sigma_1^1$. Then let $\varphi(\bar{x})$ be the following:

$$\begin{cases} S := W_2 S \ \forall \bar{x} S\bar{x} \leftrightarrow S\bar{x} \\ Q(\bar{x}) := \varphi(S, \bar{x}) \end{cases}$$

is a definition in FO + W_2 on which the *Disjoint* semantics is defined and $Disjoint(Q, \mathbf{I}) = \{\bar{a} \mid \mathbf{I} \models \exists S \ \Psi(S) \wedge \varphi(S, \bar{a})\}$.

2. *Co-Disjoint*(FO+W_2)=$\mathrm{U}\Pi_1^1$ results from the fact that for any definition $\varphi(\bar{x})$ of Q in FO + W_2, and each structure **I**, $\bar{a} \in Co\text{-}Disjoint(Q, \mathbf{I})$ iff $\bar{a} \notin Disjoint(\neg Q, I)$.

3. *Unique*(FO+W_2) = $\mathrm{U}\Delta_1^1$ = IMP is a consequence of the two previous results and of Proposition 5.3. □

We are now interested in the correspondences between logics and complexity classes.

Definition 6.4

A query language $\mathcal{L}$ captures a class N$\mathcal{QC}$ if the two following conditions hold (we identify the class $\mathcal{C}$ of problems with the machine which computes them):

1. $(\forall \mathbf{P}) \ \ (\forall \varphi \in \mathcal{L}) \ \ (\exists M \in \mathcal{C}) \ \ (\forall \mathbf{I} \in \text{inst}(\mathbf{P})) \ \ \text{Effect}(M, \mathbf{I}) = \text{Effect}(\varphi, \mathbf{I})$
2. $(\forall \mathbf{P}) \ \ (\forall M \in \mathcal{C}) \ \ (\exists \varphi \in \mathcal{L}) \ \ (\forall \mathbf{I} \in \text{inst}(\mathbf{P})) \ \ \text{Effect}(M, \mathbf{I}) = \text{Effect}(\varphi, \mathbf{I})$

It is easy to verify that the fixpoint query language FO+IFP+W_1 captures the class N$\mathcal{Q}$NP^{TA}, of total accepting queries of N$\mathcal{Q}$NP. A query is *total accepting* if on each input, **all** computations lead to an accepting state. The inclusion FO+IFP+W_1 $\subseteq$ N$\mathcal{Q}$NP^{TA} is clear. Indeed, queries in FO+IFP+W_1 are computable in polynomial time on non-deterministic TM's. They are total, and each choice for a W_1 expression leads to an output. The machine computing such a query can be designed such that the non-determinism in the transitions exactly reflects the non-determinism of the W_1 expressions. Therefore, all branches lead to an accepting state.

Conversely, the computation of a non-deterministic machine computing a query in $\mathrm{N\mathcal{Q}NP}^{TA}$ can be simulated by a program in FO+IFP+W_1 in a classical way. The combined use of the fixpoint and the choice construct permits the definition of an order relation. The main difference with Fagin's proof [Fag74] relies in the fact that FO+IFP+W_1 only expresses total queries, such that all choices lead to an output. The assumption that the initial query is total accepting fits this requirement.

The previous characterization result seems to imply that the class NDB-P introduced by Abiteboul, Simon and Vianu [ASV90], coincides with the class $\mathrm{N\mathcal{Q}NP}^{TA}$, of total accepting queries of N$\mathcal{Q}$NP. Indeed, they claim that FO+IFP+W_1 captures NDB-P.

We next prove that the fixpoint is not needed with the second-order choice to capture $\mathrm{N\mathcal{Q}NP}^{T}$.

Theorem 6.5
$\mathrm{N\mathcal{Q}NP}^{T}$ = FO+W_2.

Proof : Let Q be a total non-deterministic query computable in non-deterministic polynomial time. The non-deterministic Turing machine, M_Q, which computes Q can be simulated in FO+W_2 by a formula of the form

$$\begin{cases} S := W_2 S\ \Psi(S) \\ Q(\bar{x}) := \varphi(S,\bar{x}) \end{cases}$$

where $W_2 S\ \Psi(S)$ only encodes in S the relation simulating the transitions of the machine which lead to an accepting state. $\varphi(S,\bar{x})$ is used to read the output-tape of the machine. We follow the lines of Fagin's proof [Fag74]. There are two main differences. First, only accepting computations are encoded. Second, the machines considered are constructive (computation vs. recognition).

Conversely, let Q be a k-ary query defined by

$$\begin{cases} S := W_2 S\ \Psi(S) \\ Q(\bar{x}) := \varphi(S,\bar{x}) \end{cases}$$

in FO + W_2 where S is of arity m. Let M_Q be the following machine. It first checks if its input is the encoding of an instance. Then makes several non-deterministic transitions in order to choose and write on the working-tape a sequence S of m-ary tuples. Then verifies $\Psi(S)$. If $\Psi(S)$ is satisfied, M_Q enumerates all k-ary tuples $\bar{a}_1,\ldots,\bar{a}_n$, otherwise, it rejects. Successively for each $\bar{a}_i$, M_Q verifies $\varphi(S,\bar{a}_i)$ and writes $\bar{a}_i$ on the output-tape when $\varphi(S,\bar{a}_i)$ is satisfied. Finally M_Q accepts. M_Q is a non-deterministic polynomial-time constructive Turing machine which computes Q. And so, Q is in $\mathrm{N\mathcal{Q}NP}^{T}$. □

Since FO+W_2 coincides with $\mathrm{N\mathcal{Q}NP}^{T}$ (Theorem 6.5), *Possible-is-Certain*(FO+W_2) $= \Delta^1_1 =$ GIMP (Theorem 6.3) is a consequence of *Possible-is-Certain*($\mathrm{N\mathcal{Q}NP}^{T}$) = funct-$\mathrm{N\mathcal{Q}NP}^{T}$ (Proposition 3.2), funct-$\mathrm{N\mathcal{Q}NP}^{T}$ = $\mathcal{Q}$NP∩co-NP (Theorem 4.8), $\mathcal{Q}$NP∩co-NP $= \Delta^1_1$ (Fagin's result [Fag74]), and finally $\Delta^1_1 =$ GIMP (Proposition 5.1 [GLL95]).

Note that the impact of the deterministic semantics defined with the multi-effect of the device is different on the language FO+W_2 and on the complexity based class $\mathrm{N\mathcal{Q}NP}^{T}$. On one side, *Unique*(FO+W_2) = IMP (Theorem 6.4) and on the other side, *Unique*(N$\mathcal{Q}$NP^T) = $\mathcal{Q}$UP$\cap$co-UP (Theorem 4.11). Remark that the deterministic semantics on N$\mathcal{Q}$NP^T also captures an implicit logic since $\mathcal{Q}$UP$\cap$co-UP = LIMP [GLL95]. This dichotomy results from the use of a linear order by the machine which cannot be simulated by a formula of IMP.

7 Conclusion

We have studied different non-deterministic formalisms to express database queries. Most of our results focus on deterministic queries. The numerous equivalences between different formalisms, machine or language based, show the robustness of the classes of queries considered.

Non-deterministic queries can only be computed on non-deterministic computing devices. We focused on non-deterministic Turing machines, with polynomially bounded time complexity, and analyzed the machines and their non-deterministic behavior. Other computing devices, such as RAMs, could have been used as well. It would be very interesting to consider relational machines [AV91a] in this context.

For the languages, we considered four logical formalisms, with a more or less obvious intrinsic non-deterministic flavor. The two *choice logics*, FO+W_1 and FO+W_2, contain an explicit non-deterministic mechanism. In *implicit logic* and its extensions, relations satisfying first-order conditions have to be guessed. Finally, *second-order logic*, and even more its unambiguous versions, suppose again mechanisms to guess relations. In Figure 6, most of the correspondences between these formalisms are presented. Classes of deterministic queries on the same line have the same expressive power over the class of all finite structures.

complexity		SO	implicit logic	FO+W_2
		UΔ^1_1	IMP	Unique(FO+W_2)
Unique(N$\mathcal{Q}$NP^T)	$\mathcal{Q}$UP $\cap$ co-UP		LIMP	
funct-N$\mathcal{Q}$NP^T	$\mathcal{Q}$NP $\cap$ co-NP	Δ^1_1	GIMP	funct-FO+W_2
Disjoint(N$\mathcal{Q}$NP^T)	$\mathcal{Q}$UP			
Co-Disjoint(N$\mathcal{Q}$NP^T)	$\mathcal{Q}$co-UP			
Possible(N$\mathcal{Q}$NP^T)	$\mathcal{Q}$NP	Σ^1_1		Possible(FO+W_2)
Certain(N$\mathcal{Q}$NP^T)	$\mathcal{Q}$co-NP	Π^1_1		Certain(FO+W_2)

Figure 6: Classes of Deterministic Queries

The only classes in Figure 6 which are known to be recursively enumerable are $\mathcal{Q}$NP and $\mathcal{Q}$co-NP. It is open for all the others. Our results provide new insights on some classical questions. It was shown in [GLL95] that the relation between GIMP and LIMP is related to different sorts of choice. These extensions are based on the ability to non-deterministically choose an (intermediate) relation R satisfying a sentence $\Phi(R)$, and use it to (deterministically) compute a query, independent of the particular choice of R.

The first extension GIMP, in which an arbitrary relation is non-deterministically chosen, captures NP $\cap$ co-NP. The second extension LIMP, in which only an order relation on the domain is non-deterministically chosen, captures UP $\cap$ co-UP. The notion of *implicit definition* of a query is related to the nature of the non-determinism of the machine which computes it.

The functional fragments of N$\mathcal{Q}$NP^T and N$\mathcal{Q}$UP^T are respectively captured by GIMP and LIMP. It might seem that the non-deterministic choice of an order relation should be enough to capture the full expressive power of GIMP (since it rigidifies the structure), and that the definitions of LIMP could be seen as a normal form for GIMP queries. Nevertheless, the non-determinism can be reduced to the choice of an order relation if and only if UP $\cap$ co-UP $=$ NP $\cap$ co-NP.

We have also considered the expressive power of non-deterministic query languages. In particular, we have characterized the fundamental classes N$\mathcal{Q}$NP^T and N$\mathcal{Q}$UP^T with logical languages, respectively FO$+W_2$ and LIMP.

complexity classes	logical languages	functional fragments
N$\mathcal{Q}$UP^T	LIMP	$\mathcal{Q}$UP$\cap$co-UP
N$\mathcal{Q}$NP^T	FO$+W_2$	$\mathcal{Q}$NP$\cap$co-NP

Like most of the time in descriptive complexity theory, the order plays a fundamental role. We have seen that the difference between the two implicit languages, LIMP and GIMP, relies on restricting the choice to an order, versus to an arbitrary relation. GIMP corresponds to the functional fragment of N$\mathcal{Q}$NP^T. In this later class, the output of a query is unique, but can be obtained through distinct computations. LIMP corresponds to the *Unique* semantics of N$\mathcal{Q}$NP^T. We have seen that in this later case, the output of a query is not only unique but also has a unique computation. This is not surprising in the light of the difference between LIMP and GIMP. Indeed, the fact that the choice is restricted to an order at the language level, corresponds at the machine level to no choice at all.

The importance of the order is best revealed by the *Unique* semantics. Consider for instance the class N$\mathcal{Q}$NP^T $=$ FO$+W_2$. The effect of the *Unique* semantics is different on the machine based class and on the language based class. This is at first surprising, but it makes sense. On one hand, we have seen that *Unique*(N$\mathcal{Q}$NP^T) $=$ LIMP. As we said, *Unique* at the level of the machine, does not allow any choice, but benefits from the order of the tape. Therefore it corresponds to LIMP, which precisely, has the choice of an order as only choice. On the other hand, we have seen that *Unique*(FO$+W_2$) $=$ IMP. *Unique* at the level of the language, implies the existence of a unique relation satisfying the formula (it can therefore not be in particular an order relation), which is precisely what implicit definitions of IMP are.

Acknowledgments: The authors wish to thank Victor Vianu for fruitful discussions and the anonymous referees for their insightful suggestions.

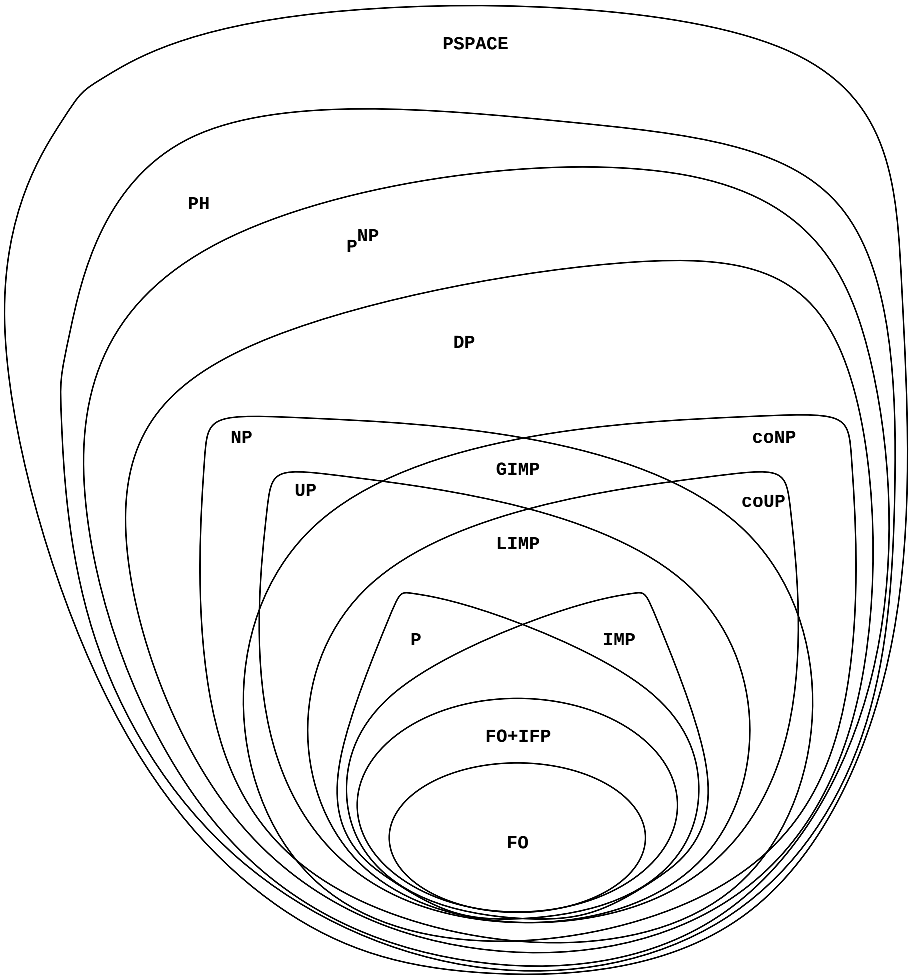

Figure 7: Onion of classes of queries

References

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1994.

[ASV90] S. Abiteboul, E. Simon, and V. Vianu. Non-deterministic languages to express deterministic transformations. In *Proc. 9th ACM Symp. on Principles of Database Systems*, 1990.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and datalog like languages. In *Proc. 4th Symp. on Logic in Computer Science*, pages 71–79, 1989.

[AV91a] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. ACM Symp. on Theory of Computing*, New Orleans, May 1991.

[AV91b] S. Abiteboul and V. Vianu. Non-determinism in logic-based languages. *Annals of Math. and AI*, 3:151–186, 1991.

[Bet53] E. W. Beth. On Padoa's method in the theory of definition. *Indag. Math.*, 15:330–339, 1953.

[BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55:80–88, 1982.

[Boa90] P. Van Emde Boas. *Handbook of Theorical Computer Science*, volume A, chapter Machine models and simulations, pages 1–66. North Holland, J. Van Leeuwen Ed., 1990.

[CH80] A. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21(2):156–178, Oct. 1980.

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

[DHK95] A. Dawar, L. Hella, and P. Kolaitis. Implicit definability and infinitary logic in finite model theory. In *Proc. 22nd International Colloquium on Automata, Languages and Programming - ICALP 95*, 1995.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49, 1961.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computations, SIAM-AMS Proceedings 7*, pages 43–73, 1974.

[Fag76] R. Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, 1976.

[Fag93] R. Fagin. Finite model theory - a personal perspective. *Theoretical Computer Science*, 116:3–31, 1993.

[Fra54] R. Fraïssé. Sur les classifications des systèmes de relations. *Publ. Sci. Univ Alger*, I:1:35–182, 1954.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability A Guide to the theory of NP-Completeness*. Freeman, 1979.

[GLL95] S. Grumbach, Z. Lacroix, and S. Lindell. Implicit definitions on finite structures. In *Computer Science Logic*, Paderborn, 1995.

[GPSZ91] F. Giannotti, D. Pedreschi, D. Saccà, and C. Zaniolo. Nondeterminsm in deductive databases. In *Proc. 2nd DOOD Conference*, pages 129–146, 1991.

[GS85] Y. Gurevich and S. Shelah. Fixed-point extensions of first order logic. In *Proc IEEE Foundations of Computer Science*, 1985.

[GS88] J. Grollmann and S. Selman. Complexity measures for public-key cryptosystems. In *SIAM J. Computing*, volume 17, pages 309–335, 1988.

[GSZ95] S. Greco, D. Saccà, and C. Zaniolo. Datalog queries with stratified negation and choice: from p to dp. In G. Gottlob and M. Vardi, editors, *Proc. Int. Conf. on Database Theory*, pages 82–96, Prague, 1995. Springer Verlag. LNCS 893.

[Gur84] Y. Gurevich. *Computation and Proof Theory (M. Richter et al. Ed.)*, chapter Towards Logic tailored for Computational Complexity, pages 175–216. Lecture Notes in Mathematics, 1104, 1984.

[Gur88] Y. Gurevich. *Current Trends in Theoretical Computer Science, E. Borger Ed.*, chapter

Logic and the Challenge of Computer Science, pages 1–57. Computer Science Press, 1988.

[HH88] J. Hartamis and L.A. Hemachandra. Complexity classes without machines: on complete languages for up. *Theoretical Computer Science*, 58:129–142, 1988.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Inf. and Control*, 68:86–104, 1986.

[Joh90] D.S. Johnson. *Handbook of Theorical Computer Science*, volume A, chapter A catalog of complexity classes, pages 67–162. North Holland, J. Van Leeuwen Ed., 1990.

[Kan90] P. Kanellakis. *Handbook of Theorical Computer Science*, volume B, chapter 17 Elements of Relational Database Theory, pages 1073–1156. North Holland, J. Van Leeuwen Ed., 1990.

[Kol90] P. Kolaitis. Implicit definability on finite structures and unambiguous computations. In *Proc. 5th Symp. of Logic in Computer Science*, 1990.

[KR90] R. Karp and V. Ramachandran. *Handbook of Theorical Computer Science*, volume A, chapter Parallel Algorithms for Shared Memory machines, pages 869–941. North Holland, J. Van Leeuwen Ed., 1990.

[Lac96] Z. Lacroix. *Bases de Données: des Relations Implicites aux Relations Contraintes.* PhD thesis, Université Paris-Sud, 1996.

[PY82] C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *Proc. of the 14th Annual ACM Symposium on Theory of Computing*, San Francisco, 1982.

[PY84] C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28:244–259, 1984.

[Sel94] A.L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48:357–381, 1994.

[SZ90] D. Saccà and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 205–217, 1990.

[Tra50] B. A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *Doklady Akademii Nauk SSSR*, 70:569–572, 1950.

[Val76] L. Valiant. Relative complexity of checking and evaluating. *Information Processing*, 5:20–23, 1976.

[Var82] M. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.