Default Logic as a Query Language

Marco Cadoli, Thomas Eiter, Georg Gottlob

Abstract— **Research in non-monotonic reasoning has focused largely on the idea of representing knowledge about the world via rules that are generally true but can be defeated. Even if relational databases are nowadays the main tool for storing very large sets of data, the approach of using non-monotonic AI formalisms as relational database query languages has been investigated to a much smaller extent. In this work we propose a novel application of Reiter's default logic by introducing a *default query language* (DQL) for finite relational databases, which is based on default rules. The main result of this paper is that DQL is as expressive as $SO_{\exists\forall}$, the existential-universal fragment of second-order logic. This result is not only of theoretical importance: We exhibit queries –which are useful in practice– that can be expressed with DQL and can not with other query languages based on non-monotonic logics such as DATALOG with negation under the stable model semantics. In particular, we show that DQL is well-suited for diagnostic reasoning.**

Keywords— **Non-monotonic reasoning, Default logic, Query languages, Relational databases, Expressive power.**

I. Introduction

FOR the purpose of Knowledge Representation, non-monotonic reasoning (NMR henceforth) formalisms can be used in two different ways:

- as languages for representing knowledge about the world, via rules that are generally true but can be defeated. Retrieving information from a non-monotonic knowledge base of this kind amounts to prove a theorem.
  As an example, we can use default logic to state that "birds generally fly". In order to prove that the bird Tweety flies, we try to prove that a specific formula follows –in the default logic semantics– from the set of general rules plus a set of specific facts;
- as relational database query languages. Retrieving information amounts to computing the set of tuples belonging to an intensional relation, starting from some extensional relations. As an example, we can query a relational database by means of a DATALOG$^\neg$ program –i.e., a DATALOG program with negated literals in the body of the rules– equipped with a specific semantics for negation.

Research in NMR has focused largely on the former idea, and remarkable results about the computational complexity of several formalisms have been obtained by many authors (cf. [1] for a survey on this topic).

Even if relational databases are nowadays the main tool for storing very large sets of data, the latter approach has been investigated to a much smaller extent.

One of the most important aspects of a query language for relational databases is its *expressive power*, i.e., the set of relations that we can compute by querying. The expressive power of relational database query languages has been studied for some twenty years now (cf. [2]). Research has focused mainly on monotonic query languages, i.e., languages such that if the extensional relations grow then the intensional ones grow as well.

Recently some interesting works investigating the expressive power of non-monotonic query languages appeared. Kolaitis and Papadimitriou study in [3] the expressive power of two semantics for DATALOG$^\neg$ programs. In particular they prove that DATALOG$^\neg$ with fixed-point semantics is as expressive as $SO_\exists$, the existential fragment of second-order logic. Schlipf proves in [4] an analogous result for DATALOG$^\neg$ with stable model semantics for logic programs [5] (stable models are called default models in [6], [7]); in the following, we refer to this variant of DATALOG as DATALOG$^\neg_{stable}$. Saccà gives in [8] further insight on the expressive power of interesting variants of DATALOG$^\neg_{stable}$. Van Gelder analyzes in [9] the expressive power of DATALOG$^\neg$ with well-founded semantics [10]. In all these papers, databases are modeled as *finite structures*, i.e., finite interpretations of theories.

In this work we are concerned with *default logic as a query language*. Default logic [11] is one of the most popular NMR formalisms and has been extensively investigated both from the semantical and the computational point of view. It has also been proposed in [7] as a tool for inferencing in logical databases (i.e., databases which are theories). Anyway the behavior of default logic on finite structures (i.e., on relational databases) has not been analyzed so far.

Here we propose a novel application of default logic by introducing a *default query language* (DQL) for finite relational databases, which is based on default rules. The main result of this paper is that DQL is more expressive than DATALOG$^\neg_{stable}$. In particular DQL is as expressive as $SO_{\exists\forall}$, the existential-universal fragment of second-order logic. This result is not only of theoretical importance: We exhibit queries –which are useful in practice– that can be expressed with DQL and can not with DATALOG$^\neg_{stable}$. Two of the queries are taken from the realm of economics, while another one deals with diagnostic reasoning on a circuit; it appears that DQL allows for an easy formalization of this process.

An alternative way of describing our main result is to say that DQL "captures" the complexity class Σ_2^p of the

A preliminary and partial version of this paper appears in the Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94), Bonn, Germany, May 1994. Morgan Kaufmann Publishers, Inc. San Francisco, Ca.

M. Cadoli is with the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy. E-mail: cadoli@dis.uniroma1.it

T. Eiter and G. Gottlob are with the Information Systems Department, Technical University of Vienna, Paniglgasse 16, A-1040 Wien, Austria. E-mail: (eiter|gottlob)@dbai.tuwien.ac.at

polynomial hierarchy, while $\mathrm{DATALOG}^{\neg}_{stable}$ "just" captures the class NP. Therefore DQL is more expressive than $\mathrm{DATALOG}^{\neg}_{stable}$ provided $\Sigma_2^p \neq \mathrm{NP}$, i.e., provided the polynomial hierarchy does not collapse –a property that has been widely conjectured and that will be assumed throughout this work.

We remind that Σ_2^p-completeness of credulous propositional default reasoning has been recently proven [12], [13]. It is therefore important to remark that the expressive power of a language is not necessarily the same as its complexity. Several languages with this property are known, cf. [14], [15]. As an example, a language which does not capture NP –even if it has an underlying NP-complete problem– has been shown by Stewart in [16].

To show that DQL is well-suited for formulating useful and practical queries in different domains, we present several examples. One of the examples deals with troubleshooting for electric circuits. It appears that DQL allows for an elegant implementation of advanced diagnostic reasoning principles (in particular, abductive model-based diagnosis [17], [18]). Other examples deal with relevant queries in business administration.

The structure of the paper is the following. Section II provides some motivating examples. In Section III we state necessary preliminaries on query languages for relational databases and provide a brief introduction to default logic. In Section IV we give the definition of the query language DQL, providing syntax and semantics. In Section V we give a formal proof of the fact that DQL captures Σ_2^p. Moreover, we briefly address the computational complexity of DQL, and we consider the sublanguage of normal DQL queries, which is analogous to normal default theories. In Section VI we show how complex problems like those presented in Section II can be solved in DQL. We draw some conclusions and compare DQL with some query languages arising from logic programming in the final Section VII. In order to increase the readability, the proofs of all results have been moved to an appendix.

II. Motivating examples

Before diving into the technical part of the paper, let us motivate with some examples, why query languages with higher expressiveness than $\mathrm{DATALOG}^{\neg}_{stable}$ are needed. These examples will be dealt with in much more detailed way in Section VI.

Our first example here is from the domain of model-based diagnosis (MBD) of electric devices. This very promising field is an active research area in AI cf. [19], [20]. It would therefore be interesting to integrate the paradigm of MBD into relational databases, as the descriptions of large electric systems (e.g., of power networks) should beneficially be stored in a database.

Example 1: **(Electric circuit troubleshooting)** An electric circuit has several components. All of them might be in some way malfunctioning, or even misdesigned (e.g., the resistance value of a resistor could be too small).

In the (simplified) electric circuit represented in Figure 1, we have a battery, two resistors and two fuses, and a control light with a bulb. The higher the battery's voltage the higher the current's amperage. Similarly, the smaller the resistor's resistance the higher the amperage. If the amperage in the circuit exceeds a certain amount, one of the fuses melts; the circuit is interrupted, and the control light is off. It is not predictable which one of the fuses will melt, or even if both of them will.

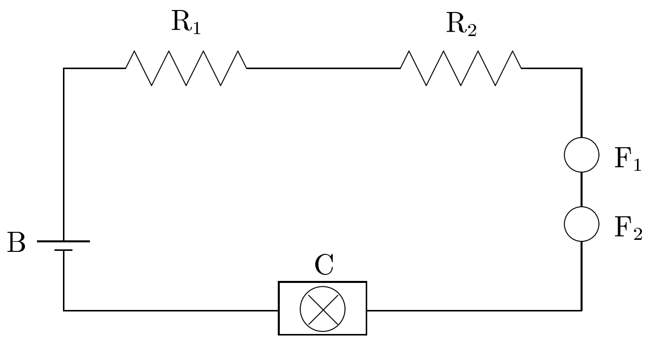

Fig. 1. Electric circuit

In the kind of MBD known as *abductive diagnosis*, it is customary to to say that when a component does not work properly, one or more *effects* can show up. Moreover, an effect is originated by a set of *causes*, and a set of causes does not originate univocally an effect. In the above circuit, melting of a fuse is a effect, and battery's voltage being too high is a cause. Similarly, a melted fuse –as well as a broken bulb– is a cause for the effect of the control light being off (cf. [21], [17], [18], [22] for a background and overview on results on abductive diagnosis).

In abductive diagnosis, knowledge about a circuit such as the above one is typically described by a diagnostic theory that consists of rules of the form

$$C_1 \wedge \cdots \wedge C_n \rightarrow E_1 \vee \cdots \vee E_m, \tag{1}$$

where the C_i's are causes and the E_j's are effects. As an example, the following rule states that melting of one of the fuses will occur whenever both resistance values are low

$$\mathrm{R}_1\text{-low} \wedge \mathrm{R}_2\text{-low} \rightarrow \mathrm{F}_1\text{-melts} \vee \mathrm{F}_2\text{-melts} \tag{2}$$

Not necessarily all of the antecedents of a rule of the kind (1) are possible causes. As an example, we might be sure that the first resistor is not underdimensioned; in such a case, R_1-low would not be a possible cause. The possible causes are called *hypotheses*. Finally, in each specific situation we make a set of *observations*, e.g., the control light is off.

Given a situation, described by a set of rules of the kind (1), a set of hypotheses and a set of observations, abductive diagnosis looks for an *explanation* of the observations we make. An explanation is a minimal set of hypotheses (wrt inclusion), such that their validity implies through the rules all observations. In a general situation it is possible that one, many, or no explanations exist. The hypotheses that belong to at least one explanation of the diagnostic problem are called *relevant* for the problem; computing the relevant facts of a diagnostic problem is an important subtask in troubleshooting, for being able to focus fault localization on a subset of the possible causes.

It is easy to conceive a database that stores rules on causes and effects, as well as hypotheses and observations. Anyway, from results on the complexity of propositional abduction in [22], it follows easily that computing the relevant facts for troubleshooting problems is a problem that is hard for the complexity class Σ_2^p of the polynomial hierarchy (cf. [23]); in fact, already deciding if a specific fact is relevant is hard. This means that it is not possible to compute the relevant facts in $\mathrm{DATALOG}^{\neg}_{stable}$ (the reason why is discussed in Section III-A). Similarly, it is not possible to formulate a Boolean query in $\mathrm{DATALOG}^{\neg}_{stable}$ that decides if a specific fact is relevant. On the other hand, in Section VI we will show how it is possible to write a DQL query that computes all facts relevant for the problem. □

The remaining examples in this section deal with business applications in a somewhat simplified economic world.

Example 2: **(Strategic companies)** Suppose a holding owns some companies. Each company produces a set of products. Each product is produced by at most two companies. As an example, pasta is produced by Barilla and Saiwa, while wine is produced only by Barilla [1].

Suppose the holding experiences a crisis and has to sell one company. The holding's policy is to keep on producing all products. This clearly makes it impossible to sell some companies –as an example the Barilla company in the above situation, because it would be impossible to produce wine. Anyway the managers are even more cautious: They know that in the future it may be necessary to sell more companies, and they do not want to get into a situation in which they will not be able to produce all products. More formally, they are interested in the minimal sets of companies that produce all products. A company is *strategic* if it is in at least one of such minimal sets. Therefore a query which is very relevant to the managers is whether a company is strategic or not: They prefer to sell a non-strategic company first, because after the transaction the minimal sets of companies that produce all products remain the same.

Now let us consider a slightly more complex situation, in which up to three (say) companies can *control* another company. As an example, companies Barilla and Saiwa together have control over company Frutto.

Assume now that the following constraint is imposed for determining strategic companies: A company A which is controlled by the companies $B_1, \ldots, B_i$ can be sold only if also one of the companies $B_1, \ldots, B_i$ is sold. This constraint completely changes the minimal sets of companies that produce all products.

In the former case –no controlled companies– the problem of deciding whether a company is strategic is in the complexity class NP (cf. [24]), while in the latter case the same problem is easily shown to be complete for the class Σ_2^p of the polynomial hierarchy (cf. [25]). As a consequence (cf. Section III-A), the former query is expressible in $\mathrm{DATALOG}^{\neg}_{stable}$, while the latter is not. In Section VI we will show how it is possible to write two DQL queries that compute the sets of the strategic companies in the two cases. □

[1]foodnote: Any coincidence with existing companies is purely casual.

Example 3: **(Maximal trust)** In a set of companies it is possible to make agreements. A company may have an agreement with another company, and of course it can have simultaneously several agreements with different companies. A *trust* is a set T of companies such that each one has an agreement with each other company in T; a trust T is *maximal* if there is no trust T' that has more companies than T, i.e., $|T'| > |T|$. Now, the following query rises: which companies belong to a maximal trust?

Computing this query is an NP-hard problem, and also a co-NP-hard problem. As a consequence (cf. Section III-A) the query can not be written in $\mathrm{DATALOG}^{\neg}_{stable}$ (unless some unexpected collapse of complexity classes). On the other hand, the query can be expressed in DQL. □

III. Preliminaries

A. Relational databases

For a background on relational databases and query languages, the reader is referred to [26], [27]. A *database schema* R is a finite set $\{R_1, \ldots, R_n\}$ of relation schemata. A *relation schema* R_i has a name N_i and a finite list of attributes $L_i = \langle A_1, \ldots, A_{l_i} \rangle$. It will sometimes be denoted as $R_i(A_1, \ldots, A_{l_i})$. The number l_i is the arity of the relation schema R_i. We assume that there is an underlying set $\mathcal{U}$ of objects that can be used in relations (the so-called *domain*). The domain $\mathcal{U}$ is arbitrarily large but finite. Given a relation schema R_i, a *relation instance* is a set of tuples of the form $\langle a_1, \ldots, a_{l_i} \rangle$, where $a_j \in \mathcal{U}$ $(1 \leq j \leq l_i)$. A *database instance* W is a set of relation instances. The set of objects occurring in a database (the so-called *active domain*) is a subset –possibly not strict– of $\mathcal{U}$.

A *database query* (or often, simply query) is a mapping from the set of all database instances of a fixed database schema R (intuitively, the input relations) to the database instances of a fixed database schema S (the output relations). It is assumed that the mapping is computable and generic (i.e., invariant under renamings of the constants in $\mathcal{U}$).

A *query language*, with its syntax and semantics, evaluates a well-formed formula (the *query*) over any database instance and returns an answer, which is a database instance; thus, each query defines a database query. The answer of an *I/O* query is a set of relation instances, while the answer of a *boolean* query is either `yes` or `no` (in this case, the schema S consists of a single propositional letter, or technically, of a 0-ary relation).

The *expressive power* of relational database query languages is one of the most studied topics in database theory. Basically one is interested in knowing which relations can be expressed by a query language and which relations can not. A relation is expressible in a query language, if there is a query in the language that, on any input database instance returns precisely the desired relation as the answer. For an example, transitive closure of a graph is expressible in DATALOG [26].

One way of presenting a result in this area is to say that a query language can/can not express a specific relation. For example, it is well-known that relational calculus can not express the transitive closure of a relation [28]; the relation of satisfiable propositional clause sets can be computed by a fixed program in $\text{DATALOG}^{\neg}_{stable}$ but not in DATALOG (unless P=NP), cf. [4].

Measuring the complexity of Boolean queries is straightforward. Usually, this is defined by referring to the following problem: Given a database instance, decide if the query evaluates to `yes`. The complexity of a query language is the complexity of this problem for the queries of this language. This is also termed as *data complexity*. If the query is not fixed, i.e., the query and the database are given for input, we analogously obtain the *combined complexity* of the query language, which is often much higher.

In order to deal with the complexity and expressive power of I/O queries, we use the key concept of query *recognizability*.

Definition III.1: (cf. [29]) Let $\mathbf{C}$ be a complexity class. A query mapping database instances over R into database instances over S is $\mathbf{C}$-*recognizable*, if deciding whether a tuple $\mathbf{t}$ belongs to a certain output relation $S_i \in S$ is in $\mathbf{C}$.

Then, the data and combined complexity of an I/O query language can be defined in terms of query-recognizability similar as for Boolean queries.

Typically, the expressive power of a query language is represented as a set of logical sentences. As an example, the results in [4] entail that the expressive power of $\text{DATALOG}^{\neg}_{stable}$ under brave semantics is $\text{SO}_{\exists}$, the existential fragment of second-order logic, i.e., the set of sentences

$$(\exists \mathbf{S})\phi(\mathbf{S}),$$

where $\mathbf{S}$ is a list of predicate symbols and $\phi(\mathbf{S})$ is a function-free first-order formula in which (among possibly others) the predicates in $\mathbf{S}$ occur.

The traditional notion of a query language *capturing* a complexity class $\mathbf{C}$ is that the queries definable in this language are precisely those which are recognizable in $\mathbf{C}$. For example, Fagin's celebrated theorem is that class of $\text{SO}_{\exists}$-queries captures the class NP [30].

Figure 2 describes well-known relations between the expressive powers of several query languages and correspondences to complexity classes, cf. [31], [32], [2]. Each edge denotes inclusion, i.e., less expressive power, which –assuming that complexity classes do not collapse– is always strict.

As well-known, DATALOG does not express all queries computable in polynomial time, and is incomparable to first-order logic. Stratified DATALOG and inflationary DATALOG have increasing expressive power; they can express all first-order queries, but still not all polynomial time computable queries queries. The latter is possible –already for DATALOG with negation of input relations– if a linear ordering $<$ of the universe is given, cf. [33].

First-order logic is equivalent with relational algebra [28]. Note that DATALOG can express queries whose computation is P-complete, and hence, unless P = NC, DATALOG is not included in the class of queries computable by NC-circuits. The queries definable in first-order logic with a linear ordering $<$ are the same as the AC^0-queries [31] (AC^0 is the class of problems solvable by uniform families of constant depth circuits of polynomial size, where the fan-in of gates is unbounded). The AC^0-queries are a proper subclass of the NC-queries (NC is the class of problems solvable in polylog parallel time with polynomial amount of total work), which are a –most likely proper– subclass of the queries computable in polynomial time.

The While-queries constitute a class of queries definable in a programming-style query language, which provides assignments of first-order expressions to tuple variables, sequencing of statements, and while statements. This class appears to be quite powerful, since PSPACE-hard queries can be expressed. On the other hand, not all queries computable in PSPACE can be expressed (e.g., if the universe has even size); however, this is possible if a linear ordering $<$ of the universe is provided.

As already mentioned above, $\text{DATALOG}^{\neg}_{stable}$ under brave semantics is equivalent to the $\text{SO}_{\exists}$ fragment of second-order logic and thus captures NP. The queries definable in the existential-universal second-order logic, $\text{SO}_{\exists\forall}$, i.e., the set of sentences

$$(\exists \mathbf{S})(\forall \mathbf{T})\phi(\mathbf{S}, \mathbf{T}), \tag{3}$$

on a relational function-free vocabulary where $\mathbf{S}, \mathbf{T}$ are disjoint lists of predicate symbols and $\phi(\mathbf{S}, \mathbf{T})$ is first-order, are of particular interest in this paper. The query language based on default logic we present in Section IV, DQL, can express exactly $\text{SO}_{\exists\forall}$. This fragment of second-order logic captures the class Σ_2^p of the polynomial hierarchy, which consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time with use of an oracle for a problem (i.e., a subroutine for solving this problem in unit time) in NP (cf. [23]). Consequently, unless $\Sigma_2^p = \text{NP}$ (which is widely conjectured to be false), DQL is much more expressive than $\text{DATALOG}^{\neg}_{stable}$. Finally, the queries definable in full second-order logic are those computable within the polynomial hierarchy [34], [35].

B. Default logic

Default logic has been introduced by Reiter in [11]; it is one of the most extensively studied non-monotonic formalisms. For a detailed treatment of this formal system, the reader is referred to [36]. Interesting relations between default logic and database theory have been shown by Bidoit and Froidevaux in [37]. They used default logic for defining a semantics for negation in deductive databases.

In default logic the knowledge about the world is divided into two parts, representing certain knowledge and defeasible rules, respectively. The first part (denoted with W)

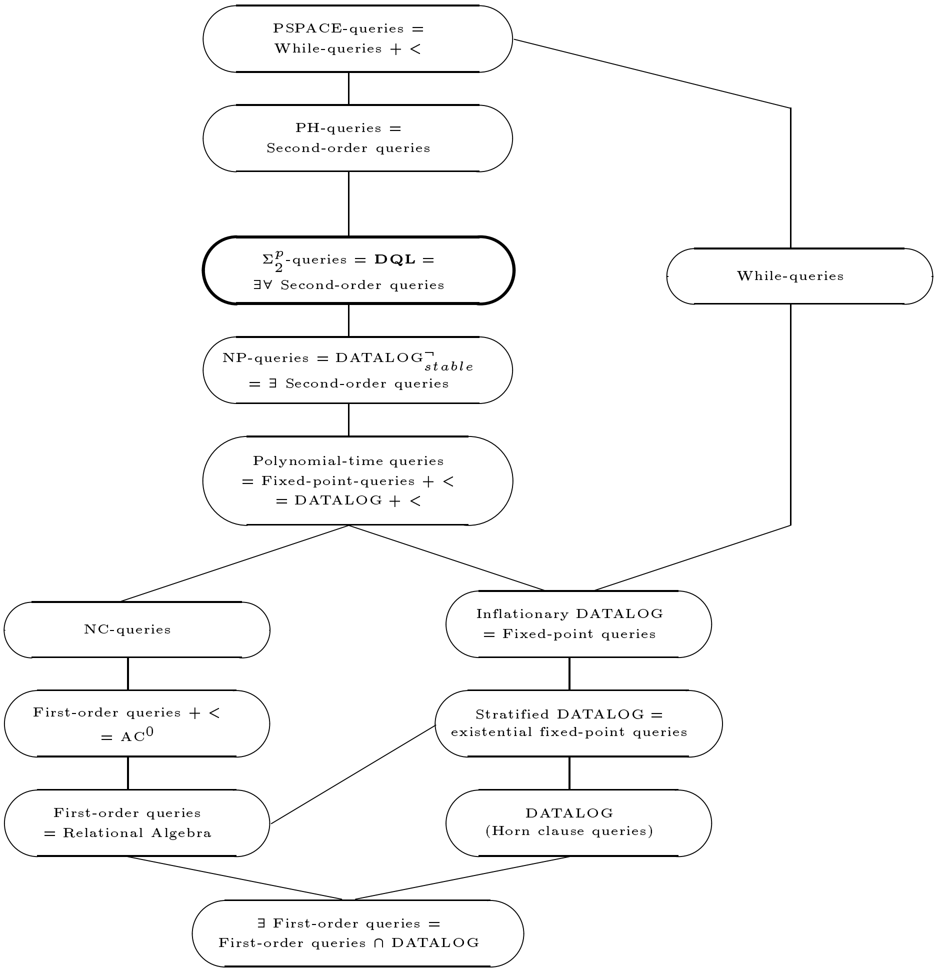

Fig. 2. Query languages (noncollapsing complexity classes)

is a set of closed first-order formulae, while the second one (denoted with D) is a collection of special inference rules called defaults. A *default* is a rule of the form

$$\frac{\alpha(\mathbf{x}) : \beta_1(\mathbf{x}), \ldots, \beta_n(\mathbf{x})}{\gamma(\mathbf{x})}$$

where $\alpha(\mathbf{x}), \beta_1(\mathbf{x}), \ldots, \beta_n(\mathbf{x}), \gamma(\mathbf{x})$ are well-formed formulas whose free variables are among those of $\mathbf{x} = x_1, \ldots, x_m$. $\alpha(\mathbf{x})$ is called the *prerequisite* of the default, $\beta_1(\mathbf{x})$, ..., $\beta_n(\mathbf{x})$ ($n \geq 0$) are called *justifications* and $\gamma(\mathbf{x})$ is the *consequence*. When $n = 0$, then the propositional constant $\top$ (true) is implicitly assumed as the justification of the default. For convenience, we will omit writing $\alpha(\mathbf{x})$ if the prerequisite is $\top$. A default is *closed* if none of $\alpha, \beta_1, \ldots, \beta_n, \gamma$ contains free variables. A default theory $\langle D, W\rangle$ is closed iff all the defaults in D are closed. A default or default theory which is not closed is called *open*.

The semantics of a closed default theory $\langle D, W\rangle$ is based on the notion of *extension*, which is a possible state of the world according to the knowledge base. Formally, an extension E can be defined using a quasi-inductive construction as follows. Define that for any set E of first order formulae, $E_0 = W$, and for $i \geq 0$

$E_{i+1} = Cons(E_i) \cup \{\, \gamma \mid \alpha : \beta_1, \ldots, \beta_n/\gamma \in D, \alpha \in E_i, \forall_j \neg\beta_j \notin E \,\}$,

where $Cons()$ denotes classical deductive closure. Then, E is an extension of $\langle D, W\rangle$ iff $E = \bigcup_{i=0}^{\infty} E_i$. [2] Notice that E is deductively closed, and hence an infinite object.

Each extension E of $\langle D, W\rangle$ is identified by its *generating defaults*, which allow for a compact representation of E. The generating defaults of E, which we denote by $GD(E, \langle D, W\rangle)$, are the defaults $\frac{\alpha\, :\, \beta_1, \ldots, \beta_n}{\gamma}$ from D such that $\alpha \in E$ and $\neg\beta_j \notin E$, for all $j = 1, \ldots, n$. Then, E is constructible from W and $GD(E, \langle D, W\rangle)$ as follows:

Lemma 1: ([11, Theorem 2.5]) Let E be an extension of the closed default theory $\Delta = \langle D, W\rangle$. Then, $E = Cons(W \cup \{\, \gamma \mid \frac{\alpha\, :\, \beta_1, \ldots, \beta_n}{\gamma} \in GD(E, \Delta)\})$.

The definition of extension is extended to open default theories by assuming that the defaults with free variables implicitly stand for the infinite set of closed defaults obtained by replacing the free variables with terms of the Herbrand Universe of the default theory.

A default theory can have one, multiple or no extensions

[2] Actually, this is a an equivalent characterization for extensions rather than the original definition in terms of the fixed-points of an operator Γ.

in general. Therefore, defining entailment of a formula ϕ from a default theory $\langle D, W\rangle$ is not straightforward. The standard variants are *credulous entailment*, under which ϕ is entailed if ϕ belongs to at least one extension of $\langle D, W\rangle$, and *skeptical entailment*, under which ϕ follows if ϕ belongs to all extensions of $\langle D, W\rangle$. From the computational side, credulous and skeptical reasoning have been extensively studied in the literature [38], [12], [13].

We conclude this section with a well-known example for a default theory.

Example 4: (**Nixon diamond**) Assume that

$$D = \left\{ \frac{Q : P}{P}, \frac{R : \neg P}{\neg P} \right\}, \quad W = \{ R, Q \},$$

where P, Q, and R are propositional atoms with meaning that someone is a pacifist, a quaker, and a republican, respectively. The default theory $\Delta = \langle D, W\rangle$ has two extensions: $Cons(\{ P, Q, R \})$ and $Cons(\{ \neg P, Q, R \})$. In the first extension, a republican that is also a quaker is a pacifist, while in the second he is not; the former extension has the (single) generating default $\frac{Q:P}{P}$, and the latter $\frac{R:\neg P}{\neg P}$. Thus, under credulous semantics, we can conclude that the person is a pacifist, as well as that (s)he is not a pacifist; under cautious semantics, we can conclude neither. □

IV. Default Query Language (DQL)

In this section we give syntax and semantics of the default query language DQL, and we consider a simple example for illustration; more examples and applications will be considered in Section VI.

A. Syntax

A *DQL Input/Output query* Q is pair (B, D) of a set B of first-order formulas (the background knowledge) and a set D of open default rules, where the first-order language is function-free and quantifier-free, plus a set of *output relation schemata* $S = \{S_1, \ldots, S_m\}$. The set of predicate symbols occurring in the defaults of Q contains all the names of the relation schemata of the database (the *extensional* relations) and possibly other symbols (the *intensional* relations). Output relations are intensional. The intuitive meaning of the query is the following: We want to compute all tuples in the S_i relations which can be inferred under the credulous default semantics. (See the next section for a formal definition.) In particular we apply the credulous default semantics to the propositional instantiation of the open defaults and the background knowledge in the query, plus the database.

A *DQL Boolean query* is a set of open default rules plus a ground formula ϕ. The intuitive meaning of the query is the following: We want to know whether ϕ follows –under the credulous default semantics– from the propositional instantiation of the defaults in the query plus the database.

Example 5: **(Mary's office)** We have two relation schemes PROG (for "programmer") and MGR (for "manager") which both have a single attribute NAME. The database instance W is as in the following table:

PROG

NAME
Peter
Paul
Mary

MGR

NAME
Mary

The I/O query Q has (B, D) as follows: the background knowledge B is empty, and

$$D = \left\{ \frac{prog(x)\ :\ \neg ex(x)}{small\text{-}office(x)}, \frac{:\neg ex(x)}{\neg ex(x)}, \frac{prog(x) \wedge mgr(x):}{ex(x)} \right\};$$

S consists of the single relation scheme SMALL-OFFICE(NAME).

The relational database states that Peter, Paul, and Mary are programmers and that Mary is a manager. The query is made out of three open defaults. The first one states that a person who is provably a programmer –and that can not be proven to be exceptional– has a small office by default. The second default states that people who can not be proven to be exceptional should be regarded as not exceptional. The third default states the rule that people who are provably both programmers and managers are exceptional. The intuitive meaning of the query is that we want to know the set of people having small offices.

For an example of a Boolean query, consider the same set of defaults, plus the ground formula $small\text{-}office(Peter)$. The intuitive meaning of the query is that we want to know whether Peter has a small office or not. □

B. Semantics

The semantics of a DQL Q is defined in terms of a default theory obtained by instantiating the query over the domain of the database, cf. [2], [14].

Let W be a database instance over the set of relation schemata $\{R_1, \ldots, R_n\}$. For each relation instance R_i, let $R_i|W$ be the set of tuples in W belonging to R_i. We denote as $COMP(W)$ the *completion* of the database, i.e., the set of the following ground literals:

- $R_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i}\rangle \in R_i|W$;
- $\neg R_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i}\rangle \in \mathcal{U}^{l_i} \setminus R_i|W$.

(This is the standard translation from databases-as-models into databases-as-complete-theories (essentially), as shown for example in [39]).

Let $\phi(\mathbf{x})$ be a formula whose free variables are among $\mathbf{x} = x_1, \ldots, x_n$, and let $\zeta = \zeta_1, \ldots, \zeta_n$ be a list of objects from $\mathcal{U}$. Then, we denote by $\phi[\mathbf{x}/\zeta]$ the result of simultaneously substituting ζ_i for x_i in ϕ, for all $i = 1, \ldots, n$.

Let (B, D) be pair of background knowledge and open defaults of a (Boolean or I/O) query Q. We denote by $INST(B)$ the *instantiation* of B, which is the set of ground formulas

$$\{ \phi[\mathbf{x}/\zeta] \mid \phi(\mathbf{x}) \in B, \, \zeta \in \mathcal{U}^n \}.$$

Similarly, we denote by $INST(D)$ the *instantiation* of D, which is the set of ground defaults

$$\left\{ \frac{\alpha[\mathbf{X}/\zeta] : \beta_1[\mathbf{X}/\zeta], \ldots, \beta_n[\mathbf{X}/\zeta]}{\gamma[\mathbf{X}/\zeta]} \mid \frac{\alpha(\mathbf{X}) : \beta_1(\mathbf{X}), \ldots, \beta_n(\mathbf{X})}{\gamma(\mathbf{X})} \in D, \, \zeta \in \mathcal{U}^n \right\}.$$

Then, for any database W for R over domain $\mathcal{U}$, we denote by $Q{+}W$ the default theory with defaults $INST(D)$ and first-order formulas $COMP(W) \cup INST(B)$, i.e.,

$$Q{+}W = \langle INST(D), COMP(W) \cup INST(B)\rangle.$$

Now, the *credulous semantics* of query Q is defined as follows.

If Q is an I/O query, i.e., the pair (B, D) plus a set $S = \{S_1, \ldots, S_m\}$ of output relations, then the answer to Q is the database instance W' for S over domain $\mathcal{U}$ defined as follows: For each $S_i \in S$, $S_i|W'$ is the set of all ground tuples $\mathbf{t}$ over $\mathcal{U}$ such that $S_i(\mathbf{t})$ follows from $Q + W$ under credulous semantics, i.e., it is in at least one extension of $Q + W$.

If Q is a Boolean query, then the answer is yes if the distinguished ground formula ϕ in Q follows from $Q + W$ under the credulous default semantics; otherwise, the answer is no.

The *skeptical semantics* of query Q is defined analogously. (That is, if Q is an I/O query, the $S_i|W'$ contains all tuples $\mathbf{t}$ such that $S_i(\mathbf{t})$ follows from $Q + W$ under the skeptical semantics; if Q is Boolean, the answer is yes if ϕ follows from $Q + W$ under skeptical semantics and no otherwise.)

Remark: We notice that, in the semantics for DQL queries, two sorts of non-monotonic reasoning are involved: first of all, the database is completed ($COMP(W)$), secondly, default rules are applied ($INST(D)$). Completion of the database prohibits that new positive facts are concluded about input predicates. This is a usual requirement in a setting of strict relational databases.

In fact, the whole mechanism could be made homogeneous by using default rules for obtaining completion of the database as well. One way to achieve this is to use the following method:

- for each extensional relation R_i, introduce a new predicate R'_i (of the same arity) that does not occur elsewhere;
- build a set D' consisting of the following defaults ($1 \leq i \leq n$):

$$\frac{:\neg R'_i(\mathbf{x})}{\neg R'_i(\mathbf{x})}, \quad \frac{R'_i(\mathbf{x}):}{R_i(\mathbf{x})}, \quad \frac{\neg R'_i(\mathbf{x}):}{\neg R_i(\mathbf{x})}$$

- define $G(W)$ as the set of all ground atoms $R'_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i}\rangle \in R_i|W$ and $R_i \in R$.

It can be shown that the default theory

$$\Delta = \langle INST(D \cup D'); G(W) \cup INST(B)\rangle$$

provides the same answers (to both Boolean and I/O queries) as $Q + W$. Intuitively, the R'_i predicates serve to transfer the extension of the input relations to the respective predicate letters R_i. In fact, in any extension of Δ, R'_i is complete (i.e., every ground atom is true or false) by the first default, and R'_i must coincide with R_i by the second and third default.

Note that if one does not allow occurrence of extensional relations in the conclusions of user defaults (a similar restriction is often made in logical query languages, e.g. in DATALOG) and in B, then also the default theory

$$\langle INST(D \cup \{\, :\neg R_i(\mathbf{x})/\neg R_i(\mathbf{x})\}); W \cup INST(B)\rangle,$$

where W is seen as set of ground atoms $R_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i}\rangle \in R_i$ and $R_i \in R$, provides the same answers as $Q + W$.

Note that the background theory can be used to state integrity constraints on the input. This is e.g. possible by using a designated atom "*invalid*", whose derivability indicates that the integrity constraints are violated. E.g., a functional dependency constraint on relation R can be implemented by the formula $r(x, y) \wedge r(x, z) \wedge y \neq z \rightarrow invalid$.

It appears that the credulous and the skeptical variant of DQL are dual with respect to complements in their expressive capability. In fact, all forthcoming results about complexity and expressiveness of credulous DQL apply to skeptical DQL in the dualized form, i.e., if each complexity class is replaced by its complementary complexity class. For instance, forthcoming Theorem 1 for credulous DQL can be rephrased for skeptical DQL by saying that the Boolean (skeptical) DQL queries precisely capture the class co-$\Sigma_2^p = \Pi_2^p$. We will not carry out such rephrasements, but leave this (as well as adaptations of proofs) to the interested reader. Moreover, we will for brevity mainly use credulous DQL in our examples and applications; notice that credulous semantics and skeptical semantics coincide if the $Q + W$ has only one extension, which is often the case.

Let us see how the semantics of DQL works in the example shown in the previous subsection.

Example 5 (Mary's office –continued) We assume that the domain is $\{Peter, Paul, Mary\}$, i.e., that the domain is the same as the active domain. Then,

$$\begin{aligned} COMP(W) &= \{prog(Peter), \neg mgr(Peter), \\ &\quad prog(Paul), \neg mgr(Paul), prog(Mary), mgr(Mary)\}; \\ INST(B) &= \emptyset; \\ INST(D) &= \left\{ \frac{prog(Mary):\neg ex(Mary)}{small\text{-}office(Mary)}, \frac{:\neg ex(Mary)}{\neg ex(Mary)}, \right. \\ &\quad \frac{prog(Mary)\wedge mgr(Mary):}{ex(Mary)}, \frac{prog(Paul):\neg ex(Paul)}{small\text{-}office(Paul)}, \\ &\quad \frac{:\neg ex(Paul)}{\neg ex(Paul)}, \frac{prog(Paul)\wedge mgr(Paul):}{ex(Paul)}, \frac{:\neg ex(Peter)}{\neg ex(Peter)} \\ &\quad \left. \frac{prog(Peter):\neg ex(Peter)}{small\text{-}office(Peter)}, \frac{prog(Peter)\wedge mgr(Peter):}{ex(Peter)} \right\}. \end{aligned}$$

The default theory $Q + W$ has one extension, whose generating defaults are:

$$\frac{prog(Mary)\wedge mgr(Mary):}{ex(Mary)}, \frac{prog(Paul):\neg ex(Paul)}{small\text{-}office(Paul)}$$
$$\frac{prog(Peter):\neg ex(Peter)}{small\text{-}office(Peter)}, \frac{:\neg ex(Peter)}{\neg ex(Peter)}, \frac{:\neg ex(Paul)}{\neg ex(Paul)}.$$

The answer to the I/O query (under credulous as well as skeptical semantics) is the relation instance:

SMALL-OFFICE

NAME
Peter
Paul

In other words, Peter and Paul have small offices, and Mary does not have a small office. The answer to the Boolean query is yes. □

V. Expressive power of DQL

In the previous section we have seen that default logic is suitable as a query language, i.e., as a language for manipulating relations. A very interesting question is the following: Which relations can be computed by DQL? Which relations can not? In other words, what is the expressive power of DQL?

In this section we show that the expressive power of DQL is $SO_{\exists\forall}$, the existential universal fragment of second order logic. This result is derived for Boolean queries as well as for general I/O queries. In order to do so, we first derive the result on Boolean DQL queries and, by using this result, derive then the result for general DQL I/O queries.

Having established the expressive power of DQL, we will then analyze the expressive power of two natural sublanguages of DQL, namely normal DQL and semi-normal DQL, which allow only for normal and semi-normal defaults in the query, respectively. These restrictions of DQL correspond to the classes of normal and semi-normal default theories, which constitute important fragments of default logic.

A. Boolean DQL queries

We are now ready to prove our main result, which concerns DQL Boolean queries. The following theorem says that DQL is capable of all and only those Boolean queries whose complexity is in the class Σ_2^p. For the practitioner, this provides very helpful information about what can be expressed in DQL and what not. As soon as we know the complexity of a query that we want to implement, we can often immediately tell whether this query is feasible in DQL or not. For example, all queries in Section II have complexity in Σ_2^p. As a consequence, they all can be expressed in DQL (cf. Section VI). On the other hand, if the complexity is higher than Σ_2^p, then it is impossible (or extremely unlikely) that the query can be implemented in DQL. As a consequence, a query that tells whether a player has a win in a given situation of the GO-GAME (stored in the database) can not be written in DQL, since this is a PSPACE-complete problem (cf. e.g., [40]).

Theorem 1: The Boolean DQL queries precisely capture the class Σ_2^p.

Notice that there are query languages that can express part of the queries in Σ_2^p –and even Σ_2^p-hard queries– but fail to express very simple queries [15]. For such languages, it is in general not easy to tell whether a certain (even simple) query can be expressed, and leaves the programmer with the uncertainty whether the query can be implemented at all.

From the result on the expressiveness of DQL, we can immediately derive a result on the complexity of DQL. In particular, we obtain the following characterization of the *data complexity*, i.e., of evaluating a fixed query Q over varying databases. The following theorem tells us that this is a Σ_2^p-complete problem in general. Roughly speaking, this means that query evaluation is still NP-complete, even if we have an oracle (i.e., a subprogram) for solving NP-complete problems available and do not account for calls to this oracle. Practically speaking, this means that we can not reduce query evaluation to solving an NP-complete problem (e.g., checking classical satisfiability of a set of propositional clauses) efficiently. In particular, an efficient reduction to integer programming methods, which have been investigated in the context of logic programming [41], is not feasible.

Theorem 2: The data complexity of Boolean DQL is Σ_2^p-complete.

We remark that complexity grows a lot if the query is not fixed, i.e., in case of evaluating a given query on a given database (*combined complexity*). In fact, there is an exponential increase in complexity, and the combined complexity can be shown to be complete for the class $\mathrm{NEXPTIME}^{\mathrm{NP}}$, which is the exponential analogue for $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$.

B. DQL I/O queries

Now we consider DQL queries that compute output relations. Using Theorem 1, we show the following more general theorem, which tells that all and only the I/O queries with complexity in Σ_2^p can be expressed in DQL. As in the case of Boolean queries, this gives again valuable information about what can be implemented and what not. For example, we immediately know that a query computing the transitive closure of a graph can be implemented in DQL, since its complexity is polynomial.

Theorem 3: A database query is Σ_2^p-recognizable if and only if it is definable as a DQL I/O query.

C. Normal DQL queries

Fragments of default logic resulting by imposing syntactical restrictions on the default rules, have been considered already by Reiter in his seminal paper [11].

Common restrictions are rules of the forms $\frac{\alpha(\mathbf{X})\,:\,\beta(\mathbf{X})}{\beta(\mathbf{X})}$, called *normal* default rules, and $\frac{\alpha(\mathbf{X})\,:\,\beta(\mathbf{X})\wedge\gamma(\mathbf{X})}{\beta(\mathbf{X})}$, which are called *seminormal* default rules; a default theory is called *normal* (resp. *semi-normal*) iff every default is normal (resp. semi-normal).

The classes of normal and semi-normal default theories are important and well-studied fragments of default logic. In particular, normal default theories model jumping to a conclusion if and only if it is possible. Semi-normal defaults allow for expressing priorities between otherwise normal defaults. In many applications default knowledge can be represented by a normal or semi-normal default theory (cf. e.g. [42] for an extensive treatment.)

This motivates to consider the sublanguages of DQL which correspond to normal and semi-normal default theories, respectively.

Definition V.1: A default query Q is called *normal* (resp. *semi-normal*) iff each default in D is normal (resp. semi-normal), i.e., of the form $\frac{\alpha(\mathbf{X})\,:\,\beta(\mathbf{X})}{\beta(\mathbf{X})}$ (resp. $\frac{\alpha(\mathbf{X})\,:\,\beta(\mathbf{X})\wedge\gamma(\mathbf{X})}{\beta(\mathbf{X})}$).

Clearly, normal DQL queries are also semi-normal, and thus a proper syntactical restriction.

Example 6: **(Students and employees)** Assume that R consists of the two relation scheme R relation schemes MARRIED and STUDENT, which both have a single attribute NAME. The I/O query specified by (B, D) is used to compute instances of the output relations

ADULT(NAME) and EMPLOYEE(NAME):

$$B = \{ \mathit{married}(x) \rightarrow \mathit{adult}(x) \},$$
$$D = \left\{ \frac{\mathit{student}(x) : \neg \mathit{employee}(x)}{\neg \mathit{employee}(x)}, \frac{\mathit{adult}(x) : \mathit{employee}(x)}{\mathit{employee}(x)} \right\}.$$

The background knowledge is that married people are adults. The first default in D states that students are typically not employed, and the second that adults are usually employed. Consider the following instance W of R, where the domain is $\{ \mathit{John}, \mathit{Sue}, \mathit{Betty} \}$:

MARRIED

NAME
Sue

STUDENT

NAME
John
Sue

The query computes the following output:

ADULT

NAME
Sue

EMPLOYEE

NAME
Sue

□

The question rises if normal DQL and semi-normal DQL are less powerful than general DQL. It turns out that normal DQL is already as powerful as general DQL, and thus can express all queries in Σ_2^p. This can be shown by slight modifications of the default theories constructed in the proofs of Theorems 1 and 3. Again, these theorems sharply describe what can be implemented in this language and what not; the advantage is that from the complexity of a query, one can mostly immediately tell whether it can be implemented or not (although, this does not give us a clue of how the query program may look like).

Theorem 4: The Boolean normal DQL queries precisely capture the class Σ_2^p.

Theorem 5: The normal I/O DQL queries precisely capture the class of Σ_2^p-recognizable queries.

As an immediate corollary, we obtain analogous results for seminormal DQL.

Corollary 1: The seminormal Boolean DQL queries (resp. I/O queries) capture the class Σ_2^p (resp. the class of Σ_2^p-recognizable queries).

We remark that Theorems 4 and 5 can be proved, though less instructive, by combining results in [37], [43] and recent results of some of the authors [15].

VI. Applications of DQL

In this section we will show how to write DQL queries solving some of the problems mentioned in Section II.

Example 1 (Electric circuit troubleshooting – continued) Rules on causes and effects of the kind (1) are stored in an appropriate relation.

Note that we can assume that all rules are of a uniform type

$$C_1 \wedge C_2 \rightarrow E_1 \vee E_2,$$

since we can encode rules with more than two causes and/or effects easily by a small number of such rules using new dummy facts. A conjunction $C_1 \wedge C_2$ (resp. disjunction $D_1 \vee D_2$) can be represented by a new fact F (resp. G), and the rules

$$\begin{array}{lll} C_1 \wedge C_2 \rightarrow F \vee \mathcal{F}, & \text{(resp.} & G \wedge \mathcal{T} \rightarrow D_1 \vee D_2, \\ F \wedge \mathcal{T} \rightarrow C_1 \vee \mathcal{F}, & & D_1 \wedge \mathcal{T} \rightarrow G \vee \mathcal{F}, \\ F \wedge \mathcal{T} \rightarrow C_2 \vee \mathcal{F} & & D_2 \wedge \mathcal{T} \rightarrow G \vee \mathcal{F}). \end{array}$$

Then, for instance the rule $C_1 \wedge C_2 \wedge C_3 \rightarrow D_1 \vee D_2 \vee D_3$ is rewritten to $F \wedge C_3 \rightarrow G \vee D_3$; it should be clear how other rules can be rewritten. Note that a general clause set can be transformed by this method into a uniform clause set in polynomial time.

It is assumed that symbols $\mathcal{T}$ and $\mathcal{F}$ represent the empty cause (which is intuitively always true) and the empty effect (which is always false), respectively, and that there are special relations TRUE(CAUSE) and FALSE(EFFECT) which contain always only the single tuple $\mathcal{T}$ and $\mathcal{F}$, respectively.

In our example, the following relation instances describe rules for causes and effects of the circuit in Figure 1:

TRUE

CAUSE
$\mathcal{T}$

FALSE

EFFECT
$\mathcal{F}$

RULE

CAUSE #1	CAUSE #2	EFFECT #1	EFFECT #2
B-high	$\mathcal{T}$	F_1-melts	F_2-melts
R_1-low	R_2-low	F_1-melts	F_2-melts
F_1-melts	$\mathcal{T}$	Light-off	$\mathcal{F}$
F_2-melts	$\mathcal{T}$	Light-off	$\mathcal{F}$
Bulb-broken	$\mathcal{T}$	Light-off	$\mathcal{F}$

The meaning of the first tuple in RULE is: whenever the battery's voltage is too high, one of the fuses or both of them might melt, even if the resistors are ok. The second tuple states that melting of fuses will occur whenever both resistors are low (cf. rule(2)). The other tuples state that a melted fuse or a broken bulb causes the control light to be off.

The set of possible causes are stored in another relation. As already mentioned, not necessarily all of the names cited in the first three columns of the above relation are possible causes.

HYPOTHESIS

FACT
B-high
R_2-low
Bulb-broken

This organization of knowledge is useful when one wants to decouple static knowledge (such as the knowledge on the circuit) and dynamic knowledge (such as knowing whether a specific device is malfunctioning).

The set of observations we are making in a specific situation is stored in a third relation, OBSERVATION(SYMPTOM).

OBSERVATION

SYMPTOM
Light-off

As already mentioned, an explanation is a minimal set S of facts (wrt inclusion), taken from the relation HYPOTHESIS, such that their validity implies from the causes-effects relationships in RULE all facts in the relation OBSERVATION, i.e., in every truth value assignment to the remaining facts which is compatible with all causes-effects relationships, the facts from OBSERVATION must be true.

In our example, the set {Bulb-broken} is an explanation: all of its elements occur in the HYPOTHESIS relation, and the fact Light-off is explained from the rule represented by the last tuple in RULE. Note that the set {Bulb-broken, R_2-low} is not an explanation, since it is not minimal wrt to inclusion. {B-high} is the only alternative explanation possible.

The set of explanations of a diagnosis instance can be computed by means of the following background knowledge B and open defaults D:

$$B \;=\; \{\, exp(x) \rightarrow hypothesis(x) \wedge t(x)\,, \quad exp'(x,y) \leftrightarrow exp(x) \wedge x \neq y \;\;,\;\; exp'(x,y) \rightarrow t'(x,y)\,\}\,;$$

$$D \;=\; \left\{ \frac{:exp(x)}{exp(x)} \;,\; \frac{rule(x,y,u,w):}{t(x)\wedge t(y)\rightarrow t(u)\vee t(w)} \;,\; \frac{:\neg exp(x)}{\neg exp(x)} \;,\; \frac{observation(x):\neg t(x)}{\perp} \;,\; \frac{rule(x,y,u,w):}{t'(x,z)\wedge t'(y,z)\rightarrow t'(u,z)\vee t'(w,z)} \;,\; \frac{observation(x):\neg t'(x,y)}{\neg remove(y)} \;,\; \frac{exp(y):remove(y)}{\perp} \right\}.$$

The intuitive meaning of $t(x)$ is "x is true", i.e., it can be proven using knowledge on the circuit and observations, and the intuitive meaning of $exp(x)$ is "x is part of the explanation". The other predicates $exp'(x,y)$, $t'(x,y)$, and $remove(x)$ are used for assuring the minimality of an explanation (we will explain this in more detail below).

The first and the second default in D represent a choice whether a fact F is considered to be in the explanation or out; the first formula in B represents the constraint that in the former case, F must appear in HYPOTHESIS and is considered to be true (by assumption). The third default in D clarifies how a fact can be inferred to be true using the diagnostic rules; the fourth default states that each fact F in OBSERVATION must be provable. (For if not, i.e., it is consistent to assume the negation $\neg F$, then we can infer a contradiction.)

The remaining part of the default theory serves for assuring that the explanation is indeed minimal wrt inclusion; it exploits the simple fact that a set of facts S entailing all observations is an explanation, if for any fact F from S, after removal of F no longer all observed facts are entailed (cf. [22] for the easy proof). Now, for each fact F, $exp'(x,F)$ represents by the second formula in B the set S described by exp after removal of F; the last formula in B states that the facts from this set are considered to be true. The fifth default in D produces a copy of the diagnostic rules for F, which is used to verify that, if F occurs in S, $S-\{F\}$ does not entail all observations. This test is implemented by the last two defaults in D: if F occurs in S, then $\neg remove(F)$ must be provable.

It can be shown that the extensions of the instantiated default theory correspond one-to-one to the explanations, such that fact F belongs to an explanation exactly if $exp(F)$ belongs to the corresponding default extension. Consequently, the credulous DQL I/O query with output relation EXP(FACT) computes all the facts that belong to at least one explanation of the diagnostic problem, i.e., the relevant facts. Notice that the analog cautious version of this DQL I/O query computes all facts *necessary* for an explanation, i.e., the facts that belong to every explanation.

As a final remark, we note that requiring minimality of explanations is not a source of complexity. By the results in [22], Σ_2^p-hardness of deciding relevance holds also if non-minimal explanations are admissible; in fact, a default query for this case would be even much simpler. □

Example 2 (Strategic companies –continued) A possible situation for products and companies is described by the following database instance:

PRODUCERS

PRODUCT	COMPANY #1	COMPANY #2
Pasta	Barilla	Saiwa
Tomatoes	Frutto	Barilla
Wine	Barilla	–
Bread	Saiwa	Panino

Here, the symbol "–" means that the entry is void (formally, we represent this by a special object in the domain, which is specified by a relation VOID(OBJECT)).

In such a situation, both Barilla and Saiwa are strategic, because {Barilla, Saiwa} produce all products, while neither {Barilla} nor {Saiwa} do that. On the other hand, Frutto is not strategic.

The following relation instance describes control of companies over other companies:

CONTROL

CONTROLLED	CONT #1	CONT #2	CONT #3
Frutto	Barilla	Saiwa	–

The meaning of the tuple is that companies Barilla and Saiwa together have control over Frutto.

Taking into account that Frutto can be sold only if either Barilla or Saiwa are also sold, the minimal sets of companies that produce all products are completely changed. As an example, {Barilla, Saiwa} is no longer such a set, while {Barilla, Saiwa, Frutto} is.

In the case there are no controlled companies, a manager can easily express a DQL Boolean query whose answer tells if a specific company C is strategic, by writing the set of open defaults D:

$$\frac{producers(x,y,z):}{strat(y)\vee strat(z)} \;,\; \frac{:\neg strat(x)}{\neg strat(x)} \;,\; \frac{void(x):}{\neg strat(x)},$$

and adding as ϕ the formula $strat(C)$ (B is empty). The intuitive meaning of the defaults is that for each product x at least one of the producers y,z are strategic companies, and that companies are non-strategic by default. The answer to the above Boolean query is `yes` precisely if the company C is strategic.

If we add the default

$$\frac{control(w,x,y,z) \wedge strat(x) \wedge strat(y) \wedge strat(z):}{strat(w)}$$

to D, then the Boolean query having $strat(C)$ as ϕ gives the desired answer for the case in which there are controlled companies.

As a final remark, we note that in this example we could allow unbounded numbers of producers for each product and controllers for each company, although the queries would get more involved. □

Example 3 (Maximal trust –continued) We can straightforwardly represent agreements in our database of companies by using a relation AGREE(COMP#1, COMP#2), where a tuple (Barilla, Saiwa) in the relation stands for an agreement between the companies Barilla and Saiwa.

The query: "which companies belong to a maximal trust?" (which computes an output relation MAX(COMP) listing the searched companies) is Δ_2^p-recognizable, where Δ_2^p is the class of problems decidable by a deterministic Turing machine in polynomial time with an oracle for an NP problem (cf. [23]). Thus, since $\Delta_2^p \subseteq \Sigma_2^p$, the query is Σ_2^p-recognizable, and can be expressed in DQL. In fact, a little extra work shows that the query is even $\Delta_2^p[O(\log n)]$-recognizable, where $\Delta_2^p[O(\log n)]$ is the subclass of Δ_2^p in which the number of oracle calls is bound by $O(\log n)$, where n is the size of the input (cf. [23]).

On the other hand, we have that the maximal trust query can not be expressed in $\text{DATALOG}^{\neg}_{stable}$ unless NP $= \Delta_2^p[O(\log n)]$, which is widely assumed to be false. Indeed, computing the maximal trust query is $\Delta_2^p[O(\log n)]$-hard, since deciding, given an instance of AGREE and a certain company C, whether C belongs to a maximal trust is an $\Delta_2^p[O(\log n)]$-hard problem; this follows trivially from the $\Delta_2^p[O(\log n)]$-hardness of deciding whether a given node belongs to some clique of maximum size in a graph [44]. □

VII. Conclusions

In this paper we have defined DQL, a query language for relational databases based on default logic. The expressiveness and complexity of DQL have been investigated both for boolean queries and for I/O queries. The results we have shown are not only of theoretical importance: We have presented queries which are useful in practice that can be handled with DQL, but can not with other query languages based on non-monotonic logics such as $\text{DATALOG}^{\neg}_{stable}$.

Let us compare and contrast DQL's expressive power in more detail with expressive power and computational paradigms for logic programs with stable semantics; the results are displayed in Figure 3.

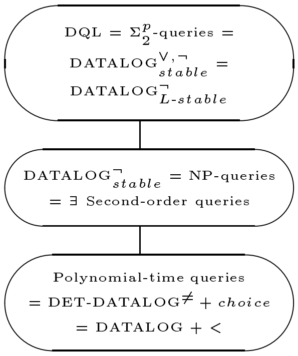

Fig. 3. Expressiveness of stable models (noncollapsing complexity classes)

The expressive power of $\text{DATALOG}^{\neg}_{stable}$ under credulous semantics is NP, i.e., precisely those queries can be formulated by general logic programs using stable models, whose complexity is in NP (this follows by the results in [4]). This expressiveness result has been preceded by complexity results for credulous reasoning from propositional logic programs, which is NP-complete for the stable semantics [37], [45]. These results also explain that it is possible to reduce the computation of stable models efficiently to linear programming, which has been shown as a promising approach for implementing stable model semantics [41]. On the other hand, the expressive power of DQL is Σ_2^p. This means that strictly more queries can be formulated, and that linear programming methods can not be efficiently applied for implementing DQL queries (unless complexity classes collapse that are strongly believed to be different). Disjunctive logic programs are endowed with the same expressive power as DQL: As shown in [15], the expressive power of disjunctive DATALOG programs using stable models ($\text{DATALOG}^{\vee,\neg}_{stable}$) is Σ_2^p. The same expressive power is also available with so called least undefined partial stable (L-stable) models [46] for normal logic programs ($\text{DATALOG}^{\neg}_{L-stable}$), which are a relaxation of the concept of stable model.

An alternative to allow negation in DATALOG in order to increase expressive power is the choice operator introduced in [47]. Roughly, the choice operator allows for atoms $choice(\mathbf{x}, \mathbf{y})$ in rule bodies, which choose precisely one instantiation of the $\mathbf{y}$ variables for each instantiation of the $\mathbf{x}$ variables. Since several possibilities exist for that in general, the language computes non-deterministic queries (i.e., several outputs are possible) rather than deterministic ones (which we consider here). It is shown in [48], the semantics of the choice operator is subsumed by stable models, and in [49], that $\text{DATALOG}^{\neq}$ augmented with dynamic choice expresses exactly the non-deterministic queries computable in polynomial time. The deterministic fragment of this language (DET-$\text{DATALOG}^{\neq}$ + *choice*) expresses precisely all queries computable in polynomial time (cf.[50]); however, it can not be recognized. Notice that it is conjectured that no query language for precisely the polynomial time queries exists (see [29]).

We conclude by outlining possible issues for future work on DQL. In the definition of query –Section IV-A– open defaults are quantifier and function-free. One direction is a generalization of DQL by allowing first-order formulas with quantifiers and/or functions. Another direction is identifying (more) tractable fragments of DQL, in particular fragments which resemble a subclass of the polynomial-time computable queries, and determining their expressive powers. A good starting point for this project is [38], where several important tractable fragments of default logic have been identified.

Appendix

Theorem 1 *The Boolean DQL queries precisely capture the class Σ_2^p.*

PROOF The easy part is to show that every Boolean DQL can be expressed as a Σ_2^p-recognizable Boolean query. To this end, it is sufficient to prove that the data complexity of DQL (i.e., regarding the query as being fixed) is in Σ_2^p. We notice that the semantics of DQL given in Section IV-B transforms query answering into credulous reasoning in a propositional default theory. The transformation is clearly polynomial in the size of the database. It has been proven in [12], [13] that the problem of credulous inference in propositional default theories is in Σ_2^p; hence, this part of the proof is complete.

The more difficult part is to show that each Boolean query expressible as a sentence of $SO_{\exists\forall}$ can be expressed in DQL. As we already noticed in the introduction, we can not take advantage of the fact that propositional credulous default reasoning is Σ_2^p-hard, because the expressiveness of a query language is not necessarily the same as its complexity.

Without loss of generality, we assume that sentence (3) is of form

$$(\exists\mathbf{S})(\forall\mathbf{T})(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y}), \qquad (4)$$

where $\mathbf{S}$, $\mathbf{T}$ are lists of predicate symbols, $\mathbf{x},\mathbf{y}$ are lists of individual variables, and ψ is a first order formula in which no function symbol or quantifier occurs. The pass from (3) to (4), which is well-known in Logic, is justified in the report [51].

Now we have to show that for each query $Q_{SO_{\exists\forall}}$ of the form (4) there is a DQL query Q_{DQL} such that the two queries give the same answer on all possible database instances W over the unquantified relations in (4).

We outline the idea for Q_{DQL}. The formula $(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$ is encoded as follows. We use a predicate $<$ that defines a linear order on the set of all $\mathbf{y}$-tuples, together with associated predicates $F(\mathbf{y})$, $S(\mathbf{y},\mathbf{y}')$, $L(\mathbf{y})$ which state that $\mathbf{y}$ is the first tuple, $\mathbf{y}'$ is the successor of $\mathbf{y}$, and $\mathbf{y}$ is the last tuple in $<$, respectively. Furthermore, we use a predicate $Z(\mathbf{x},\mathbf{y})$ which intuitively states that for each tuple $\mathbf{y}'$ from the initial segment of $<$ up to $\mathbf{y}$, the formula $\psi(\mathbf{x},\mathbf{y}')$ is true. A designated propositional letter A indicates if for some $\mathbf{x}$-tuple $\mathbf{a}$, $Z(\mathbf{a},\mathbf{b})$ is true, where $\mathbf{b}$ is the last $\mathbf{y}$-tuple. Then, $(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$ will be true just in case A is derivable. We encode this by default rules, such that in every extension that contains A, a valuation for the $\mathbf{S}$-predicates is defined and for every valuation of the $\mathbf{T}$-predicates some $Z(\mathbf{a},\mathbf{b})$ is true, i.e., the sentence (4) is true over the underlying database W.

Formally, Q_{DQL} consists of the ground formula A and the pair (B,D) as follows. The background knowledge B contains a set of axioms which state that $<$ is a linear order, i.e.,

$$B = \{\neg(\mathbf{y} < \mathbf{y}' \wedge \mathbf{y}' < \mathbf{y}), (\mathbf{y} < \mathbf{y}') \vee (\mathbf{y}' < \mathbf{y}) \vee (\mathbf{y} = \mathbf{y}'), (\mathbf{y} < \mathbf{y}'' \wedge \mathbf{y}'' < \mathbf{y}') \rightarrow \mathbf{y} < \mathbf{y}'\}.$$

(Equality between tuples $\mathbf{y} = (y_1,\ldots,y_m)$ and $\mathbf{y}' = (y_1,\ldots,y'_m)$, i.e., $(\mathbf{y} = \mathbf{y}')$, is expressed by $(\,(y_1 = y'_1) \wedge \cdots \wedge (y_m = y'_m)\,)$.

The set D contains the following open defaults.

- For each predicate P from $\mathbf{S}$ and for $<$:

$$\frac{:P(\mathbf{x})}{P(\mathbf{x})}, \quad \frac{:\neg P(\mathbf{x})}{\neg P(\mathbf{x})}, \quad \frac{:\mathbf{y} < \mathbf{y}'}{\mathbf{y} < \mathbf{y}'}, \quad \frac{:\neg(\mathbf{y} < \mathbf{y}')}{\neg(\mathbf{y} < \mathbf{y}')};$$

- rules for the associated predicates:

$$\frac{:\ F(\mathbf{y})}{F(\mathbf{y})}, \frac{:\ L(\mathbf{y})}{L(\mathbf{y})}, \frac{:\ S(\mathbf{y},\mathbf{y}')}{S(\mathbf{y},\mathbf{y}')},$$

$$\frac{\mathbf{y} < \mathbf{y}' :}{\neg F(\mathbf{y}')}, \quad \frac{\mathbf{y} < \mathbf{y}' :}{\neg L(\mathbf{y})}, \quad \frac{\neg(\mathbf{y} < \mathbf{y}') \vee (\mathbf{y} < \mathbf{y}'' \wedge \mathbf{y}'' < \mathbf{y}') :}{\neg S(\mathbf{y},\mathbf{y}')};$$

- derivation of A (i.e., checking if $(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$ is true):

$$\frac{:F(\mathbf{y})}{\psi(\mathbf{x},\mathbf{y}) \rightarrow Z(\mathbf{x},\mathbf{y})}, \quad \frac{:L(\mathbf{y})}{Z(\mathbf{x},\mathbf{y}) \rightarrow A},$$

$$\frac{:S(\mathbf{y},\mathbf{y}')}{Z(\mathbf{x},\mathbf{y}) \wedge \psi(\mathbf{x},\mathbf{y}') \rightarrow Z(\mathbf{x},\mathbf{y}')}.$$

Let W be any database instance, and denote by Δ the default theory $Q_{DQL} + W$.

It is easy to see that Δ has only consistent extensions. (Formally, this can be easily proved by Lemma 1.)

We claim that the atom A belongs to an extension of Δ if and only if $W \models (\exists\mathbf{S})(\forall\mathbf{T})(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$.

"$\Leftarrow$". Assume that $W \models (\exists\mathbf{S})(\forall\mathbf{T})(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$. That is,

$$W, \mathbf{S}_0 \models (\forall\mathbf{T})(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y}) \qquad (5)$$

for some valuation $\mathbf{S}_0$ of the $\mathbf{S}$ predicates. Define a set $\mathcal{F}$ of formulas as follows. Let $<_0$ be an arbitrary linear order of all $\mathbf{y}$-tuples, and let F_0, S_0, L_0 be the associated extensions for F, S, and L. In what follows, let $\mathbf{a},\mathbf{a}',\mathbf{b}$ denote tuples of appropriate arities over the domain. The set $\mathcal{F} = \bigcup_{i=1}^{10}\mathcal{F}_i$ contains the following formulas:

$$\begin{aligned}
\mathcal{F}_1 &= COMP(W) \cup INST(B)\\
\mathcal{F}_2 &= \{\mathbf{a} < \mathbf{a}' \mid (\mathbf{a},\mathbf{a}') \in <_0\} \cup \{\neg(\mathbf{a} < \mathbf{a}') \mid (\mathbf{a},\mathbf{a}') \notin <_0\}\\
\mathcal{F}_3 &= \{\neg L(\mathbf{a}) \mid \mathbf{a} \notin L_0\}\\
\mathcal{F}_4 &= \{\neg F(\mathbf{a}) \mid \mathbf{a} \notin F_0\}\\
\mathcal{F}_5 &= \{\neg S(\mathbf{a},\mathbf{a}') \mid (\mathbf{a},\mathbf{a}') \notin S_0\}\\
\mathcal{F}_6 &= \{L(\mathbf{a}) \mid \mathbf{a} \in L_0\} \cup \{F(\mathbf{a}) \mid \mathbf{a} \in F_0\} \cup\\
&\quad\ \{S(\mathbf{a},\mathbf{a}') \mid (\mathbf{a},\mathbf{a}') \in S_0\}\\
\mathcal{F}_7 &= \{P(\mathbf{a}) \mid P \in \mathbf{S},\, \mathbf{a} \in P_0\} \cup\\
&\quad\ \{\neg P(\mathbf{a}) \mid P \in \mathbf{S},\, \mathbf{a} \notin P_0\}\\
\mathcal{F}_8 &= \{\psi(\mathbf{a},\mathbf{b}) \rightarrow Z(\mathbf{a},\mathbf{b}) \mid \mathbf{b} \in F_0\}\\
\mathcal{F}_9 &= \{Z(\mathbf{a},\mathbf{b}) \wedge \psi(\mathbf{a},\mathbf{b}') \rightarrow Z(\mathbf{a},\mathbf{b}') \mid (\mathbf{b},\mathbf{b}') \in S_0\}\\
\mathcal{F}_{10} &= \{Z(\mathbf{a},\mathbf{b}) \rightarrow A \mid \mathbf{b} \in L_0\}
\end{aligned}$$

Claim: $E = Cons(\mathcal{F})$ is an extension of Δ such that $A \in E$. To show this, we first note that E is consistent. Indeed, extend the valuations given by W, $\mathbf{S}_0$, $<_0$, F_0, S_0, L_0 by assigning "*true*" to A, by letting Z_0 be the set of all possible tuples, and by letting the remaining predicates $\mathbf{T}$ have an arbitrary extension. This valuation satisfies $\mathcal{F}$. Hence, $E = Cons(\mathcal{F})$ is consistent.

Moreover, E is an extension of Δ. This can be easily shown from the definition of extension: we obtain that

$$\begin{aligned}
E_0 &= COMP(W) \cup INST(B) = \mathcal{F}_1 \\
E_1 &= Cons(E_0) \cup \mathcal{F}_2 \cup \mathcal{F}_6 \cup \mathcal{F}_7 \cup \mathcal{F}_8 \cup \mathcal{F}_9 \cup \mathcal{F}_{10} \\
E_2 &= Cons(E_1) \cup \mathcal{F}_3 \cup \mathcal{F}_4 \cup \mathcal{F}_5 \\
E_3 &= Cons(E_2) = Cons(\mathcal{F}) \\
E_4 &= E_3 \\
&\vdots
\end{aligned}$$

Hence, $\bigcup_{i=0}^{\infty} E_i = Cons(\mathcal{F}) = E$. Thus, E is an extension of Δ.

To prove the claim, it remains to show that $A \in E$. Since $E = Cons(\mathcal{F})$, it suffices to show that A is true in every valuation that satisfies $\mathcal{F}$. Consider an arbitrary valuation that satisfies $\mathcal{F}$. Let $\mathbf{T}_0$ be the valuation of $\mathbf{T}$ and Z_0 the valuation of Z. Then, from (5) we have that

$$W, \mathbf{S}_0, \mathbf{T}_0 \models (\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y})$$

Let $\mathbf{a}$ be an arbitrary tuple such that

$$W, \mathbf{S}_0, \mathbf{T}_0 \models (\forall \mathbf{y})\psi(\mathbf{a}, \mathbf{y}) \tag{6}$$

By finite induction on $<_0$, we show that $(\mathbf{a}, \mathbf{b}) \in Z_0$, for all $\mathbf{b}$.

(Basis). Let $\mathbf{b}_0$ be the first tuple of $<_0$. Since $\mathbf{b}_0 \in F_0$, we have $\psi(\mathbf{a}, \mathbf{b}_0) \to Z(\mathbf{a}, \mathbf{b}_0) \in \mathcal{F}$; hence, by (6), $\psi(\mathbf{a}, \mathbf{b}_0)$ is true. Thus, $(\mathbf{a}, \mathbf{b}_0) \in Z_0$.

(Induction). Assume the statement holds for tuple $\mathbf{b}$. We show that it holds also for $\mathbf{b}'$, where $(\mathbf{b}, \mathbf{b}') \in S_0$. We have that $Z(\mathbf{a}, \mathbf{b}) \wedge \psi(\mathbf{a}, \mathbf{b}') \to Z(\mathbf{a}, \mathbf{b}') \in \mathcal{F}$; by the induction hypothesis, $(\mathbf{a}, \mathbf{b}) \in Z_0$, and by (6), $\psi(\mathbf{a}, \mathbf{b}')$ is true. Hence, it follows that $Z(\mathbf{a}, \mathbf{b}')$ is true, i.e., $(\mathbf{a}, \mathbf{b}') \in Z_0$. This shows the induction case.

Since $(\mathbf{a}, \mathbf{b}) \in Z_0$ for all $\mathbf{b}$, in particular we have that $(\mathbf{a}, \mathbf{b}_1) \in Z_0$ where $\mathbf{b}_1$ is the last tuple in $<_0$, i.e., $\mathbf{b}_1 \in L_0$. However, we have that $Z(\mathbf{a}, \mathbf{b}_1) \to A \in \mathcal{F}$; thus, it follows that A has value "*true*" in the valuation. Consequently, we have shown that $A \in E$.

This concludes the "$\Leftarrow$" part of the proof.

"$\Rightarrow$": Let E be an extension of Δ such that $A \in E$. Notice that E is consistent.

E defines a valuation $\mathbf{S}_0$ for the $\mathbf{S}$ predicates, i.e., for each P from $\mathbf{S}$, we have $P(\mathbf{a}) \in E$ or $\neg P(\mathbf{a}) \in E$ for each tuple $\mathbf{a}$. This follows since the defaults $\frac{:P(\mathbf{a})}{P(\mathbf{a})}$, $\frac{:\neg P(\mathbf{a})}{\neg P(\mathbf{a})}$ are in $INST(D)$. Moreover, E defines a valuation $<_0$ for $<$ such that $<_0$ satisfies the axioms for a linear order of all tuples of the arity of $\mathbf{y}$.

Furthermore, it follows from the instantiated default rules for the predicates associated to $<$ that E defines a total valuation of F, S, and L, and that $\neg F(\mathbf{a}) \in E$ iff $\mathbf{a}$ is not the first tuple in $<_0$, that $\neg S(\mathbf{a}, \mathbf{a}') \in E$ iff $\mathbf{a}'$ is not the successor of $\mathbf{a}'$ in $<_0$, and that $\neg L(\mathbf{a}) \in E$ iff $\mathbf{a}$ is not the last tuple in $<_0$.

We claim that

$$W, \mathbf{S}_0 \models (\forall \mathbf{T})(\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y}) \tag{7}$$

To prove this, assume this is false. Hence, there exists a valuation $\mathbf{T}_0$ of $\mathbf{T}$ such that

$$W, \mathbf{S}_0, \mathbf{T}_0 \models (\forall \mathbf{x})(\exists \mathbf{y})\neg\psi(\mathbf{x}, \mathbf{y})$$

We extend this valuation to a model of E that has A false. Let for each tuple $\mathbf{a}$ for $\mathbf{x}$ be $m(\mathbf{a})$ the first tuple $\mathbf{b}$ for $\mathbf{y}$ in $<_0$ such that $W, \mathbf{S}_0, \mathbf{T}_0 \models \neg\psi(\mathbf{a}, \mathbf{b})$. Notice that $m(\mathbf{a})$ exists for each $\mathbf{a}$.

Define a valuation Z_0 for Z by $Z_0 = \{ (\mathbf{a}, \mathbf{b}) \mid \mathbf{b} <_0 m(\mathbf{a}) \}$, and assign A the value "*false*".

This valuation of the predicates defines a model of E. To prove this, it is by Lemma 1 sufficient to show that the valuation is a model of

$$COMP(W) \cup INST(B) \cup G,$$

where $G = \{\gamma \mid \alpha : \beta / \gamma \in GD(E, \Delta)\}$. Since $COMP(W) \cup INST(B) = \mathcal{F}_1$ and $\mathcal{F}_1 \subseteq E$, we are left to show that G is satisfied. It is easy to see that every γ in which Z does not occur is satisfied. Consider any remaining γ (i.e., a conclusion of a default for deriving A). Then, we have three cases:

(*a*) γ is $\psi(\mathbf{a}, \mathbf{b}) \to Z(\mathbf{a}, \mathbf{b})$. Then, $\mathbf{b}$ is the first tuple in $<_0$, and the formula is certainly satisfied.

(*b*) γ is $Z(\mathbf{a}, \mathbf{b}) \wedge \Psi(\mathbf{a}, \mathbf{b}') \to Z(\mathbf{a}, \mathbf{b}')$. Then, $\mathbf{b}'$ is the successor of $\mathbf{b}$ in $<_0$. By considering the three cases (1) $\mathbf{b}, \mathbf{b}' <_0 m(\mathbf{a})$, (2) $\mathbf{b}' = m(\mathbf{a}$, and (3) $m(\mathbf{a}) <_0 \mathbf{b}, \mathbf{b}'$ it is readily checked from the definition of Z_0 that the formula is satisfied.

(*c*) γ is $Z(\mathbf{a}, \mathbf{b}) \to A$. Then, $\mathbf{b}$ is the last tuple in $<_0$. By definition of Z_0, we have that $Z(\mathbf{a}, \mathbf{b})$ is false; hence, since A is false, the formula is satisfied.

Thus, the valuation satisfies $COMP(W) \cup INST(B) \cup G$, and hence also E. Since A is false in this model of E, we have that $A \notin E$. This is a contradiction, however. Hence, claim (7) is proved. Now claim (7) means

$$W \models (\exists \mathbf{S})(\forall \mathbf{T})(\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y}).$$

This concludes the "$\Rightarrow$" part of the proof.

Remark: All defaults in D can be made prerequisite-free by deleting the prerequisite α and rewriting the conclusion γ as $\alpha \to \gamma$. □

Theorem 2 *The data complexity of Boolean DQL is Σ_2^p-complete.*

PROOF From Theorem 1, we have that every Boolean DQL query defines a Σ_2^p database property; this gives the membership part. For the hardness part, we notice that for every $SO_{\exists\forall}$ sentence as in (3), an equivalent sentence of the form (4) in the proof of Theorem 1 can be constructed (cf. Appendix), for which the equivalent default query Q_{DQL} can be easily constructed (even in polynomial time). Consequently, the problem of deciding whether a *fixed* sentence (3) is valid in a given database instance W, which is Σ_2^p-hard, is transformable to a Boolean DQL query in polynomial time. $\square$

Theorem 3 *A database query is Σ_2^p-recognizable if and only if it is definable as a DQL I/O query.*

PROOF "$\Leftarrow$": Deciding whether a tuple $\mathbf{t}$ belongs to a certain output relation $S_i \in S$ transforms in polynomial time into credulous reasoning in a propositional default theory (cf. Proof of Theorem 1). Hence, Σ_2^p-recognizability follows.

"$\Rightarrow$": Let $q : R \rightarrow S$ be a Σ_2^p-recognizable query. We define from q a database property $\mathcal{P}$ on the relation scheme $R' = R \cup S$ in the following way: For each instance W of R', $\mathcal{P}(W)$ is true if and only if $S_i|W \subseteq S_i|q(W_R)$ for each $S_i \in S$, where W_R denotes the restriction of W to the relations of R. That is, $\mathcal{P}(W)$ is true iff each tuple $\mathbf{t}$ that belongs to $S_i|W$ is computed by q on the instance W_R of R.

Notice that $\mathcal{P}$ is in fact a database property. Moreover, it is easy to see that $\mathcal{P}$ is Σ_2^p-recognizable. Consequently, by Theorem 1, $\mathcal{P}$ can be expressed by a Boolean DQL query $Q^{\mathcal{P}}_{DQL}$. Without loss of generality, we may assume that $Q^{\mathcal{P}}_{DQL}$ has the form of the query Q_{DQL} constructed in the proof of Theorem 1; notice that the database relations of $Q^{\mathcal{P}}_{DQL}$ are $R \cup S$. From $Q^{\mathcal{P}}_{DQL}$ we construct a DQL I/O query Q^{q}_{DQL} as follows. The background knowledge is the one of $Q^{\mathcal{P}}_{DQL}$, i.e. B, and the open defaults D' are defined by

$$D' = D \cup \left\{ \frac{:S_i(\mathbf{x})}{S_i(\mathbf{x})}, \frac{:\neg S_i(\mathbf{x})}{\neg S_i(\mathbf{x})} \mid S_i \in S \right\} \cup \left\{ \frac{:\ \neg A}{\bot} \right\}$$

The database relations are given by R and the output relations by S.

The rules $\frac{:S_i(\mathbf{x})}{S_i(\mathbf{x})}, \frac{:\neg S_i(\mathbf{x})}{\neg S_i(\mathbf{x})}$ enforce a total valuation of the S_i predicates. Intuitively, these defaults simulate all extensions of an instance W of R to an instance W' of $R' = R \cup S$, which is then processed by the defaults of query $Q^{\mathcal{P}}_{DQL}$. The default $\frac{:\neg A}{\bot}$ cutts off all default extensions in which the atom A is not contained; hence, only default extensions $E_{W'}$ for W' are left such that $\mathcal{P}(W')$ is true. The query Q^{q}_{DQL} collects under the brave semantics all tuples $\mathbf{t}$ for S_i such that $S_i(\mathbf{t})$ belongs to some default extension $E_{W'}$; from the definition of $\mathcal{P}$, this means that exactly those tuples $\mathbf{t}$ are collected that q computes for S_i on the database W. In other words, Q^{q}_{DQL} computes on database W the same as q, i.e., Q^{q}_{DQL} defines q in DQL.

More formally, we show the following. Let W be an instance of R, and define the set of formulas $\mathcal{G}$ by

$$\begin{aligned} \mathcal{G} \;=\; & \mathcal{F} \cup \{ S_i(\mathbf{t}) \mid \mathbf{t} \in S_i|q(W),\, S_i \in S \} \cup \\ & \{ \neg S_i(\mathbf{t}) \mid \mathbf{t} \notin S_i|q(W)\, S_i \in S \}, \end{aligned}$$

where $\mathcal{F}$ is as in the proof of Theorem 1. Let W' be the extension of W to $R' = R \cup S$ such that $S_i|W' = S_i|q(W)$, for $S_i \in S$. Then $\mathcal{P}(W')$ is true. Therefore, $E = Cons(\mathcal{G})$ is an extension of $Q^{\mathcal{P}}_{DQL} + W'$ such that $A \in E$. Hence, it is easy to see from the definition of extension that E is also an extension of $Q^{q}_{DQL} + W$. It follows that

$$S_i|q(W) \subseteq S_i|Q^{q}_{DQL}(W), \quad \text{for } S_i \in S.$$

To show that Q^{q}_{DQL} and q compute the same on W, it thus remains to show that

$$S_i|Q^{q}_{DQL}(W) \subseteq S_i|q(W), \quad \text{for } S_i \in S.$$

Let E be an extension of $Q^{q}_{DQL} + W$. Notice that $A \in E$. Then, it is easy to see that E is also an extension of $Q^{\mathcal{P}}_{DQL} + W'$, where W' is the extension of W to R' such that $\mathbf{t} \in S_i|W'$ iff $S_i(\mathbf{t}) \in E$. Consequently, $\mathcal{P}(W')$ is true. This means that

$$\{\mathbf{t} \mid S_i(\mathbf{t}) \in E\} \subseteq S_i|q(W), \quad \text{for } S_i \in S.$$

It follows that $S_i|Q^{q}_{DQL}(W) \subseteq S_i|q(W)$.

Thus we have shown that Q^{q}_{DQL} defines q, which concludes the "$\Rightarrow$" part of the proof. This proves the theorem. $\square$

Theorem 4 *The Boolean normal DQL queries precisely capture the class Σ_2^p.*

PROOF Reconsider the Boolean DQL query Q_{DQL} constructed in the proof of Theorem 1. Let Q'_{DQL} be the DQL query resulting from Q_{DQL} by replacing (B, D) with (B', D'), where D' and B' are obtained from D and B, respectively, by removing every non-normal default $\frac{\alpha :}{\gamma}$ for the associated predicates and by adding $\alpha \rightarrow \gamma$ to B.

Notice that each α is a Boolean formula built on atoms from $<$; since the defaults $\frac{:\mathbf{y}<\mathbf{y}'}{\mathbf{y}<\mathbf{y}'}$ and $\frac{:\neg(\mathbf{y}<\mathbf{y}')}{\neg(\mathbf{y}<\mathbf{y}')}$ belong to both D and D', it is easy to see that the default theories $Q_{DQL} + W$ and $Q'_{DQL} + W$ have the same extensions for every instance W, and thus express the same query.

Now let D'' be the result of replacing in D' every non-normal default, i.e. rule $\frac{:\beta}{\gamma}$ for deriving A, by $\frac{:\beta\wedge\gamma}{\beta\wedge\gamma}$, and let Q''_{DQL} be the query obtained from Q'_{DQL} by replacing D' with D''. Notice that Q''_{DQL} is a normal DQL query.

One can easily show that, for every database instance W of R, a set E is an extension of $Q''_{DQL} + W$ if and only if E is an extension of $Q'_{DQL} + W$.

Hence, it follows that $Q''_{DQL} + W$ has an extension E such that $A \in E$ iff $W \models (\exists\mathbf{S})(\forall\mathbf{T})(\exists\mathbf{x})(\forall\mathbf{y})\psi(\mathbf{x},\mathbf{y})$. The result follows. $\square$

Theorem 5 *The normal I/O DQL queries precisely capture the class of Σ_2^p-recognizable queries.*

PROOF We show this by a modification of the construction in the proof of Theorem 3.

Without loss of generality, we assume that the Boolean DQL query $Q^{\mathcal{P}}_{DQL}$ for expressing $\mathcal{P}$ has the form of the query Q''_{DQL} in Theorem 4 (instead of Q_{DQL} in Theorem 1). Then, by analogous lines as in the proof Theorem 3, we obtain that the I/O query Q^{q}_{DQL} constructed from $Q^{\mathcal{P}}_{DQL}$ by adding the defaults $\frac{:S_i(\mathbf{X})}{S_i(\mathbf{X})}$, $\frac{:\neg S_i(\mathbf{X})}{\neg S_i(\mathbf{X})}$, $S_i \in S$, and $\frac{:A}{\perp}$ expresses the query q.

Now let Q be the query obtained from Q^q_{DQL} by replacing each occurrence of every $S_i \in S$ by S'_i, where S'_i is a new predicate of the same arity, and by replacing the default $\frac{:\neg A}{\perp}$ with the defaults

$$\frac{A \ : \ S_i(\mathbf{x}) \leftrightarrow S'_i(\mathbf{x})}{S_i(\mathbf{x}) \leftrightarrow S'_i(\mathbf{x})}, \qquad S_i \in S.$$

Let W be an instance of R. Notice that A belongs to every extension of $Q^q_{DQL} + W$, and that an atom $S_i(\mathbf{t})$ belongs to an extension E of $Q + W$ only if A belongs to E. Thus, it is easy to see that a set E is an extension of $Q^q_{DQL} + W$ only if

$$E' = E \cup \{\, S'_i(\mathbf{t}_0),\, \neg S'_i(\mathbf{t}_1) \mid S_i(\mathbf{t}_0),\, \neg S_i(\mathbf{t}_1) \in E,\, S_i \in S \,\}$$

is an extension of $Q + W$. On the other hand, if E is an extension of $Q + W$ such that $A \in E$, then

$$E' = E - \{\, S'_i(\mathbf{t}),\, \neg S'_i(\mathbf{t}) \mid S_i \in S \,\}$$

is an extension of $Q^q_{DQL} + W$. Thus, it follows that Q computes on W the same as Q^q_{DQL}.

Consequently, Q and Q^q_{DQL} represent the same query. Since Q is a normal DQL I/O query, the result follows. $\square$

Acknowledgements

The authors are grateful to Torsten Schaub for interesting comments on semantics of DQL. They also appreciate the very useful comments of the anonymous reviewers, which helped to improve the reading of the paper. The first author has been partially supported by ESPRIT Basic Research Action 6810 COMPULOG II and by *Progetto Finalizzato Informatica* of the CNR (Italian Research Council). The second and third authors have been partially supported by the *Christian Doppler Laboratory for Expert Systems*.

References

[1] M. Cadoli and M. Schaerf, "A Survey of Complexity Results for Non-monotonic Logics", *Journal of Logic Programming*, vol. 17, pp. 127–160, 1993.

[2] P. Kanellakis, "Elements of Relational Database Theory", in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B, chapter 17. Elsevier Science Publishers B.V. (North-Holland), 1990.

[3] Ph.G. Kolaitis and Ch. H. Papadimitriou, "Why Not Negation By Fixpoint ?", *Journal of Computer and System Sciences*, vol. 43, pp. 125–144, 1991.

[4] J.S. Schlipf, "The Expressive Powers of Logic Programming Semantics", Tech. Rep. CIS-TR-90-3, Computer Science Department, University of Cincinnati, 1990, Preliminary version in *Proceedings PODS-90*, pp. 196–204. To appear in *Journal of Computer and System Sciences*.

[5] M. Gelfond and V. Lifschitz, "The Stable Model Semantics for Logic Programming", in *Proceedings Fifth Logic Programming Symposium*, Cambridge Mass., 1988, pp. 1070–1080, MIT Press.

[6] N. Bidoit and C. Froidevaux, "Minimalism Subsumes Default Logic and Circumscription in Stratified Logic Programming", in *Proceedings LICS-87*. 1987, pp. 89–97, IEEE Computer Science Press.

[7] N. Bidoit and Ch. Froidevaux, "General Logic Databases and Programs: Default Semantics and Stratification", *Information and Computation*, vol. 19, pp. 15–54, 1991.

[8] D. Saccà, "Multiple Stable Models are Needed to Solve Unique Solution Problems", in *Informal Proceedings of the Second Compulog Net Meeting on Knowledge Bases (CNKBS-93)*, Athens, April 1993.

[9] A. Van Gelder, "The Alternating Fixpoint of Logic Programs With Negation", in *Proceedings PODS-89*, 1989, pp. 1–10.

[10] A. van Gelder, K.A. Ross, and J.S. Schlipf, "The Well-Founded Semantics for General Logic Programs", *Journal of the ACM*, vol. 38, no. 3, pp. 620–650, 1991.

[11] R. Reiter, "A Logic for Default Reasoning", *Artificial Intelligence*, vol. 13, pp. 81–132, 1980.

[12] Georg Gottlob, "Complexity Results for Nonmonotonic Logics", *Journal of Logic and Computation*, vol. 2, no. 3, pp. 397–425, June 1992.

[13] J. Stillman, "The Complexity of Propositional Default Logic", in *Proceedings AAAI-92*, 1992, pp. 794–799.

[14] S. Abiteboul and V. Vianu, "Expressive Power of Query Languages", in *Theoretical Studies in Computer Science*, J. Ullman, Ed. Academic Press, 1992, Festschrift in Honour of Seymour Ginsburg's 64th Birthday.

[15] T. Eiter, G. Gottlob, and H. Mannila, "Adding Disjunction to Datalog", in *Proceedings of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, May 1994, pp. 267–278.

[16] I. Stewart, "Comparing the Expressibility of Languages Formed Using NP-Complete Operators", *Journal of Logic and Computation*, vol. 1, no. 3, pp. 305–330, 1991.

[17] D. Poole, "Normality and Faults in Logic Based Diagnosis", in *Proceedings IJCAI-89*, 1989, pp. 1304–1310.

[18] L. Console, D. Theseider Dupré, and P. Torasso, "On the Relationship Between Abduction and Deduction", *Journal of Logic and Computation*, vol. 1, no. 5, pp. 661–690, 1991.

[19] Georg Gottlob and Wolfgang Nejdl, Eds., *Proceedings International Workshop on Expert Systems in Engineering*, number 462 in Lecture notes in AI, September 1990.

[20] Walter Hamscher, Ed., *Readings in Model-Based Diagnosis*, Morgan Kaufman Publ., 1992.

[21] Y. Peng and J.A. Reggia, *Abductive Inference Models for Diagnostic Problem Solving*, Symbolic Computation – Artificial Intelligence. Springer, 1990.

[22] Thomas Eiter and Georg Gottlob, "The Complexity of Logic-Based Abduction", *Journal of the ACM*, vol. 42, no. 1, pp. 3–42, January 1995.

[23] David S. Johnson, "A Catalog of Complexity Classes", in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. A, chapter 2. Elsevier Science Publishers B.V. (North-Holland), 1990.

[24] Marco Cadoli and Maurizio Lenzerini, "The Complexity of Propositional Closed World Reasoning and Circumscription", *Journal of Computer and System Sciences*, vol. 43, pp. 165–211, April 1994.

[25] T. Eiter and G. Gottlob, "Complexity Aspects of Various Semantics for Disjunctive Databases", in *Proceedings of the Twelth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-93)*, June 1993, pp. 158–167.

[26] J. D. Ullman, *Principles of Database and Knowledge Base Systems*, vol. 1, Computer Science Press, 1989.

[27] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[28] A.V. Aho and J.D. Ullman, "Universality of Data Retrieval Languages", in *Proceedings ACM Symposium on Principles of Programming Languages*, 1979, pp. 110–117.

[29] Y. Gurevich, "Logic and the Challenge of Computer Science", in *Trends in Theoretical Computer Science*, E. Börger, Ed., chapter 1. Computer Science Press, 1988.

[30] R. Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets", in *Complexity of Computation*, R. M. Karp, Ed. 1974, pp. 43–74, AMS.

[31] Neil Immerman, "Languages that Capture Complexity Classes", *SIAM J. Comp.*, vol. 16, pp. 760–778, 1987.

[32] Ashok K. Chandra, "Theory of Database Queries", in *Proceedings PODS-88*, 1988.

[33] C. H. Papadimitriou, "A Note on the Expressive Power of Datalog", *Bulletin of the EATCS*, vol. 26, pp. 21–23, 1985.

[34] L. J. Stockmeyer, "The Polynomial-Time Hierarchy", *Theoretical Computer Science*, vol. 3, pp. 1–22, 1977.

[35] J.F. Lynch, "Complexity Classes and Theories of Finite Models", *Mathematical Systems Theory*, vol. 15, pp. 127–144, 1982.

[36] W. Marek and M. Truszczyński, *Nonmonotonic Logics – Context-Dependent Reasoning*, Springer, 1993.

[37] N. Bidoit and Ch. Froidevaux, "Negation by Default and Unstratifiable Programs", *Theoretical Computer Science*, vol. 78, pp. 85–112, 1991.

[38] H. Kautz and B. Selman, "Hard Problems for Simple Default Logics", *Artificial Intelligence*, vol. 49, pp. 243–279, 1991.

[39] R. Reiter, "Towards a Reconstruction of Logical Databases", in *On Conceptual Modelling*, J.W. Schmidt M.L. Brodie, J.L. Mylopoulos, Ed., New York, 1984, pp. 163–189, Springer.

[40] Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[41] C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian, "Implementing Stable Semantics by Linear Programming", *Journal of the ACM*, vol. 42, no. 1, January 1995.

[42] W. Lukasiewicz, *Non-Monotonic Reasoning*, Ellis Horwood Limited, Chichester, England, 1990.

[43] Ch. Sakama and K. Inoue, "Relating Disjunctive Logic Programs to Default Theories", in *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)*, Luìs-Moniz Pereira and Anil Nerode, Eds., Lisbon, Portugal, July 1993, pp. 266–282, MIT Press.

[44] Klaus Wagner, "Bounded Query Classes", *SIAM J. Comp.*, vol. 19, no. 5, pp. 833–846, 1990.

[45] W. Marek and M. Truszczyński, "Autoepistemic Logic", *Journal of the ACM*, vol. 38, no. 3, pp. 588–619, 1991.

[46] Domenico Saccá, "The Expressive Powers of Stable Models for Bound and Unbound DATALOG Queries", *Journal of Computer and System Sciences*, 1995, to appear.

[47] R. Krishnamurthy and S.A. Naqvi, "Nondeterministic Choice in Datalog", in *Proceedings 5th Intl. Conf. on Data and Knowledge Bases*, Los Altos, 1988, pp. 416–424, Morgan Kaufmann Publishers, Inc.

[48] D. Saccà and C. Zaniolo, "Stable Models and Non-Determinism in Logic Programs with Negation", in *Proceedings PODS-90*, 1990, pp. 205–218.

[49] L. Corciulo, F. Giannotti, and D. Pedreschi, "Datalog with Nondeterministic Choice computes NBD-TIME", in *Proceedings Intl. Conf. on Deductive and Object-Oriented Databases (DOOD-93)*, 1993.

[50] S. Abiteboul and V. Vianu, "Non-Determinism in Logic-Based Languages", *Annals of Mathematics and Artificial Intelligence*, vol. 3, pp. 151–186, 1991.

[51] M. Cadoli, T. Eiter, and G. Gottlob, "Default Logic as a Query Language", Tech. Rep. CD-TR 94/65, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, May 1994, Preliminary version in: Proceedings Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94), May 1994.

Marco Cadoli received a Laurea degree in Electrical Engineering (1989) and a Doctorate in Computer Science (1993) from *Università "La Sapienza"* in Rome, Italy. Since 1994 he is Assistant Professor of Engineering at the same university. In September 1991-May 1992 he was a visiting scholar to the Computer Science Department of Stanford University. Cadoli's main research interests are computational aspects of data and knowledge representation, in particular non-monotonic reasoning, concept description languages and planning. He has published several papers in journals and conference proceedings in the fields of Artificial Intelligence, Theoretical Computer Science, Logic Programming and Database Theory. An extended version of his doctoral thesis appears as a volume in Springer Lecture Notes in AI with the title "Tractable Reasoning in Artificial Intelligence". He has served in the Program Committees of several workshops and conferences on AI and Databases. Since April 1995 he is secretary of the IFIP Working Group on Knowledge Representation.

Thomas Eiter received the Dipl.-Ing. degree, in 1989, the doctoral degree, in 1991, and the Dozent degree, in 1995, from the Technical University of Vienna, Vienna, Austria. From 1990 to 1991, he was a research assistant in the Christian Doppler Laboratory for Expert Systems. Since 1991 until now, he is with the Institute for Information Systems at the Technical University of Vienna and the Christian Doppler Laboratory for Expert Systems. He currently holds a tenured faculty position at the Technical University of Vienna. His current research interests are foundations of databases, artificial intelligence (mainly, knowledge representation and reasoning), and applied discrete mathematics. He has published several papers on these subjects, and served as a program committee member in conferences in the field of databases and expert systems. He also was enrolled as a consultant in projects about expert systems and a system for model-based hardware diagnosis.

Georg Gottlob is a Professor of Computer Science at the Technical University of Vienna, Austria, where he currently chairs the department of Information Systems and directs the industry-funded Christian Doppler Expert Systems Lab. His research interests are database theory (in particular, query languages), nonmonotonic reasoning, finite model theory, and computational complexity. On the more applied side, he supervises a number of industry projects dealing with expert systems and with multimedia information systems. Dr. Gottlob holds his current position since 1988. Before that, he was affiliated with the Italian National Research Council in Genoa, Italy, and before with the Politecnico di Milano, Italy. During the summer terms 1985–1987, he lectured at Stanford University.