In: Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94), Bonn, Germany, May 24–27, 1994.

Default Logic as a Query Language

Marco Cadoli
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli@assi.dis.uniroma1.it

Thomas Eiter **Georg Gottlob**
Christian Doppler Laboratory for Expert Systems
Technical University of Vienna
Paniglgasse 16, A-1040 Wien, Austria
{eiter|gottlob}@vexpert.dbai.tuwien.ac.at

Abstract

Research in NMR 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 formalisms as relational database query languages has been investigated to a much smaller extent. In this work 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 as expressive as $SO_{\exists\forall}$, the existential universal fragment of second order logic. This result is not only of theoretical importance: We show queries –which are useful in practice– that can be expressed with DQL and cannot with other query languages based on non-monotonic logics such as $\mathrm{DATALOG}^{\neg}_{stable}$. Another result in this paper concerns the *combined* complexity of DQL, i.e., when it is assumed that the query is part of the input; for this problem, $\mathrm{NEXPTIME}^{\mathrm{NP}}$-completeness is shown.

1 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 Tweety the bird 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 $\mathrm{DATALOG}^{\neg}$ 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. [Cadoli and Schaerf1993] 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. [Kannelakis1990]). 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 [Kolaitis and Papadimitriou1991] the expressive power of two semantics for $\mathrm{DATALOG}^{\neg}$ programs. In particular they prove that $\mathrm{DATALOG}^{\neg}$ with fixpoint semantics is as expressive as $SO_{\exists}$, the existential fragment of second order logic. Schlipf proves in [Schlipf1990] an analogous result for $\mathrm{DATALOG}^{\neg}$ with stable model semantics ($\mathrm{DATALOG}^{\neg}_{stable}$ henceforth). Saccà gives in [Saccà1993] further insight on the expressive power of $\mathrm{DATALOG}^{\neg}_{stable}$. Van Gelder analyzes in [Van Gelder1989] the expressive power of DATALOG$^\neg$ with well-founded semantics. 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 [Reiter1980] 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 [Bidoit and Froidevaux1991] as a tool for inferencing in logical databases (i.e., databases which are theories). Anyway the behavior of default logic on finite structures 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 show queries –which are useful in practice– that can be expressed with DQL and cannot with DATALOG$^\neg_{stable}$. Our queries are taken from the realm of economics.

An alternative way of describing our main result is to say that DQL "captures" the complexity class Σ_2^p of the polynomial hierarchy, while DATALOG$^\neg_{stable}$ "just" captures the class NP. Therefore DQL is more expressive than DATALOG$^\neg_{stable}$ provided $\Sigma_2^p \neq$ 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 [Gottlob1992, Stillman1992]. It is therefore important to remark that the expressive power of a language is not necessarily the same as its complexity. 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 [Stewart1991].

Another result shown in this paper concerns the *combined* complexity of DQL, i.e., when it is assumed that the query is part of the input; in particular, NEXPTIME$^{\mathrm{NP}}$-completeness is proven.

The structure of the paper is the following. In Section 2 we give the definition of the query language DQL, providing syntax, semantics and some simple examples. In Section 3 we give a formal proof of the fact that DQL captures Σ_2^p. In Section 4 we show how to use DQL for expressing queries to a relational database. In particular we show queries relative to a (somewhat simplified) economic world that are expressible in DQL but are not expressible in DATALOG$^\neg_{stable}$. In Section 5 we briefly address the issue of combined complexity of DQL, and in Section 6 we draw some conclusions.

2 DEFINITION OF DQL

2.1 SYNTAX

A *database schema* (cf. [Ullman1988]) 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 $\mathcal{U}$.

An *open default* (cf. [Reiter1980]) is a sentence of the form

$$\frac{\alpha(\mathbf{x}) : \beta(\mathbf{y})}{\gamma(\mathbf{z})}$$

where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are (not necessarily disjoint) lists of individual variables and $\alpha(\mathbf{x}), \beta(\mathbf{y}), \gamma(\mathbf{z})$ are quantifier-free formulae of first order logic (without functions) such that the free variables are those in $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$.

A *DQL Input/Output query* Q is a set of open default rules 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 in the query, plus the database.

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

Example 1: **(Tweety flies)** We have two relation schemes BIRD and SMALL-WINGS which both have a single attribute NAME. The database instance W_1 is as in the following table:

BIRD	NAME
	Tweety
	Sam
	Fred

SMALL-WINGS	NAME
	Fred

The I/O query is the following set of defaults D_1:

$$\left\{ \frac{bird(x) : \neg abnormal(x)}{flies(x)} \ , \ \frac{: \neg abnormal(x)}{\neg abnormal(x)}, \ \frac{bird(x) \wedge small\text{-}wings(x) :}{abnormal(x)} \right\}$$

plus the unary relation FLIES(NAME). The relational database states that Tweety, Sam, and Fred are birds. The query is made out of three open defaults. The first one states that an object that is provably a bird –and that cannot be proven to be abnormal– flies by default. The second default states that objects that cannot be proven to be abnormal should be regarded as not abnormal. The third default states the rule that objects that are provably birds and have small wings are abnormal. The intuitive meaning of the query is that we want to know the set of flying objects.

The boolean query has the same set of defaults, plus the ground formula $flies(Tweety)$. The intuitive meaning of the query is that we want to know whether Tweety flies or not. □

2.2 SEMANTICS

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 [Reiter1984]).

Let D be the set of open defaults in a query (either I/O query or boolean query). We denote as $INST(D)$ the *instantiation* of D, i.e., the set of ground defaults obtained in the following way:

- for each open default
$$\frac{\alpha(\mathbf{x}) : \beta(\mathbf{y})}{\gamma(\mathbf{z})}$$
in D there is a set of defaults
$$\frac{\alpha(\xi) : \beta(\tau)}{\gamma(\zeta)},$$
in $INST(D)$, where ξ, τ, and ζ are all possible lists –of the appropriate length– of objects taken from $\mathcal{U}$.

We remark that instantiating the query over the domain is common in databases, and it is assumed in [Schlipf1990, Kolaitis and Papadimitriou1991].

Let Q be an I/O query, i.e., a set D of defaults plus a set $S = \{S_1, \ldots, S_m\}$ of output relations. Let W be a database on the domain $\mathcal{U}$. The answer to Q is defined as follows. For each relation $S_i \in S$ with arity k_i, the answer is the set of tuples $\langle t_1, \ldots, t_{k_i} \rangle \in \mathcal{U}^{k_i}$ such that $S_i(t_1, \ldots, t_{k_i})$ follows under the credulous default semantics from the ground default theory $\langle INST(D); COMP(W) \rangle$, i.e. it is in at least one of the extensions of such a default theory (cf. [Reiter1980] for the definition of extension of a default theory).

The semantics of a boolean query Q is even simpler. Let γ be the distinguished ground formula in Q. If γ follows under the credulous default semantics from the ground default theory $\langle INST(D); COMP(W) \rangle$, then the answer is `yes`; otherwise, the answer is `no`.

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 ($INST(D)$). 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 ($1 \le i \le n$):
$$\frac{: \neg R'_i(\mathbf{x})}{\neg R'_i(\mathbf{x})} \ , \ \frac{R'_i(\mathbf{x}) :}{R_i(\mathbf{x})} \ , \ \frac{\neg R'_i(\mathbf{x}) :}{\neg R_i(\mathbf{x})}$$
- define $DB(W)$ as $R'_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i} \rangle \in R_i|W$.

It can be shown that the default theory $T = \langle INST(D \cup D'); DB(W) \rangle$ provides the same answers (to both boolean and I/O queries) as $\langle INST(D); COMP(W) \rangle$. 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 T, 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), then also the default theory

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

where W is seen as $R_i(a_1, \ldots, a_{l_i})$, for each tuple $\langle a_1, \ldots, a_{l_i}\rangle \in R_i$, provides the same answers as $\langle INST(D); COMP(W)\rangle$.

Our semantics for DQL is based on credulous default reasoning, and one may argue if this is the most appropriate way of answering to a query. In general, nothing prevents us from grounding our definitions on *skeptical* default reasoning, i.e. on drawing an inference iff a formula is in *all* the extensions of the relevant default theory. In fact, all forthcoming results about complexity and expressiveness would hold for the complementary complexity classes. As an example, forthcoming Theorem 1 could be rephrased by saying that the boolean (skeptical) DQL queries precisely capture the class Π_2^p.

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

Example 1 (continued) We assume that the domain is the set $\{Tweety, Sam, Fred\}$, i.e., that the domain is the same as the active domain. Then,

$COMP(W_1) =$

$$\{ bird(Tweety), \neg small\text{-}wings(Tweety), bird(Sam), \neg small\text{-}wings(Sam), bird(Fred), small\text{-}wings(Fred) \};$$

$INST(D_1) =$

$$\left\{ \frac{bird(Tweety) : \neg abnormal(Tweety)}{flies(Tweety)}, \frac{: \neg abnormal(Tweety)}{\neg abnormal(Tweety)}, \frac{bird(Tweety) \wedge small\text{-}wings(Tweety) :}{abnormal(Tweety)}, \frac{bird(Sam) : \neg abnormal(Sam)}{flies(Sam)}, \frac{: \neg abnormal(Sam)}{\neg abnormal(Sam)}, \frac{bird(Sam) \wedge small\text{-}wings(Sam) :}{abnormal(Sam)}, \frac{bird(Fred) : \neg abnormal(Fred)}{flies(Fred)}, \frac{bird(Fred) \wedge small\text{-}wings(Fred) :}{abnormal(Fred)} \right\}.$$

The answer to the I/O query is the relation instance:

FLIES

NAME
Tweety
Sam

In other words, Tweety and Sam fly, but Fred does not fly. The answer to the boolean query is `yes`. □

3 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 cannot? In other words, what is the *expressive power* of DQL? We recall that expressive power of relational database query languages is one of the most studied topics in database theory. One way of presenting a result in this area is to say that a query language can/cannot express a specific relation. For example, it is well-known that relational calculus cannot express the transitive closure of a relation [Aho and Ullman1979], while such a relation can be expressed in DATALOG [Ullman1988]; 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. [Schlipf1990].

Typically, the expressive power of a query language is represented as a set of logical sentences. As an example the expressive power of relational calculus is the set of first order sentences, while the expressive power of $\text{DATALOG}^{\neg}_{stable}$ is $\text{SO}_\exists$ [Schlipf1990], 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. In this section we show that the expressive power of DQL is $\text{SO}_{\exists\forall}$, the existential universal fragment of second order logic, on a relational vocabulary, i.e., the set of sentences

$$(\exists \mathbf{S})(\forall \mathbf{T})\phi(\mathbf{S}, \mathbf{T}), \tag{1}$$

where $\mathbf{S}, \mathbf{T}$ are disjoint lists of predicate symbols and $\phi(\mathbf{S}, \mathbf{T})$ is a function-free first order formula in which at least the predicates in $\mathbf{S}, \mathbf{T}$ occur.

Following the traditional notion of a set of logical sentences *capturing* a complexity class, we can say that the set $\text{SO}_\exists$ captures the class NP (cf. [Fagin1974]), while the set $\text{SO}_{\exists\forall}$ captures Σ_2^p (cf. [Lynch1982, Stockmeyer1977]). (See [Garey and Johnson1979, Johnson1990] for a definition of the classes NP and Σ_2^p of the polynomial hierarchy).

We are now ready to prove our main result, which concerns DQL boolean queries. We refer to the following useful lemmas. Let $Cons()$ denote classical deductive closure.

Lemma 1 ([Reiter1980, Theorem 2.1]) *Let $\Delta = \langle D, W\rangle$ be a (closed) default theory. A set E is an extension of Δ iff $E = \bigcup_{i=0}^{\infty} E_i$, where $E_0 = W$ and $E_{i+1} = Cons(E_i) \cup \{\gamma | \alpha : \beta/\gamma \in D,\, \alpha \in E_i,\, \beta \notin E\}$, for all $i \geq 0$.*

Lemma 2 ([Reiter1980, Theorem 2.5]) *Let E be an extension of the default theory $\Delta = \langle D, W\rangle$. Denote by $GD(E, \Delta) = \{\alpha : \beta/\gamma \in D \mid \alpha \in E, \neg\beta \notin E\}$ the generating defaults of E. Then,*

$$E = Cons(W \cup \{\gamma \mid \alpha : \beta/\gamma \in GD(E, \Delta)\})$$

Theorem 1 *The boolean DQL queries precisely capture the Σ_2^p database properties.*

PROOF (sketch): The easy part is to show that every query of DQL can be expressed as a Σ_2^p-recognizable database property. We notice that the semantics of DQL given in Section 2.2 transforms query answering into credulous reasoning in a propositional default theory. The transformation is polynomial in the size of the database, i.e., its data complexity is polynomial. Moreover it has been proven in [Gottlob1992, Stillman1992] 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 query expressible as a sentence of $\mathrm{SO}_{\exists\forall}$ can be expressed in DQL. As we already noticed in the introduction, *we cannot take advantage of the fact that propositional credulous default reasoning is Σ_2^p-hard*, because the expressiveness of a language is not necessarily the same as its complexity (cf. [Stewart1991] for a specific example).

Without loss of generality, we assume that sentence (1) is of form

$$(\exists \mathbf{S})(\forall \mathbf{T})(\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y}), \qquad (2)$$

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 (1) to (2) is justified in the appendix.

Now we have to show that for each query $Q_{SO_{\exists\forall}}$ of the form (2) 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 (2).

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 (2) is true over the underlying database W.

Formally, Q_{DQL} consists of the ground formula A and the set D containing the following open defaults.

- For each predicate P from $\mathbf{S}$ and for $<$:
$$\frac{:P(\mathbf{x})}{P(\mathbf{x})}, \quad \frac{:\neg P(\mathbf{x})}{\neg P(\mathbf{x})}, \quad \frac{:\mathbf{y} < \mathbf{y}'}{\mathbf{y} < \mathbf{y}'}, \quad \frac{:\neg(\mathbf{y} < \mathbf{y}')}{\neg(\mathbf{y} < \mathbf{y}')};$$
- linear order axioms for $<$:
$$\frac{:}{\neg(\mathbf{y} < \mathbf{y}' \wedge \mathbf{y}' < \mathbf{y})},$$
$$\frac{:}{(\mathbf{y} < \mathbf{y}') \vee (\mathbf{y}' < \mathbf{y}) \vee (\mathbf{y} = \mathbf{y}')},$$
$$\frac{:}{(\mathbf{y} < \mathbf{y}'' \wedge \mathbf{y}'' < \mathbf{y}') \rightarrow \mathbf{y} < \mathbf{y}'};$$
- rules for the associated predicates:
$$\frac{\mathbf{y} < \mathbf{y}' :}{\neg F(\mathbf{y}')}, \quad \frac{\mathbf{y} < \mathbf{y}' :}{\neg L(\mathbf{y})},$$
$$\frac{\neg(\mathbf{y} < \mathbf{y}') \vee (\mathbf{y} < \mathbf{y}'' \wedge \mathbf{y}'' < \mathbf{y}') :}{\neg S(\mathbf{y}, \mathbf{y}')};$$
- derivation of A (i.e., checking if $(\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y})$ is true):
$$\frac{:F(\mathbf{y})}{\psi(\mathbf{x}, \mathbf{y}) \rightarrow Z(\mathbf{x}, \mathbf{y})}, \quad \frac{:S(\mathbf{y}, \mathbf{y}')}{Z(\mathbf{x}, \mathbf{y}) \wedge \psi(\mathbf{x}, \mathbf{y}') \rightarrow Z(\mathbf{x}, \mathbf{y}')},$$
$$\frac{:L(\mathbf{y})}{Z(\mathbf{x}, \mathbf{y}) \rightarrow A}.$$

(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 $((y_1 = y_1') \wedge \cdots \wedge (y_m = y_m'))$.)

Let W be any database instance, and denote by Δ the default theory $\langle INST(D); COMP(W)\rangle$.

It is easy to see that Δ has only consistent extensions. (Formally, this can be easily proved by Lemma 2.)

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,

$$W, \mathbf{S}_0 \models (\forall \mathbf{T})(\exists \mathbf{x})(\forall \mathbf{y})\psi(\mathbf{x}, \mathbf{y}) \qquad (3)$$

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 $\mathcal{F}$ contains the following formulas:

$\mathcal{F}_1$: $COMP(W)$

$\mathcal{F}_2$: $\{\mathbf{a} < \mathbf{a}' \mid (\mathbf{a}, \mathbf{a}') \in <_0\} \cup \{\neg(\mathbf{a} < \mathbf{a}') \mid (\mathbf{a}, \mathbf{a}') \notin <_0\}$

$\mathcal{F}_3$: $\{\neg L(\mathbf{a}) \mid \mathbf{a} \notin L_0\}$

$\mathcal{F}_4$: $\{\neg F(\mathbf{a}) \mid \mathbf{a} \notin F_0\}$

$\mathcal{F}_5$: $\{\neg S(\mathbf{a}, \mathbf{a}') \mid (\mathbf{a}, \mathbf{a}') \notin S_0\}$

$\mathcal{F}_6$: $\{P(\mathbf{a}) \mid P \in \mathbf{S}, \mathbf{a} \in P_0\} \cup \{\neg P(\mathbf{a}) \mid P \in \mathbf{S}, \mathbf{a} \notin P_0\}$

$\mathcal{F}_7$: all ground formulas obtained by instantiation of the linear order axioms for $<$ over the domain

$\mathcal{F}_8$: $\{\psi(\mathbf{a}, \mathbf{b}) \to Z(\mathbf{a}, \mathbf{b}) \mid \mathbf{b} \in F_0\}$

$\mathcal{F}_9$: $\{Z(\mathbf{a}, \mathbf{b}) \wedge \psi(\mathbf{a}, \mathbf{b}') \to Z(\mathbf{a}, \mathbf{b}') \mid (\mathbf{b}, \mathbf{b}') \in S_0\}$

$\mathcal{F}_{10}$: $\{Z(\mathbf{a}, \mathbf{b}) \to A \mid \mathbf{b} \in L_0\}$

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 iterative characterization of default extensions in Lemma 1. We obtain that

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

Containment of $\mathcal{F}_8$, $\mathcal{F}_9$, and $\mathcal{F}_{10}$ in E_1 holds since for each tuples $\mathbf{b}_0 \in F_0$, $(\mathbf{b}_1, \mathbf{b}_2) \in S_0$, and $\mathbf{b}_3 \in L_0$, we have that $\neg F(\mathbf{b}_0) \notin E$, $\neg S(\mathbf{b}_1, \mathbf{b}_2) \notin E$, and $\neg L(\mathbf{b}_3) \notin E$; this follows immediately from the definition of $\mathcal{F}$ and standard interpolation properties.

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 (3) 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

$$W, \mathbf{S}_0, \mathbf{T}_0 \models (\forall \mathbf{y})\psi(\mathbf{a}, \mathbf{y}) \tag{4}$$

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 (4), $\psi(\mathbf{a}, \mathbf{b}_0)$ 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 (4), $\psi(\mathbf{a}, \mathbf{b}')$ 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 $: P(\mathbf{a}) \,/\, P(\mathbf{a})$, $: \neg P(\mathbf{a}) \,/\, \neg P(\mathbf{a})$ 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, from the instantiated default rules for the predicates associated to $<$, it follows that $\neg F(\mathbf{a}) \in E$ if $\mathbf{a}$ is not the first tuple in $<_0$, that $\neg S(\mathbf{a}, \mathbf{a}') \in E$ if $\mathbf{a}'$ is not the successor of $\mathbf{a}'$ in $<_0$, and that $\neg L(\mathbf{a}) \in E$ if $\mathbf{a}$ is not the last tuple in $<_0$. By the characterization of E in terms of its generating defaults (Lemma 2), it is easy to see that E is consistent with each of $F(\mathbf{a}_0)$, $S(\mathbf{a}_1, \mathbf{a}_2)$, and $L(\mathbf{a}_3)$ such that $\mathbf{a}_0$ is the first tuple in $<_0$, $\mathbf{a}_2$ is the successor of $\mathbf{a}_1$ in $<_0$, and $\mathbf{a}_3$ is the last tuple in $<_0$.

We claim that

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

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

$$Z_0 = \{\, (\mathbf{a}, \mathbf{b}) \mid \mathbf{b} <_0 m(\mathbf{a}) \,\},$$

assign A the value *"false"*, and complete the partial valuations F_0, S_0, and L_0 of F, S, and L defined by E by letting $\mathbf{a}_0 \in F_0$, $(\mathbf{a}_1, \mathbf{a}_2) \in S_0$, and $\mathbf{a}_3 \in L_0$ for all $\mathbf{a}_0$, $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ such that $\neg F(\mathbf{a}_0) \notin E$, $\neg S(\mathbf{a}_1, \mathbf{a}_2) \notin E$, and $\neg L(\mathbf{a}_3) \notin E$, respectively.

This valuation of the predicates defines a model of E. To prove this, it is by Lemma 2 sufficient to show that the valuation is a model of $COMP(W) \cup G$, where $G = \{\gamma \mid \alpha : \beta/\gamma \in GD(E, \Delta)\}$.

Clearly, the only formulas in G it remains to argue about are those which are conclusions of generating defaults of E with justifications $F(\mathbf{y})$, $S(\mathbf{y},\mathbf{y}')$, and $L(\mathbf{y})$.

- Assume that $\psi(\mathbf{a},\mathbf{b}) \to Z(\mathbf{a},\mathbf{b}) \in G$. Then, $\mathbf{b}$ is the first tuple in $<_0$, and the formula is certainly satisfied.
- Assume that $Z(\mathbf{a},\mathbf{b}) \wedge \Psi(\mathbf{a},\mathbf{b}') \to Z(\mathbf{a},\mathbf{b}') \in G$. 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.
- Assume that $Z(\mathbf{a},\mathbf{b}) \to A \in G$. 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 $COMP(W) \cup G$, 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 (5) is proved. Now claim (5) 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 $\alpha \to \gamma$. $\Box$

In order to give a corresponding result for DQL I/O queries, we need the concept of query recognizability. A query mapping database instances over R into database instances over S is **C**-*recognizable,* **C** a complexity class, if deciding whether a tuple **t** belongs to a certain output relation $S_i \in S$ is in **C** (cf. [Gurevich1988]). Using Theorem 1, we can show the following.

Theorem 2 *A database query is Σ_2^p-recognizable if and only if it is definable as a DQL I/O query.*

4 APPLICATIONS AND EXAMPLES

This section is devoted to illustration of queries expressible via DQL.

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. The database instance in Table 1 describes a possible situation.

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 company Alpha 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. As an example, both Alpha and Beta are strategic, because {Alpha, Beta} produce all products, while neither {Alpha} nor {Beta} do that. On the other hand Gamma is not strategic. 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.

A manager can easily express a boolean query whose answer tells if a specific company C is strategic or not by writing the set of open defaults D:

$$\frac{produces(x,y,z):}{strat(y) \vee strat(z)}, \quad \frac{:\neg strat(x)}{\neg strat(x)}$$

and the ground formula $strat(C)$. 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` iff the company C is strategic.

Now let us consider a slightly more complex situation, in which up to three (say) companies can *control* another company. As an example, we assume that the situation is described by means of the relation instance in Table 2.

The meaning of the tuple is that companies Alpha and Beta together have control over Gamma, i.e., the holding cannot own both Alpha and Beta without owning Gamma as well. Further information of this kind completely changes the minimal sets of companies that produce all products. As an example {Alpha, Beta} is no longer such a set, while {Alpha, Beta, Gamma} is. If we add the default

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

to D, then the boolean query gives the desired answer.

In the former case –no controlled companies– the problem of deciding whether a company is strategic is in NP (cf. [Cadoli and Lenzerini1991]), while in the latter case the same problem is Σ_2^p-complete (cf. [Eiter and Gottlob1993]). As a consequence, the former query is expressible in $\text{DATALOG}^{\neg}_{stable}$, while the latter is not. 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.

Table 1: Database instance of the producers

PRODUCERS

PRODUCT	COMPANY #1	COMPANY #2
Pasta	Alpha	Beta
Tomatoes	Gamma	Alpha
Wine	Alpha	Alpha
Bread	Beta	Delta

Table 2: Database instance of the controlled companies

CONTROL

CONTROLLED	CONT #1	CONT #2	CONT #3
Gamma	Alpha	Beta	Beta

Example 3: **(Maximal trust)** In a finite set of companies it is possible to make agreements. A *trust* is a set T of companies such that each one has an agreement with each other company in T. A *maximal* trust is a trust such that there is no bigger –wrt cardinality– trust. The decision problem "does a company belong to a maximal trust?" is $\mathrm{P}^{\mathrm{NP}[O(log\ n)]}$-hard, hence most likely harder than NP.

We can easily represent companies and agreements with a relational database. We can express the query "does a company belong to a maximal trust?" in DQL but we cannot in $\mathrm{DATALOG}^{\neg}_{stable}$. □

5 COMBINED COMPLEXITY OF DQL

The *combined complexity* (cf. [Vardi1982]) of Boolean DQL queries amounts to the complexity of deciding a query Q over a database instance W, where the input is arbitrary Q and W.

By the complexity results for propositional default logic [Gottlob1992, Stillman1992], the combined complexity of Boolean DQL is Σ_2^p-complete if no default is open, i.e., all defaults are already instantiated. The instantiation of a set of open defaults is in general exponentially larger, however; this intuitively suggests an exponential increase of complexity for query answering if open defaults are present. In fact, we can show that it increases to $\mathrm{NEXPTIME}^{\mathrm{NP}}$, the problems solvable by a non-deterministic Turing machine in *exponential* time using an NP-oracle (cf. [Johnson1990]).

Theorem 3 *The combined complexity of Boolean DQL is* $\mathrm{NEXPTIME}^{\mathrm{NP}}$*-complete.*

PROOF (Sketch) The transformation of the semantics of Boolean DQL into credulous reasoning in a propositional default theory is exponential ($O(2^{n^k})$, where k is a constant) in the query size, i.e., the default theory $\langle INST(D), COMP(W)\rangle$ is exponential in the size of D plus W. Credulous reasoning in $\langle INST(D), COMP(W)\rangle$ can be done by a nondeterministic Turing machine with an NP oracle in time polynomial in the size of the input default theory. Therefore, the Boolean DQL query can be decided by a non-deterministic Turing machine in exponential with an NP oracle. This proves membership in $\mathrm{NEXPTIME}^{\mathrm{NP}}$.

The hardness proof is too involved to be described here in detail. It gives a reduction from a graph coloring problem (co-$\mathrm{CERT3COL}_S$) that can be shown to be $\mathrm{NEXPTIME}^{\mathrm{NP}}$-hard, cf. [Eiter *et al.*1994]; the reduction is based on advanced complexity upgrading techniques [Balcázar *et al.*1992, Papadimitriou and Yannakakis1985] and simulation of a Boolean circuit (cf. [Kolaitis and Papadimitriou1991]) in DQL.

Problem co-CERT3COL is a follows. An instance I of size n of the problem consists of an undirected graph G on vertices $\{0,\ldots,n-1\}$, whose edges are labeled with a disjunction of two literals where each literal is over the Boolean variables $\{X_{i,j} \mid i,j = 0,\ldots,n-1\}$; I is a Yes-instance if for some truth value assignment t to the Boolean variables, the graph $t(G)$ obtained from G by including only those edges whose labels are true under t is not 3-colorable. I is encoded as a binary string representing a database instance (domain $\{0,\ldots,n-1\}$) over relations describing the edges and the graph labeling.

The succinct version of problem co-CERT3COL, co-$\mathrm{CERT3COL}_S$, is $\mathrm{NEXPTIME}^{\mathrm{NP}}$-hard, cf. [Eiter *et al.*1994]. In the succinct version – instead of a binary string w for I – the input consists of a Boolean circuit C_I with $\log|w|$ input bits, by which each bit i of w can be computed.

The transformation of co-$\mathrm{CERT3COL}_S$ to Boolean DQL roughly is as follows. For each instance I of

CERT3COL of size n, an equivalent Boolean DQL query on a *fixed* database with domain $\{0,1\}$ is constructed in polynomial time; elements of the domain are represented by tuples on $\{0,1\}$ of arity $\lceil \log n \rceil$, and each input relation R is computed by a collection of default rules, which simulates the computation of R by a Boolean circuit C_R that is easily constructed from C_I.

Remark: Since the transformation reduces co-CERT3COL to a fixed database, the same result follows for the *expression complexity* of Boolean DQL. □

Notice that NEXPTIME$^{\mathrm{NP}}$-complete problems are *provably* harder than NP-complete problems, since NP $\subset$ NEXPTIME$^{\mathrm{NP}}$; few practical such problems are known.

For the case where Q is fixed, i.e., for the *data complexity*, we immediately obtain from our expressiveness results the following.

Theorem 4 *The data complexity of Boolean DQL is* Σ_2^p*-complete.*

PROOF (Sketch) 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 (1), an equivalent sentence of the form (2) 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 (1) is valid in a given database instance W, which is Σ_2^p-hard, is transformable to a Boolean DQL query in polynomial time. □

Similar results can be derived for DQL I/O queries, measuring the complexity of query recognizability (cf. Section 3).

Theorem 5 *The combined (resp. expression) complexity of DQL I/O queries is* NEXPTIME$^{\mathrm{NP}}$*-complete and the data complexity is* Σ_2^p*-complete.*

6 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 and cannot with other query languages based on non-monotonic logics such as DATALOG$^{\neg}_{stable}$.

In the definition of query –Section 2.1– open defaults are function- and quantifier-free. While unlimited quantification cannot be allowed without loosing decidability, the impact of allowing functions remains for further investigation. Another interesting question is whether expressiveness of DQL decreases if only normal or semi-normal defaults are allowed.

APPENDIX

The pass from (1) to (2) is justified as follows.

As shown in [Kolaitis and Papadimitriou1991, p.130] (cf. also [van Benthem and Doets1983, Section 2.5.2]), for every existential second order sentence Ψ over a relational vocabulary σ, there exists an equivalent second order sentence over σ of the form $(\exists \mathbf{T})(\forall \mathbf{x})(\exists \mathbf{y})\phi(\mathbf{x},\mathbf{y})$ over the same vocabulary, where ϕ is a quantifier-free first-order formula; this sentence can be effectively constructed. Consequently, for every universal second order sentence Φ over vocabulary σ, there exists an equivalent second order sentence over σ of the form $(\forall \mathbf{T})(\exists \mathbf{x})(\forall \mathbf{y})\phi(\mathbf{x},\mathbf{y})$, where ϕ is a quantifier-free first-order formula. Let Φ be a universal second order sentence equivalent to $(\forall \mathbf{T})\phi(\mathbf{S},\mathbf{T})$. Let D be any structure for σ. Then,

$$D \models (\exists \mathbf{S})(\forall \mathbf{T})\phi(\mathbf{S},\mathbf{T})$$

iff

$$D, \mathbf{S}_0 \models (\forall \mathbf{T})\phi(\mathbf{S},\mathbf{T})$$

iff

$$D, \mathbf{S}_0 \models \Phi$$

iff

$$D \models (\exists \mathbf{S})\Phi,$$

where $\mathbf{S}_0$ is an appropriate valuation of the $\mathbf{S}$ predicates.

Consequently, the sentence in (1) is equivalent to a sentence of the form (2).

Acknowledgements

The authors are grateful to Torsten Schaub for interesting comments on semantics of DQL. The first author is partially supported by ESPRIT Basic Research Action 6810 COMPULOG II and by *Progetto Finalizzato Informatica* of the CNR (Italian Research Council).

References

[Aho and Ullman1979] A.V. Aho and J.D. Ullman. Universality of Data Retrieval Languages. In *Proceedings ACM Symposium on Principles of Programming Languages*, pages 110–117, 1979.

[Balcázar *et al.*1992] J.L. Balcázar, A. Lozano, and J. Torán. The Complexity of Algorithmic Problems on Succinct Instances. In R. Baeta-Yates and U. Manber, editors, *Computer Science*, pages 351–377. Plenum Press, New York, 1992.

[Bidoit and Froidevaux1991] N. Bidoit and Ch. Froidevaux. General logic databases and programs: Default semantics and stratification. *Information and Computation*, 19:15–54, 1991.

[Cadoli and Lenzerini1991] Marco Cadoli and Maurizio Lenzerini. The Complexity of Closed World Reasoning and Circumscription. Technical Report RAP. 17.91, Universitá di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, September 1991. Preliminary version in *Proc. of AAAI-90.* To appear in *Journal of Computer and System Sciences.*

[Cadoli and Schaerf1993] M. Cadoli and M. Schaerf. A Survey of Complexity Results for Non-monotonic Logics. *Journal of Logic Programming*, 17:127–160, 1993.

[Eiter and Gottlob1993] 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)*, pages 158–167, June 1993.

[Eiter *et al.*1994] 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. Forthcoming.

[Fagin1974] R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.

[Garey and Johnson1979] Michael Garey and David S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

[Gottlob1992] Georg Gottlob. Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Gurevich1988] Y. Gurevich. Logic and the Challenge of Computer Science. In E. Börger, editor, *Trends in Theoretical Computer Science*, chapter 1. Computer Science Press, 1988.

[Johnson1990] David S. Johnson. A Catalog of Complexity Classes. In van Leeuwen leeu-90, chapter 2.

[Kannelakis1990] P. Kannelakis. Elements of Relational Database Theory. In van Leeuwen leeu-90, chapter 17.

[Kolaitis and Papadimitriou1991] Ph. Kolaitis and Ch. H. Papadimitriou. Why Not Negation By Fixpoint ? *Journal of Computer and System Sciences*, 43:125–144, 1991.

[Lynch1982] J.F. Lynch. Complexity Classes and Theories of Finite Models. *Mathematical Systems Theory*, 15:127–144, 1982.

[Papadimitriou and Yannakakis1985] C.H. Papadimitriou and M. Yannakakis. A Note on Succinct Representations of Graphs. *Information and Computation*, 71:181–185, 1985.

[Reiter1980] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Reiter1984] R. Reiter. Towards a Reconstruction of Logical Databases. In J.W. Schmidt M.L. Brodie, J.L. Mylopoulos, editor, *On Conceptual Modelling*, pages 163–189, New York, 1984. Springer.

[Saccà1993] D. Saccà. Multiple Stable Models are Neeeded to Solve Unique Solution Problems. In *Informal Proceedings of the Second Compulog Net Meeting on Knowledge Bases (CNKBS-93)*, Athens, April 1993.

[Schlipf1990] J.S. Schlipf. The Expressive Powers of Logic Programming Semantics. Technical Report CIS-TR-90-3, Computer Science Department, University of Cincinnati, 1990. Preliminary version in *Proc. of PODS-90.* To appear in *Journal of Computer and System Sciences.*

[Stewart1991] I. Stewart. Comparing the Expressibility of Languages Formed Using NP-Complete Operators. *Journal of Logic and Computation*, 1(3):305–330, 1991.

[Stillman1992] J. Stillman. The Complexity of Propositional Default Logic. In *Proceedings AAAI-92*, pages 794–799, 1992.

[Stockmeyer1977] L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Ullman1988] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

[van Benthem and Doets1983] Johan van Benthem and Kees Doets. Higher Order Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol.I*, chapter I.4, pages 275–329. D. Reidel Publishing Company, 1983.

[Van Gelder1989] A. Van Gelder. The Alternating Fixpoint of Logic Programs With Negation. In *Proceedings PODS-89*, pages 1–10, 1989.

[van Leeuwen1990] J. van Leeuwen, editor. Handbook of Theoretical Computer Science. Elsevier Science Publishers B.V. (North-Holland), 1990.

[Vardi1982] M. Vardi. Complexity of Relational Query Languages. In *Proceedings 14th STOC*, pages 137–146, 1982.