This is a preliminary release of an article accepted by ACM Transactions on Database Systems. The definitive version is currently in production at ACM and, when released, will supersede this version.

Disjunctive Datalog

THOMAS EITER
University of Gießen
and
GEORG GOTTLOB
Technical University of Vienna
and
HEIKKI MANNILA
University of Helsinki

We consider disjunctive datalog, a powerful database query language based on disjunctive logic programming. Briefly, disjunctive datalog is a variant of datalog where disjunctions may appear in the rule heads; advanced versions also allow for negation in the bodies, which can be handled according to a semantics for negation in disjunctive logic programming. In particular, we investigate three different semantics for disjunctive datalog: the minimal model semantics, the perfect models semantics, and the stable model semantics. For each of these semantics, the expressive power and complexity are studied. We show that the possibility variants of these semantics express the same set of queries. In fact, they precisely capture the complexity class Σ_2^p. Thus, unless the Polynomial Hierarchy collapses, disjunctive datalog is more expressive than normal logic programming with negation.

These results are not only of theoretical interest; we demonstrate that problems relevant in practice such as computing the optimal tour value in the Traveling Salesman Problem and eigenvector computations can be handled in disjunctive datalog, but not datalog with negation (unless the Polynomial Hierarchy collapses).

In addition, we study modularity properties of disjunctive datalog and investigate syntactic restrictions of the formalisms.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Query Languages; D.1.6 [**Programming Techniques**]: Logic Programming; I.2.3 [**Artificial Intelligence**]: Deduction

---

This work has been partially supported by Austrian Industries in the context of the Christian Doppler Laboratory for Expert Systems. This paper substantially augments and improves an extended abstract presented at PODS '94 [Eiter et al. 1994]. Part of the third authors' work has been carried out while visiting the Technical University of Vienna.
Authors' addresses: Thomas Eiter, Computer Science Group, Faculty of Mathematics, University of Gießen, Arndtstraße 2, D-35392 Germany; email: ⟨Thomas.Eiter@informatik.uni-giessen.de⟩; Georg Gottlob, Information Systems Department, Technical University of Vienna, Paniglgasse 16, A-1040 Wien, Austria; email: ⟨gottlob@dbai.tuwien.ac.at⟩; Heikki Mannila, Department of Computer Science, University of Helsinki, P.O. Box 26, FIN-00014 Helsinki, Finland; email: ⟨mannila@cs.helsinki.fi⟩

and Theorem Proving; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms: Theory

Additional Key Words and Phrases: datalog, disjunctive logic programming, deductive systems, expressive power, finite model theory, finite structures, descriptive complexity theory.

1. INTRODUCTION

Strong interest in enhancing logic programs by the capability of disjunction emerged in the logic programming, artificial intelligence, and databases communities. This interest is documented by a number of publications (cf. [Lobo et al. 1992; Minker 1994]) and even workshops dedicated to this subject [Wolfinger 1994; Dix et al. 1996]. In this paper, we study disjunctive logic programming in the context of databases.

Datalog is a well-known database query language that uses disjunction-free logic programs, where no functions are allowed [Ullman 1989; Ceri et al. 1990]. We deal here with extensions of such programs by disjunction (and, as well as for datalog, by negation); the corresponding query languages thus amount to *disjunctive datalog.*

Briefly, a disjunctive logic program (DLP) is a logic program where disjunction may occur in the rule heads. For example, a DLP may contain rules such as

$$Male(x) \vee Female(x) \leftarrow Person(x).$$

The usefulness of disjunctive rules for knowledge representation, database querying, and for representing incomplete information, is generally acknowledged (see [Gelfond and Lifschitz 1991; Baral and Gelfond 1994]).

Various semantics of DLPs have been proposed; they are commonly based on the paradigm of minimal models, which underlies Minker's Generalized Closed World Assumption [Minker 1982]. For an overview of various semantics for DLPs, the reader may consult [Lobo et al. 1992] and Minker's more recent article [Minker 1994].

In this paper, we limit our attention to three important semantics for disjunctive logic programming (and thus for disjunctive datalog), whose precise definition will be given in Section 3:

—*Minimal model semantics.* This semantics, also termed Extended Generalized Closed World Assumption (EGCWA) [Yahya and Henschen 1985], is closely related to McCarthy's circumscription [McCarthy 1986].

—*Perfect model semantics* [Przymusinski 1988]. This semantics has been defined for general DLPs, but is particularly suited for stratified DLPs. Note that the perfect model approach is successfully used by Kifer, Lausen, and Wu [Kifer et al. 1995] for defining a semantics of object-oriented languages.

—*Disjunctive stable model semantics* [Przymusinski 1991; Gelfond and Lifschitz 1991]. This semantics generalizes the well acknowledged stable semantics for normal logic programs [Gelfond and Lifschitz 1988].

Much work has been spent on the suitability of these and other DLP semantics in various contexts, as well as on mathematical and structural properties, cf. [Dix 1992; Lobo et al. 1992; Minker 1994; Sakama and Inoue 1994]. However, until recently, little was known on the complexity and the expressive power of disjunctive logic programming.

Noticeable recursion-theoretic results on disjunctive inference formalisms over *infinite* structures where obtained by Schlipf [Schlipf 1987], who showed that general circumscription over the integers is Π^1_2-complete, and by Chomicki and Subrahmanian [Chomicki and Subrahmanian 1990], who showed that Minker's Generalized Closed World Assumption is Π^0_2-complete. These results show that on infinite structures, disjunctive logic programming is harder than both classical reasoning and normal logic programming [Chomicki and Subrahmanian 1990] (see [Schlipf 1995a] for an overview). However, most computer science applications deal with *finite structures*, in particular, with database relations. In this context, the important question of whether each DLP can be replaced by an equivalent normal logic program was open.

Recently, Eiter and Gottlob [Eiter and Gottlob 1995b] analyzed the complexity of *propositional* disjunctive logic programming (see [Eiter and Gottlob 1995b] for a detailed treatment). They show that in this setting, most forms of DLP are computationally harder than normal logic programming with negation (unless the Polynomial Hierarchy collapses). The present paper tackles the more difficult and involved task of analyzing both the complexity and the expressive power of disjunctive datalog over relational databases. In particular, we tackle the problem of characterizing the queries that a disjunctive datalog program may compute on relational databases that are supplied for input. More precisely, this means that we investigate the expressive power of a query language where the database queries are syntactically given by disjunctive datalog programs and a database consists of a finite collection of finite relations.

The main questions we address in this paper are the following:

⋆ How to define the precise semantics of disjunctive datalog based on different well-known semantics for DLPs (in particular, the three semantics from above) ?

⋆ What is the expressive power of these variants of disjunctive datalog? In particular, how is the expressive power affected by allowing disjunction in combination with other constructs like inequality ($\neq$) and negation ($\neg$) ?

⋆ What is the complexity of disjunctive datalog, measured by data and expression complexity for the various formalisms ?

⋆ Is disjunctive datalog besides its theoretical interest of practical relevance, i.e., do there exist queries arising in practice for which the power of disjunction in datalog is needed?

Our answers to these questions and the main contributions of this paper are summarized as follows:

—We give a precise definition of the syntax and the semantics of disjunctive datalog based on the minimal model semantics (*MM*), the disjunctive stable model semantics (*SM*), and the perfect model semantics (*PM*). We consider both the possibility and the certainty version of these semantics, for boolean queries and for queries computing output relations.

—We prove some new modularity properties for disjunctive logic programs, which generalize the concept of stratification and are a useful tool for the formal analysis of DLPs.

—Plain datalog augmented with disjunction (DATALOG$^\vee$) allows under all considered semantics (*MM*,*SM*,*PM*) expressing some Σ_2^p-hard queries, but not all queries in Σ_2^p. In fact, it does not allow expressing all queries computable in polynomial time. (This is shown by proving a weak form of preservation under supermodels.)

—We show that disjunctive datalog with $\neq$ captures precisely the Σ_2^p queries under the possibility version of all considered semantics. Thus, adding $\neq$ (inequality) to DATALOG$^\vee$ (giving DATALOG$^{\vee,\neq}$) suffices to gain full expressive power. This result is proved exploiting a strengthened form of Fagin's Theorem generalized to the Polynomial Hierarchy ($\Sigma_1^p = \mathrm{NP}$, $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$, ...) [Fagin 1974; Stockmeyer 1977], which exists for finite structures only and classes Σ_k^p where $k \geq 2$ [Eiter et al. 1996]. The same expressive power (Σ_2^p) is gained by adding negation ($\neg$) in case of both the *SM* and the *PM* semantics. For these semantics, symmetric results (Π_2^p-expressiveness) holds for certainty reasoning. In particular, the results show that for disjunctive datalog the *SM*-semantics and the *PM*-semantics express precisely the same set of queries; in the light of results in [Eiter and Gottlob 1995b], this is probably not true for DATALOG$^\neg$, i.e. nondisjunctive datalog with negation.

—We show that disjunctive datalog queries are Σ_2^p-complete in the data size and NEXPTIME$^{\mathrm{NP}}$-complete in the program size under the possibility version of all considered semantics. In fact, there are Σ_2^p-hard (resp. NEXPTIME$^{\mathrm{NP}}$-hard) queries defined by plain disjunctive datalog programs without $\neq$ and $\neg$. Notice that NP $\subset$ NEXPTIME$^{\mathrm{NP}}$, and hence NEXPTIME$^{\mathrm{NP}}$-complete problems are provably intractable; few natural among them are known. We establish the NEXPTIME$^{\mathrm{NP}}$-completeness results using complexity upgrade techniques that extend results in [Balcázar et al. 1992; Papadimitriou and Yannakakis 1985; Papadimitriou 1994], applied to results in [Stewart 1991].

—We demonstrate the practical relevance of disjunctive datalog by exhibiting queries which can be expressed in disjunctive datalog but not in standard datalog with negation (DATALOG$^\neg$) unless the Polynomial Hierarchy collapses. In particular, we show how computing the optimal tour value in the Traveling Salesman Problem and eigenvector computations can be handled in disjunctive datalog, but not in DATALOG$^\neg$ (unless the Polynomial Hierarchy collapses).

—We compare disjunctive datalog with other variants of datalog and well-known database query languages.

Our results provide a clear picture of the expressiveness of disjunctive datalog in the various settings. Moreover, they complement and extend previous results of important extensions of datalog and related query languages (see [Abiteboul et al. 1995; Kanellakis 1990]). Note that the major versions of DATALOG$^\neg$ [Kolaitis and Papadimitriou 1991; Schlipf 1995b; van Gelder et al. 1991] express database queries in NP (resp. co-NP) under possibility (resp. certainty) inference. Thus, disjunction adds expressive power (unless $\Sigma_2^p = \text{NP}$). A consequence of the high expressive power of disjunctive datalog is a high worst-case complexity. This trade-off between expressivity and complexity, which is not necessarily a drawback, will be discussed in the conclusion (Section 9).

The paper is organized as follows. Section 2 gives some basic concepts and notation. In Section 3 we present syntax and semantics of disjunctive datalog, based on different semantics for disjunctive logic programs. Applications and examples are given in Section 4, where programs for several queries are given that can not be implemented in DATALOG$^{\neg}$. Section 5 investigates the structure of disjunctive datalog programs, and presents a result about decomposability of a program into hierarchically organized modules. Section 6 analyses the expressive power of disjunctive datalog, which is continued in Section 7 for plain disjunctive datalog. The complexity of disjunctive datalog is determined in Section 8. Finally, Section 9 compares disjunctive datalog to common and novel database query languages, addresses the issue of why its precise expressiveness characterization is relevant, and outlines open issues for future work.

2. PRELIMINARIES

We assume familiarity with the basic concepts of complexity theory and logical query languages for databases; for a background, see [Papadimitriou 1994; Ullman 1989; Ceri et al. 1990; Abiteboul et al. 1995].

A *relational scheme* over a domain *Dom* is a (possibly empty) list $\mathbf{R} = R_1, \dots, R_k$ of relation symbols R_i, $i = 1, \dots, k$ of arity $a(R_i) \geq 0$. We assume that *Dom* is fixed and countable. A *relational database* over $\mathbf{R}$ (or *instance* of $\mathbf{R}$) is a finite structure $D = (U, r_1, \dots, r_k)$ of a finite universe $U \subseteq Dom$ and relations $r_i \subseteq U^{a(R_i)}$. Notice that for each tuple $\mathbf{t} \in U^{a(R_i)}$, $\mathbf{t} \in r_i$ iff $D \models R_i(\mathbf{t})$. By $U(D)$ (or simply U, if clear from the context) we denote the universe of D and by $D[\mathbf{R}']$ the projection of D to $\mathbf{R}'$ for any subscheme $\mathbf{R}'$ of $\mathbf{R}$. The set of all databases over $\mathbf{R}$ is denoted by $\mathcal{D}(\mathbf{R})$. As common, we use relation symbols also to denote relation instances, and say relation instead of relation symbol.

A *database property over* $\mathbf{R}$ is a predicate $\mathcal{P}$ on $\mathcal{D}(\mathbf{R})$ which associates with each $D \in \mathcal{D}(\mathbf{R})$ a truth value $\mathcal{P}(D)$ from $\{true, false\}$ and which is generic, i.e., invariant under permutations of *Dom*.

A *database mapping* or *query* is a (partial) recursive function $q : \mathcal{D}(\mathbf{R}) \to \mathcal{D}(\mathbf{S})$, where $\mathbf{R}$ and $\mathbf{S}$ are disjoint, which maps instances D of $\mathbf{R}$ (intuitively, the input relations) to instances $q(D)$ of $\mathbf{S}$ (the output relations) so that $U(D) = U(q(D))$. Usually, attention is restricted to *generic* queries [Chandra and Harel 1982] (or, less restrictively, to C-generic queries, which allow generic constants), which are invariant under permutations of the elements of *Dom*. Genericity of queries asserts data independence, i.e., data representation does not have any impact on the query result.

A query q is *boolean* if $\mathbf{S}$ is a single 0-ary relation symbol (i.e., a propositional letter) S; technically, S is *true* if (as a relation) $S = \{\,()\,\}$, and S is *false* if $S = \emptyset$. A generic total boolean query corresponds to a database property in the obvious way.

Recognizability of a query q is defined in terms of the following *query output* *tuple (QOT) problem*: Given $D \in \mathcal{D}(\mathbf{R})$, $S_i \in \mathbf{S}$, and a tuple $\mathbf{t} \in U(D)^{a(S_i)}$, decide if $q(D) \models S_i(\mathbf{t})$. Query q is $\mathbf{C}$-*recognizable*, where $\mathbf{C}$ is a complexity class, iff the QOT-problem is in $\mathbf{C}$ (cf. [Gurevich 1988]). Similarly, a database property $\mathbf{P}$ is $\mathbf{C}$-*recognizable* (or, *in* $\mathbf{C}$) iff deciding $\mathbf{P}(D) = true$ is in $\mathbf{C}$.

A *query language* is a set $\mathcal{L}$ of query expressions E and a function μ defining the semantics of $\mathcal{L}$, such that for each $E \in \mathcal{L}$, $\mu(E)$ is a query [Vardi 1982]. The *expressive power* $E(\mathcal{L})$ *of* $\mathcal{L}$ is the set of all queries definable by query expressions from $\mathcal{L}$, i.e., $E(\mathcal{L}) = \{q : q = \mu(E), E \in \mathcal{L}\}$. Of particular interest are the generic queries definable in $\mathcal{L}$. As common with query languages, we will focus in our analysis of the expressive power of disjunctive datalog on generic queries.

It is well-known that second-order definability of properties on finite structures is closely connected to computability within the Polynomial Hierarchy (PH) [Stockmeyer 1977]. Recall that the classes of PH are defined as follows: $\Sigma_0^p = \Pi_0^p = \Delta_0^p =$ P, and for $k \geq 0$, $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p}$, $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$, and $\Pi_{k+1}^p = \text{co-}\Sigma_{k+1}^p$.

For a first-order vocabulary σ, denote by $\Sigma_k^1(\sigma)$ the second-order sentences of the form $(\exists \mathbf{S}_1)(\forall \mathbf{S}_2) \cdots (Q\mathbf{S}_k)\varphi(\mathbf{S}_1, \ldots, \mathbf{S}_k)$, where the quantifiers alternate and φ is first order; each $\mathbf{S}_i = S_{i,1}, \ldots, S_{i,n_i}$ is a list of predicate variables of fixed arities.

PROPOSITION 2.1. [Fagin 1974; Stockmeyer 1977] *A database property $\mathcal{P}$ over a relational scheme $\mathbf{R}$ is in Σ_k^p, $k \geq 1$, iff there exists a $\Sigma_k^1(\mathbf{R})$-sentence Ψ such that for each $D \in \mathcal{D}(\mathbf{R})$, $D \models \Psi$ iff $\mathcal{P}(D) = true$.*

3. DISJUNCTIVE DATALOG

We assume that the reader is familiar with the basic concepts of deductive databases and logic programming [Lloyd 1984; Ullman 1989; Ceri et al. 1990]. Recall that datalog programs are function-free logic programs. We consider here the extension of datalog to disjunctive logic programs; for a background on disjunctive logic programming, see [Lobo et al. 1992; Minker 1994].

3.1 Syntax

A *disjunctive datalog rule* is a clause of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_m, \qquad 1 \leq n, \quad 0 \leq m, \tag{1}$$

over a function-free first-order language, where the a_i's are atoms forming the *head* of the clause, and the b_j's are literals (i.e., atoms or negated atoms) forming the *body*; a logical constant $=$ for equality is available, whose use is restricted to the body. For literals $\neg(t_1 = t_2)$ we write as usual $t_1 \neq t_2$. We use the terms predicates and relations (resp. relation symbols) as synonyms.

A *disjunctive datalog program* (simply program) is a finite collection π of clauses of form (1); by $RS(\pi)$ we denote the relations occurring in π. A program π is *stratified* if it is possible to decompose $RS(\pi)$ into pairwise disjoint sets $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_k$, such that: if a predicate $P \in \mathbf{P}_i$ occurs in the head of some rule of π and a predicate P' occurs in the same rule, then $P' \in \mathbf{P}_i$ if P' occurs in the head, and $P' \in \mathbf{P}_j$ if P' occurs in the body, where $j \leq i$ if it occurs positively and $j < i$ if it occurs negatively. Any such decomposition is a stratification of P; it induces a decomposition of π into subsets of rules $\pi_0, \ldots, \pi_k$ where π_i contains all rules that contain some predicate $P \in \mathbf{P}_i$ in the head.

A *query program* is a triple $(\pi, \mathbf{R}, \mathbf{S})$ of a program π and relation schemes $\mathbf{R}$, $\mathbf{S}$. The pair $(\mathbf{R}, \mathbf{S})$ is the *input/output scheme* of the query program, where $\mathbf{R}$ are the *extensional predicates* (i.e., the database relations), whose occurrence is restricted to the rule body, and $\mathbf{S}$ is a *subset* of the *intentional predicates* (i.e., the derived relations). Intuitively, the relations in $\mathbf{R}$ are the input relations and the relations in $\mathbf{S}$ are the output relations; the relations in $RS(\pi) - \mathbf{RS}$ (where $\mathbf{RS}$ are the predicates in $\mathbf{R}$ and $\mathbf{S}$) are referred to as *auxiliary relations*. An *input database* for the query program is any database $D \in \mathcal{D}(\mathbf{R})$.

For convenience, we identify a query program with π if $\mathbf{R}$ and $\mathbf{S}$ are clear from the context, and refer to it as the program π with input/output scheme $IO(\pi) = (\mathbf{R}, \mathbf{S})$.

EXAMPLE 1. Consider π with $IO(\pi) = (R, S)$:

$$\begin{aligned} S(x) \vee A(x,b) &\leftarrow R(x). \\ S(x) &\leftarrow \neg R(x), \neg B(a,x). \\ S(x) &\leftarrow A(x,b). \end{aligned}$$

Notice that π is stratified (choose $\mathbf{P}_0 = B, R$, $\mathbf{P}_1 = A, S$; then $\pi_0 = \emptyset$, $\pi_1 = \pi$). ■

We define a family of classes DATALOG^{X,Y} of query programs, based on independent syntactical restrictions X and Y concerning disjunction and negation, respectively, as follows. If X is void, then no disjunction is allowed (i.e., $n = 1$ in the rule (1)); if X is $\vee$, then disjunction is allowed. Y is one of the following increasingly liberal restrictions:

- Y is void : negation restricted to $\mathbf{R}$.
- Y is $\neq$: negation restricted to $\mathbf{R}$ and $=$ (i.e., inequality literals allowed).
- Y is $\neg_s$: stratified negation (π is stratified).
- Y is $\neg$: arbitrary negation.

For example, DATALOG$^{\vee}$, which we refer to as *plain disjunctive datalog*, allows disjunction in the head and negation of predicates from $\mathbf{R}$.[1] DATALOG$^{\vee,\neg}$, *disjunctive datalog*, is the language of all query programs.

3.2 Semantics

There are several proposals to capture the meaning of disjunctive logic programs, based on different concepts of the intended models of a program (cf. [Lobo et al. 1992; Minker 1994]). We consider minimal models [Minker 1982] (arising in the AI community in circumscription [McCarthy 1986], and in advanced forms of the Closed World Assumption, e.g. the Extended Generalized CWA [Yahya and Henschen 1985]), the perfect models [Przymusinski 1988], and the (disjunctive) stable models [Gelfond and Lifschitz 1988; Przymusinski 1991], which are among the most well-known such concepts. Using these concepts, we define different semantics μ of disjunctive datalog, which give rise to different query languages; the semantics of sublanguages is implicit.

We need some notation and concepts from (disjunctive) logic programming. For any program π, we denote by HU_π its Herbrand universe, i.e., the set of constants

[1] Notice that DATALOG$^{\vee}$ programs allow the use of positive equality atoms in the body. Equality atoms can be easily eliminated and are thus not essential.

occurring in π, and by HB_π its Herbrand base, i.e., the set of all ground atoms of predicates in $RS(\pi)$ over HU_π. The ground instantiation of π over universe U is denoted by $ground(\pi, U)$, or, if $U = HU_\pi$, simply by $ground(\pi)$. An *(Herbrand) interpretation* of π is a subset $I \subseteq HB_\pi$; it satisfies a ground rule (1) if $I \models b_i$ for every literal b_i in the body implies $I \models a_j$, for at least one a_j in the head. An interpretation I of π is a *model* of π, if it satisfies each rule in $ground(\pi)$.

DEFINITION 3.1. *A model M of π is* minimal, *if it does not contain any other model of π properly; by $MM(\pi)$ we denote the set of all minimal models of π.*

The disjunctive stable models of π are defined as follows. For any interpretation I, denote by π^I the *Gelfond-Lifschitz transform* of π w.r.t. I, which is obtained by removing from $ground(\pi)$ all clauses that contain a negative literal $\neg a$ in the body such that $a \in I$, and by removing all negative literals from the remaining clauses; Notice that π^I is a $\neg$-free grounded (i.e., variable-free) program.

DEFINITION 3.2. [Przymusinski 1991] *An interpretation M is a* (disjunctive) stable model *of π iff $M \in MM(\pi^M)$; by $SM(\pi)$ we denote the set of all stable models of π.*

Notice that each stable model of π is a minimal model, i.e., $SM(\pi) \subseteq MM(\pi)$ holds. However, the converse is not true in general.

Perfect models have been proposed earlier than disjunctive stable models in [Przymusinski 1988]. The definition of perfect models for arbitrary programs is provided in the appendix; we characterize here only the perfect models of stratified programs, for which the perfect models amount to the commonly agreed models. The characterization is known by the equivalence of perfect models and models of prioritized circumscription for stratified programs [Przymusinski 1988], and can be easily proved from properties and results below.

PROPOSITION 3.1. *An interpretation M is a* perfect model *of a stratified program π, iff for a stratification $\pi_0, \ldots, \pi_k$ of π, $M \cap HB_{\pi_i^*} \in MM({\pi_i^*}^{M_i})$, for $i = 0, \ldots, k$, where $\pi_i^* = ground(\pi_i, HU_\pi)$, $M_0 = \emptyset$ and $M_i = M \cap HB_{\pi_{i-1}^*}$, $i = 1, \ldots, k$.*

The set of all perfect models of π is denoted by $PM(\pi)$. Notice that like stable models, perfect models are always minimal models.

For convenience, we use in the sequel $\mathcal{S}$ as a generic variable for MM, PM, and SM. The following proposition states fundamental relationships between minimal, perfect, and stable models.

PROPOSITION 3.2. [Przymusinski 1991] *Let π be a stratified program. Then, $PM(\pi) = SM(\pi)$. Moreover, (i) if π is disjunction-free, then π has a unique perfect (resp. stable) model; (ii) if π is positive (i.e., $\neg$-free), then $PM(\pi) = SM(\pi) = MM(\pi)$.*

We remark that in [Bidoit and Froidevaux 1987] a semantics of stratified programs was proposed using default logic; it is equivalent to the stable (and also perfect) model semantics.

The evaluation of a query program over a database amounts to evaluating the ground program given by the facts in the database plus the instantiated rules. In order to circumvent a distinction between database predicates and intentional predicates (which is necessary for minimal models in case of negative literals involving database predicates), we reduce this ground program by propagating the database facts to rule bodies.

The instantiation of a program π on a database D, denoted by $\pi[D]$, is the program obtained from $ground(\pi, U)$ as follows. First, remove every rule that contains a literal in the body which is false in D (in particular, also equality and inequality atoms). Then, remove from bodies of the remaining rules all equality atoms, inequality atoms, and literals involving a relation from D.

Example 1 (ctd) Consider program π from above again ($IO(\pi) = (R; S)$):

$$\begin{aligned} S(x) \vee A(x,b) &\leftarrow R(x). \\ S(x) &\leftarrow \neg R(x), \neg B(a,x). \\ S(x) &\leftarrow A(x,b). \end{aligned}$$

Let $D = (\{a,b\}, \{a\}) \in \mathcal{D}(R)$. Then,

$ground(\pi, \{a,b\})$:	$\pi[D]$:
$S(a) \vee A(a,b) \leftarrow R(a).$	$S(a) \vee A(a,b) \leftarrow .$
$S(b) \vee A(b,b) \leftarrow R(b).$	
$S(a) \leftarrow \neg R(a), \neg B(a,a).$	
$S(b) \leftarrow \neg R(b), \neg B(a,b).$	$S(b) \leftarrow \neg B(a,b).$
$S(a) \leftarrow A(a,b).$	$S(a) \leftarrow A(a,b).$
$S(b) \leftarrow A(b,b).$	$S(b) \leftarrow A(b,b).$

■

Models of a program on a database are now defined as follows.

Definition 3.3. *Let π be a program and D be any database. The* minimal (resp. stable, perfect) models *of π on D, denoted by $MM(\pi, D)$ (resp. $SM(\pi, D)$, $PM(\pi, D)$), are the minimal models (resp. the stable models, perfect models) of the program $\pi[D]$.*

Example 1 (ctd) The minimal models of $\pi[D]$ are $M = \{S(a), S(b)\}$ and $M' = \{S(a), B(a,b)\}$. From these, M is perfect. Indeed, $\pi[D]$ is stratified by $\mathbf{P}_0 = B$, $\mathbf{P}_1 = A, S$, which induces $\pi_0^* = \pi_0 = \emptyset$, $\pi_1^* = \pi[D]$. Check that

$M \cap HB_{\pi_0^*} = \emptyset \in MM(\pi_0^{*M_0})$ ($\pi_0^{*M_0} = \emptyset$);

$M \cap HB_{\pi_1^*} = M \in MM(\pi_1^{*M_1})$ ($\pi_1^{*M_1} = \pi[D]^\emptyset$, which results from $\pi[D]$ by removing $\neg B(a,b)$.)

Hence, M is a perfect model of $\pi[D]$. On the other hand, M' is not perfect. Since the program is stratified, M is thus also the unique stable model. Hence $MM(\pi, D) = \{\, \{S(a), S(b)\},\ \{S(a), B(a,b)\}\, \}$ and $PM(\pi, D) = SM(\pi, D) = \{\, \{S(a), S(b)\}\, \}$. ■

The result of the query computed by a query program π (i.e., of $\mu(\pi)$) on an input database D is defined in terms of the models $\mathcal{S}(\pi, D)$, by taking as usual either the union of all models or the intersection [Abiteboul et al. 1990; Schlipf 1995b]. The former approach (union) is known as *possibility inference* (also termed brave or credulous reasoning in the AI community), and the latter (intersection) as *certainty inference* (also termed cautious or skeptical inference) [Imielinski and Lipski 1984; Abiteboul et al. 1990].

Definition 3.4. *Let inf be an inference modality from p(ossibility), c(ertainty).* $\mathrm{DATALOG}^{X,Y}_{inf,\mathcal{S}}$ *is the query language of* $\mathrm{DATALOG}^{X,Y}$ *query programs* π, $IO(\pi) = (\mathbf{R}, \mathbf{S})$, *with semantics* $\mu_{inf,\mathcal{S}}$, *such that* $\mu_{inf,\mathcal{S}}(\pi)$ *is the partial query*

$$q^{\pi}_{inf,\mathcal{S}} : \mathcal{D}(\mathbf{R}) \to \mathcal{D}(\mathbf{S})$$

such that $q^{\pi}_{inf,\mathcal{S}}(D)$ *is defined if* $HU_\pi \subseteq U$ *(i.e., all constants in* π *belong to the universe of* D*), and depending on inf, for each* $S_i \in \mathbf{S}$ *and tuple* $\mathbf{t} \in U^{a(S_i)}$,

$$\begin{array}{ll} (\textit{possibility inference}) & q^{\pi}_{p,\mathcal{S}}(D) \models S_i(\mathbf{t}) \;\textit{ iff }\; S_i(\mathbf{t}) \in \bigcup_{M \in \mathcal{S}(\pi,D)} M; \\ (\textit{certainty inference}) & q^{\pi}_{c,\mathcal{S}}(D) \models S_i(\mathbf{t}) \;\textit{ iff }\; S_i(\mathbf{t}) \in \bigcap_{M \in \mathcal{S}(\pi,D)} M. \end{array}$$

For example, $\mathrm{DATALOG}^{\vee,\neg}_{p,SM}$ denotes $\mathrm{DATALOG}^{\vee,\neg}$ with possibility stable model semantics.

Technically, we have defined the semantics of any disjunctive datalog query program for any kind $\mathcal{S}$ of model. The intended use of minimal models, however, is for $\mathrm{DATALOG}^{\vee,\neq}$, and of perfect models for $\mathrm{DATALOG}^{\vee,\neg_s}$. The relationships between minimal, perfect, and stable models of a program (Proposition 3.2) imply the coincidence of several query languages $\mathrm{DATALOG}^{X,Y}_{inf,\mathcal{S}}$, which motivates a simplification of notation ($inf = p, c$):

Simplified Notation	Collapsing query languages
$\mathrm{DATALOG}^{\vee,\neq}_{inf}$	$\mathrm{DATALOG}^{\vee,\neq}_{inf,\mathcal{S}}$, $\mathcal{S} = MM, PM, SM$
$\mathrm{DATALOG}^{\vee,\neg_s}_{inf}$	$\mathrm{DATALOG}^{\vee,\neg_s}_{inf,\mathcal{S}}$, $\mathcal{S} = PM, SM$
$\mathrm{DATALOG}^{\neq}$	$\mathrm{DATALOG}^{\neq}_{inf,\mathcal{S}}$, $\mathcal{S} = MM, PM, SM$
$\mathrm{DATALOG}^{\neg_s}$	$\mathrm{DATALOG}^{\neg_s}_{inf,\mathcal{S}}$, $\mathcal{S} = PM, SM$

The simplified notation is inherited to sublanguages (e.g., DATALOG is familiar datalog with negation of input relations).

In this paper, we will disregard $\mathrm{DATALOG}^{\vee,\neq}_{c}$, i.e., the certainty variant of the *MM*-semantics, and restrict our attention to $\mathrm{DATALOG}^{\vee,\neq}_{p}$. The reason is the well-known fact that a positive atom is in all minimal models iff it is in all models. Thus, the certainty *MM*-semantics does not provide model selection with respect to positive atoms, and computes the same query result as classical (certainty) semantics. Therefore, $\mathrm{DATALOG}^{\vee,\neq}_{c}$ falls back to query languages in co-NP which are well-understood (cf. FN-queries in [Chandra and Harel 1985]). For the other semantics (*SM*,*PM*), the certainty variant is precisely symmetric to the possibility variant if (stratified) negation is allowed. For instance, the boolean (constant-free) $\mathrm{DATALOG}^{\vee,\neg}_{p,SM}$ queries express the Σ^p_2 database properties and the boolean (constant-free) $\mathrm{DATALOG}^{\vee,\neg}_{c,SM}$ queries express the Π^p_2 database properties. Since results on the certainty variant of *SM*-semantics and *PM*-semantics are mostly straightforward from results on the possibility variant, we will keep the treatment of the certainty variant short.

4. APPLICATIONS

The extension of datalog with disjunction increases the expressive capability. It is possible to formulate many queries which can not be expressed in standard datalog. Among these queries are classical optimization problems such as Traveling Salesman, Maximum Clique, Minimal Vertex Cover etc. [Papadimitriou 1994] which are NP-hard and NP-easy, i.e., solvable in polynomial time with an oracle for NP.

In this section, we show several practical queries that can be formulated in disjunctive datalog, but not in standard datalog unless PH collapses, which is generally believed to be false (see [Papadimitriou 1994]). In our examples involving optimization problems, we use integers $0, 1, \ldots$ and the standard arithmetic operations (resp. predicates) of addition and multiplication. These operations can be easily defined from the standard successor relation; notice that we show in Section 6 how an enumeration of the database universe, which corresponds to the usual successor function on an interval of the integers, can be generated through a DATALOG$^{\vee,\neq}$ program.

For the sake of brevity, we omit in the examples detailed formal proofs for the correctness of query programs. Such proofs are not difficult to obtain, however. For this concern, we also provide a formal result on a useful modularity property of disjunctive datalog in the next section, which generalizes the concept of modularity embodied by stratified programs.

In our examples, we use the following notational convention. Rules of the form

$$\perp \leftarrow l_1, \ldots, l_m.$$

are a shorthand notation for

$$\begin{aligned} False &\leftarrow l_1, \ldots, l_m. \\ A &\leftarrow False, \neg A. \end{aligned}$$

where $False$ and A are new propositional letters. This proviso emulates an integrity constraint $\leftarrow l_1, \ldots, l_m$, as $False$ must be false in every stable model.[2]

Example 2. (Graph Coloring) Graph colorings have important applications in many areas. We show here that 3-colorability of a graph $G = (V, E)$, as well as 3-uncolorability, can be decided in DATALOG$^{\vee}$, without any use of negation. Since these queries are NP- and co-NP-complete respectively, not both are expressible in the same variant of the stable model semantics (possibility or certainty) of DATALOG$^{\neg}$ programs unless NP= co-NP.

A simple program for deciding 3-colorability is the following one:

$$\begin{aligned} Red(x) \vee Green(x) \vee Blue(x) &\leftarrow . \\ NotColored &\leftarrow E(x,y), Red(x), Red(y). \\ NotColored &\leftarrow E(x,y), Green(x), Green(y). \\ NotColored &\leftarrow E(x,y), Blue(x), Blue(y). \\ NotColored \vee Colored &\leftarrow . \end{aligned}$$

The first rule assigns each vertex of G one of three colors. If the assignment is not a legal coloring, then $NotColored$ is derived by one of the three subsequent

[2]Alternatively, we could extend the formalism by a designated atom $\perp$ which is false in every interpretation. While straightforward, for simplicity we do not consider such an extension.

rules. In this case, $Colored$ takes the value false in the minimal model. On the other hand, if $NotColored$ is not derivable from the assignment, a minimal model is obtained by including $Colored$. Thus, on any graph G, the program computes $Colored$ true under the possibility variant of the minimal model semantics iff G is 3-colorable.

Notice that 3-colorability can be expressed in DATALOG$^\neg$ under the possibility variant of the stable model semantics. A respective program can be obtained from the preceding one by replacing the clause $Red(x) \vee Green(x) \vee Blue(x) \leftarrow$ with the three clauses

$$\begin{aligned} Red(x) &\leftarrow \neg Green(x), \neg Blue(x). \\ Green(x) &\leftarrow \neg Blue(x), \neg Red(x). \\ Blue(x) &\leftarrow \neg Red(x), \neg Green(x). \end{aligned}$$

and the clause $NotColored \vee Colored \leftarrow$ with

$$Colored \leftarrow \neg NotColored.$$

A disjunctive program for deciding 3-uncolorability can be obtained from the disjunctive program for 3-colorability by the following modification: drop the rule $NotColored \vee Colored \leftarrow .$ and add instead the rules

$$\begin{aligned} Red(x) &\leftarrow NotColored. \\ Green(x) &\leftarrow NotColored. \\ Blue(x) &\leftarrow NotColored. \end{aligned}$$

These rules generate the maximal possible extension for the intentional predicates Red, $Green$, and $Blue$ if $NotColored$ is true, and effect that the (trivial) model consisting of all ground atoms is the only model if each possible assignment of colors to the vertices does not yield a 3-coloring. In fact, if an assignment is a 3-coloring, then the program has a model in which $NotColored$ is not contained; consequently, $NotColored$ is not contained in any minimal model. Thus, the graph G is not 3-colorable iff the program has on G a minimal model containing $NotColored$. Consequently, the program computes $NotColored$ true on input G under the possibility (as well the certainty) variant of the MM-semantics iff G is not 3-colorable.

Notice that no simple modification of the previous program yields a DATALOG$^\neg$ program which decides 3-uncolorability under the possibility variant of the stable model semantics; no such program exists unless NP= co-NP.

In Section 8, we will encounter a generalized 3-coloring problem that can be expressed by a simple, intuitive DATALOG$^\vee$ program; however, that query is Σ_2^p-complete, and can thus not be expressed in DATALOG$^\neg$ under either possibility or certainty semantics. ■

Example 3. (Traveling Salesman) It is well-known that finding an optimal solution to the Traveling Salesman Problem (TSP) is intractable. Recall that the problem is, given cities $c_1, \ldots, c_n$, find a round trip that visits all cities in sequence and has minimal traveling cost, i.e., a permutation τ of $1, \ldots, n$ such that

$$c(\tau) = \sum_{i=1}^{n-1} c(\tau(i), \tau(i+1)) + c(\tau(n), \tau(1))$$

is minimum, where $c(i,j)$ is the cost of traveling from c_i to c_j, given by an integer.

Computing an optimal tour is both NP-hard and co-NP-hard. In fact, in [Papadimitriou 1984] it was shown that deciding whether the cost of an optimal tour is even is Δ_2^p-complete, and in [Krentel 1988] it was shown that computing this cost (or even a single bit) is $F\Delta_2^p$-complete under so called metric reductions. Hence, this is not possible in DATALOG$^\neg$ (unless PH collapses). Here, we describe a disjunctive datalog program π which can do this under the stable semantics.

Suppose that the cities are encoded by $1,\ldots,n$ and that the intercity traveling costs are stored in a relation $C(i,j,v)$ where $v = c(i,j)$. We assume that a built-in predicate is available which tells whether $x = y+z$. In abuse of notation, we simply refer to the number n of cities (which is provided by an input relation) by itself.

The following program π_1 computes legal tours and their costs in its stable models:

$$
\begin{array}{lrl}
(1) & T(i,j) \vee \tilde{T}(i,j) & \leftarrow i \leq n, j \leq n. \\
(2) & \bot & \leftarrow T(i,j), T(i,k), j \neq k. \\
(3) & \bot & \leftarrow T(i,k), T(j,k), i \neq j. \\
(4) & Chosen\_Stop(i) & \leftarrow T(i,j). \\
(5) & \bot & \leftarrow \neg Chosen\_Stop(i), 1 \leq i \leq n. \\
(6) & P\_Value(n,x) & \leftarrow T(n,i), T(1,j), C(i,j,x). \\
(7) & P\_Value(k-1,x) & \leftarrow P\_Value(k,y), T(k-1,i), T(k,j), C(i,j,z), x = y+z. \\
(8) & Cost(x) & \leftarrow P\_Value(1,x).
\end{array}
$$

The first clause (1) guesses a tour, where $T(i,j)$ intuitively means that the i-th stop of the tour is city j and $\tilde{T}(i,j)$ that it's not.[3] By the minimality of a stable model, exactly one of $T(i,j)$ and $\tilde{T}(i,j)$ is true in it, for each i and j such that $1 \leq i,j \leq n$; in all other cases, both are false.

The subsequent clauses (2)–(5) check that the guess is proper: each stop has attached at most one city, each city can be attached to at most one stop, and every stop must have attached some city. The rules (6)–(8) compute top down the cost of the chosen tour, which is given by the (unique) atom $Cost(x)$ contained in the model.

It holds that the stable models of π_1 correspond one-to-one to the legal tours.

To reach our goal, we have to eliminate from them those which do not correspond to optimal tours. That is, we have to eliminate all tours T such that there exists a tour T' which has lower cost. This is performed by a DATALOG$^\vee$ program, which basically tests all choices for a tour T' and rules out each choice that is not a cheaper tour, which is indicated by a propositional atom $NotCheaper$. The following program, which is similar to π_1, generates all possible choices for T':

$$
\begin{array}{lrl}
(1') & T'(i,j) \vee \tilde{T}'(i,j) & \leftarrow 1 \leq i \leq n, 1 \leq j \leq n. \\
(2') & NotCheaper & \leftarrow T'(i,j), T'(i,k), j \neq k. \\
(3') & NotCheaper & \leftarrow T'(i,k), T'(j,k), i \neq j. \\
(4') & NotChosen\_Stop(i,1) & \leftarrow \tilde{T}'(i,1).
\end{array}
$$

[3]This choice can also be implemented by the two nondisjunctive clauses $T(i,j) \leftarrow \neg\tilde{T}(i,j), \leq i \leq n, j \leq n.$ and $\tilde{T}(i,j) \leftarrow \neg T(i,j), i \leq n, j \leq n.$ Note that the resulting program is nondisjunctive.

$$
\begin{array}{lrcl}
(5') & NotChosen\_Stop(i,j) & \leftarrow & \tilde{T}'(i,j), NotChosen\_Stop(i,j-1). \\
(6') & NotCheaper & \leftarrow & NotChosen\_Stop(i,n). \\
(7') & P\_Value'(n,x) & \leftarrow & T'(n,i), T'(1,j), C(i,j,x). \\
(8') & P\_Value'(k-1,x) & \leftarrow & P\_Value'(k,y), T'(k-1,i), T'(k,j), \\
(9') & Cost'(x) & \leftarrow & P\_Value'(1,x).
\end{array}
$$

The predicates T', $\tilde{T}'$, P_Value' and $Cost'$ have the rôle of the predicates T, $\tilde{T}$, P_Value and $Cost$ in π_1. Since we do not allow negation, the test that for each stop a city has been chosen (rules (4)–(5) in π_1) has to be implemented differently (rules (4′)–(6′)). $NotChosen\_Stop(i,j)$ tells whether no city $\leq j$ has been chosen for stop i. Thus, if $NotChosen\_Stop(i,n)$ is true, then no city has been chosen for stop i, and the choice for T' does not correspond to a legal tour.

The minimal models of (1′)–(9′) which do not contain $NotCheaper$ correspond one-to-one to all legal tours. By adding the following rule, each of them is eliminated which does not have smaller cost than the tour given by T:

$$
(10') \quad NotCheaper \leftarrow Cost(x), Cost'(y), x \leq y.
$$

Thus, if for a legal tour T, each choice for T' leads to the derivation of $NotCheaper$, then T is an optimal tour.

For the desired program, we add the following rules, applying a similar technique as in the program for graph 3-uncolorability from above:

$$
\begin{array}{lrcl}
(11') & \perp & \leftarrow & \neg NotCheaper. \\
(12') & P(x_1,\ldots,x_n) & \leftarrow & NotCheaper. \quad ,
\end{array}
$$

for any predicate P that occurs in a rule head of (1′)–(9′) except $NotCheaper$. The first rule enforces that $NotCheaper$ must be contained in the stable model; consequently, it must be derivable. The other rules derive the maximal extension for each predicate P if $NotCheaper$ is true, which is a trivial model for (1′)–(9′). In fact, it is for some given tour T the only model if no choice for T' leads to a tour with cost smaller than the cost of T; otherwise, there exists another model, which does not contain $NotCheaper$.

Let π_2 be the program consisting of the rules (1′)–(12′). Then, it holds that the stable models of $\pi = \pi_1 \cup \pi_2$ on any instance of TSP correspond to the optimal tours.[4] In particular, the optimal cost value, described by $Cost(x)$, is contained in each stable model. Thus, the program π computes on any instance of TSP under the possibility (as well as certainty) stable model semantics in $Cost$ the cost of an optimal tour.

Notice that the above program can be extended for further discrimination among different optimal tours by using an order (e.g., the lexicographical order on the vectors $(\pi(1),\ldots,\pi(n))$ corresponding to permutations π; see Example 4 for an implementation of lexicographic ordering) and output the first optimal tour w.r.t. the order. ■

Example 4. (Eigenvectors) The need for doing matrix computations in databases has been recently emphasized [Maier and Vance 1993]. In this context, we consider

[4]Here, we suppose that the provided universe U of the database storing the instance is sufficiently large for computing the tour values.

queries connected with eigenvector computations. Recall that an *eigenvalue* of an $n \times n$ matrix $M = (m_{i,j})$ is a number λ such that the system

$$M \cdot \mathbf{x}^T = \lambda \mathbf{x}^T \qquad \text{i.e., } x_i = \textstyle\sum_{k=1}^{n} m_{i,k} \cdot x_k,\ i = 1, \ldots, n$$

has nontrivial solutions, where $\mathbf{x}^T$ is the transpose of the vector $\mathbf{x} = (x_1, \ldots, x_n)$. Every such nontrivial solution $\mathbf{x}$ is called an *eigenvector* (w.r.t. λ). Eigenvalues/eigenvectors have important applications in many areas, e.g. to structural analysis, quantum chemistry, power system analysis, stability analysis, VLSI design, and geophysics [Cullum and Willoughby 1985].

It is well-known that the eigenvectors of a real matrix M form with the zero vector a vector space, and thus in general infinitely many eigenvectors exist. We consider here the problem of computing integer eigenvectors $\mathbf{e} = (e_1, \ldots, e_n)$ such that $|e_i| \leq c$, for every $i = 1, \ldots, n$, where c is an integer constant.[5] We exhibit natural queries on such eigenvectors which can not be expressed in DATALOG$^{\neg}$ but in disjunctive datalog (unless PH collapses).

Suppose that an integer $n \times n$ matrix M is stored in a database using a 3-ary relation M such that $(i, j, v) \in M$ iff $m_{i,j} = v$. We use numbers $1, \ldots, n$ with the usual successor relation to index the entries of M. As above, we refer in abuse of notation to the predecessor of i simply as $i - 1$, and to the dimension n of M by itself.

In the sequel, we tacitly restrict to bounded eigenvectors. For arithmetic, we use built-in predicates for telling whether $x = y + z$ and $x = y \cdot z$, respectively.

Consider the following program π_1, which computes the eigenvectors of M in its stable models:

$$\begin{array}{lrl}
(1) & E(i,x) \vee \tilde{E}(i,x) & \leftarrow 1 \leq i \leq n,\ -c \leq x \leq c. \\
(2) & \perp & \leftarrow E(i,x), E(i,y), x \neq y. \\
(3) & Chosen(i) & \leftarrow E(i,x). \\
(4) & \perp & \leftarrow \neg Chosen(i), 1 \leq i \leq n. \\
(5) & S(i,1,x) & \leftarrow M(i,1,y), E(1,z), x = y \cdot z. \\
(6) & S(i,k,x) & \leftarrow S(i,k-1,y), M(i,k,v), E(k,w), z = v \cdot w, x = y + z. \\
(7) & \perp & \leftarrow E(i,y), S(i,n,x), x \neq y.
\end{array}$$

The first rules (1) implement a choice for an eigenvector e from all bounded n-vectors.[6] Here, $E(i, x)$ represents that the i-th component of the vector has value x. By the minimality of a stable model, exactly one of the atoms $E(i, x)$ and $\tilde{E}(i, x)$ is true for each pair of i and x such that $1 \leq i \leq n$ and $-c \leq x \leq c$, and both are false in each other case. The rules (2)-(4) check whether the guess is proper, i.e., for each component of e precisely one integer has been chosen. The eigenvector condition is checked by rule (7); to this end, the value of $\sum_{j=1}^{k} m_{i,j} \cdot e_k$ is computed bottom up in $S(i, k, x)$ by the rules (5)–(6).

It holds that, on any input matrix M, the stable models of program π_1 correspond to the eigenvectors of M in the obvious way.

---

[5]Notice that in practice, integers are usually supported on an interval $[-c, c-1]$ for some (large) constant c, with arithmetic modulo c.

[6]Similar as in the previous example, this choice can alternatively be implemented through nondisjunctive clauses $E(i,j) \leftarrow \neg\tilde{E}(i,j), 1 \leq i \leq n, -c \leq j \leq c.$ and $\tilde{E}(i,j) \leftarrow \neg E(i,j), 1 \leq i \leq n, -c \leq j \leq c.$

Now assume that we are interested in eigenvectors $\mathbf{e}$ that have maximum norm, i.e., $\|\mathbf{e}\| = \sqrt{\sum_{i=1}^{n} e_i \cdot e_i}$ is maximum. In particular, we pose the query whether such an $\mathbf{e}$ exists where a particular component, say e_1, has a specific value, e.g. $e_1 = 0$.

In the program π_1 above, we essentially have to restrict the stable models to those that contain $E(1,0)$ and whose corresponding eigenvectors have maximal norm. The former is easily achieved, by simply adding to π_1 the rule

$$E(1,0) \leftarrow .$$

Implementing the latter is more cumbersome. Given a stable model corresponding to an eigenvector $\mathbf{e}$, testing whether $\mathbf{e}$ is maximal amounts to deciding that every eigenvector $\mathbf{e}'$ satisfies $\|\mathbf{e}'\| \leq \|\mathbf{e}\|$. This test is implemented by a DATALOG$^\vee$ program, which basically considers all possible choices for a bounded n-vector, and rules out each choice that does not correspond to an eigenvector $\mathbf{e}'$ with greater norm than $\mathbf{e}$ by deriving a propositional atom $NotGreater$. If all choices are rules out, then $\mathbf{e}$ is in fact maximal.

Consider the following program, which is similar to the program π_1:

$$
\begin{array}{lrl}
(1') & E'(i,x) \vee \tilde{E}'(i,x) & \leftarrow 1 \leq i \leq n, -c \leq x \leq c. \\
(2') & NotGreater & \leftarrow E'(i,x), E'(i,y), x \neq y. \\
(3') & NotChosen(i,-c) & \leftarrow \tilde{E}'(i,-c). \\
(4') & NotChosen(i,j) & \leftarrow \tilde{E}'(i,j), NotChosen(i,j-1). \\
(5') & NotGreater & \leftarrow NotChosen(i,c). \\
(6') & S'(i,1,x) & \leftarrow M(i,1,y), E(1,z), Prod(x,y,z). \\
(7') & S'(i,k,x) & \leftarrow S'(i,k-1,y), M(i,k,v), E(k,w), z = v \cdot w, x = y + z. \\
(8') & NotGreater & \leftarrow E(i,y), S'(i,n,x), x \neq y.
\end{array}
$$

Here, $E', \tilde{E}'$ and S' have the rôle of E, $\tilde{E}$ and S in π. The first rule (1′) guesses a bounded vector; the validity of the guess is checked by the rules (2′)–(5′). Since we may not use negation, the test that some value has been chosen for component i is implemented differently (rules (3′)–(5′)). Here $NotChosen(i,j)$ tells that no number $\leq j$ has been chosen for component i. Thus, if $NotChosen(i,c)$ is true, no number has been chosen for component i, and $NotGreater$ is derived.

The stable models of this program (which coincide with the minimal ones) that do not contain $NotGreater$ correspond one-to-one to the eigenvectors of M. By adding the following rules, we can rule out all eigenvectors $\mathbf{e}'$ with $\|\mathbf{e}'\| \leq \|\mathbf{e}\|$:

$$
\begin{array}{lrl}
(9') & N(1,x) & \leftarrow E(1,y), x = y \cdot y. \\
(10') & N(i,x) & \leftarrow N(i-1,v), E(i,y), z = y \cdot y, x = v + z. \\
(11') & N'(1,x) & \leftarrow E'(1,y), x = y \cdot y. \\
(12') & N'(i,x) & \leftarrow N'(i-1,v), E'(i,y), z = y \cdot y, x = v + z. \\
(13') & NotGreater & \leftarrow N(n,x), N'(n,y), y \leq x.
\end{array}
$$

The first two rules compute bottom up $\sum_{i=1}^{n} e_i \cdot e_i = \|\mathbf{e}\|^2$ in $N(n,x)$, and analogously the subsequent two rules $\sum_{i=1}^{n} e_i' \cdot e_i' = \|\mathbf{e}'\|^2$ in $N'(n,x)$. The last rule derives $NotGreater$ iff $\|\mathbf{e}'\|^2 \leq \|\mathbf{e}\|^2$, which is equivalent with $\|\mathbf{e}'\| \leq \|\mathbf{e}\|$.

Thus, whenever for the eigenvector $\mathbf{e}$ given by a stable model of π, $NotGreater$ is derived upon all possible choices for an eigenvector $\mathbf{e}'$, then $\mathbf{e}$ is maximal. The checking part is added, together with the following linking rules, where P ranges over all predicates in rule heads of the checking part except $NotGreater$:

$$
\begin{array}{lr}
(14') & \perp \leftarrow \neg NotGreater. \\
(15') & P(x_1, \dots, x_n) \leftarrow NotGreater.
\end{array}
$$

The first rule states that $NotGreater$ must be derivable; the other rules generate, applying a similar technique as in the program for 3-uncolorability in Example 2, the maximal extension for every predicate P defined in the checking part if $NotGreater$ is true. This is trivially a model, and in fact the only one if $NotGreater$ is in each model of the checking part; this, however, is true precisely if the choice for E corresponds to a maximal eigenvector **e**.

Let π_2 be the program comprising the rules $(1')$–$(15')$. Then, the stable models of the overall program $\pi = \pi_1 \cup \pi_2$ correspond one-to-one to the maximal eigenvectors **e** of M that have $e_1 = 0$. Since $NotGreater$ is contained in every stable model, it follows that π computes $NotGreater$ true on input M and I under the possibility stable model semantics iff such an eigenvector **e** exists.

Notice that the use of disjunction in the checking part of π is crucial as opposed to the part generating all eigenvectors **e**, and can most likely not be eliminated. One can show that the computed query is both NP- and co-NP-hard; precisely, the query is complete for the class Θ_2^p $(= \Delta_2^p[O(\log n)]$, i.e. the class of problems solvable in polynomial time with logarithmically many NP oracle queries) [Eiter and Gottlob 1995a]. Thus, the query can not be formulated in DATALOG$^{\neg}$ (unless PH collapses).

Computing a single maximal eigenvector does not specify a generic query, unless this vector is uniquely determined. By discrimination among the maximal eigenvectors, it is possible to single out a unique such vector. A natural discrimination is by lexicographical ordering, so that the task is to compute the first maximal eigenvector in this order. This can be accomplished by the following clauses:

$$
\begin{array}{ll}
(20) & LexEqual(1) \leftarrow E(1,x), E'(1,x). \\
(21) & LexEqual(i) \leftarrow LexEqual(i-1), E(i,x), E'(i,x). \\
(22) & NotBefore \leftarrow E(1,x), E'(1,y), x < y. \\
(23) & NotBefore \leftarrow LexEqual(i-1), E(i,x), E'(i,y), x < y. \\
(24) & NotBefore \leftarrow LexEqual(n). \\
(25) & NotGreater \leftarrow NotBefore.
\end{array}
$$

The propositional letter $NotBefore$ is derived if the vector corresponding to E' is not listed before the one corresponding to E in the lexicographic ordering, and rules out the choice of E' for a proof that E is not the first maximal eigenvector. Note that computing the lexicographically first maximal eigenvector is an $\mathrm{F}\Delta_2^p$-complete query [Eiter and Gottlob 1995a], and is thus not expressible in DATALOG$^{\neg}$ (unless PH collapses).

Even harder queries on eigenvectors can be expressed. For example, suppose eigenvectors that are maximal modulo a specified subset I of the components are considered, i.e., the maximal eigenvectors w.r.t. to the partial order $\preceq_I$, where $\mathbf{x} \preceq_I \mathbf{y}$ iff **x** and **y** coincide on the components in I and $\|\mathbf{x}\| \leq \|\mathbf{y}\|$. The query whether a maximal such eigenvector **e** exists where $e_1 = 0$ can be expressed by adding to the program π_1 from above the clause

$$(30) \qquad NotGreater \leftarrow I(i), E(i,x), E'(i,y), x \neq y.$$

(assuming that I is provided by a unary input relation). This query is Σ_2^p-complete [Eiter and Gottlob 1995a], and thus much harder than the previous queries; it is among the hardest queries that can be expressed in disjunctive datalog. ■

In the above examples, candidate solutions are generated by one component of the program and tested by another. This is reminds of datalog extended with a choice construct, by which such candidates can be nondeterministically generated. As shown by Saccá and Zaniolo [Saccà and Zaniolo 1990], *choice* can be be formulated using stable models of a normal program that has unstratified rules. However, in this setting, the testing phase where the validity of a choice is checked is limited to a polynomial time computation. On the other hand, as shown by Examples 3 and 4, disjunctive datalog is capable of expressing problems whose testing phase after the choice is in co-NP and might even bear an co-NP-complete problem. Therefore, unless the Polynomial Hierarchy collapses, those examples can not be formulated in datalog with choice. Note that choice can be implemented using head-cycle-free disjunction [Ben-Eliyahu and Dechter 1994]. Loosely speaking, a program is headcycle-free if no two distinct atoms which occur together in the head of a rule mutually depend on each other. Since, by the results in [Ben-Eliyahu and Dechter 1994; Eiter et al. 1996], headcycle-free programs capture NP (resp. co-NP) under the possibility (resp. certainty) variant of the stable model semantics, it is not surprising that the programs exhibited in Example 3 and 4 are not headcycle-free. In fact, for the respective queries no such programs exist (unless the Polynomial Hierarchy collapses).

5. MODULARITY OF DISJUNCTIVE DATALOG

In the examples of the previous section, we have designed query programs in a hierarchical fashion. Following the guess and check paradigm, we designed a basic module π_1 which generates a number of possible solution candidates, and a module π_2 that takes the output of π_1 (i.e., the possible solution candidates) and eliminates from them those which do not fulfill certain criteria. Such a methodology, which is quite natural, should undoubtedly be possible in a reasonable query language. In fact, if a query program grows more complex, it must be possible to structure the program into modules in order not to lose the general view and to understand more easily how the program works. This requirement is well-acknowledged, and led to the formulation of desirable modularity properties for normal and disjunctive logic programs, cf. [Schlipf 1995b; Dix 1992; Ullman 1989; Ross 1994; Lifschitz and Turner 1994]. The notion of modularity of programs is related to invariance under unfolding (cf. [Brass and Dix 1995]) which was independently established for the static semantics [Przymusinski 1994; Przymusinski 1995] and for the *D-WFS* semantics [Brass and Dix 1995].

Notice that stratified programs enjoy such a modularity property. If $\pi_0, \pi_1, \ldots, \pi_k$ is a stratification of a program π, then each stratum π_i can be seen as a module which is on top of all lower strata, i.e. $\bigcup_{j<i} \pi_j$. We consider here a more general concept, in which modules can be even unstratified programs (cf. [Ullman 1989]);

it shows that hierarchical structured query programming is possible in disjunctive datalog using stable models. This makes the language, besides its declarative nature, appealing as a powerful extension of familiar datalog.

The following definition states a syntactic condition for the intuition that a module π_2 is placed on top of a module π_1.

Definition 5.1. *Let π_1 and π_2 be programs. We say that π_2 potentially uses π_1, or π_1 is independent of π_2 (denoted $\pi_2 \triangleright \pi_1$), iff each predicate that occurs in some rule head of π_2 does not occur (positively or negatively) in π_1.*

Example 5. Consider the following ground programs π_1 and π_2:

$$\begin{array}{llll} \pi_1: & D(a) \vee B(c) \leftarrow A(c,a), \neg A(c,b). & \pi_2: & E(a,a) \vee F(a,b) \leftarrow \neg C(b), D(a). \\ & A(c,a) \leftarrow \neg C(c), \neg B(a). & & \end{array}$$

It holds that $\pi_2 \triangleright \pi_1$. Notice that for the split of π_1 into the subprograms $\pi_{1,1}$ and $\pi_{1,2}$ consisting of the first and the second rule, respectively, $\pi_{1,1} \not\triangleright \pi_{1,2}$ and $\pi_{1,2} \not\triangleright \pi_{1,1}$. (In fact, π_1 is unstratifiable.) ■

Further examples are the programs for the Traveling Salesman Problem and the eigenvector problems in Section 4. There, we have that $\pi_2 \triangleright \pi_1$.[7] Notice that, if $\pi = \pi_1 \cup \pi_2$ such that $\pi_2 \triangleright \pi_1$, then $ground(\pi_2, C) \triangleright ground(\pi_1, C)$ for every set of constants C; in particular, $\pi_2[D] \triangleright \pi_1[D]$ for every database D.

Intuitively, if $\pi_2 \triangleright \pi_1$ and $\pi = \pi_1 \cup \pi_2$ is a ground, then the stable models of $\pi_1 \cup \pi_2$ are obtained by computing the stable models of π_2, augmented with all facts $a \leftarrow .$ such that atom a is true in a particular stable model of π_1. Indeed, the rules of π_2 cannot have any effect on the rules of π_1. This intuition is formally justified by the next lemma. For any program π and interpretation M, let $M \cup \pi$ denote the program $\pi \cup \{ a \leftarrow . \mid a \in M \}$.

Lemma 5.1. [8] *Let $\pi = \pi_1 \cup \pi_2$ be a ground program such that $\pi_2 \triangleright \pi_1$. Then,*
(*i*) *for every $M \in SM(\pi)$, $M \cap HB_{\pi_1} \in SM(\pi_1)$.*
(*ii*) *$SM(\pi) = \bigcup_{M \in SM(\pi_1)} SM(M \cup \pi_2)$.*

Proof.
(*i*): Let $M \in SM(\pi)$ and let $M_1 = M \cap HB_{\pi_1}$. Assume $M_1 \notin SM(\pi_1)$. Clearly, M_1 is a model of $\pi_1^{M_1}$; hence, there exists an $M' \in MM(\pi_1^{M_1})$ such that $M' \subset M_1$. Let $M^* = (M - HB_{\pi_1}) \cup M'$; note that $M^* \subset M$. We show that M^* is a model of π^M, which raises a contradiction to the hypothesis $M \in SM(\pi)$. Clearly, M^* satisfies $\pi_1^{M_1}$. Since $M_1 \subseteq M$, we have that $\pi_1^M \subseteq \pi_1^{M_1}$; hence, M^* also satisfies π_1^M. Moreover, M^* also satisfies each clause of $\pi^M - \pi_1^M$. In fact, consider any such clause $C = r_1 \vee \cdots \vee r_n \leftarrow q_1, \cdots, q_m$. Assume C is not satisfied by M^*, i.e., each q_j is satisfied by M^* but none of the r_i. Since $\pi_2 \triangleright \pi_1$ and π_1, π_2 are ground, each r_i belongs to $HB_{\pi_2} - HB_{\pi_1}$; by definition, M^* and M coincide on $HB_{\pi_2} - HB_{\pi_1}$, thus M does not satisfy r_i. On the other hand, since each q_j is a

[7]In lack of a designated constant $\perp$ for falsity, we make here the implicit assumption that each denial $\perp \leftarrow l_1, \ldots, l_m$ is rewritten using local fresh propositional atoms A and $False$ (cf. Section 4).
[8]The lemma can be generalized to nonground programs, provided that in general constants occurring only in π_2 (resp. π_1) are accessible to variables in π_1 (resp. π_2). This is e.g. trivially true if $ground(\pi) = ground(\pi_1) \cup ground(\pi_2)$.

positive literal and $M^* \subset M$, M satisfies each q_j. Consequently, M does not satisfy C, which implies $M \notin MM(\pi^M)$, and hence $M \notin SM(\pi_1 \cup \pi_2)$; thus we arrived at the desired contradiction. It follows $M_1 \in SM(\pi_1)$.

(ii): $(\subseteq)$. Let $M_0 \in SM(\pi)$ and let $M = M_0 \cap HB_{\pi_1}$. By (i), $M \in SM(\pi_1)$; it thus suffices to show that $M_0 \in SM(M \cup \pi_2)$. Clearly, M_0 is a model of $(M \cup \pi_2)^{M_0}$. Assume there exists a smaller model $M^* \subset M_0$ of $(M \cup \pi_2)^{M_0}$. Clearly, $M \subseteq M^* \cap HB_{\pi_1}$; hence, it follows that $M \subseteq M^* \cap HB_{\pi_1} \subseteq M_0 \cap HB_{\pi_1} = M$, which implies $M^* \cap HB_{\pi_1} = M$. It follows that $M^* \cap HB_{\pi_1}$ is a model of $\pi_1^{M_0}$; since π_1, π_2 are ground, it follows that M^* is a model of π^{M_0}. This is in contradiction to the hypothesis $M_0 \in SM(\pi)$. Therefore, $M_0 \in MM(M \cup \pi_2)^{M_0}$, which means $M_0 \in SM(M \cup \pi_2)$. This proves the $\subseteq$-direction.

$(\supseteq)$. Let $M_0 \in SM(M \cup \pi_2)$ for some $M \in SM(\pi_1)$. We have to show that $M_0 \in MM(\pi_1 \cup \pi_2)^{M_0}$. Clearly M contains only atoms from HB_{π_1} and therefore $\pi_2 \triangleright M$. Thus, by (i), $M_0 \cap HB_{\pi_1} \in SM(M)$. Since $M \subseteq M_0$, it follows $M_0 \cap HB_{\pi_1} = M$. From this it follows $\pi_1^{M_0} = \pi_1^M$; since M_0 satisfies $\pi_2^{M_0}$ and π_1, π_2 are ground, it follows that M_0 is a model of π^{M_0}. It remains to show that M_0 is minimal. Towards a contradiction, assume there exists a smaller model $M^* \subset M_0$ of π^{M_0}. $M^* \cap HB_{\pi_1}$ is obviously a model of $\pi_1^{M_0}$. If $M^* \cap HB_{\pi_1} \subset M_0 \cap HB_{\pi_1} = M$, then $M^* \cap HB_{\pi_1}$ would be a smaller model of $\pi_1^{M_0} = \pi_1^M$ than M, which would contradict the hypothesis that M is a stable model of π_1. Thus $M^* \cap HB_{\pi_1} = M$. Consequently, M^* is a model of $(M \cup \pi_2)^{M_0}$. This is a contradiction to the hypothesis that M_0 is a minimal model of $(M \cup \pi_2)^{M_0}$. It follows that M_0 is a minimal model of π^{M_0}. Consequently, $M_0 \in SM(\pi_1 \cup \pi_2)$. This completes the proof. ∎

EXAMPLE 5 (CTD) π_1 has the unique stable model $\{ A(c,a), D(a) \}$. Thus, $SM(\pi_1 \cup \pi_2) = \bigcup_{M \in SM(\pi_1)} SM(M \cup \pi_2) = SM(\{ A(c,a), D(a) \} \cup \pi_2) = \{ \{A(c,a), D(a), F(a,b)\}, \{A(c,a), D(a), E(a,a)\} \}$. ∎

As a consequence of Lemma 5.1, we can indeed write query programs that use stable models in a modular fashion, such that modules π_2 use the output of modules π_1. On the other hand, it is possible to break up any given query program π efficiently into components and then compute the stable models bottom up, according to a dependency relation between components. This property is important for implementation, and has been exploited already in systems based on datalog [Ullman 1989].

Say that a predicate P *directly depends* in π on predicate Q, if P occurs in the head of some rule of π and Q occurs (positively or negatively) in the same rule; P depends on Q (in π) if the pair (P, Q) is in the transitive closure of the direct dependency relation (cf. [Dix 1992]). P and Q are *mutually recursive* if P depends on Q and vice versa. Intuitively, if P depends on Q, the extension of Q may influence derivability of ground atoms on P; if P and Q are mutually recursive, they ought to be computed in the same module. Guided by this intuition, we introduce a formal notion of program decomposition.

A subset C of the rules of π is called a *complete component* of π (cf. [Ross 1994]), if for every predicate P that occurs in the head of any rule from C, every rule from π belongs to C that has P in the head or some predicate Q mutually recursive with P. A predicate Q is *defined* by a complete component C, if Q occurs in the head of some rule of C. The following proposition, which is easily proved from the definitions, relates complete components with relation $\rhd$.

PROPOSITION 5.2. *Let $\pi = \pi_1 \cup \pi_2$ such that $\pi_2 \rhd \pi_1$. Then, π_1 and π_2 are complete components of π.*

Call a collection $\mathcal{C} = \{\pi_1, \ldots, \pi_n\}$ of complete components π_i of program π a *decomposition* of π if the π_i are pairwise disjoint and $\pi = \bigcup_{i=1}^{n} \pi_i$. Notice that each stratification of π constitutes a decomposition (but not vice versa); thus, the concept of decomposition is more general. The complete component π_j depends on π_i (denoted $\pi_i \preceq \pi_j$) iff $i = j$ or some predicate defined in π_j depends on some predicate defined in π_i. Notice that $\preceq$ defines a partial order on $\mathcal{C}$. Intuitively, the relations computed by π_i serve as input to π_j; the stable models of π, restricted to the predicates defined in π_j, are computable as the stable models of π_j using the output (i.e., the stable models) of all components on which π_i depends as input. The following theorem expresses this for components on top of a decomposition (i.e., those on which no other components depend).

THEOREM 5.3. *Let $\mathcal{C}$ be a decomposition of a program π, let $\pi_t = \{\pi_1, \ldots, \pi_k\} \subseteq \mathcal{C}$ be a subset of those complete components maximal under $\preceq$, (i.e., on which no other component depends), and let $\pi_{\bar{t}} = \pi - \pi_t$. Then, given any database D s.t. $HU_\pi \subseteq U(D)$,*

(i) for every $M \in SM(\pi, D)$, $M \cap HB_{\pi_{\bar{t}}[D]} \in SM(\pi_{\bar{t}}, D)$;
(ii)

$$\begin{aligned} SM(\pi, D) &= \bigcup_{M \in SM(\pi_{\bar{t}}, D)} SM(M \cup \pi_t, D) \\ &= \bigcup_{M \in SM(\pi_{\bar{t}}, D)} \{M_1 \cup \cdots \cup M_k \mid M_i \in SM(M \cup \pi_i, D), 1 \le i \le k\}. \end{aligned}$$

PROOF. Note that for any such database D, $\pi[D] = \bigcup_{\pi_i \in \mathcal{C}} \pi_i[D]$ and $\pi_t[D] \rhd \pi_{\bar{t}}[D]$. Moreover, the different complete components π_i, π_j define different predicates. Thus the theorem follows from Lemma 5.1. ∎

COROLLARY 5.4. *For $\mathcal{C}$ and π as in Theorem 5.3,*
(a) if $\pi_t = \pi$, then for every database D s.t. $HU_\pi \subseteq U(D)$,

$$SM(\pi, D) = \{M_1 \cup \cdots \cup M_k \mid M_i \in SM(\pi_i, D), i = 1, \ldots, k\};$$

(b) if $D \in \mathcal{D}(\mathbf{P})$ s.t. $HU_\pi \subseteq U(D)$ and $P \in \mathbf{P}$ for each predicate P where $\neg P$ occurs in the body of some rule in π, then

(i) $SM(\pi, D) = MM(\pi, D)$ and for all $M \in MM(\pi, D)$, $M \cap HB_{\pi_{\bar{t}}[D]} \in MM(\pi_{\bar{t}}, D)$;
(ii) $SM(\pi, D) = \bigcup_{M \in MM(\pi_{\bar{t}}, D)} \{M_1 \cup \cdots \cup M_k \mid M_i \in MM(M \cup \pi_i, D), 1 \le i \le k\}$.

As a consequence, the stable models of π on D can be computed bottom up according to the ranking $r_{\mathcal{C}}$ of the π_i's in $\mathcal{C}$ induced by $\preceq$, i.e., $r_{\mathcal{C}}(\pi_j) = 0$ for each π_j which is minimal under $\preceq$, and $r_{\mathcal{C}}(\pi_j) = \max_{\pi_i \prec \pi_j} r_{\mathcal{C}}(\pi_i) + 1$ otherwise. The stable models of $\pi^{\le r}[D] = \bigcup_{r_{\mathcal{C}}(\pi_j) \le r} \pi_j[D]$ are computed from the stable models

M of $\pi^{\leq r-1}[D]$, by computing for each such M all unions $M' = M_1 \cup \cdots \cup M_k$ of stable models M_i of $(M \cup \pi_i)[D]$, where $\pi_1, \ldots, \pi_k$ are the complete components with rank r; for $r = 0$, the M_i are the stable models of $\pi_i[D]$. Notice that the M_i can be computed in parallel. The stable models of $\pi[D]$ are given by $\pi^{\leq r}[D]$ for the highest rank r.

With regard of computation, decompositions that break up π into small components are preferable; on the one hand, they lead to smaller programs for which stable models have to be computed, and on the other hand, a larger degree of parallelism is possible. Say that decomposition $\mathcal{C}'$ is a refinement of another decomposition $\mathcal{C}$ iff for every component π in $\mathcal{C}$, there is some component π' in $\mathcal{C}'$, such that C' is a subset of C. It is easily checked that the collection of all complete components that are minimal with respect to inclusion (called *strongly connected components* (SCCs) in [Ullman 1989; Ross 1994]), is a decomposition and a refinement of any other decomposition, and hence the unique most refined decomposition of π.

Example 5 (ctd) Since $\pi_2 \triangleright \pi_2$, by Proposition 5.2 π_1 and π_2 are complete components of $\pi = \pi_1 \cup \pi_2$. In fact, they are the SCCs of π. (For a more involved example, see the program π_1 computing an enumeration of the universe in the proof of Theorem 6.3). ■

Notice that the SCCs of π can be determined efficiently. Thus, one can decompose π into its SCCs (even at compile time), and then compute the stable models bottom up. In particular, the decomposition into the SCCs supports parallelization of model computation to the highest degree among all decompositions, and is thus appealing for an implementation of datalog that exploits parallel processing for query evaluation.

We remark that in [Lifschitz and Turner 1994], independent of [Eiter et al. 1994] and at the same time, a modularity property for the answer set semantics to extended disjunctive logic programs (which contain two kinds of negation) has been stated which is very similar (and in fact equivalent) to Lemma 5.1, based on the notion of splitting set. With respect to our framework, a set of atoms S is a splitting set of a ground program π if every rule that has an atom from S in the head has all its atoms in S; denote by $split(\pi, S)$ the collection of all such rules. Clearly, if $\pi = \pi_1 \cup \pi_2$ and $\pi_2 \triangleright \pi_1$, then the set of atoms in π_1 is a splitting set of π; conversely, if S is a splitting set of π, then $(\pi - split(\pi, S)) \triangleright split(\pi, S)$ if the ground atoms are seen as propositional letters. Under this view, the main result of [Lifschitz and Turner 1994] (Splitting Set Theorem) amounts for programs with only one kind of negation precisely to Lemma 5.1 (Lemma 6.1 in [Eiter et al. 1994]). It is no surprise that the same modularity property has been independently found in slightly different settings, since it is quite natural. Moreover, Lifschitz and Turner extend their result to iterated splittings of a program (Splitting Sequence Theorem), which decomposes the program into a chain of modules. This result corresponds to the above observation about bottom up computability of the stable models, but does not exploit parallelism in this process which is possible by the partial ordering of modules through dependency.

6. EXPRESSIVE POWER OF DISJUNCTIVE DATALOG

In this section, we address the issue of the expressive power of disjunctive datalog. The main result is that disjunctive datalog captures $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ under possibility stable model semantics, that is, from the generic total queries, it can express precisely those which are Σ_2^p-recognizable. Thus the expressive power of disjunctive datalog is one level above the expressive power of normal datalog under stable model semantics [Schlipf 1995b] or fixpoint semantics [Kolaitis and Papadimitriou 1991], which is NP.

An interesting result is that the full expressive power is already available with DATALOG$^{\vee,\neq}$ programs; hence, also the DATALOG$_p^{\vee,\neq}$ queries capture Σ_2^p, provided that inequality literals are allowed in the body. Without inequality literals, this is not the case; we discuss this in detail in the next section. This result also shows that the expressive power of circumscription [McCarthy 1986], applied to function-free first-order theories, is Σ_2^p under brave (possibility) inference.

Our plan for this section is a follows. We consider first boolean (generic) queries and then general (generic) queries. In order to assure genericity of queries, we focus on disjunctive datalog programs that are constant-free. Constants in queries are not an issue, since they can be provided by designated input relations (see [Abiteboul et al. 1995, Section 9]).

The following proposition on the complexity of minimal, stable, and perfect models of propositional programs is well-known.

Proposition 6.1. [Eiter and Gottlob 1995b] *Let π be a propositional (i.e., ground) program and let p be a propositional atom. Deciding if $p \in M$ for some $M \in \mathcal{S}(\pi)$ is Σ_2^p-complete for $\mathcal{S} = MM, SM, PM$.*

Proof. (Membership) A guess for a model $M \in \mathcal{S}(\pi)$ such that $p \in M$, which is polynomial in the size of π, can be checked in polynomial time with an NP oracle. This proves membership in $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$. ■

We obtain from this easily the following complexity bound on the recognizability of disjunctive datalog queries.

Lemma 6.2. *The* DATALOG$_{p,\mathcal{S}}^{\vee,\neg}$ *queries are Σ_2^p-recognizable, for $\mathcal{S} = MM$, SM, PM.*

Proof. The QOT problem (cf. Section 2) for query $q_{p,\mathcal{S}}^{\pi}$ defined by the DATALOG$^{\vee,\neg}$ program π, trivially reduces to deciding, given a database D, if a grounded atom a belongs to some $M \in \mathcal{S}(\pi, D) = \mathcal{S}(\pi[D])$. For fixed π, the size of $ground(\pi, U)$ is polynomial in the size of D; hence, $\pi[D]$ can be clearly constructed in polynomial time. Thus the lemma follows from Proposition 6.1. ■

On the other hand, we will show that every Σ_2^p-recognizable query can be expressed in disjunctive datalog under possibility inference. In fact, we show that this holds already for the fragment of DATALOG$_p^{\vee,\neq}$ queries, i.e., under possibility minimal model inference.

We start by exploring the expressiveness of boolean DATALOG$_p^{\vee,\neq}$ queries, which is useful for deriving the expressiveness of general DATALOG$_p^{\vee,\neq}$ queries with output relations.

THEOREM 6.3. *The boolean constant-free* DATALOG$_p^{\vee,\neq}$ *queries coincide with the class of all database properties in* Σ_2^p.

PROOF. By Lemma 6.2, it remains to show that every database property $\mathcal{P}$ in Σ_2^p can be expressed by a boolean DATALOG$_{p,MM}^{\vee,\neq}$ query.

For any such $\mathcal{P}$, we construct a constant-free boolean DATALOG$^{\vee,\neq}$ query program π with $IO(\pi) = (\mathbf{R}, W)$, such that for $D \in \mathcal{D}(\mathbf{R})$, $q^{\pi}_{p,MM}(D) \models W$ iff $\mathcal{P}(D) = true$.

The construction uses a lemma, whose proof can be found in [Eiter et al. 1994; Eiter et al. 1996]. We needs some concepts first. Assume that S and F, L are distinguished relation symbols which do not occur in $\mathbf{R}$. The class of *enumerated databases over* $\mathbf{R}$ is the collection of all databases D over $\mathbf{R}, S, F, L$ such that the binary relation S describes an enumeration of all elements in U, where $(x, y) \in S$ means that y is the successor of x, and the unary relations F and L contain the first and the last element in the enumeration, respectively.

LEMMA 6.4. [Eiter et al. 1994; Eiter et al. 1996] *A database property* $\mathcal{P}$ *on* $\mathbf{R}$ *is in* Σ_2^p *iff* $\mathcal{P}$ *is definable on the enumerated databases over* $\mathbf{R}$ *by a* Σ_2^1 *formula* Ψ *over vocabulary* $\mathbf{R}, S, F, L$ *of the form*

$$(\exists \mathbf{S})(\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x}), \tag{2}$$

where $\varphi(\mathbf{x}) = \vartheta_1(\mathbf{x}) \vee \cdots \vee \vartheta_k(\mathbf{x})$ *and the* ϑ_i *are conjunctions of literals.*

We compose the program π of two modules π_1 and π_2 such that $\pi_2 \triangleright \pi_1$. The lower module, π_1, computes an enumeration of the universe of the given database in a minimal model, such that the minimal models containing enumerations are those in which a designated propositional atom *Order* is true; every enumeration is in some minimal model. The upper module, π_2, evaluates the formula Ψ for $\mathcal{P}$ on the database, by using the enumeration output by π_1.

Program π_1 is as follows:

(1)	$x < y \vee y < x \leftarrow x \neq y.$	(7)	$\tilde{F}(x) \leftarrow y < x.$
(2)	$x < z \leftarrow x < y,\ y < z.$	(8)	$F(x) \vee \tilde{F}(x) \leftarrow .$
(3)	$\tilde{S}(x, z) \leftarrow z < x.$	(9)	$\tilde{L}(x) \leftarrow x < y.$
(4)	$\tilde{S}(x, z) \leftarrow x < y, y < z.$	(10)	$L(x) \vee \tilde{L}(x) \leftarrow .$
(5)	$\tilde{S}(x, x) \leftarrow .$	(11)	$G(x) \leftarrow F(x).$
(6)	$S(x, y) \vee \tilde{S}(x, y) \leftarrow .$	(12)	$G(y) \leftarrow G(x), S(x, y).$
		(13)	$Order \leftarrow G(x), L(x).$

The predicates defined in the strongly connected components of π_1, ordered by dependency, are shown in Figure 1.

Informally, a minimal model of π_1 on U is built bottom-up as follows. Denote for any linear order $<$ on U by $S_<, F_<$, and $L_<$ the relations that give the successor, the first, and the last element of U with respect to $<$. The lowest component (rules (1)–(2)) computes in $<$ a linear order on U. The relations $\tilde{S}$,$\tilde{F}$, and $\tilde{L}$ are computed such that they are supersets of the complements of $F_<$, $S_<$, and $L_<$, and S, F, and L are computed as their complements. Dependent on that, G is computed, such that $G(x)$ means that F gives $F_<$ and S gives $S_<$ from the first element in $<$ up to

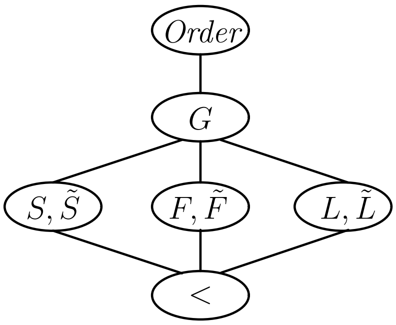

Fig. 1. Predicates defined by the strongly connected components of π_1.

x. Finally, the rule deriving *Order* is evaluated, which fires precisely if F, S and L give $F_<$, $S_<$, and $L_<$, respectively.

More formally, let for any linear order $<$ on U be $M^U_<$ the interpretation of $ground(\pi_1, U)$ in which *Order* is true, $G = U$, $<$ gives itself, and P (resp. $\tilde{P}$) for $P = S, F, L$ is $P_<$ (resp. the complement of $P_<$). Then, we have the following.

LEMMA 6.5. *Let U be a finite universe, and let M be an interpretation of the program $ground(\pi_1, U)$ such that Order $\in M$. Then, $M \in MM(ground(\pi_1, U))$ iff $M = M^U_<$ for some $<$.*

PROOF. Let P denote $ground(\pi_1, U)$.
($\Rightarrow$) It is easy to see that $M^U_<$ is a model of P, and that no atoms can be removed from $M^U_<$ without loosing the model property. Hence, $M^U_< \in MM(P)$.
($\Leftarrow$) From Corollary 5.4, it follows by the rules (1)–(2) that $<$ is in M a linear order on U; moreover, by the rules (6), (8), and (10) the relations S, F, and L must be in M the complements of $\tilde{S}$, $\tilde{F}$, and $\tilde{L}$, respectively. Consequently, from the rules (3)–(5), (7), and (9), it follows that F, S and L must be in M subsets of $F_<$, $S_<$, and $L_<$, respectively. Since *Order* $\in M$, it follows that $G(a), L(a) \in M$ for some $a \in U$; it follows that F is in M the relation $F_<$. By induction downwards the order $<$, we obtain for each $b \in U$ whose predecessor w.r.t. $<$ is a, that $G(a) \in M$ and $S(a, b) \in M$ and that $F(a) \in M$ where a is the first element of U w.r.t. $<$. Consequently, S and F are in M the relations $S_<$ and $F_<$, respectively, and $G = U$. Thus, $M = M^U_<$. ■

The program π_2, which evaluates the formula Ψ on an enumerated database over $\mathbf{R}$, is as follows (arities of tuple variables are understood):

$$\begin{array}{lrl}
(1) & S_i(\mathbf{x}) \vee \tilde{S}_i(\mathbf{x}) \leftarrow . & \text{for each } S_i \in \mathbf{S}; \\
(2) & T_j(\mathbf{x}) \vee \tilde{T}_j(\mathbf{x}) \leftarrow . & \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{for each } T_j \in \mathbf{T}; \\
(3) & T_j(\mathbf{x}) \leftarrow W. & \\
(4) & \tilde{T}_j(\mathbf{x}) \leftarrow W. & \\
(5) & W \leftarrow \eta_i(\mathbf{x}). & \text{for } i = 1, \ldots, k;
\end{array}$$

In this program, the $\tilde{S}_i$ and $\tilde{T}_j$ are new predicates of the arity of S_i and T_j, respectively, and W is a new propositional letter. The η_i are constructed from the

ϑ_i in Ψ as follows. Suppose that

$$\vartheta_i(\mathbf{x}) = l_{i,1} \wedge \cdots \wedge l_{i,n_i};$$

then,

$$\eta_i(\mathbf{x}) = l'_{i,1}, \ldots, l'_{i,n_i}, \mathit{Order},$$

where $l'_{i,j} = \tilde{P}(\mathbf{x})$, if $l_{i,j} = \neg P(\mathbf{x})$ and $P \notin \mathbf{R}, =$; otherwise, $l'_{i,j} = l_{i,j}$.

Intuitively, program π_2 works as follows. The clauses (1) choose an extension for the predicates in $\mathbf{S}$, and assign their complements to $\tilde{\mathbf{S}}$. For every such extension $\mathbf{S}$, choosing W true and for all predicates in $\mathbf{T}, \mathbf{T}'$ the maximal extensions (i.e., all tuples) yields an interpretation $M_{\mathbf{S}}$ which is clearly a model of π_2. This interpretation is in fact a minimal model, if there exists no choice for the predicates in $\mathbf{T}$ (implemented by the clauses (2)), such that all instances of the clauses (5) have a false body. Thus, on an enumerated database D (extended by the complements $\tilde{F}$, $\tilde{S}$, $\tilde{L}$, of F S, and L, respectively, and by the true letter *Order*), $M_{\mathbf{S}}$ is a minimal model precisely if $(D, \mathbf{S}) \models (\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x})$. This means that π_2 has a minimal model containing W iff $D \models \Psi$.

More formally, we have the following. Let $\mathbf{R}' = \mathbf{R}, (P, \tilde{P} \mid P = F, S, L), \mathit{Order}$ and adopt $\mathit{IO}(\pi_2) = (\mathbf{R}', W)$.

Lemma 6.6. *For every $D' \in \mathcal{D}(\mathbf{R}')$, such that D' is an enumerated database over $\mathbf{R}$, $D' \models (\forall \mathbf{x})(\tilde{P}(\mathbf{x}) \leftrightarrow \neg P(\mathbf{x}))$, $P = F, S, L$ and $D' \models \mathit{Order}$, it holds that $q^{\pi_2}_{p,MM}(D') \models W$ iff $D' \models \Psi$.*

Proof. ($\Rightarrow$) If $q^{\pi_2}_{p,MM}(D') \models W$, then some $M \in \mathit{MM}(\pi_2, D')$ exists with $W \in M$. From Corollary 5.4 and clauses (1), (3)–(4) it is clear that $M = M_{\mathbf{S}}$ for the predicates $\mathbf{S}$ as interpreted in M. Suppose $(D', \mathbf{S}) \not\models (\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x})$. Then, there exist relations for $\mathbf{T}$ such that $(D', \mathbf{S}, \mathbf{T}) \models (\forall \mathbf{x})\neg\varphi(\mathbf{x})$. Let M' be the interpretation of $\pi_2[D']$ in which W is false, the predicates $\mathbf{S}, \mathbf{T}$ are as in $(D', \mathbf{S}, \mathbf{T})$, and the predicates $\tilde{\mathbf{S}}, \tilde{\mathbf{T}}$ are their complements. Then, M' is a model of $\pi_2[D']$. However, $M' \subset M$; this is a contradiction to $M \in \mathit{MM}(\pi_2, D')$. It follows $(D', \mathbf{S}) \models (\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x})$, and hence $D' \models \Psi$.

($\Leftarrow$) Assume $D' \models \Psi$. Hence, there are relations $\mathbf{S}$ such that $(D', \mathbf{S}) \models (\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x})$. Consider the interpretation $M_{\mathbf{S}}$. Clearly, $M_{\mathbf{S}}$ is a model of $\pi_2[D]$. Suppose that $M_{\mathbf{S}}$ is not minimal. Hence, there exists a model $M' \subset M_{\mathbf{S}}$. Clearly, by the clauses (1), M' coincides on $\mathbf{S}$ with M, and hence by the clauses (3)–(4) $W \notin M'$. By the clauses (2), either $T_j(\mathbf{t}) \in M'$ or $\tilde{T}(\mathbf{t}) \in M'$ for each $T_j \in \mathbf{T}$ and tuple $\mathbf{t} \in U^{a(T_j)}$, and since M' is minimal, not both is true. Thus, by the clauses (5), it follows for the relations $\mathbf{S}, \mathbf{T}$ as in M that $(D', \mathbf{S}, \mathbf{T}) \models (\forall \mathbf{x})\neg\varphi(\mathbf{x})$; the latter implies $(D', \mathbf{S}) \not\models (\forall \mathbf{T})(\exists \mathbf{x})\varphi(\mathbf{x})$. Thus, we arrived at a contradiction. Hence, $M_{\mathbf{S}} \in \mathit{MM}(\pi_2, D')$. Since $W \in M$, it follows $q^{\pi_2}_{p,MM}(D') \models W$. This proves the lemma. ■

Now let $\pi = \pi_1 \cup \pi_2$; notice that $\pi_2 \rhd \pi_1$. We show that, given any database $D \in \mathcal{D}(\mathbf{R})$, $q^{\pi}_{p,MM}(D) \models W$ iff $\mathcal{P}$ is true on D.

($\Rightarrow$) If $q^{\pi}_{p,MM}(D) \models W$, then there exists $M \in \mathit{MM}(\pi, D)$ such that $W \in M$. It is easily seen that $\mathit{Order} \in M$. Thus, by Corollary 5.4 and Lemma 6.5, it follows that $M \cap \mathit{HB}_{ground(\pi_1,U)} = M^U_<$ for some linear order $<$ on U. By Corollary 5.4, $M \in \mathit{MM}(M^U_< \cup \pi_2, D)$. Thus, for the database D' over $\mathbf{R}'$ such that $D'[\mathbf{R}] = D$

and D' coincides on all predicates in $\mathbf{R}' - \mathbf{R}$ with $M^U_<$, clearly $M \cap HB_{\pi_2[D']} = MM(\pi_2, D')$ and $q^{\pi_2}_{p,MM}(D') \models W$. Thus by Lemma 6.6, $D' \models \Psi$. It follows from Lemma 6.4 that $\mathcal{P}$ is true on D.

($\Leftarrow$) Assume that $\mathcal{P}$ is true on D. Then, from Lemmata 6.4 and 6.6, it follows that $q^{\pi_2}_{p,MM}(D') \models W$ for every $D' \in \mathcal{D}(\mathbf{R}')$ as described such that $D'[\mathbf{R}] = D$. Fix any such D', and let $M \in MM(\pi_2, D')$ such that $W \in M$. For the linear order $<$ on U such that S in D' is $S_<$, it follows from Corollary 5.4 and Lemma 6.5 that $M \cup M^U_< \in MM(\pi, D)$. Thus, $q^{\pi}_{p,MM}(D) \models W$.

This completes the proof of the theorem. ∎

A similar result can be derived for DATALOG$^{\vee,\neg}$ queries under the p,*SM* (resp. *PM*) -semantics. Notice that the expressive power of DATALOG$^{\vee,\neg}_{p,MM}$ queries without inequality literals is the same as for DATALOG$^{\vee}_{p}$ queries, since negative literals involving non-input predicates in the body of a clause can be moved to the head.

Using negation, the equality and inequality can be computed easily, e.g. by the program $\pi_{Eq,Neq}$

$$\begin{aligned} Neq(x,y) &\leftarrow \neg Eq(x,y). \\ Eq(x,x) &\leftarrow . \end{aligned}$$

Notice that $\pi_{Eq,Neq}$ is stratified. Hence for every universe U, $ground(\pi_{Eq,Neq}, U)$ has a unique perfect model $M^U_{Eq,Neq}$, which coincides with the unique stable model, in which Eq and Neq are equality and inequality on U respectively. We obtain the following.

Lemma 6.7. *For every* DATALOG$^{\vee,\neg}$ *query program* π *exists a* DATALOG$^{\vee,\neg}$ *query program* π' *such that* $q^{\pi}_{inf,SM} = q^{\pi'}_{inf,SM}$, $inf = p, c$ *and that* π' *is stratified if* π *is stratified.*

Proof. Let $IO(\pi) = (\mathbf{R}, \mathbf{S})$, and assume that Neq and Eq do not occur in $RS(\pi), \mathbf{R}, \mathbf{S}$ (use otherwise fresh predicates otherwise). Let π_2 be the program resulting from π by replacing each equality atom $x_i = x_j$ with x_i, x_j and each inequality atom $x_i \neq x_j$ with $Neq(x_i, x_j)$. Let $\pi = \pi_2 \cup \pi_{Eq,Neq}$ and $IO(\pi') = IO(\pi)$. Notice that $\pi_2 \rhd \pi_1$, and that for every $D \in \mathcal{D}(\mathbf{R})$, $M^U_{Eq,Neq}$ is the unique stable model of $\pi_{Eq,Neq}[D]$. Thus from Corollary 5.3, it follows that $q^{\pi'}_{p,SM}(D) = q^{\pi'}_{p,SM}(D)$ for every $D \in \mathcal{D}(\mathbf{R})$, which means $q^{\pi'}_{p,SM} = q^{\pi}_{p,SM}$. Clearly, π' is stratified if π is. ∎

Theorem 6.8. *The boolean constant-free* DATALOG$^{\vee,\neg_s}_{p}$ *queries precisely capture the class of all database properties in* Σ^p_2*, even if no equality and inequality literals are allowed.*

Proof. Follows from Theorem 6.3 and Lemma 6.7. ∎

6.1 I/O queries

Since Σ^p_2 is closed under conjunction, it is not difficult to extend our results for boolean queries to the case of general I/O-queries. However, proofs of such generalizations are rarely found in the literature. Therefore, we will give a proof here.

Theorem 6.9. *A generic total query* q *is* Σ^p_2*-recognizable iff it is expressible in* DATALOG$^{\vee,\neq}_{p}$ *through a constant-free query-program.*

PROOF. By Lemma 6.2, it remains to show the if-direction.

Let $q : \mathcal{D}(\mathbf{R}) \to \mathcal{D}(\mathbf{S})$ be a Σ_2^p-recognizable generic total query. Let $\mathcal{P}$ be the database property over $\mathbf{R}' = \mathbf{RS}$ defined as follows: for every $D' \in \mathcal{D}(\mathbf{R}')$, $\mathcal{P}(D') =$ *true* iff $D'[\mathbf{S}] \subseteq q(D'[\mathbf{R}])$, where $\subseteq$ is understood componentwise. Notice that $\mathcal{P}$ is in fact a database property. For any instance D' of $\mathbf{R}'$, $\mathcal{P}(D')$ is equivalent to a conjunction of QOT-problems for q as follows:

$$\mathcal{P}(D') = \bigwedge_{S_i \in \mathbf{S}} \; \bigwedge_{D' \models S_i(\mathbf{t})} q(D'[\mathbf{R}]) \models S_i(\mathbf{t}).$$

Since q is Σ_2^p-recognizable, it follows that $\mathcal{P}$ is in Σ_2^p. Thus, by Theorem 6.3, there exists a constant-free DATALOG$^{\vee,\neq}$ query program π' where $IO(\pi') = (\mathbf{R}', W)$ for some 0-ary predicate W, such that for every $D' \in \mathcal{D}(\mathbf{R}')$, $q^{\pi'}_{p,MM}(D') \models W$ iff $\mathcal{P}(D') = $ *true*. We assume that π' has exactly the form of the DATALOG$^{\vee,\neq}$ query program in the proof of Theorem 6.3. Now let π be the DATALOG$^{\vee,\neq}$ query program with $IO(\pi) = (\mathbf{R}, \mathbf{S})$ obtained by modifying π' as follows. For each $S_i \in \mathbf{S}$, replace each positive (resp. negative) occurrence of S_i in π' with S'_i (resp. $\tilde{S}'_i$), where S'_i and $\tilde{S}'_i$ are new predicates of the same arity as S_i, and add the rules

$$\begin{aligned} S_i(\mathbf{x}) &\leftarrow S'_i(\mathbf{x}), W. \\ S'_i(\mathbf{x}) \vee \tilde{S}'_i(\mathbf{x}) &\leftarrow \; . \end{aligned}$$

Intuitively, the first rule allows derivation of S_i only if W is in the interpretation, and the second rule simulates all possible instances of S'_i.

For every $D \in \mathcal{D}(\mathbf{R})$, $q^{\pi}_{p,MM}$ essentially computes the union of all $M \in MM(\pi, D)$ with $W \in M$. By our construction, for each such M the database $D' \in \mathcal{D}(\mathbf{R}')$ such that $D'[\mathbf{R}] = D$ and $D'[\mathbf{S}]$ coincides with M, the program π' has a minimal model on D' that contains W, and hence $\mathcal{P}(D') = $ *true* (i.e., $D'[\mathbf{S}] \subseteq q(D'[\mathbf{R}])$). On the other hand, given that $M' \in MM(\pi', D')$ such that $W \in M'$, where $D' \in \mathcal{D}(\mathbf{R}')$ and $D'[\mathbf{R}] = D$, the interpretation $N = M' \cup \{S_i(\mathbf{t}), S'_i(\mathbf{t}) \mid D' \models S_i(\mathbf{t})\} \cup \{S''_i(\mathbf{t}) \mid D' \not\models S_i(\mathbf{t})\}$ is clearly a minimal model of π on D. Consequently, $q^{\pi}_{p,MM}(D)$ $= q(D)$; it follows $q^{\pi}_{p,MM} = q$. ■

THEOREM 6.10. *A generic total query q is Σ_2^p-recognizable iff it is expressible in* DATALOG$^{\vee,\neg_s}_{p,\mathcal{S}}$, *$\mathcal{S} = PM, SM$, even without equality and inequality literals, through a constant-free query program.*

PROOF. Follows from Theorem 6.9 and Lemma 6.7. ■

Symmetric results hold for the certainty semantics, if negation is allowed.

THEOREM 6.11. *A generic query total q is Π_2^p-recognizable iff it is expressible in* DATALOG$^{\vee,\neg_s}_{c}$ *through a constant-free query program.*

PROOF. That each query $q^{\pi}_{c,SM}$ (resp. $q^{\pi}_{c,PM}$) is Π_2^p-recognizable follows from the known fact that, given a propositional program π and an atom p, deciding if $p \in M$ for all $M \in SM(\pi)$ (resp. $M \in PM(\pi)$) is in Π_2^p (cf. [Eiter and Gottlob 1995b]; the proof is dual to the membership part of Proposition 6.1).

It remains to show that each generic total query $q : \mathcal{D}(\mathbf{R}) \to \mathcal{D}(\mathbf{S})$ that is Π_2^p-recognizable can be expressed in DATALOG$^{\vee,\neg_s}_{c}$. Let $\overline{q}$ be the query that computes the complement of q, i.e., for each $S_i \in \mathbf{S}$ and suitable tuple $\mathbf{t}$, it holds $\overline{q}(D)S_i(\mathbf{t})$

iff $q(D) \not\models S_i(\mathbf{t})$. Since $\overline{q}$ is Σ_2^p-recognizable, by Theorem 6.10, $\overline{q}$ is expressible by a $\mathrm{DATALOG}_p^{\vee,\neg_s}$ query program π' without equality and inequality literals.

Let π be the query program with $IO(\pi) = (\mathbf{R}, \mathbf{S})$ obtained from π' by first replacing in the clauses each occurrence of every $S_i \in \mathbf{S}$ with $\tilde{S}_i$, where each $\tilde{S}_i$ is a new predicate letter with arity $a(S_i)$, and by afterwards adding the clauses

$$S_i(\mathbf{x}) \leftarrow \neg\tilde{S}_i(\mathbf{x}), \qquad S_i \in \mathbf{S}.$$

Then, for any $D \in \mathcal{D}(\mathbf{R})$, $S_i \in \mathbf{S}$, and $\mathbf{t} \in U^{a(S_i)}$, we have that

$$\begin{aligned} q^{\pi}_{c,SM}(D) \models S_i(\mathbf{t}) &\Leftrightarrow S_i(\mathbf{t}) \in M, \quad \text{for all } M \in SM(\pi[D]) \\ &\Leftrightarrow \tilde{S}_i(\mathbf{t}) \notin M, \quad \text{for all } M \in SM(\pi[D]) \\ &\Leftrightarrow S_i(\mathbf{t}) \notin M, \quad \text{for all } M \in SM(\pi'[D]) \\ &\Leftrightarrow \overline{q}(D) \not\models S_i(\mathbf{t}) \\ &\Leftrightarrow q(D) \models S_i(\mathbf{t}). \end{aligned}$$

Consequently, $q^{\pi}_{c,SM}$ is identical to q. Since π' is stratified, it follows by Proposition 3.2 that $q^{\pi}_{c,PM} = q$. This proves the theorem. ∎

7. PLAIN DISJUNCTIVE DATALOG

In the previous section, we have seen that adding disjunction to DATALOG is enough to gain the full expressive power of the formalism, provided that inequality literals $t_1 \neq t_2$ may be used. It is interesting to see what happens if we disallow for such literals, and look at plain disjunctive datalog ($\mathrm{DATALOG}^{\vee}$). On a first guess, the expressive power remains unaffected. However, this intuition is wrong, as we show that $\mathrm{DATALOG}_p^{\vee}$ can not express simple P-time computable queries, and hence not all Σ_2^p-recognizable queries. This means that extending plain disjunctive datalog by inequality literals results in the maximal increase of the expressive power, as opposed to nondisjunctive datalog, where the increase is minor [Kolaitis and Vardi 1995].

On the other hand, compared to (plain) DATALOG, which expresses a strict subclass of the P-time queries, the expressive power of $\mathrm{DATALOG}^{\vee}$ increases a lot, since $\mathrm{DATALOG}_p^{\vee}$ can express some Σ_2^p-hard queries (cf. Section 8). Analogous to DATALOG, which expresses all P-time queries on enumerated databases (i.e., databases with a successor relation on the universe) [Vardi 1982; Immerman 1986], $\mathrm{DATALOG}_p^{\vee}$ expresses all Σ_2^p-recognizable queries on enumerated databases, since the inequality predicate can be easily defined from the enumeration of the universe. Notice that if we have a linear order $<$ rather than an enumeration, then $\mathrm{DATALOG}_p^{\vee}$ also expresses all of Σ_2^p, while DATALOG then expresses only a strict subset of P.

The rôle of inequality literals in disjunctive datalog is complemented by an interesting phenomenon: The expressive power of $\mathrm{DATALOG}_p^{\vee}$ is strictly higher than the expressive power of $\mathrm{DATALOG}^{\neq}$ (which is limited to P-time queries). In particular, the inequality predicate is computable by a query in $\mathrm{DATALOG}_p^{\vee}$. This sounds paradoxical, and is explained as follows: while it is possible to compute the inequality predicate in a minimal model of a program π_1, it is not possible to recognize this model and pass it to a program π_2 that uses π_1. This contrasts with computing an enumeration of the database universe (see Section 6).

The first result mentioned above is that the $\mathrm{DATALOG}_p^{\vee}$ queries do not include all P-time queries. In particular, we show that the boolean Even-query

$$q_{Even} : \mathcal{D}(R) \to S,$$

which tells if the relation R has an even number of elements, can not be expressed in $\mathrm{DATALOG}_{p,MM}^{\vee,\neg}$ without inequality literals. This is established by exploiting a weak form of monotonicity of the query programs.

Notice that for normal (i.e., nondisjunctive) datalog, nonexpressiveness results have been shown using the tool of infinitary logic with finitely many variables ($L^{\omega}_{\infty\omega}$) [Kolaitis and Vardi 1995; Kolaitis and Vardi 1992]. It is known that many prominent query languages (e.g., stratified datalog, fixpoint queries, while queries etc) can only express queries that are definable in $L^{\omega}_{\infty\omega}$ [Kolaitis and Vardi 1992; Kolaitis and Vardi 1995; Abiteboul et al. 1995]. Moreover, it is known that $L^{\omega}_{\infty\omega}$ has a 0-1 law, i.e., every query from this language is either almost surely true or almost surely false, if the size of the universe grows to infinity [Kolaitis and Vardi 1992]. Since the Even-query does not have this property, it can not be expressed in $L^{\omega}_{\infty\omega}$, and hence also not in any of the query languages subsumed by $L^{\omega}_{\infty\omega}$.

Unfortunately, the powerful tool of infinitary logic (prove inexpressibility in a query language embedded in $L^{\omega}_{\infty\omega}$ by inexpressibility in $L^{\omega}_{\infty\omega}$) can not be applied to disjunctive datalog. In fact, as shown in Section 4, graph 3-colorability can be expressed in $\mathrm{DATALOG}_p^{\vee}$; on the other hand, it has been shown that graph 3-colorability can not be expressed in $L^{\omega}_{\infty\omega}$ [Dawar 1995]. Thus,

Proposition 7.1. *(Constant-free)* $\mathrm{DATALOG}_p^{\vee}$ *is not equivalent to any fragment of* $L^{\omega}_{\infty\omega}$.

As shown below, the inequality relation can not be recognized by a $\mathrm{DATALOG}_p^{\vee}$ query. Thus, $\mathrm{DATALOG}_p^{\vee}$ is even incomparable with any fragment from $L^{\omega}_{\infty\omega}$ that can recognize inequality (e.g., stratified datalog, the fixpoint queries, and the while queries).

The following lemma implies a weak form of monotonicity for $\mathrm{DATALOG}_{p,MM}^{\vee,\neg}$, in the sense that minimal models of programs without input relations and inequality literals grow monotonically.

Lemma 7.2. *Let* π *be a* $\mathrm{DATALOG}^{\vee,\neg}$ *program without inequality literals, and let* $U_1 \subseteq U_2$ *such that* $|U_1| \geq 1$. *Then, for every* $M_1 \in MM(ground(\pi, U_1))$ *there exists an* $M_2 \in MM(ground(\pi, U_2))$ *such that* $M_1 \cap M_2 = M_1$.

Proof. We prove the claim for $U_2 = U_1 \cup \{a\}$ where $a \notin U_1$; the general result follows by a simple induction argument.

In what follows, let $P_1 = ground(\pi, U_1)$ and $P_2 = ground(\pi, U_2)$. Consider $M_1 \in MM(P_1)$. Let c be an arbitrary constant from U_1, and denote for any formal object X (formula, rule, term, etc) by $X[a/c]$ the object resulting from X by uniformly replacing a with c. Let $M = \{L \in HB_{P_2} : L[a/c] \in M_1\}$. Of course, M is an interpretation of P_2; notice that $M_1 \subseteq M$.

We show that M is a model of P_2. Assume to the contrary that M is not a model of P_2. Then there exists a clause $C = H \leftarrow B$ in $P_2 - P_1$ such that C is not satisfied by M. This means that B is satisfied by M but no disjunct of H is in M. Clearly $C[a/c] \in P_1$. By definition of M, since B is satisfied by M, $B[a/c]$ must be satisfied by M_1. Since M_1 is a model of P_1, also $H[a/c]$ must be satisfied by M_1. Therefore there exists an atom $G \in M_1$ such that G occurs as disjunct in $H[a/c]$. Thus there exists an atom G^* in H with $G^*[a/c] = G$ and $G^* \in M$ (by definition of M). This shows that H is satisfied by M. We arrived at a contradiction.

We thus have shown that M is a model of P_2. Consequently, there must exist a minimal model $M_2 \subseteq M$ of P_2. Clearly, $M_2 \cap HB_{P_1}$ is a model of P_1. Since $M_2 \cap HB_{P_1} \subseteq M_1$, but M_1 is already minimal, it holds that $M_2 \cap HB_{P_1} = M_1$; thus, $M_2 \cap M_1 = M_1$. This proves the lemma. ■

THEOREM 7.3. *The Even-query is not expressible in* $\text{DATALOG}^{\vee,\neg}_{p,MM}$ *without inequality literals.*

PROOF. Assume that $q_{Even} : R \to S$ is computed by a $\text{DATALOG}^{\vee,\neg}$ query program π without inequality literals. Let π' be the program resulting from π by removing all rules that contain negative literals involving R and all positive literals involving R from the remaining rules. For every $D = (U, U)$ the following condition (*) holds: $\pi[D] = ground(\pi', U)$.

Let $D_1 = (U_1, U_1)$ such that $|U_1| \geq 1$ is even. Then, $q^{\pi}_{p,MM}(D_1) = true$; from (*), it follows that $M \in MM(ground(\pi', U_1))$ exists with $S \in M$. Now choose $D_2 = (U_2, U_2)$ such that $U_1 \subseteq U_2$ and $|U_2|$ is odd. From Lemma 7.2, it follows that $M \in ground(\pi', U_2)$ exists with $S \in M$; thus, by (*), $q^{\pi}_{p,MM}(D_2) = true$. However, this means that $q^{\pi}_{p,MM}$ is different from q_{Even}. We reached a contradiction. ■

COROLLARY 7.4. *The Even-query is not expressible in* $\text{DATALOG}^{\vee}_{p}$.

Using Theorem 7.3, it is easily shown that also the complement of the Even-query can not be expressed in $\text{DATALOG}^{\vee}_{p}$. Analogous results can be shown for certainty inference; in particular, the Even-query (as well as its complement) can not be expressed in $\text{DATALOG}^{\vee}_{c}$.

As mentioned above, the inequality predicate can be easily computed by a $\text{DATALOG}^{\vee}_{p}$ query. Consider the following program $\pi_{\neq}$:

$$
\begin{array}{lr}
(1) & A(x,y) \vee Neq(x,y) \leftarrow . \\
(2) & A(y,z) \leftarrow Neq(x,x). \\
(3) & Neq(y,z) \leftarrow Neq(x,x).
\end{array}
$$

For any universe U, the minimal models of $ground(\pi_{\neq}, U)$ are the interpretations M in which Neq is a subset of the inequality relation on U, and A is the complement of Neq. In particular, there is a unique minimal model in which Neq coincides with the inequality relation, which we denote by $M^U_{\neq}$. Thus, for $IO(\pi_{\neq}) = (\emptyset, Neq)$, $q^{\pi_{\neq}}_{p,MM}$ computes in Neq the inequality relation on U.

An interesting consequence is that each query in $\text{DATALOG}^{\neq}$ can be expressed in $\text{DATALOG}^{\vee}_{p}$.

THEOREM 7.5. *The expressive power of* $\text{DATALOG}^{\neq}$ *is strictly included in the expressive power of* $\text{DATALOG}^{\vee}_{p}$.

PROOF. Let $q : \mathbf{R} \to \mathbf{S}$ be a query expressed by a DATALOG$^{\neq}$ query program π'. Assume that Neq, A are fresh predicates. Denote by π_2 the program obtained from π' by replacing each occurrence of inequality literals $t_1 \neq t_2$ by $Neq(t_1, t_2)$. Let $\pi = \pi_2 \cup \pi_{\neq}$ and $IO(\pi) = (\mathbf{R}, \mathbf{S})$. Since $\pi_2 \triangleright \pi_{\neq}$, by Corollary 5.4 it follows $MM(\pi, D) = \bigcup_{M \in MM(\pi_{\neq}, D)} MM(M \cup \pi_2, D)$ for every $D \in \mathcal{D}(\mathbf{R})$. Each program $M \cup \pi_2$ is a DATALOG$^{\neq}$ program, and thus has a unique minimal model N on D. It is easily seen that the predicate A has in N the same extension as in M, and that N increases monotonically with M. As A does not occur in π_2 and $MM(\pi_{\neq}, D) = MM(ground(\pi_{\neq}, U))$, it follows that for $M = M^U_{\neq}$, $N \cap HB_{\pi_2[D]}$ is maximal w.r.t. inclusion. Consequently, $q^{\pi}_{p,MM}(D) = q^{\pi'}(D)$, and hence $q^{\pi}_{p,MM} = q^{\pi'}$. This shows that every query in DATALOG$^{\neq}$ is expressible in DATALOG$^{\vee}_{p}$. The result thus follows from Proposition 7.1. ∎

On the other hand, the inequality relation can not be recognized in plain disjunctive datalog. Let $q_{\neq} : Neq \to W$ be the boolean query which tells, given $D = (U, Neq)$, whether Neq is the inequality relation on U.

THEOREM 7.6. *$q_{\neq}$ is not expressible in* DATALOG$^{\vee}_{p}$.

PROOF. We show that expressibility of $q_{\neq}$ in DATALOG$^{\vee}_{p}$ would entail that every query in DATALOG$^{\vee,\neq}_{p,MM}$ (in particular, the Even-query) can be expressed in DATALOG$^{\vee}_{p}$. This raises a contradiction to Theorem 7.3.

Suppose that $q_{\neq} = q^{\pi_0}_{p,MM}$ for some DATALOG$^{\vee}$ query program π_0. Let $\pi_1 = \pi_0 \cup \pi_{\neq}$, where $\pi_{\neq}$ is the program from above. Notice that $\pi_0 \triangleright \pi_{\neq}$. Corollary 5.4 implies the following property (*): for every $D \in \mathcal{D}(\mathbf{R})$ such that $\mathbf{R}$ is disjoint with $RS(\pi_1)$, some $M \in MM(\pi_1, D)$ exists with $W \in M$, and Neq is the inequality relation on U in each $M \in MM(\pi_1, D)$ with $W \in M$.

Consider now any query $q : \mathbf{R} \to \mathbf{S}$, defined by some DATALOG$^{\vee,\neq}_{p}$ query program π'. Assume that the predicates in π_1 are distinct from those in $RS(\pi'), \mathbf{R}, \mathbf{S}$ (otherwise, use fresh predicates). Let π_2 be the program obtained from π be replacing each inequality atom $t_1 \neq t_2$ with $Neq(t_1, t_2)$, and by adding the atom W to the body of each rule. Let $\pi = \pi_2 \cup \pi_1$ and $IO(\pi) = (\mathbf{R}, \mathbf{S})$. Since $\pi_2 \triangleright \pi_1$, it follows from Corollary 5.4 that for every $D \in \mathcal{D}(\mathbf{R})$,

$$\{\, M - \{W\} \mid M \in MM(\pi, D),\, W \in M \,\} \cap HB_{\pi'[D]} \;=\; MM(\pi'[D]).$$

It is easy to see that, if $W \notin M$ for $M \in MM(\pi, D)$, then $S_i(\mathbf{t}) \notin M$, for every $S_i \in \mathbf{S}$ and tuple $\mathbf{t}$. This implies $q^{\pi}_{p,MM}(D) = q^{\pi_2}_{p,MM}(D)$; since D was arbitrary, it follows $q^{\pi}_{p,MM} = q^{\pi_2}_{p,MM}$. Since no inequality literals occur in π_2, this means that q is expressible in DATALOG$^{\vee}_{p}$. However, for $q = q_{Even}$ this is not true (Theorem 7.3). This proves the theorem. ∎

8. COMPLEXITY OF DISJUNCTIVE DATALOG

An important aspect of a query language is its computational complexity, which is traditionally measured by data and expression complexity, cf. [Vardi 1982]. The *data* (resp. *expression*) *complexity* of a query language $\mathcal{L}$ is defined as the complexity of the QOT-problem in Section 2 for the queries defined by expressions $E \in \mathcal{L}$, where E is part of the input and E (resp. D) is *fixed*.

The data complexity is usually closely tied to the expressive power of a query language. In particular, data complexity provides an upper bound for the expressive power. On the other hand, the fact that hard queries can be formulated does not necessarily imply that all queries within the same complexity can be formulated. A well-known example is datalog, which can express P-time hard queries, but not all P-time queries [Kanellakis 1990]. Such query languages are not balanced in the sense that the complexity dominates the expressive power. As will be shown in the following, plain disjunctive datalog suffers this property, while $\mathrm{DATALOG}_p^{\vee,\neq}$ is balanced. Notice that allowing stratified negation in nondisjunctive datalog programs does not establish the balance of the language, since still only a strict subset of the P-time computable queries can be expressed, where each level of stratification adds expressive power [Kolaitis 1991]. On the other hand, inspection of the proofs in Section 6 yields that a single level of stratified negation in disjunctive programs gives the full expressive power.

Similar with respect to expressibility, we observe that constants in query programs are not an issue with respect to the complexity of disjunctive datalog, since constants can be provided by designated input relations. Thus, the complexity results established below apply also to disjunctive datalog without constants.

8.1 Data complexity

From Lemma 6.2, we have

Theorem 8.1. *The data complexity of* $\mathrm{DATALOG}_{p,\mathcal{S}}^{\vee,\neg}$ *queries is in* Σ_2^p *for* $\mathcal{S} = MM, SM, PM$.

From Theorem 6.3 it follows that $\mathrm{DATALOG}_{p,MM}^{\vee,\neq}$ queries are Σ_2^p-hard in the data size. In fact, this can be sharpened to plain disjunctive datalog queries defined by programs without $\neq$ and $\neg$. Thus, adding disjunction to pure Horn clauses allows one to express Σ_2^p-hard queries. The proof is via a program π for deciding whether an atom a occurs in some minimal model of a given propositional program (cf. [Eiter and Gottlob 1995b]).

Theorem 8.2. *The boolean* $\mathrm{DATALOG}_p^{\vee}$ *queries are* Σ_2^p*-hard in the data size, even if no negation occurs in query programs.*

Proof. Let π be a program consisting of clauses of the form

$$a_1 \vee a_2 \leftarrow t_1, t_2, t_3 \qquad (*)$$

where all a_1, a_2 and t_1, t_2, t_3 are (not necessarily distinct) propositional letters (i.e., 0-ary predicates). By Corollary 6 in [Eiter and Gottlob 1995b] it follows that, given π, deciding if for a fixed propositional letter P, it holds $P \in M$ for some $M \in MM(\pi)$ is Σ_2^p-hard, provided that a constant for *true*, given by $\top$, is available.

We define a fixed $\mathrm{DATALOG}^{\vee}$ query program π_1 with $IO(\pi_1) = (C; W)$, where C is a 5-ary relation and W is a 0-ary relation, such that $q_{p,MM}^{\pi_1}$ solves this problem on the input database D_π in which π is encoded as follows.

The universe U of D is given by $U = RS(\pi) \cup \{T, P\}$, i.e., by the set of propositional letters occurring in π and T, P, and C contains tuples $(a_1, a_2, t_1, t_2, t_3)$ for the clauses (*) in π. Program π consists of the following three clauses:

$$
\begin{aligned}
V(\top) &\leftarrow . \\
V(x_1) \vee V(x_2) &\leftarrow C(x_1, x_2, x_3, x_4, x_5), V(x_3), V(x_4), V(x_5). \\
W &\leftarrow V(P).
\end{aligned}
$$

Intuitively, V corresponds to a valuation of the propositional letters, where $V(Q)$ means that Q is true.

It is easily checked that $\pi_1[D_\pi]$ has a minimal model in which W occurs (i.e., $q^{\pi}_{p,MM}(D_\pi) = \mathit{true}$) iff π has a minimal model M with $P \in M$. Since D_π is polynomial-time constructible from π, the result follows. ■

8.2 Expression complexity

Intuitively, the expression complexity of disjunctive datalog is exponentially higher than the data complexity, since the size of $\pi[D]$ can be exponential in the size of π, as opposed to polynomial size in D. In fact, the expression complexity of DATALOG$^{\vee,\neg}_{p,MM}$ turns out to be complete for the exponential analog of $\Sigma^p_2 = \mathrm{NP}^{\mathrm{NP}}$, which is $\Sigma^{exp}_2 = \mathrm{NEXPTIME}^{\mathrm{NP}}$. This class consists of all problems which can be decided on a nondeterministic Turing machine in single exponential time (i.e., in time $O(2^{p(n)})$, where $p(n)$ is polynomial), with the use of an NP-oracle. Notice that few natural Σ^{exp}_2-complete problems are known (see [Kolaitis and Vardi 1995] for an example in the context of second-order logic).

Theorem 8.3. *The expression complexity of* DATALOG$^{\vee,\neg}_{p,\mathcal{S}}$ *is in* Σ^{exp}_2 *for* $\mathcal{S} = $ *MM*, *SM*, *PM*.

Proof. $\pi[D]$ is polytime constructible from D and $\mathit{ground}(\pi, U)$, whose size is clearly bounded by $n^n = 2^{n \log n}$, where n is the size of π plus D. By Proposition 6.1, deciding if $S_i(\mathbf{a}) \in M$ for some $M \in \mathcal{S}(\pi[D])$ is nondeterministically possible with an NP oracle in time polynomial in the size $\pi[D]$. Thus, deciding whether $S_i(\mathbf{a}) \in M$ for some $M \in \mathcal{S}(\pi, D)$ is possible by a nondeterministic Turing machine in single exponential time with respect to the size of π plus D by using an NP oracle. Thus, the QOT-problem (Section 2) is in $\Sigma^{exp}_2 = \mathrm{NEXPTIME}^{\mathrm{NP}}$.[9] ■

Analogous to the data complexity, hardness of the expression complexity for Σ^{exp}_2 already shows up in the fragment of DATALOG$^{\vee}_p$ queries, even if no negation occurs in the query program. The proof of this result relies on the use of complexity upgrade techniques that extend the results in [Balcázar et al. 1992; Papadimitriou and Yannakakis 1985], applied to results in [Stewart 1991], and the simulation of boolean circuits by a DATALOG$^{\vee}$ program without $\neq$ and negation under minimal model semantics. In [Kolaitis and Papadimitriou 1991], a similar simulation was described in DATALOG$^{\neg,\neq}$ using stratified negation and fixpoint semantics.

Briefly, the proof consists in a reduction of a problem on graphs, co-CERT3COL [Stewart 1991], in its succinct version co-CERT3COL$_S$ [Balcázar et al. 1992], i.e., the

[9] In fact, for fixed D, the size of $\mathit{ground}(\pi, U)$ is bounded by 2^{cn}; thus, the problem is even in $\mathrm{NE}^{\mathrm{NP}}$, the class of problems decidable on a nondeterministic Turing machine in linear exponential (i.e., $O(2^{cn})$) time with an NP oracle, and, as follows from Theorem 8.8, also complete for this class.

problem input consists of a boolean circuit C_w by which the bits of the standard problem input w (a binary string encoding the graph) can be computed. Using the complexity upgrade techniques, we show that co-CERT3COL$_S$ is Σ_2^{exp}-complete. This problem is reduced to evaluating boolean queries $q^{\pi}_{p,MM}$ over a fixed database D_0, defined by DATALOG$^{\vee}$ programs π that are constructed in polynomial time.

8.2.1 *Succinct representation.* It is well-known that the input representation (e.g., numbers in unary or binary notation) can have a drastic effect on the problem complexity. Such effects were investigated for the "succinct versions" of graph-theoretic problems in [Papadimitriou and Yannakakis 1985], where a graph whose vertices are elements of $\{0,1\}^n$ is represented by a boolean circuit with $2n$ input bits; the circuit outputs 1 if and only if the input represents two vertices which are connected by an edge.

Formally, a boolean circuit is a finite set of triples (gates) $\{g_i = (a_i, j, k) : 1 \leq i \leq t\}$, where $a_i \in \{in, and, or, not\}$ is the kind of the gate, and g_j, g_k, $j, k < i$ are the inputs of the gate, unless g_i is an *in*-gate, in which case, say, $j = k = 0$. For *not*-gates, $j = k$ (cf. [Kolaitis and Papadimitriou 1991]). As shown in [Papadimitriou and Yannakakis 1985], the succinct problem setting often leads to an exponential complexity increase.

The results and techniques of [Papadimitriou and Yannakakis 1985] (see also [Papadimitriou 1994]) were generalized in [Balcázar et al. 1992] to the succinct version of any decision problem A encoded as a language over $\{0,1\}$. The succinct representation of a binary word w is a boolean circuit that on input of a number i (in binary), outputs whether $i \leq |w|$ and in that case, the i-th bit of w ($|w|$ denotes the length of w). The succinct version A_S of A is as follows: Given a boolean circuit C_w describing a word w, decide whether $w \in A$.

The following theorem shows how to obtain hardness results under polynomial-time transformability ($\leq_m^p$) for the succinct problem version from hardness of the standard problem under polylogtime transformability ($\leq_m^{PLT}$). This theorem is a generalization of [Balcázar et al. 1992, Theorem 5], where the theorem is formulated for logtime transformability ($\leq_m^{LT}$). Roughly, a $\leq_m^{PLT}$-transformation is a problem mapping f such that, given w and an integer i, the size of $f(w)$ is bounded by a polynomial and the i-th bit of $f(w)$ is computable in polylogarithmic time; note that every such mapping defines a polynomial-time transformation. For problem A, let $long(A)$ be defined as follows: $w \in long(A)$ iff the binary expression of the length of w is in $\{1w : w \in A\}$.

THEOREM 8.4. *Let* **C1** *and* **C2** *be arbitrary complexity classes such that for every* $A \in \mathbf{C1}$, $long(A) \in \mathbf{C2}$. *Then, for every* B, *if* B *is hard for* **C2** *under* $\leq_m^{PLT}$*-transformation, then* B_S *is hard for* **C1** *under* $\leq_m^p$*-transformation.*

PROOF. (Sketch) The theorem can be easily shown along the lines of the proof of Theorem 5 in [Balcázar et al. 1992] by using a generalization of the following Conversion Lemma in [Balcázar et al. 1992]: If $A \leq_m^{LT} B$ then $A_S \leq_m^p B_S$. It can be seen, by carefully looking at the proof in [Balcázar et al. 1992], that the Conversion Lemma remains valid if $A \leq_m^{LT} B$ is replaced by $A \leq_m^{PLT} B$.

To show the theorem, assume that B is hard for **C2** under $\leq_m^{PLT}$-transformation, and let $A \in \mathbf{C1}$ be arbitrary. By hypothesis, $long(A) \in \mathbf{C2}$ and therefore $long(A) \leq_m^{PLT} B$. By the generalized Conversion Lemma, $long(A)_S \leq_m^p B_S$. Furthermore,

$A \leq_m^p long(A)_S$ ([Balcázar et al. 1992, Lemma 4]); by transitivity, $A \leq_m^p B_S$, and thus B_S is $\leq_m^p$-hard for **C1**. ■

A strengthening of this theorem from $\leq_m^p$-transformation to logspace-transformation is shown in [Veith 1994; Gottlob et al. 1995].

The proof of Theorem 8.8 uses this complexity upgrade technique. It gives a polynomial-time transformation of the succinct version co-CERT3COL$_S$ of the graph problem co-CERT3COL, to evaluating boolean $\neg$-free DATALOG$_p^\vee$ query programs over a fixed database. The query program involves a subprogram which simulates boolean circuits by which the relations in the input structure encoding the graph can be computed.

Problem co-CERT3COL is as follows (cf. [Stewart 1991]):

Instance of size n: a graph G on the vertices $\{0, 1, \ldots, n-1\}$ such that every edge is labeled with a disjunction of two literals, where each literal is over the boolean variables $\{X_{i,j} : i, j = 0, \ldots, n-1\}$.
Yes-instance of size n: an instance G of size n such that for some truth value assignment t to the boolean variables, the graph $t(G)$ obtained from G by including only those edges set true under t is not 3-colorable.

Stewart showed that the complementary problem CERT3COL (i.e., negated condition for Yes-instances) is complete for Π_2^p under projection translations. It can be easily seen that projection translatability implies $\leq_m^{PLT}$-transformability (cf. [Veith 1994]). Thus, co-CERT3COL is complete for Σ_2^p under $\leq_m^{PLT}$-transformation. Applying the complexity upgrade theorem, we thus obtain

THEOREM 8.5. co-CERT3COL$_S$ *is complete for* Σ_2^{exp} *under* $\leq_m^p$*-transformation.*

We next describe disjunctive datalog programs for deciding co-CERT3COL and for simulating a boolean circuit. The programs are used in the proof of Theorem 8.8.

8.2.2 *Labeled graph coloring program.* Problem co-CERT3COL can be decided by a boolean DATALOG$^\vee$ query program π_{Col} in which no negation occurs using possibility stable model semantics. For an instance of co-CERT3COL of size n, the input database of π_{Col} consists in a structure with universe $\{0, \ldots, n-1\}$ and certain relations encoding the labeled graph.

We may assume that co-CERT3COL is encoded using a binary relation E and 4-ary relations P and N, such that $E(x, y) \vee E(y, x)$ (resp. $P(x, y, u, v) \vee P(y, x, u, v)$, $N(x, y, u, v) \vee N(y, x, u, v)$) encodes that an edge between x and y exists (resp. the edge between x and y is labeled by $X_{u,v}$, $\neg X_{u,v}$).

Thus, the input relations of π_{Col} are E, N and P; the output relation of π_{Col} is the 0-ary relation W. The program π_{Col} uses in addition binary predicates T and $\tilde{T}$, where $T(u, v)$ means that $X_{u,v}$ has value *true*; $\tilde{T}$ is intended to be the complement of T. Furthermore, π_{Col} uses unary predicates R, B, and G, which encode the color of a node. π_{Col} consists of the following clauses ($C = R, G, B$):

$$
\begin{array}{ll}
(1) & T(u,v) \vee \tilde{T}(u,v) \leftarrow . \\
(2) & R(x) \vee G(x) \vee B(x) \leftarrow . \\
(3) & C(x) \leftarrow W. \\
(4) & W \leftarrow P(x,y,u,v), T(u,v), C(x), C(y). \\
(5) & W \leftarrow N(x,y,u,v), \tilde{T}(u,v), C(x), C(y).
\end{array}
$$

Intuitively, clause (1) chooses a truth value for $X_{u,v}$. Dependent on that choice, W is computed true iff the resulting graph $t(G)$ is not 3-colorable.

LEMMA 8.6. *Let $D_I = (\{0,\ldots,n-1\}, P, N)$ be the input database for π_{Col} for an instance I of* co-CERT3COL*. Then, $W \in M$ for some $M \in MM(\pi_{Col}, D)$ iff I is a Yes-instance.*

PROOF. ($\Leftarrow$). Let t be a valuation of the $X_{u,v}$ such that $t(G)$ is not 3-colorable. Let M be the interpretation of $\pi_{Col}[D]$ such that W is true, $T(u,v) \in M$ iff $t(X_{u,v}) = \mathit{true}$, $\tilde{T}(u,v) \in M$ iff $t(X_{u,v}) = \mathit{false}$, for all $u, v \in U$, and $R = G = B = U$. As easily checked, M is a minimal model of $\pi_{Col}[D]$.
($\Rightarrow$). Assume that M is a minimal model of $\pi_{Col}[D]$ in which W is true. From Corollary 5.4, we can infer that for each $u, v \in U$, either $T(u,v) \in M$ or $\tilde{T}(u,v) \in M$, but not both. Since M is a minimal model, we infer that for the valuation t of the $X_{u,v}$ defined by $t(X_{u,v}) = \mathit{true}$ if $T(u,v) \in M$ and $t(X_{u,v}) = \mathit{false}$ if $T(u,v) \notin M$, for each assignment of one of the colors red, green and blue to every node, two adjacent nodes of the graph $t(G)$ are colored equally. Hence, I is a Yes-instance. ■

8.2.3 *Boolean circuit program.* Let $C = \{g_i = (a_i, j, l), 1 \le i \le t\}$ be a boolean circuit that decides a k-ary predicate R over $U_0 = \{0,1\}$, i.e., for any tuple $\mathbf{t} \in U_0^k$ supplied to C as input, a designated output gate of C, which we assume is g_t, has value 1 iff $\mathbf{t} \in R$.

We describe a DATALOG$^\vee$ program π_C that simulates C using the universe U_0. For each gate g_i, π_C uses a k-ary predicate G_i, where $G_i(\mathbf{x})$ informally states that on input of tuple $\mathbf{x}$ to C, the circuit computation sets the output of g_i to 1. Moreover, it uses a propositional letter *False*, which is true in those models in which the G_i do not have the intended interpretation; none of these models will be minimal.

The clauses of π_C are the following ones. For each gate $g_i = (a_i, j, l)$ of C, it contains the clause

$$(00) \quad G_i(\mathbf{x}) \leftarrow \mathit{False}.$$

Depending on the type a_i, π_C contains for g_i additional clauses:

$$
\begin{array}{lll}
a_i = \mathit{in}: & (01) & G_i(x_1,\ldots,x_{j-1},1,x_{j+1},\ldots,x_k) \leftarrow\ . \\
 & (02) & \qquad\qquad \mathit{False} \leftarrow G_i(x_1,\ldots,x_{j-1},0,x_{j+1},\ldots,x_k) \\[6pt]
a_i = \mathit{not}: & (03) & G_i(\mathbf{x}) \vee G_j(\mathbf{x}) \leftarrow\ . \\
 & (04) & \qquad\qquad \mathit{False} \leftarrow G_j(\mathbf{x}), G_i(\mathbf{x}). \\[6pt]
a_i = \mathit{and}: & (05) & \qquad\quad G_i(\mathbf{x}) \leftarrow G_j(\mathbf{x}), G_l(\mathbf{x}). \\
 & (06) & \qquad\quad G_j(\mathbf{x}) \leftarrow G_i(\mathbf{x}). \\
 & (07) & \qquad\quad G_l(\mathbf{x}) \leftarrow G_i(\mathbf{x}). \\[6pt]
a_i = \mathit{or}: & (08) & \qquad\quad G_i(\mathbf{x}) \leftarrow G_j(\mathbf{x}). \\
 & (09) & \qquad\quad G_i(\mathbf{x}) \leftarrow G_l(\mathbf{x}). \\
 & (10) & G_j(\mathbf{x}) \vee G_l(\mathbf{x}) \leftarrow G_i(\mathbf{x}).
\end{array}
$$

The clauses (00) ensure that if a model of $\mathit{ground}(\pi_C, U_0)$ contains *False*, it is the maximal interpretation (which is trivially a model of π_C). In fact, this is the only model of π_C that contains *False*. Let M_C denote the interpretation of $\mathit{ground}(\pi_C, U_0)$ given by $M_C = \{G_i(\mathbf{t}) \mid g_i \in C,\ \mathbf{t} \in U_0^k$, and g_i takes value 1 on input $\mathbf{t}$ to $C\ \}$.

Lemma 8.7. *For any boolean circuit C, M_C is the unique minimal model of $\mathit{ground}(\pi_C,\ U_0)$.*

Proof. In what follows, let $P = \mathit{ground}(\pi_C, U_0)$. It is easily checked that M_C is a model of P. Since HB_P is the only model M of P such that $\mathit{False} \in M$, it suffices to show that M_C is the only model M of P such that $\mathit{False} \notin M$.

We need additional concepts. The *level* $\ell_C(g_i)$ of gate $g_i = (a_i, j, l)$ in C is the length of the longest path from any input gate to g_i. Formally,

$$
\ell_C(g_i) = \begin{cases} 0, & \text{if } a_i = \mathit{in}; \\ \max(\ell_C(g_j), \ell_C(g_l)) + 1, & \text{otherwise.} \end{cases}
$$

For each model M of P, let for each $d \geq 0$,

$$
M^{(d)} = M \cap \{G_i(\mathbf{t}) : \ell_C(g_i) \leq d, \mathbf{t} \in \{0,1\}^k\ \},
$$

i.e., M restricted to all predicates G_i such that g_i occurs in C at level $r \leq d$.

We show by induction on the levels d of C the following: For each model M of P with $\mathit{False} \notin M$, it holds $M^{(d)} = M_C^{(d)}$; this clearly implies $M = M_C$.

(Base) Let $d = 0$. It is easy to see that $M_C^{(0)} \subseteq M^{(0)}$. Assume that $M^{(0)} \not\subseteq M_C^{(0)}$. Then, for some input gate $g_i = (\mathit{in}, j, l)$ of C and tuple $\mathbf{t}$, we have $G_i(\mathbf{t}) \in M^{(0)}$ and $G_i(\mathbf{t}) \notin M_C^{(0)}$. From the definition of M_C and clause (02), we infer that the j-th component of $\mathbf{t}$ is 0; consequently, from clause (03) we infer that $\mathit{False} \in M$, which is a contradiction. It follows that $M^{(0)} \subseteq M_C^{(0)}$, and hence $M^{(0)} = M_C^{(0)}$.

(Induction) Assume that the statement holds for $d \geq 0$ and consider $g_i = (a_i, j, l)$ such that $\ell_C(g_i) = d+1$. Consider for $\mathbf{t} \in \{0,1\}^k$ the two possibilities for membership of $G_i(\mathbf{t})$ in M_C.

Case 1: $G_i(\mathbf{t}) \in M_C^{(d+1)}$. We show that in this case, $G_i(\mathbf{t}) \in M^{(d+1)}$. There are three possibilities for the type a_i of gate g_i.

(1) $a_i = not$: By definition, we have that $G_j(\mathbf{t}) \notin M_C$. Hence, it follows from the induction hypothesis that $G_j(\mathbf{t}) \notin M$; thus from clause (03), we infer that $G_i(\mathbf{t}) \in M^{(d+1)}$.

(2) $a_i = and$: By definition, $G_j(\mathbf{t}), G_l(\mathbf{t}) \in M_C$; hence, by the induction hypothesis, it follows that $G_j(\mathbf{t}), G_l(\mathbf{t}) \in M$. Thus from clause (05), we infer that $G_i(\mathbf{t}) \in M^{(d+1)}$.

(3) $a_i = or$: By definition, we have that $G_j(\mathbf{t}) \in M_C$ or $G_l(\mathbf{t}) \in M_C$. If $G_j(\mathbf{t}) \in M_C$, it follows from the induction hypothesis that $G_j(\mathbf{t}) \in M$. Hence, from clause (08) we infer that $G_i(\mathbf{t}) \in M^{(d+1)}$. The case $G_l(\mathbf{t}) \in M_C$ is analogous.

Case 2: $G_i(\mathbf{t}) \notin M_C^{(d+1)}$. We show that in this case, $G_i(\mathbf{t}) \notin M^{(d+1)}$.

(1) $a_i = not$: By definition, we have that $G_j(\mathbf{t}) \in M_C$, and thus by the induction hypothesis, it follows that $G_j(\mathbf{t}) \in M$. From clause (04) we infer that $G_i(\mathbf{t}) \notin M^{(d)}$.

(2) $a_i = and$: By definition, we have that $G_j(\mathbf{t}) \notin M_C$ or $G_l(\mathbf{t}) \notin M_C$. Assume that $G_j(\mathbf{t}) \notin M_C$. From the induction hypothesis, it follows that $G_j(\mathbf{t}) \notin M$. From clause (06), we infer that $G_i(\mathbf{t}) \notin M^{(d+1)}$. The case $G_l(\mathbf{t}) \notin M_C$ is analogous.

(3) $a_l = or$: By definition, we have that $G_j(\mathbf{t}), G_l(\mathbf{t}) \notin M_C$, and, by the induction hypothesis, we infer that $G_j(\mathbf{t}), G_l(\mathbf{t}) \notin M$. From clause (10), we infer that $G_i(\mathbf{t}) \notin M^{(d+1)}$. qed.

Thus, the claimed statement holds for $d + 1$. This concludes the proof. ∎

Theorem 8.8. *The boolean* DATALOG$^{\vee}_{p}$ *queries are* Σ_2^{exp}*-hard in the program size.*

Proof. Let C_I be the boolean circuit of an instance I of co-CERT3COL of size n. We describe a $\leq_m^p$-transformation of deciding I into evaluating a program π_I on the fixed database $D_0 = (U_0)$.

From C_I, boolean circuits C_P and C_N are constructible in polynomial time that compute P and N, respectively, where each argument x, y, u, and v is encoded by a string of $\lceil \log n \rceil$ bits. The circuits output 1 if a tuple is contained in P (resp. N) and 0 otherwise; for any string representing m with $n \leq m < 2^{\lceil \log n \rceil}$, the output is 0. Semantically, this means that we have added isolated nodes $n, n+1, \ldots, 2^{\lceil \log n \rceil}$ to the graph of I which are not labeled and hence never included in the graph $t(G)$.

C_P and C_N can be simulated by polynomial-time constructible DATALOG$^{\vee}$ programs π_{C_P} and π_{C_N} using U_0 as shown above, which use predicates G_i^P, False^P and G_i^N, False^N, respectively; assume that $G_{t_P}^P$ and $G_{t_N}^N$ are the predicates corresponding to the output gates $g_{t_P}^P$ and $g_{T_N}^N$ of C_P and C_N, respectively.

Program π_{Col} can be easily adapted to a variant π_{Col}^n where the elements from $\{0, \ldots, n-1\}$ are represented by binary strings of length $\lceil \log n \rceil$ and all occurrences of P, N are replaced by $G_{t_P}^P$ and $G_{t_N}^N$, respectively, in time polynomial in $\log n$. For the database D_I of I, let $D_I^* \in \mathcal{D}(G_{t_P}^P, G_{t_N}^N)$ be the database such that $U(D^*) = U_0$ and for each $\mathbf{t}' \in U_0^{4\lceil \log n \rceil}$,

$$D^* \models G_{t_P}^P(\mathbf{t}') \quad \text{iff} \quad \mathbf{t}' \text{ encodes a tuple } \mathbf{t}, \text{ and } D \models P(\mathbf{t}),$$
$$D^* \models G_{t_N}^N(\mathbf{t}') \quad \text{iff} \quad \mathbf{t}' \text{ encodes a tuple } \mathbf{t}, \text{ and } D \models N(\mathbf{t}).$$

Notice the following property (*): $W \in M$ for some $M \in MM(\pi_{Col}^n, D_I^*)$ iff $W \in M$ for some $M \in MM(\pi_{Col}, D_I)$.

Now let $\pi_I = \pi^n_{Col} \cup \pi_{C_N} \cup \pi_{C_P}$ and adopt $IO(\pi_I) = (\emptyset; W)$. Notice that $\pi^n_{Col} \rhd (\pi_{C_N} \cup \pi_{C_P})$, and that π_{C_N} and π_{C_P} do not share any predicates.

It holds that $W \in M$ for some $M \in MM(\pi_I, D_0)$, i.e., $q^{\pi_I}_{p,MM}(D_0) \models W$ iff I is a Yes-instance. Indeed, from Corollary 5.4 and Lemma 8.7, we can infer that for every $M \in MM(\pi_I, D_0)$, $M \cap HB_{\pi_{C_P}[D_0]} = M_{C_P}$ and $M \cap HB_{\pi_{C_N}[D_0]} = M_{C_N}$, respectively. Hence, from Corollary 5.4 and (*), it follows that $W \in M$ for some $M \in MM(\pi_I, D_0)$ iff $W \in M$ for some $M \in MM(\pi^n_{Col}, D^*_I)$ iff $W \in M$ for some $M \in MM(\pi_{Col}, D_I)$. Thus, by Lemma 8.6, $W \in M$ for some $M \in MM(\pi_I, D_0)$ iff I is a Yes-instance.

Since π_I can be constructed from C_I in polynomial time, the result follows. ■

Notice that evaluating a given DATALOG$^{\vee,\neg}$ query program under $p, \mathcal{S}$ semantics on a given database (termed *combined complexity* in [Vardi 1982]), is Σ^{exp}_2-complete as well.

Remarks. Without loss of generality, we could bypass *or*-gates in the boolean circuit simulation, by simulating *or*-gates with *and*-gates and *not*-gates. Moreover, the transformation described in the proof of Theorem 8.8 can be performed in logspace. Thus, by using the strengthening of Theorem 8.4 from $\leq^p_m$-transformation to logspace-transformation [Veith 1994; Gottlob et al. 1995], one can obtain a strengthening of Theorem 8.8 to logspace-transformation.

In fact, the function of a circuit can be computed by a program that uses *neither disjunction nor negation*, i.e., a pure datalog program; this improves on the proof approach in [Kolaitis and Papadimitriou 1991], where negation was needed. The following observation is crucial for this improvement.

Fact. For every Boolean circuit C, there exists an equivalent Boolean circuit C' that has $O(|C|)$ size, so that C' has negation only immediately after input gates; moreover, such a C' can be easily constructed from C in logspace.

Indeed, introduce for each gate g_i a new gate g^*_i that computes the complement of g_i; if g_i is an *in*-gate, g^*_i is a *not*-gate which is fed by g_i; if g_i is an *and*-gate (resp. *or*-gate) with input gates g_j and g_k, then g^*_i is an *or*-gate (resp. *and*-gate) with inputs g^*_j and g^*_k; finally, if g_i is a *not*-gate fed by g_j, then it is changed to an *or*-gate whose both inputs are g^*_j, and g^*_i is an *or*-gate with g_j for both inputs. Clearly, this yields the desired C', which can be constructed in logspace.

Negation on input gates can be realized quite easily in a program for circuit simulation. If g_i is a *not*-gate fed by an *in*-gate $g_k = (a_k, j, l)$, then use the rule

$$(20) \quad G_i(x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_k) \leftarrow \ .$$

To simulate the circuit C' that has only input-negation, it is sufficient to use beside such rules the rules (01), (05) and (08) from the above circuit program for *in*-gates, *and*-gates, and *or*-gates, respectively. The resulting program, $\pi^*_{C'}$, is a pure datalog program; it is equivalent to the standard program $\pi_{C'}$ for C', i.e., $\pi^*_{C'}$ has $M_{C'}$ as its unique minimal model on U_0.

Notice that this way we obtain, using complexity upgrading techniques, a simple proof of the folklore result that pure DATALOG has EXPTIME-complete expression complexity, which is implicit already in [Vardi 1982; Chandra and Harel 1982]. Indeed, the exponential upgrade of P is EXPTIME; problems that are complete for P under $\leq^{PLT}_m$ reductions (or even stronger notions of reductions) can be expressed in pure datalog. An example is the CIRCUIT VALUE problem (compute the output of a Boolean circuit on a given input). This problem is complete for P under projection translations, even if negation is restricted to input gates [Veith 1996] (essentially, this can be seen as an encoding of the MONOTONE CIRCUIT VALUE problem in [Papadimitriou 1994]). A pure datalog program solving this problem is straightforward. This implies EXPTIME-hardness; membership is obvious since the data complexity is well-known polynomial.

9. DISCUSSION AND CONCLUSION

The results of this paper give a precise picture of the computational and expressive power of various semantics of disjunctive datalog over relational databases (i.e., finite structures). Moreover, by our complexity-theoretic characterization of the expressiveness of disjunctive datalog (Σ_2^p), we are able to compare disjunctive datalog with well-known database query languages. These relationships are depicted in Figure 2. Proofs and/or references for the other query languages appearing in this figure can found in [Abiteboul et al. 1995]; here, $\mathrm{DATALOG}^{\neg}_{wf}$ denotes nondisjunctive datalog with well-founded negation [van Gelder et al. 1991]. Solid lines represent proper inclusion ($\subset$), while dashed lines stand for possibly improper inclusion ($\subseteq$). In the latter case, it is currently not known, but strongly believed that the inclusion is proper. Note that recent results of Saccà show that normal logic programming using maximal or least undefined partial stable models can have the same expressive power as disjunctive logic programming [Saccá 1995].

$\mathrm{DATALOG}^{\vee,\neg}$ appears to be a language of very high expressive power, and therefore, the worst-case complexity of answering queries in this formalism is very high, which may be seen as a serious drawback. A similar comment applies to several other expressive query languages studied in the literature, for example, to the language of While-Queries studied by Chandra and Harel [Chandra and Harel 1982] and the language of Partial Fixpoint Queries considered by Abiteboul and Vianu (cf. [Abiteboul et al. 1995]). The latter two languages both capture the highly intractable complexity-class PSPACE. This raises the question of whether it makes sense to study such expressive, and therefore complex, query languages. For the following reasons, we believe that it indeed does make sense, in particular, in the case of $\mathrm{DATALOG}^{\vee,\neg}$.

First, $\mathrm{DATALOG}^{\vee,\neg}$ is a natural and conservative extension of the tractable query language DATALOG. Thus, whatever query can be posed in DATALOG (or even in $\mathrm{DATALOG}^{\neg_s}$) can be posed in exactly the same way in $\mathrm{DATALOG}^{\vee,\neg}$ and can be answered in polynomial time. Therefore, with respect to plain DATALOG and $\mathrm{DATALOG}^{\neg_s}$ queries, $\mathrm{DATALOG}^{\vee,\neg}$ inherits all good properties.

Moreover, several problems of high practical relevance can be formulated in $\mathrm{DATALOG}^{\vee,\neg}$, but presumably not in plain DATALOG (for examples, see Section 4). Clearly, we cannot expect a database engine to solve, say, the Traveling Salesman Problem (TSP), in polynomial time. However, such problems occur in practice and have to be solved. Disjunctive datalog allows us to *formulate* such hard queries very elegantly, and gives us a chance to get an answer for small instances in reasonable time. In plain datalog, unless the Polynomially Hierarchy collapses, it is not possible to formulate any NP-hard problem, even on enumerated databases unless proper restrictions are imposed on the problem description. It appears that also in

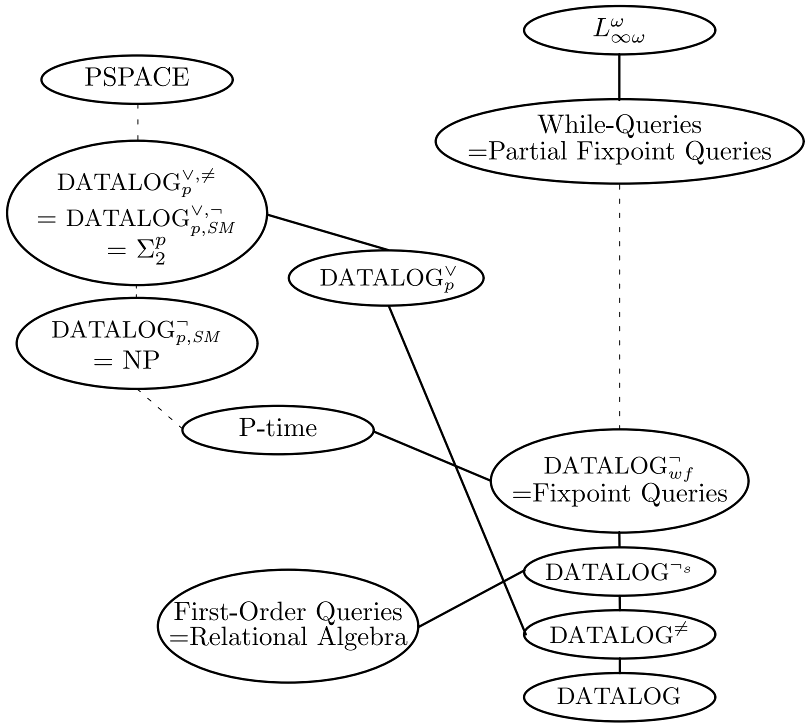

Fig. 2. Expressive powers of disjunctive datalog and other query languages

this case, the plain datalog formulation of a query can be rather cumbersome; for example, if the number of cities in the TSP is bounded by a constant.

We advocate that a modular approach to the expressiveness/complexity trade-off should be taken. A solid kernel query language (in our case, $\text{DATALOG}^{\neg_s}$) guarantees polynomial response time at the cost of a limited expressiveness. Extending this kernel, a much more expressive language (in our case $\text{DATALOG}^{\vee,\neg}$) is made available, in which hard queries can be formulated. Clearly, even if this extended language is implemented by smart procedures that behave satisfactorily in many cases, no guarantee about response time can be given. The extended language should be used *at the user's own risk*. For the effectiveness of this approach, the core language must be easily distinguishable (at the syntactic level) from the extended language. This is indeed the case with $\text{DATALOG}^{\vee,\neg}$: it is easily (polynomially) recognizable whether a query does or does not contain disjunctions or unstratified negation. For a discussion of similar modular approaches in the field of terminological databases, cf. Section 6 in [Borgida 1995]; for a related discussion of the *expressiveness versus complexity* question, see [Doyle and Patil 1991].

Finally, we believe that the exact determination of the expressiveness and complexity of $\text{DATALOG}^{\vee,\neg}$ at the second level of the Polynomial Hierarchy is important for two reasons. The expressiveness results allow us to separate the queries that can be represented in our query language from those that (presumably) can't. For example, important optimization problems such as computing optimal tours for the TSP or computing maximal eigenvectors are expressible, while playing games like GO (which is PSPACE-complete, cf. [Papadimitriou 1994]) is not possible in DATALOG$^{\vee,\neg}$ unless the Polynomial Hierarchy collapses. The complexity results give us useful information about the algorithmic nature of the query-answering problem. As demonstrated in Section 4, a Σ_2^p-complete problem can be solved by a guess-and-check algorithm that generates guesses and checks their appropriateness via a co-NP procedure. We can thus identify two independent and, in a sense, orthogonal sources of complexity: guessing suitable candidates for stable models and checking them. This insight is important for the design of practical algorithms and heuristics that attack these two sources of complexity as much as possible, and, in particular, for the identification and recognition of tractable subcases. Several current projects deal with the implementation of disjunctive datalog or, more generally, of disjunctive logic programming, in which the design of such algorithms and methods is a primary goal.

For example, the group of Dix and Furbach [Dix and Müller 1993] in Koblenz (Germany), in an ambitious project, implements the disjunctive stable semantics and other semantics of disjunctive logic programming; in particular, they implement the recently proposed static semantics [Przymusinski 1994; Przymusinski 1995], which is very similar to the *D-WFS* semantics [Brass and Dix 1995; S.Brass et al. 1996b; S.Brass et al. 1996a] and the well-founded circumscription semantics [You and Yuan 1993]; in fact, for disjunctive datalog, static semantics and well-founded circumscription semantics coincide (cf. [You and Yuan 1995]). The static semantics associates with each program a set of modal formulas as its meaning; it appears to be the best candidate for a natural extension of the well-founded semantics to disjunctive logic programs. Notice that, since the static semantics coincides on $\neg$-free disjunctive logic programs with the minimal model semantics [S.Brass et al. 1996b], it is clear from our results that DATALOG$^{\vee,\neg}$ programs can express under this semantics every query in Π_2^p. Moreover, preliminary results based on characterizations in [S.Brass et al. 1996b] indicate that it can only express such queries, and thus has the same expressive power as DATALOG$^{\vee,\neg}$ under the certainty variant of the stable model semantics; we leave this for further study.

Seipel [Seipel 1994] in Würzburg (Germany) implements various semantics and methods developed by Minker and his students [Lobo et al. 1992; Minker 1994] at College Park (Maryland, U.S.A.). There, V.S. Subrahmanian and his coworkers [Bell et al. 1994; Nerode et al. 1995; Bell et al. 1996], use integer programming methods for implementing normal and disjunctive logic programming. Ben-Eliyahu [Ben-Eliyahu and Dechter 1994; Ben-Eliyahu and Palopoli 1994] (Ben Gurion University and Technion, Israel) has recently started a project of implementing a subclass of disjunctive logic programs (so called headcycle-free programs) that is less complex (thus also less powerful) than the full class; see [Eiter et al. 1996] for more details.

Last but not least, a project for implementing a prototype of disjunctive datalog based on the stable model semantics (funded by the Austrian Science Foundation (FWF)) was most recently started at the information systems department of TU Vienna. Results and techniques from [Leone et al. 1995] are a starting point for this work.

The current and future research of these groups is centered around the development of smart algorithms for disjunctive logic programming. Such algorithms should be able to recognize cases where the high worst case complexity of disjunction can be avoided, possibly using efficient rewriting techniques.

On the more theoretical side, we currently investigate the expressive power of nonmonotonic formalisms that are related to disjunctive logic programming.

One of the most popular nonmonotonic logics is Reiter's *default logic* [Reiter 1980]. This logic was developed as a knowledge representation formalism and was originally not conceived as a database query language. However, in [Cadoli et al. 1994] a suitable setting was defined in which default logic can be used as a query language for relational databases (*Default Query Language, DQL*). It was shown in [Cadoli et al. 1994] that DQL has over relational databases the same expressive power as disjunctive datalog. This naturally raises the question about how these formalisms are related, in particular with regard to possible translations.

On one hand, note that each disjunctive logic program can be translated into an equivalent default theory. A concrete translation, related to earlier work in [Bidoit and Froidevaux 1991], was described by Sakama and Inoue in [Sakama and Inoue 1993]. Their translation is rather intuitive and *modular*, which means that the translation of a disjunctive logic program is obtained by the union of the independent translations of the single rules. In this translation, each rule is rewritten to a default and further normal defaults enforcing the closed world assumption are added. A related proposal for handling disjunction by using normal defaults has been recently pointed out in [Turner 1996].

Based on the translation mapping of [Sakama and Inoue 1993], a translation from $\text{DATALOG}^{\vee,\neg}$ under the stable semantics into DQL can be obtained. Therefore, as already remarked in [Cadoli et al. 1994], the fact that DQL expresses at least as much as $\text{DATALOG}^{\vee,\neg}$ under the stable semantics follows from Theorems 6.9, 6.11 of the present paper in combination with the translation result by Sakama and Inoue. Notwithstanding, a direct expressiveness proof for DQL, which does not rely on disjunctive datalog but exploits particular syntactic features of DQL was given in the appendix of [Cadoli et al. 1994]. That proof is instructive to users of default logic, because it shows how database properties can be expressed in DQL via their semantics in second-order logic.

On the other hand, to the best of our knowledge, no translation from default logic to disjunctive logic programming is known in the literature. Therefore, the expressivity results in Section 6 of the present paper cannot be obtained from the results in [Cadoli et al. 1994] by means of translations. Genuine – and more involved – proofs were in order to determine the expressive power of disjunctive datalog (in particular, for $\neg$-free disjunctive datalog).

By the results of the present paper it follows that it is possible to devise a translation from DQL to $\text{DATALOG}^{\vee,\neg}$ (even to $\text{DATALOG}^{\vee,\neq}$), which also gives us a translation from default logic to disjunctive logic programming. In fact, each DQL query can be easily expressed by a Σ_2^1 second-order formula, which, as shown in [Eiter et al. 1996] can be brought into the normal form used in Theorem 6.3, and thus translated into $\text{DATALOG}^{\vee,\neq}$. However, this translation has two shortcomings. First, it is not a true *embedding* since it extends the language by new predicate symbols. Second, it is not modular.

The study of modular embeddings between database languages appears to be important for understanding their relationship. If a modular embedding from query language $\mathcal{L}_1$ to $\mathcal{L}_2$ exists, then, in a sense, $\mathcal{L}_1$ can be seen as a fragment or sublanguage of $\mathcal{L}_2$.

Interestingly, there exists no modular embedding of DQL into disjunctive datalog. This can be seen as follows. In [Lifschitz and Schwarz 1993; Chen 1993], modular embeddings of disjunctive logic programming into autoepistemic logic [Moore 1985] are exhibited. Thus, if there existed a modular embedding of default logic into disjunctive logic programming, by composition, there would also exist a modular embedding of default logic into autoepistemic logic. This would contradict a result in [Gottlob 1995] which proves that there exists no modular embedding of default logic into autoepistemic logic.

The existence of a modular embedding of DATALOG$^{\vee,\neg}$ into DQL and the nonexistence of a modular embedding in the reverse direction can be interpreted as follows. Loosely speaking, DATALOG$^{\vee,\neg}$ amounts to a fragment of DQL which, as shown in the present paper, provides the full expressive power over relational databases. On the other hand, DQL provides more complicated syntactic features that cannot be mapped into DATALOG$^{\vee,\neg}$ by a simple translation.

From a practical point of view, in the context of deductive databases disjunctive datalog seems to be the more suitable extension of DATALOG$^{\neg}$ than DQL. Due to its plain syntax, DATALOG$^{\vee,\neg}$ is amenable for automatic program analysis and optimization. In particular, modularity properties (as described in Section 5) and natural syntactic restrictions on DATALOG$^{\vee,\neg}$ programs (such as head-cycle freeness) can be exploited to improve efficiency [Ben-Eliyahu and Dechter 1994; Leone et al. 1995]. On the other hand, due to its rich (and in some sense more succinct) syntax, DQL is far less accessible to such methods. For example, the concept of stratification for default theories is much more intricated than for logic programs [Cholewinski 1995]. The strength of DQL is that certain AI problems such as model-based diagnostic reasoning about complex devices can be more naturally expressed in that language [Cadoli et al. 1994]. It seems that DQL is a suitable language for bridging the impedance mismatch between advanced knowledge representation and relational databases.

Another aspect highlighting the difference between disjunctive datalog and DQL is their expressive power over disjunctive (clausal) databases. As shown in [Bonatti and Eiter 1996], the expressive powers of the two languages are different in that context.

Autoepistemic logic [Moore 1985] has the same expressive power as default logic [Bonatti and Eiter 1996]. Promising variants of default logic such as stable class default logic [Baral and Subrahmanian 1992] remain to be explored.

ACKNOWLEDGMENTS

The authors would like to thank several people for their comments on this work. In particular, we are indebted to P.A. Bonatti, J. Dix, K. Inoue, Ph. Kolaitis, J.S. Schlipf, H. Veith, and V. Vianu for important hints and comments. Furthermore, we are grateful to the anonymous referees for their constructive comments and helpful suggestions for improvement.

APPENDIX

Perfect models

The perfect models of a program π are defined as follows (cf. [Przymusinski 1988]). A priority relation $<$ is defined on HB_π using an auxiliary relation $\preceq$ as follows. For each clause

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg c_1, \cdots, \neg c_m$$

from $ground(\pi)$, it holds that

(i) $a_i < c_j$, for all $1 \leq i \leq n$, $1 \leq j \leq m$,
(ii) $a_i \preceq b_j$, for all $1 \leq i \leq n$, $1 \leq j \leq k$, and
(iii) $a_i \preceq a_j$ for all $1 \leq i, j \leq n$.

Now $<$ and $\preceq$ are the smallest extensions to (i)–(iii) which satisfy $a < b \Rightarrow a \preceq b$ and which are closed under transitivity, i.e. $a \preceq b, b \preceq c \Rightarrow a \preceq c$ and $a \preceq b$, $b < c$ (resp. $d < a$) $\Rightarrow a < c$ (resp. $d < b$). Intuitively, $a < b$ means that b has higher priority than a.

Now define the *preference order* $\ll$ on the models of π by $M_1 \ll M_2$ iff $M_1 \neq M_2$ and for each $a \in M_1 - M_2$ there exists $b \in M_2 - M_1$ such that $a < b$.

DEFINITION .1. *A model M of π is* perfect *iff there is no model M' of π such that $M' \ll M$; the collection of all perfect models of π is denoted by $PM(\pi)$.*

Notice that $M_1 \subset M_2$ implies $M_1 \ll M_2$; thus, $PM(\pi) \subseteq MM(\pi)$.

REFERENCES

ABITEBOUL, A., SIMON, E., AND VIANU, V. 1990. Non-Deterministic Languages to Express Deterministic Transformations. In *Proceedings PODS-90* (1990), pp. 218–229.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases.* Addison-Wesley.

BALCÁZAR, J., LOZANO, A., AND TORÁN, J. 1992. The Complexity of Algorithmic Problems on Succinct Instances. In R. BAETA-YATES AND U. MANBER Eds., *Computer Science*, pp. 351–377. New York: Plenum Press.

BARAL, C. AND GELFOND, M. 1994. Logic Programming and Knowledge Representation. *Journal of Logic Programming 19/20*, 73–148.

BARAL, C. AND SUBRAHMANIAN, V. 1992. Stable and Extension Class Theory for Logic Programs and Default Logic. *Journal of Automated Reasoning 8*, 345–366.

BELL, C., NERODE, A., NG, R., AND SUBRAHMANIAN, V. 1994. Mixed Integer Programming Methods for Computing Non-Monotonic Deductive Databases. *Journal of the ACM 41*, 6 (November), 1178–1215.

BELL, C., NERODE, A., NG, R., AND SUBRAHMANIAN, V. 1996. Implementing Deductive Databases by Mixed Integer Programming. *ACM Trans. on Database Syst. 21*, 2, 238–269.

BEN-ELIYAHU, R. AND DECHTER, R. 1994. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence 12*, 53–87.

BEN-ELIYAHU, R. AND PALOPOLI, L. 1994. Reasoning with Minimal Models: Efficient Algorithms and Applications. In *Proceedings Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)* (1994), pp. 39–50.

BIDOIT, N. AND FROIDEVAUX, C. 1987. Minimalism Subsumes Default Logic and Circumscription in Stratified Logic Programming. In *Proceedings LICS-87* (1987), pp. 89–97. IEEE Computer Science Press.

BIDOIT, N. AND FROIDEVAUX, C. 1991. General Logic Databases and Programs: Default Semantics and Stratification. *Information and Computation 19*, 15–54.

BONATTI, P. A. AND EITER, T. 1996. Querying Disjunctive Databases Through Nonmonotonic Logics. *Theoretical Computer Science 160*, 321–363.

BORGIDA, A. 1995. Description Logics in Data Management. *IEEE Transactions on Knowledge and Data Engineering 7*, 5, 671–682.

BRASS, S. AND DIX, J. 1995. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. In L. STERLING Ed., *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo* (June 1995), pp. 199–213. MIT Press.

CADOLI, M., EITER, T., AND GOTTLOB, G. 1994. Using Default Logic as a Query Language. In *Proceedings Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)* (1994), pp. 99–108.

CERI, S., GOTTLOB, G., AND TANCA, L. 1990. *Logic Programming and Databases*. Springer.

CHANDRA, A. AND HAREL, D. 1982. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences 25*, 99–128.

CHANDRA, A. AND HAREL, D. 1985. Horn Clause Queries and Generalizations. *Journal of Logic Programming 2*, 1–15.

CHEN, J. 1993. Minimal Knowledge + Negation as Failure = Only Knowing (Sometimes). In L.-M. PEREIRA AND A. NERODE Eds., *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)* (Lisbon, Portugal, July 1993), pp. 132–150. MIT Press.

CHOLEWINSKI, P. 1995. Reasoning with Stratified Default Theories. In W. MAREK, A. NERODE, AND M. TRUSZCZYŃSKI Eds., *Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-95)*, Number 928 in LNCS (1995), pp. 273–286. Springer.

CHOMICKI, J. AND SUBRAHMANIAN, V. 1990. Generalized Closed World Assumption is Π^0_2-Complete. *Information Processing Letters 34*, 289–291.

CULLUM, J. AND WILLOUGHBY, R. 1985. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Volume 1. Birkhäuser, Boston.

DAWAR, A. 1995. A Restricted Second Order Logic for Finite Structures. In D. LEIVANT Ed., *Proceedings International Workshop on Logic and Computational Complexity (LCC '94)*, LNCS 960 (Indianapolis, IN, 1995). Springer.

DIX, J. 1992. Classifying Semantics of Disjunctive Logic Programs. In K. APT Ed., *Proceedings JICSLP-92* (Washington DC, November 1992), pp. 798–812. MIT Press.

DIX, J., LOVELAND, D., MINKER, J., AND WARREN, D. Eds. 1996. *Workshop on Disjunctive Logic Programming and Databases: Nonmonotonic Aspects* (Schloß Dagstuhl, Germany, July 1996). Seminar Report 150.

DIX, J. AND MÜLLER, M. 1993. Implementing Semantics of Disjunctive Logic Programs Using Fringes and Abstract Properties. In L.-M. PEREIRA AND A. NERODE Eds., *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)* (Lisbon, Portugal, July 1993), pp. 43–59. MIT Press.

DOYLE, J. AND PATIL, R. 1991. Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the Utility of Representation Services. *Artificial Intelligence 48*, 261–297.

EITER, T. AND GOTTLOB, G. 1995a. Note on the Complexity of some Eigenvector Problems. Technical Report CD-TR 95/89 (December), Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria.

EITER, T. AND GOTTLOB, G. 1995b. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence 15*, 3/4, 289–323.

EITER, T., GOTTLOB, G., AND GUREVICH, Y. 1996. Normal Forms for Second-Order Logic over Finite Structures, and Classification of NP Optimization Problems. *Annals of Pure and Applied Logic 78*, 111–125.

EITER, T., GOTTLOB, G., AND MANNILA, H. 1994. Adding Disjunction to Datalog. In *Proceedings PODS '94* (May 1994), pp. 267–278.

Eiter, T., Leone, N., and Saccà, D. 1996. Expressive Power of Partial Models for Disjunctive Deductive Databases. In *Proceedings International Workshop on Logic in Databases (LID '96)*, Number 1154 in LNCS (1996), pp. 245–264. Springer.

Fagin, R. 1974. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp Ed., *Complexity of Computation* (1974), pp. 43–74. AMS.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium* (Cambridge, Mass., 1988), pp. 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–385.

Gottlob, G. 1995. Translating Default Logic into Standard Autoepistemic Logic. *Journal of the ACM 42*, 4, 711–740.

Gottlob, G., Leone, N., and Veith, H. 1995. Second-Order Logic and the Weak Exponential Hierarchies. In J. Wiedermann and P. Hajek Eds., *Proceedings of the 20th Conference on Mathematical Foundations of Computer Science (MFCS '95)*, LNCS 969 (Prague, 1995), pp. 66–81. Full paper CD-TR 95/80, Christian Doppler Lab for Expert Systems, Information Systems Department, TU Wien.

Gurevich, Y. 1988. Logic and the Challenge of Computer Science. In E. Börger Ed., *Trends in Theoretical Computer Science*, Chapter 1. Computer Science Press.

Imielinski, T. and Lipski, W. 1984. The Relational Model of Data and Cylindric Algebras. *Journal of Computer and System Sciences 28*, 80–102.

Immerman, N. 1986. Relational Queries Computable in Polynomial Time. *Information and Control 68*, 86–104.

Kanellakis, P. 1990. Elements of Relational Database Theory. In J. van Leeuwen Ed., *Handbook of Theoretical Computer Science*, Volume B, Chapter 17. Elsevier Science Publishers B.V. (North-Holland).

Kifer, M., Lausen, G., and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM 42*, 4, 740–843.

Kolaitis, P. 1991. The Expressive Power of Stratified Logic Programs. *Information and Computation 90*, 50–66.

Kolaitis, P. and Papadimitriou, C. H. 1991. Why Not Negation By Fixpoint ? *Journal of Computer and System Sciences 43*, 125–144.

Kolaitis, P. and Vardi, M. 1992. Fixpoint Logic vs. Infinitary Logic in Finite-Model Theory. In *Proceedings LICS-92* (1992), pp. 61–71. IEEE Computer Science Press.

Kolaitis, P. and Vardi, M. 1995. On the Expressive Power of Datalog: Tools and a Case Study. *Journal of Computer and System Sciences 51*, 1, 110. Abstract in Proc. PODS 90, pp. 61–71.

Krentel, M. 1988. The Complexity of Optimization Problems. *Journal of Computer and System Sciences 36*, 490–509.

Leone, N., Rullo, P., and Scarcello, F. 1995. Declarative and Fixpoint Characterizations of Disjunctive Stable Models. In *Proceedings of the International Logic Programming Symposium (ILPS '95)* (Portland, Oregon, 1995), pp. 399–413. MIT Press.

Lifschitz, V. and Schwarz, G. 1993. Extended Logic Programs as Autoepistemic Theories. In L.-M. Pereira and A. Nerode Eds., *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)* (Lisbon, Portugal, July 1993), pp. 101–114. MIT Press.

Lifschitz, V. and Turner, H. 1994. Splitting a Logic Program. In *Proceedings ICLP-94* (Santa Margherita Ligure, Italy, June 1994), pp. 23–38. MIT-Press.

Lloyd, J. 1984. *Foundations of Logic Programming*. Springer, Berlin.

Lobo, J., Minker, J., and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA.

Maier, D. and Vance, B. 1993. A Call to Order. In *Proceedings PODS-93* (1993), pp. 1–16.

McCarthy, J. 1986. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence 28*, 89–116.

Minker, J. 1982. On Indefinite Data Bases and the Closed World Assumption. In D. Loveland Ed., *Proceedings* 6^{th} *Conference on Automated Deduction (CADE '82)*, Number 138 in Lecture Notes in Computer Science (New York, 1982), pp. 292–308. Springer.

Minker, J. 1994. Overview of Disjunctive Logic Programming. *Annals of Mathematics and Artificial Intelligence 12*, 1–24.

Moore, R. 1985. Semantical Considerations on Nonmonotonic Logics. *Artificial Intelligence 25*, 75–94.

Nerode, A., Ng, R., and Subrahmanian, V. 1995. Computing Circumscriptive Databases. I: Theory and Algorithms. *Information and Computation 116*, 58–80.

Papadimitriou, C. 1984. The Complexity of Unique Solutions. *Journal of the ACM 31*, 492–500.

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

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

Pereira, L.-M. and Nerode, A. Eds. 1993. *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)* (Lisbon, Portugal, July 1993). MIT Press.

Przymusinski, T. 1988. On the Declarative and Procedural Semantics of Stratified Deductive Databases. In J. Minker Ed., *Foundations of Deductive Databases and Logic Programming*, pp. 193–216. Washington DC: Morgan Kaufman.

Przymusinski, T. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing 9*, 401–424.

Przymusinski, T. 1994. A knowledge representation framework based on autoepistemic logic of minimal beliefs. In *Proc. Twelfth National Conference on Artificial Intelligence (AAAI '94)* (1994), pp. 952–959.

Przymusinski, T. 1995. Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence 14*, 323–357.

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

Ross, K. 1994. Modular Stratification and Magic Sets for Datalog Programs with Negation. *Journal of the ACM 41*, 6, 1216–1267.

Saccá, D. 1995. Deterministic and Nondeterministic Stable Model Semantics for Unbound DATALOG Queries. In *Proceedings 5th Intl. Conference on Database Theory (ICDT-95)*, LNCS 893 (January 1995), pp. 353–367.

Saccà, D. and Zaniolo, C. 1990. Stable Models and Non-Determinism in Logic Programs with Negation. In *Proceedings PODS-90* (1990), pp. 205–218.

Sakama, C. and Inoue, K. 1993. Relating Disjunctive Logic Programs to Default Theories. In L.-M. Pereira and A. Nerode Eds., *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR-93)* (Lisbon, Portugal, July 1993), pp. 266–282. MIT Press.

Sakama, C. and Inoue, K. 1994. An Alternative Approach to the Semantics of Disjunctive Logic Programs and Deductive Databases. *Journal of Automated Reasoning 13*, 145–172.

S.Brass, Dix, J., and Przymusinski, T. 1996a. A comparison of the static and disjunctive well-founded semantics. (In preparation.), University of California at Riverside.

S.Brass, Dix, J., and Przymusinski, T. 1996b. Super logic programs. In *Proceedings Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR '96)* (Vancouver, Canada, 1996), pp. 529–540.

Schlipf, J. 1987. Decidability and Definability with Circumscription. *Annals of Pure and Applied Logic 35*, 173–191.

Schlipf, J. 1995a. Complexity and Undecidability Results in Logic Programming. *Annals of Mathematics and Artificial Intelligence 15*, 3/4, 257–288.

Schlipf, J. 1995b. The Expressive Powers of Logic Programming Semantics. *Journal of Computer and System Sciences 51*, 1, 64–86. Abstract in Proc. PODS 90, pp. 196–204.

SEIPEL, D. 1994. Non-Monotonic Reasoning Based on Minimal Models and its Efficient Implementation. In B. WOLFINGER Ed., *Innovationen bei Rechen- und Kommunikationssystemen (13th IFIP World Computer Congress, proceedings workshop FG2: Disjunctive Logic Programming and Disjunctive Databases)* (Berlin, 1994), pp. 53–60. Springer.

STEWART, I. 1991. Complete Problems Involving Boolean Labelled Structures and Projection Transactions. *Journal of Logic and Computation 1*, 6, 861–882.

STOCKMEYER, L. J. 1977. The Polynomial-Time Hierarchy. *Theoretical Computer Science 3*, 1–22.

TURNER, H. 1996. Representing Actions in Default Logic: A Situation Calculus Approach. In *Proceedings Common Sense '96* (1996).

ULLMAN, J. D. 1989. *Principles of Database and Knowledge Base Systems*. Computer Science Press.

VAN GELDER, A., ROSS, K., AND SCHLIPF, J. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM 38*, 3, 620–650.

VARDI, M. 1982. Complexity of relational query languages. In *Proceedings 14th STOC* (San Francisco, 1982), pp. 137–146.

VEITH, H. 1994. Logical Reducibilities in Finite Model Theory. Master's thesis, Information Systems Department, TU Vienna, Austria.

VEITH, H. 1996. Succinct Representation and Leaf Languages. In *11th Annual IEEE Conference on Computational Complexity* (Philadelphia, PA, May 1996), pp. 118–126. Full paper in *Electronic Colloquium on Computational Complexity, Report TR95-048*.

WOLFINGER, B. Ed. 1994. Workshop: *Disjunctive Logic Programming and Disjunctive Databases* (Berlin, August 1994). German Society for Computer Science (GI): Springer. 13^{th} IFIP World Computer Congress, Hamburg, Germany.

YAHYA, A. AND HENSCHEN, L. 1985. Deduction in Non-Horn Databases. *Journal of Automated Reasoning 1*, 2, 141–160.

YOU, J.-H. AND YUAN, L. 1993. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning 10*, 143–160.

YOU, J.-H. AND YUAN, L. 1995. On the Extension of Logic Programming with Negation through Uniform Proofs . In W. MAREK, A. NERODE, AND M. TRUSZCZYŃSKI Eds., *Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-95)*, Number 928 in LNCS (1995), pp. 231–244. Springer.