Disjunctive Logic Programming: A Survey and Assessment

In Honor of Robert A. Kowalski

Jack Minker[1] and Dietmar Seipel[2]

[1] Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742, USA

[2] Department of Computer Science
University of Würzburg
Am Hubland, D – 97074 Würzburg, Germany

Abstract

We describe the fields of disjunctive logic programming and disjunctive deductive databases from the time of their inception to the current time. Contributions with respect to semantics, implementations and applications are surveyed.

In the last decade many semantics have been proposed out of which we highlight what we believe to be the most influential ones and compare them. Basic ideas have been borrowed from the semantics of normal logic programs such as stable model semantics and well–founded semantics, which have been generalized in various ways to obtain semantics of disjunctive logic programs.

We discuss disjunctive systems such as DLV and Smodels, and related non–disjunctive systems such as XSB and DeReS, that have been implemented. We also describe applications of disjunctive logic programming: reasoning about declarative specifications, reasoning about actions, diagnosis (e.g. in medicine or biology), and in data integration that have resource predicates defined by multiple rules. We discuss the future needs to make the field practical: e.g. integrating concepts from databases (such as aggregation), optimization methods, and object orientation.

In Section 7 we discuss the influence that Bob Kowalski had on our work.

1 Introduction

The field of disjunctive logic programming (DLP) had its beginnings in 1982 and is nearing the completion of its second decade. Work prior to 1982 focused primarily on Horn theories of logic programming as described by Kowalski [Kow74]. The first result in the field of disjunctive logic programming (DLP) was by

Minker [Min82] who developed a consistent theory for default negation in disjunctive theories, the generalized closed world assumption (GCWA). The paper also set forth the concept of minimal models for computing answers both for positive and negative atoms. The GCWA reduces to the CWA for Horn theories developed by Reiter [Rei78].

There was no work in DLP until 1986 when renewed interest arose as a consequence of a *Workshop on Foundations of Deductive Databases and Logic Programming*, organized by Minker [Min88b]. Following the workshop, Minker and Rajasekar [MR90], developed fixpoint, model theoretic and proof theoretic semantics for DLP. This work extended the results of van Emden and Kowalski [vEK76], for the Horn theory of logic programming. It led to many theoretical developments in disjunctive and normal disjunctive logic programs, including theories of negation and disjunctive deductive databases. Much of this work in the late 1980s and early 1990s was performed at the University of Maryland and is the subject of a research monograph, *Foundations of Disjunctive Logic Programming* [LMR92]. The theoretical results contained in that monograph extend what is known in the theory of logic programs as developed in the monograph by Lloyd [Llo87] and the paper by Apt [Apt90]. The work in DLP has also been shown to be important for representing and implementing nonmonotonic and abductive reasoning. In this paper, we describe the developments that have taken place in disjunctive logic programming since 1982.

The paper is organized as follows. In Section 2, we provide the basic definitions and background needed for the paper. In Section 3, we briefly discuss developments in logic programming that led to the developments in disjunctive logic programming. In Section 4, we discuss the theoretical developments that have taken place in DLP and in the related area of disjunctive deductive databases (DDDBs). One of the major accomplishments in the field has been the implementation of systems to handle normal logic programs, DLPs and DDDBs. This work is discussed in Section 5. We discuss several areas for applications in DLP in Section 6. Finally, in Section 7 we provide a summary and an assessment of the field of DLP. We end the paper with a Tribute to Robert Kowalski whose pioneering research was significant for our research.

2 Background

We assume that the reader has a background in logic programming as described in [Kow78,SS86], or other books on logic programming, and is familiar with the major theoretical results given in [Apt90,Llo87]. A personal perspective on the development of disjunctive logic programming is given in [Min89]. Kowalski [Kow88] provides a personal perspective of the history of the field of logic programming.

Throughout the paper we will refer to different classes of clauses. These are *definite Horn clauses, normal clauses, extended clauses, positive–disjunctive clauses, disjunctive clauses*, and *extended–disjunctive clauses*. Associated with each type of clause is a class of logic programs. These are termed, respectively, *definite logic programs, normal logic programs, extended logic programs, positive–disjunctive logic programs, disjunctive logic programs*, and *extended–disjunctive logic programs*. These logic programs are defined as follows.

Given a first order language $\mathcal{L}$, a *disjunctive logic program* $\mathcal{P}$ consists of logical inference rules of the form

$$r = A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n, \tag{1}$$

where A_i, B_i and C_i are atoms in the language $\mathcal{L}$; $k, m, n \in I\!N_0$, and *not* denotes negation–by–default, rather than logical negation. A rule r of the form (1) is denoted for short as:

$$r = \alpha \leftarrow \beta \wedge not\cdot \gamma, \tag{2}$$

where $\alpha = A_1 \vee \ldots \vee A_k$, $\beta = B_1 \wedge \ldots \wedge B_m$, $\gamma = C_1 \vee \ldots \vee C_n$.[1] Sometimes α, β, and γ will be considered as sets (e.g. in Table 3 of Section 4). α is called the *head*, β is called the *positive body*, and $not\cdot \gamma$ is called the *negative body* of r.

- A rule is called a *fact* if $m = n = 0$.
- A rule is called a *definite clause* or a *Horn clause*, if $n = 0$ and $k = 1$, i.e. it neither contains disjunction nor default negation.
- A rule is called *positive–disjunctive*, if $n = 0$, i.e. it does not contain default negation.
- A rule is called *normal*, if $k = 1$, i.e. it does not contain disjunction.
- A rule is called a *denial rule*, if $k = 0$. Denial rules can be used for representing *integrity constraints* and *queries* to disjunctive logic programs.
- A rule is called a *range–restricted* if all variable symbols occurring in the head also occur in the positive body.

A rule of the form (1) is called *extended–disjunctive rule*, if A_i, B_i and C_i are literals in the language, i.e. atoms or atoms preceded by classical negation $\neg$. If a (disjunctive) logic program does not contain any function symbols, then it is called a *(disjunctive) deductive database*.

The Herbrand base $H\!B_{\mathcal{P}}$ of a disjunctive logic program $\mathcal{P}$ contains all ground atoms over the language of $\mathcal{P}$. The set of all *ground instances* of the rules and facts in $\mathcal{P}$ is denoted by $gnd\,(\mathcal{P})$.

Partial Herbrand Interpretations

A *partial (three–valued) Herbrand interpretation* of $\mathcal{P}$ is given by a mapping I: $H\!B_{\mathcal{P}} \to \{\, \mathsf{t}, \mathsf{f}, \mathsf{u} \,\}$ that assigns a truth value "t" (*true*), "f" (*false*) or "u" (*undefined*) to each ground atom in $H\!B_{\mathcal{P}}$. Obviously, I can be represented by a pair $\langle I_{\mathsf{t}}, I_{\mathsf{f}} \rangle$, such that

$$I_{\mathsf{t}} = \{\, A \in H\!B_{\mathcal{P}} \mid I(A) = \mathsf{t} \,\},$$
$$I_{\mathsf{f}} = \{\, A \in H\!B_{\mathcal{P}} \mid I(A) = \mathsf{f} \,\}.$$

[1] Note that γ is a disjunction, and, according to De Morgan's law, $not\cdot \gamma$ is taken to be a conjunction.

I is called a *total Herbrand interpretation*, if $I_{\mathsf{t}} \cup I_{\mathsf{f}} = H_{B_{\mathcal{P}}}$, i.e. if all atoms $A \in H_{B_{\mathcal{P}}}$ are mapped to one of the classical truth values t or f. In this case, it is sufficient to represent I by its set I_{t} of *true* atoms alone.

There are two important *partial orderings on truth values*: in the truth ordering $\leq_{\mathsf{k}}$ it holds $\mathsf{f} \leq_{\mathsf{t}} \mathsf{u}$, $\mathsf{u} \leq_{\mathsf{t}} \mathsf{t}$, and in the knowledge ordering $\leq_{\mathsf{k}}$ it holds $\mathsf{u} \leq_{\mathsf{k}} \mathsf{f}$, $\mathsf{u} \leq_{\mathsf{k}} \mathsf{t}$, cf. Fitting [Fi91] and Figure 1. These partial orderings have been

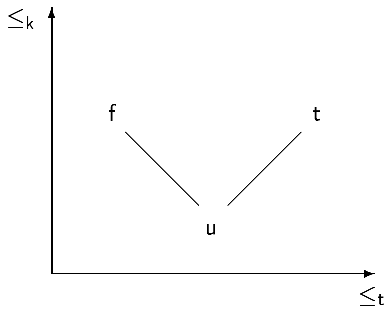

Fig. 1. Truth Ordering and Knowledge Ordering

generalized (pointwise) to partial orderings on partial Herbrand interpretations as follows. For $\mathsf{x} \in \{\, \mathsf{t}, \mathsf{k} \,\}$: $I_1 \leq_{\mathsf{x}} I_2$, iff $(\, \forall A \in H_{B_{\mathcal{P}}} : I_1(A) \leq_{\mathsf{x}} I_2(A)\,)$.

The *Boolean operations* "$\vee$", "$\wedge$" and "$\neg$" on truth values are defined based on the truth ordering, cf. Figure 2. The truth value of a disjunction $v_1 \vee v_2$ and a conjunction $v_1 \wedge v_2$ of truth values are constructed by taking the *maximum* and the *minimum* of v_1 and v_2, respectively. "$\vee$" and "$\wedge$" both are commutative and associative, and thus can be generalized to disjunctions and conjunctions, respectively, of more than one truth value. Let M be a partial Herbrand interpretation and let $A_i \in H_{B_{\mathcal{P}}}$ be ground atoms. For a connective $\otimes \in \{\, \vee, \wedge \,\}$ we define $M(\, A_1 \otimes \ldots \otimes A_k\,) = M(A_1) \otimes \ldots \otimes M(A_k)$. For $k = 0$, the empty disjunction (i.e. $\otimes = \vee$) evaluates to f, whereas the empty conjunction (i.e. $\otimes = \wedge$) evaluates to t.

$\wedge$	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

$\vee$	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

$\neg$	
t	f
f	t
u	u

Fig. 2. Boolean operations in three–valued logic

Models and Partial Models, Minimality

A total Herbrand interpretation M satisfies a ground rule $r = \alpha \leftarrow \beta \wedge not \cdot \gamma$ if $M(\beta) \wedge \neg M(\gamma) = \mathtt{t}$ implies $M(\alpha) = \mathtt{t}$. This can be generalized to partial Herbrand interpretations M by requiring that

$$M(\beta) \wedge \neg M(\gamma) \leq_t M(\alpha). \tag{3}$$

A total (partial) Herbrand interpretation M is called a *total (partial) model* of $\mathcal{P}$ if M satisfies of all ground instances $r \in gnd(\mathcal{P})$ of all rules of $\mathcal{P}$. M is called a *partial minimal model* of $\mathcal{P}$ if M is a partial model of $\mathcal{P}$ and there is no other partial model I of $\mathcal{P}$ such that $I \leq_{\mathtt{t}} M$ (truth ordering). A partial minimal model M of $\mathcal{P}$ that is total is called a *minimal model* of $\mathcal{P}$. The set of minimal models of $\mathcal{P}$ is denoted by $\mathcal{MM}_2(\mathcal{P})$.

Semantics of Disjunctive Logic Programs

Each alternative type of logic program leads to a different theory. We are concerned with the semantics of each type of logic program and how one computes in the theory. The computation of negation is particularly important for nonmonotonic reasoning. There are many relationships between logic programming and alternative formulations of nonmonotonic reasoning by circumscription [McCa80], default reasoning [Rei80], and autoepistemic reasoning [MT91,MT93,Moo84], [Moo85]. We do not cover this topic here. For details on work in nonmonotonic reasoning, see [Gin87]. For a discussion of the relationships between nonmonotonic theories and logic programming, see [Min93]. We discuss this topic briefly. See Reiter [Rei78] for an early survey of work in nonmonotonic reasoning.

In the case of logic program theories that are definite, there exists one semantics that is generally accepted as the meaning of the program. This is the unique minimal Herbrand model of the definite theory. When one comes to the alternative types of logic programs, different interpretations may be given to the programs. That is, each theory may or may not provide a unique minimal model; however, even when there is a unique meaning ascribed to the program, the meaning may be different between the alternative approaches. In the rest of this section we define some of the most prominent approaches in detail: stable and partial stable models for disjunctive logic programs, the well–founded model for normal logic programs, and answer sets for extended–disjunctive logic programs. In the following sections we provide a broader overview of the theoretical results that are known in each class of logic programs and we give some examples. We discuss the possible alternative semantics within each class.

Stable and Partial Stable Models. The *Gelfond–Lifschitz transformation* (GL–transformation, [GL88,Prz91]) $\mathcal{P}^M$ of a disjunctive logic program $\mathcal{P}$ w.r.t. a partial Herbrand interpretation M is obtained from the ground instance $gnd(\mathcal{P})$ of $\mathcal{P}$ by replacing in every ground rule $r = \alpha \leftarrow \beta \wedge not \cdot \gamma \in gnd(\mathcal{P})$ the negative body by its truth value w.r.t. M.[2] We define $r^M = \alpha \leftarrow \beta \wedge \neg M(\gamma)$ and $\mathcal{P}^M = \{\, r^M \mid r \in \mathit{gnd}\,(\mathcal{P})\,\}$. The GL–transformation $\mathcal{P}^M$ is a ground positive–disjunctive logic program that has as additional atoms the truth values t, f and u.[3] Based on $\mathcal{P}^M$ the concepts of stable models (cf. Gelfond and Lifschitz, [GL88]) and partial stable models (cf. Pryzmusinski, [Prz91]) have been defined:

1. A total Herbrand interpretation M is called a *stable model* of $\mathcal{P}$ if M is a minimal model of $\mathcal{P}^M$.
2. A partial Herbrand interpretation M is called a *partial stable model* of $\mathcal{P}$ if M is a partial minimal model of $\mathcal{P}^M$.

It can be shown that stable models are always also partial stable models. That is, the semantics of stable models is always stronger than the semantics of partial stable models. An example will be given in Section 3.

Well–Founded Semantics. For normal logic programs there always exists a (unique) least partial stable model M in the knowledge ordering ([Fi91]). This partial model M coincides with the *well–founded model*, which had been defined by Van Gelder, Ross and Schlipf, and which can be characterized using the following *alternating fixpoint approach* ([VaGe89]):

$$M_0 = \emptyset,\ M_{n+1} = \mathcal{MM}_2(\mathcal{P}^{M_n}).$$

This recursion generates an increasing sequence $(M_{2n})_{n \in \mathbb{N}_0}$ and a decreasing sequence $(M_{2n+1})_{n \in \mathbb{N}_0}$ of sets of atoms, such that $M_{2n} \subseteq M_{2n+1}$, for all $n \in \mathbb{N}_0$. For $M_\cup = \bigcup_{n=0}^{\infty} M_{2n}$ and $M_\cap = \bigcap_{n=0}^{\infty} M_{2n+1}$ it holds that $M_\cup \subseteq M_\cap$. The well–founded model M is a partial Herbrand interpretation given by $M = \langle\, M_\cup, \mathit{HB}_\mathcal{P} \setminus M_\cap \,\rangle$, i.e. an atom A is *true* under the well–founded model semantics if it is in $M_\cup$ and *false* if it is not in $M_\cap$. An example will be given in Section 3.

Answer Set Semantics. Gelfond and Lifschitz ([GL90,GL91]) have extended the concept of stable models to extended–disjunctive logic programs $\mathcal{P}$, i.e. logic programs that may contain *classical negation*. They propose the so–called *answer set semantics*, which can be defined as follows: $\mathcal{P}$ is transformed to a disjunctive logic program $\mathcal{P}'$ free of classical negation, which is obtained by replacing negative literals $\neg A = \neg p(t_1, \ldots, t_n)$ by positive literals $A' = p'(t_1, \ldots, t_n)$ over new predicate symbols. Every stable model M' of $\mathcal{P}'$ defines an answer set M of $\mathcal{P}$, which is a set of literals: Let

$$L = \{\, A \in \mathit{HB}_\mathcal{P} \mid M'(A) = \mathsf{t} \,\} \cup \{\, \neg A \in \neg \mathit{HB}_\mathcal{P} \mid M'(A') = \mathsf{t} \,\};$$

if L does not contain any complementary pair A and $\neg A$ of literals, then $M = L$, otherwise $M = \mathit{HB}_\mathcal{P} \cup \neg \mathit{HB}_\mathcal{P}$ is the set of all ground literals.

[2] If this truth value is "t", then "t" can be deleted from the body. If it is "f", then the whole rule can be deleted from $\mathcal{P}^M$.

[3] Note that these truth values must evaluate to themselves under all partial Herbrand interpretations I of $\mathcal{P}^M$.

3 Theory of Logic Programming

Since we treat, primarily, disjunctive logic programs, we sketch, briefly, only the results in logic programming, except for the case of normal and extended logic programs. Logic programming began in approximately 1971 [Kow88]. Theoretical results were obtained first in 1976, with the publication of the landmark paper by van Emden and Kowalski [vEK76] in which they detailed model theoretic, fixpoint, and operational semantics of logic programs as a programming language. They introduced an operator, $\mathcal{T}_{\mathcal{P}}$, and demonstrated that the operator has a fixpoint. The semantics of the fixpoint corresponds to model theory, while operational semantics corresponds to proof theory. The semantics in all three cases are identical. They provided a formal semantics of a definite clause logic formula, viewed as a statement in a programming language. Their use of the least model and least fixpoint constructions, as well as the procedural interpretation, provide the foundations of the field of logic programming. Their ideas also led to the concept of negation. If one subtracts the minimal model of a program from the Herbrand base, the atoms that remain can be considered *false*. This provides a model characterization of the *CWA*, first propounded by Reiter [Rei78] in proof theoretic terms.

Apt and van Emden [AvE82] built upon the theoretical treatment of van Emden and Kowalski. They renamed Hill's inference system [Hil74], LUSH resolution (Linear Resolution with Unrestricted Selection function based on Horn clauses), to be SLD (*SL* resolution for Definite Horn clauses). They also characterized the finite failure set of an atom relative to a program in terms of the van Emden/Kowalski fixpoint operator, $\mathcal{T}_{\mathcal{P}}$ [vEK76]. Lassez and Maher[LM84] then proved that the finite failure set is characterized by the difference between the Herbrand base and the fixpoint operator described by Apt and van Emden. Clark [Cla78] ties these results to the completion of a logic program. He shows the soundness of the negation–as–failure rule for any Horn logic program, $\mathcal{P}$, augmented by $comp(\mathcal{P})$ and equality axioms. He shows that every goal G with a finitely failed SLD–tree is a logical consequence of $comp(\mathcal{P})$ and equality axioms (actually his results extend to general programs using safe computation rules, i.e. rules that select only ground negative literals). Jaffar, Lassez and Lloyd [JLL83] prove the completeness result and show that if a goal G is a logical consequence of $comp(\mathcal{P})$, then there is a finitely failed SLD–tree for G. See Shepherdson [She88], for work on negation, and Lloyd [Llo87] and Apt [Apt90] for the theoretical results in logic programming. Apt and Bol [AB94] and Dix [Dix95d] update the Shepherdson survey on negation in logic programming [She88] and briefly describe theories of negation in disjunctive logic programs.

In 1988, Apt, Blair and Walker [ABW88] extended the class of logic programs to a subset of normal logic programs, termed *stratified logic programs*. Such programs are normal logic programs for which the rules do not have recursion through negation. When this occurs, the predicates can be placed into strata so that one can compute over the strata. Independently, and at the same time, Van Gelder [VaGe88], also described this class of programs. The theoretical results obtained were that there was a fixpoint theory that characterized the semantics of this class of programs, and that there was a unique model that could be obtained by iterating over the strata. Przymusinski [Prz88], showed that the model obtained was the so–called *perfect model*. He also extended the work to include *locally stratified logic programs*.

The semantics of definite and stratified logic programs lead to unique minimal models, which are generally accepted to be the semantics of the two classes of logic programs. However, this is not the case when we come to the class of *normal logic programs*; then there are several possible ways to determine the semantics. Each alternative leads to a different semantics. Van Gelder, Ross and Schlipf [VGRS91], Gelfond and Lifschitz [GL88], and Baral, Lobo and Minker [BLM89,BLM91]* ([Min99])[4], have developed alternative semantics.

Well–Founded and Stable Model Semantics

Van Gelder, Ross and Schlipf [VGRS91], were the first to extend the work by Apt, Blair and Walker [ABW88], to the class of *normal logic programs*. Unlike stratified logic programs, the program may not be stratifiable. This occurs when there is recursion through negation. The *well–founded semantics* (WFS) of Van Gelder et al. is given in terms of three possible truth values: *true, false*, and *unknown*. For the normal logic program

$$\mathcal{P} = \{\, a \leftarrow not\, b,\ b \leftarrow not\, a,\ c \leftarrow a,\ c \leftarrow b \,\}$$

Van Gelder et al. obtain the semantics $I = \langle\, \emptyset, \emptyset \,\rangle$. The semantics may be understood as follows: since, from the first two clauses it cannot be decided if either a, or b, are either *true* or *false*, they are determined to be *unknown*. Since a and b are both *unknown*, c must be assigned to be *unknown*.

Chen and Warren [CW93] develop a variant of *SLS*–resolution [Prz88] to obtain a procedural semantics for the WFS. This procedure, called *XOLDTNF*, is sound and complete. Warren and his group at Stony Brook have developed an implementation that handles most cases of the WFS, termed XSB [War99]. The system can handle large sets of data and rules. XSB is a sophisticated extension to PROLOG, containing many of its features. We discuss this and other implementations in logic programming and disjunctive logic programming in Section 5. Zukowski et al. ([ZBF97,ZF99,BDFZ01]) present a transformation–based approach for the bottom–up computation of the well–founded model.

Alternatively, the *stable model semantics* [GL88], has been developed by Gelfond and Lifschitz, as discussed in Section 2. The meaning of the program is taken as the set of positive clauses that are *true* in all minimal models obtained by the GL–transformation. For the program given above, there are two stable models: $\{a, c\}$, and $\{b, c\}$. To be *true* in the stable model semantics, an atom must be *true* in every stable model; to be *false*, it must be *false* in every stable model; to be *unknown*, it must neither be *true* nor *false*. The stable model semantics for the

[4] To conserve space, we sometimes write references as, [—]* ([Min99]), to denote that the reference in [—]* can be found in ([Min99]).

above program $\mathcal{P}$ is thus given by the following partial Herbrand interpretation $I' = \langle \{c\}, \emptyset \rangle$, i.e. a and b are *unknown* and c is *true*. An iterative method for finding the stable models of a program has been developed by Fernández, Lobo, Minker and Subrahmanian [FLMS93]. Their approach transforms a normal logic program into a disjunctive logic program with integrity constraints, called the *evidential transformation*. The minimal models of this evidential transformation are the same as the stable models of the original disjunctive logic program. Bell, Nerode, Ng and Subrahmanian [BNNS94] have developed a linear programming based implementation of the stable model semantics. The work of Gelfond and Lifschitz can be extended to include disjunctions of positive atoms in the head of a clause as shown by Przymusinski [Prz91]. The approach by [FLMS93] to find the stable models of normal logic programs extends to the case where there are disjunctions in the head of a clause. Fernández and Lobo [FL93]* ([Min99]) have developed a proof procedure for computing answers to queries under the stable model semantics. Niemela and Simons [NS97] have developed an efficient system, Smodels, that computes the stable model semantics, cf. Section 5.

Other semantics developed at approximately that time, [BLM89,BLM91]* ([Min99]) are not discussed here since it is clear that the two dominant theories for normal logic programs are the WFS and the stable model semantics.

Although in the above example, the *stable* and the *GWFS* semantics are the same, this is not always the case. Various criteria have been specified [Dix92a] to determine which semantics should be chosen. These criteria relate to properties that the semantics satisfy. Based on these properties one may argue the pros and cons of a particular semantics. In addition, motivated by finding semantics with the appropriate properties, several semantics have been developed. A summary of these semantics is given in [Min94]. Although of considerable interest, the well–founded semantics and the stable model semantics remain the dominant semantics in use. Perhaps one of the reasons for this is that efficient systems have been developed for them, namely XSB and Smodels.

Gelfond and Lifschitz [GL90,GL91] introduce classical negation in logic programs, i.e. they represent extended–disjunctive clauses. The semantics proposed for such a logic program, which is called *answer set semantics*, is based on the stable model semantics. They show that an extended–disjunctive program can be viewed as a default theory in which every justification and conclusion is a literal, and every precondition is a conjunction of literals. In [GL91], they show that some facts of commonsense knowledge can be represented by logic programs and disjunctive databases more easily when classical negation is available. Extended–disjunctive programs are shown to be identical to a special case of default theories in the sense of Reiter [Rei80].

4 Theory of Disjunctive Logic Programming

Work in disjunctive theories was pursued seriously after a workshop organized by Minker in 1986 [Min86]* ([Min99]). The field of *disjunctive logic programming (DLP)* started approximately in 1982 with the the paper by Minker [Min82], who devised a consistent theory of negation for *disjunctive deductive databases (DDDBs)*. Shepherdson [She88] showed that Minker's theory of negation for DDDBs also applied to DLPs. For a historical perspective of DLPs and DDDBs, see [Min88a]. There is a major difference for *deductive databases (DDBs)* and those for DDDBs. Whereas DDBs have a unique minimal model that describes the meaning of the database, DDDBs generally have multiple minimal models.

As shown in [Min82] it is sufficient to answer positive queries over *DDDBs* by showing that the query is satisfied in every minimal model. Thus, for $\mathcal{P} = \{ a \vee b \}$, there are two minimal models, $\{a\}$ and $\{b\}$. The query $\leftarrow a$ is not satisfied in the model $\{b\}$, and hence, a cannot be *true*. However, the query $\leftarrow a \vee b$ is satisfied in both minimal models and hence the answer to the query $\leftarrow a \vee b$ is *yes*. To answer negated queries, it is not sufficient to use Reiter's *CWA* [Rei78] since, as he noted, from the theory $\mathcal{P} = \{ a \vee b \}$, it is not possible to prove a, and it is not possible to prove b. Hence, by the *CWA*, *not a* and *not b* follow. But, $\{ a \vee b, \mathit{not}\ a, \mathit{not}\ b \}$ is not consistent. The *Generalized Closed World Assumption (GCWA)*, [Min82] resolves this problem by specifying that a negated atom be considered *true* if the atom does not appear in any minimal model of the database. This provides a model theoretic definition of negation. An equivalent proof theoretic definition, also in [Min82], is that a ground atom A may be considered to be *false* if, whenever $A \vee \alpha$ may be proven from the database, then α may be proven from the database, where α is an arbitrary ground disjunction of atoms.

For related work on negation in disjunctive theories see [YH85,GPP86,Cha93], [Sak89,RT88,RLM89]. For surveys on negation see [She88,AB94], [Dix95d,Min93].

Fixpoint Approaches

In LPs, it is natural for the fixpoint operator to map atoms to atoms. However, for DLPs, it is natural to map positive disjunctions to positive disjunctions. A set of positive disjunctions is referred to as a *state*. A *model state* is a state all of whose minimal models satisfy the DLP. The concept of a state was defined by Minker and Rajasekar [MR90] as the domain of a fixpoint operator $\mathcal{T}^s_{\mathcal{P}}$ whose least fixpoint characterizes the semantics of a disjunctive logic program $\mathcal{P}$. The operator is shown to be monotonic and continuous, and hence converges in ω iterations. The fixpoint computation operates bottom–up and yields a minimal model state logically equivalent to the set of minimal models of the program. The Minker/Rajasekar fixpoint operator $\mathcal{T}^s_{\mathcal{P}}$ is an extension of the van Emden/Kowalski fixpoint operator $\mathcal{T}_{\mathcal{P}}$. If one considers all model states of a DLP and intersects them, the resultant is a model state, and among all model states it is minimal. Hence, one obtains a unique minimal model in a Horn database, while one obtains a unique model state in a DLP.

In [FM91b,FM95,SMR97] another generalization of the fixpoint operator $\mathcal{T}_{\mathcal{P}}$ has been investigated: the operator $\mathcal{T}^{INT}_{\mathcal{P}}$ works in sets of Herbrand interpretations rather than states. It generates the set of minimal Herbrand models as its least fixpoint. In [SMR97] a useful relationship has been established between state generation by $\mathcal{T}^s_{\mathcal{P}}$ and model generation by $\mathcal{T}^{INT}_{\mathcal{P}}$: for a set $\mathcal{I}$ of Herbrand interpretations, the set $\mathcal{T}^{INT}_{\mathcal{P}}(\mathcal{I})$ of interpretations is a subset of the set of models of the state $S \cup \mathcal{T}_{\mathcal{P}}^{s}(S)$, if $\mathcal{I}$ is a subset of the set of Herbrand models of the disjunctive Herbrand state S. Furthermore, it is shown that $\mathcal{T}_{\mathcal{P}}^{INT}$ is monotonic but not continuous, but that is still converges towards its least fixpoint in at most ω iterations.

Decker [Dec91] develops an alternative fixpoint operator for *DDDBs* which reduces to the Minker/Rajasekar fixpoint operator [MR90]. At each iteration of his operator, he finds partial models of the database. In the limit, he obtains the set of minimal models of the database. If one takes an atom from each minimal model and forms a disjunction, the resulting set of all such disjunctions is equivalent to the minimal model state of the *DDDBs*.

Query Answering

Answering queries in *DDDBs* has been studied by a number of individuals. Grant and Minker [GM86] were among the first to address the problem of computing answers to queries in *DDDBs*. They investigated the case where the database consists exclusively of ground positive disjuncts. Yahya and Henschen [YH85] developed a deductive method to determine whether or not a conjunction of ground atoms can be assumed *false* in a *DDDB* under the *Extended Generalized Closed World Assumption (EGCWA)*. The *EGCWA* is an extension of the *GCWA*. Bossu and Siegel [Boss85]* ([Min99]) developed a deductive method to answer a query by subimplication (a generalization of the *GCWA* that handles databases that have no minimal models). Henschen and Park [Hens86]* ([Min99]) answer *yes/no* questions in a database that consists of an *EDB*, an *IDB* and *ICs* that are all function–free. In addition, they allow negated unit clauses to be part of the database. The axioms in the *IDB* may be recursive. Yahya [Yah97] discusses how to answer queries defined as sets of clauses in implication form in a *DDDB*. Liu and Sunderraman [LS90b] generalize the relational model to represent disjunctive data. They develop a data–structure, called *M–table* to represent the data. Their generalized relational algebra operates on *M-tables*, however, it is sound, but not complete. Yuan and Chiang [YC89] developed a generalized relational algebra that is a sound and complete query evaluation algorithm for *DDDBs* that do not contain recursive *IDB* rules.

Fernández and Minker [FM91b] developed the concept of a *model tree*. They incrementally compute sound and complete answers to queries in *hierarchical DDDBs*. An example of a *model tree* is shown in Figure 3. The model tree represents the two minimal models $\{ a_1, a_2, b_1 \}$ and $\{ a_1, b_2 \}$ of the positive–disjunctive database $\{ a_1, a_2 \vee b_2, b_1 \vee b_2 \}$. A *DDDBs* is hierarchical if it contains no recursion. In [FLMS93] they develop a fixpoint operator over trees to capture the meaning of a *DDDB* that includes recursion. The tree representation of the fixpoint is equivalent to the Minker/Rajasekar fixpoint [MR90]. They compute the model tree of the extensional *DDDB* once for all queries. To answer queries *intensional database* rules may be invoked. Their approach to compute answers generalizes both to stratified and normal *DDDBs*. In [Sei94] a tree data structure for Herbrand states (rather than models) is given, which is called *clause tree*, and it is shown how the fixpoint operator $\mathcal{T}_{\mathcal{P}}^{s}$ can work on clause trees.

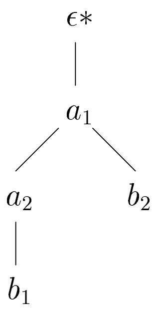

Fig. 3. Model Tree

The above approaches to answering queries in *DDDBs* have the following limitations. [GM86] can only compute answers to queries that contain a disjunctive extensional database. [YH85] can only answer *yes/no* questions. [LS90b] provide sound, but not complete answers to queries. [YC89] essentially compute the fixpoint of the entire *DDB* to answer each query. [FM91b] compute the model tree of the extensional *DDDB* once. To answer queries *IDB* rules may be invoked. However, the models of the extensional disjunctive part of the database do not have to be generated for each query.

Loveland et al. [Llo87,Lov87,SL88,RL90,RLS91] developed a top–down case–based reasoner that uses PROLOG when the database is *near Horn*. Loveland et al. [LRW93] introduced a relevancy detection algorithm to be used with SATCHMO, developed by Manthey and Bry [MB88], for automated theorem proving. Their system, SATCHMORE (SATCHMO with RElevancy), improves SATCHMO by limiting the uncontrolled use of forward chaining. Another approach is by Stickel [Sti88] using a PROLOG *Technology Theorem Prover (PTTP)*.

For disjunctive logic programs a huge amount of work has been done on bottom–up fixpoint evaluations using state generation or model generation. But so far very few top–down approaches exist for disjunctive logic programs. In deductive databases efficient query evaluation is achieved by mixing bottom–up and top–down techniques: The *magic sets technique* incorporates the top–down binding passing known from SLD–resolution into a bottom–up, breadth–first fixpoint computation to achieve a goal–oriented query evaluation, which focuses on relevant derivations as described by Bry [Bry90]. Yahya [Yah00] investigates the problem of efficiently answering positive ground queries to disjunctive deductive databases without default negation. He applies model generation to the *dual database*, which is obtained by reversing the rule arrows in all rules. This generalizes SLD–resolution, and it yields *goal–oriented query processing* which is driven by the facts that are derived from the query goal. Hasegawa et al. [HIOK97,OIH98] have proposed an extension of magic sets for range–restricted disjunctive logic programs. Also Greco, cf. [Gre98,Gre99], presents a rewriting technique for the optimization of bound queries to disjunctive deductive databases. Experiments have shown that the rewriting greatly reduces the number of models to be considered to answer a query.

Complexity Results

Imielinski and Vadaparty [IV89], Vardi [Var82] and Imielinski [Imi91] have investigated the complexity of answering queries in disjunctive logic programs. Chomicki and Subrahmanian [CS90] discuss the complexity of the *GCWA*. For disjunctive theories that are tractable, see [BED94]. For complexity results for disjunctive propositional logic programs see Eiter and Gottlob [EG95,DEGV97]. A summary of complexity results, drawn from [EG95], is given in Table 1.

	Propositional		First Order over Herbrand models no function symbols[5]	
Semantics	Complexity	Ref.	Data Complexity	Ref.
Positive Consequences				
Minimal Models	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
Negation				
GCWA	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
WGCWA	co–$\mathcal{NP}$–complete	[Cha93]		
Stratified Programs				
Perfect	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
Locally Stratified Programs				
Perfect	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
Normal Programs				
Stable	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
Partial Stable	Π_2^P–complete	[EG93a]	Π_2^P–complete	[EGM94]
Extended Programs				
Stable	Π_2^P–complete	[?]	Π_2^P–complete	[EGM94]
Partial Stable	Π_2^P–complete	[MR94]	Π_2^P–complete	[EGM94]

Table 1. (Taken from [EG93a,EG95]) Complexity of Cautious Reasoning with Disjunctive Logic Programs (with Integrity Constraints)

The development of model theoretic, fixpoint and proof procedures has placed the semantics of DDDBs on a firm foundation. Methods to handle DDDBs have started and are discussed in Section 5. The GCWA and alternative theories of negation have enhanced our understanding of default negation in DDDBs. Complexity results provide an understanding of the difficulties to find answers to queries in such systems.

Disjunctive Deductive Databases with Default Negation

Fernández and Minker [FM95] present a new fixpoint characterization of the minimal models of *DDDBs* and stratified *DDDBs*. They prove that by applying the operator iteratively, in the limit, it constructs the perfect models semantics (Przymusinski [Prz88]) of stratified *DDDBs*. Given the equivalence between the *perfect models semantics of stratified programs* and *prioritized circumscription* [Prz88] their fixpoint characterization captures the meaning of the corresponding circumscribed theory. They present a bottom–up evaluation algorithm for stratified *DDDBs* using the *model–tree* data structure to represent and to compute answers to queries. In [FM92], they develop the theory of *DDDBs* using model trees. Work on updates in *DDDBs* is described in [GHLM93,FGM96].

Four alternative semantics were developed for non–stratifiable normal DLPs at approximately the same time: Ross [Ros89], Baral et al. [BLM90a,BLM90b]* ([Min99]), and two semantics by Przymusinski [Prz90a,Prz90c]. Ross termed his semantics the *strong well–founded semantics*, Baral et al. defined their semantics the *Generalized Disjunctive Well–Founded Semantics (GDWFS)*. They defined a fixpoint operator, and gave model and proof theoretic semantics for such *DDDBs*. Przymusinski [Prz90c] extends *stable model semantics* for normal *DDDBs*. He also defined in [Prz90a] the *stationary semantics*. As in the case of normal *DDBs* it will be necessary to develop effective bottom–up computing techniques to answer queries in these theories.

In addition, other important semantics have been developed. Przymusinski [Prz95] describes a new *semantic framework* for disjunctive logic programs and introduces the *static expansions* of disjunctive programs. The class of static expansions extends both the classes of stable, well–founded and stationary models of normal programs and the class of minimal models of disjunctive programs. Any static expansion of a program $\mathcal{P}$ provides the corresponding semantics for $\mathcal{P}$ consisting of the set of all sentences logically implied by the expansion. The stable model semantics has also been extended to disjunctive programs [GL91,Prz91]. Leone et al., [LRS96], develop an algorithm for solving the (co–$\mathcal{NP}$–hard decision) problem of checking if a model is stable. It runs in polynomial time (in the worst case) on the class of *head–cycle free programs* (discussed below), and in the case of general disjunctive logic programs limits the inefficient part of the computation only to components of the program which are not head–cycle free. Leone et al. [LRS95,LRS97] extend the notion of unfounded sets from normal to disjunctive logic programs and provide a declarative characterization of disjunctive stable models in terms of unfounded sets. They define an algorithm to compute stable model semantics of disjunctive logic programs. [BLR97] extend Disjunctive DATALOG to include integrity constraints and so–called weak constraints that are satisfied if possible.

Several types of *program transformations* have been used for characterizing existing semantics of DLPs and for defining new semantics based on existing ones. Brass and Dix [BD95a,BD99] have used *partial evaluation* and *program simplification* for defining and computing their *disjunctive well–founded semantics* D-WFS; this semantics is of interest as it permits a general approach to bottom–up computation in disjunctive programs. In addition, their transformation approach leads to several confluent calculi ([BD98]) which leads both to a better understanding ([BD97]) and an efficient computation of such semantics ([DFN99]). In [DS98] this approach was extended to first–order programs and coupled with constraint logic programming techniques. A restriction of the transformation approach to normal programs yields an implementation of the WFS which is provably better than the alternating fixpoint procedure and is linear for almost all programs occurring in practice ([BDFZ01]).

Seipel et al. [SMR97b] have characterized the partial stable models of a disjunctive logic program $\mathcal{P}$ as the stable models of a transformed disjunctive logic program $\mathcal{P}^{\mathsf{tu}}$, which is called the tu–*transformation*. Fernández et al. [FLMS93] and Seipel [Sei97] have used the *evidential transformation* for characterizing *total stable models*, and *evidential stable models*, respectively. In [Sei00] a disjunctive logic program $\mathcal{P}$ is mapped to a normal logic program $\mathcal{P}^{\mathsf{cd}}$ with function symbols for lists, such that the *total stable models* of $\mathcal{P}$ correspond to the stable models of $\mathcal{P}^{\mathsf{cd}}$. Moreover, a new semantics for DLPs, which is called *stable state semantics*, is defined based on the stable models of a variant of $\mathcal{P}^{\mathsf{cd}}$.

Eiter et al. [ELS97,ELS98] summarize results for partial stable models [Prz91]; maximum stable models (M–stable) which are the maximal partial stable models in the knowledge ordering [Sac96]; regular models of You and Yuan [YY94] which are similar in spirit to M–stable models, but based on a weaker concept[6]; and least undefined stable models (L–stable) [Sac96] which are the partial stable models with the minimal degree of undefinedness. In [YWY97], You et al. presented another semantics for DLPs, called regular extension semantics, for which a generalization of the abductive proof procedure of Eshghi and Kowalski [EK89] is used as a top–down method for query answering [YYG00].

Special Classes of DDDBs and Complexity Issues

As noted previously, there are semantics both for extended *DDBs* and extended *DDDBs*. A user of such a system has the problem of selecting the appropriate semantics for his needs. Which semantics should be used, and under what circumstances? There have been no guidelines developed. However, many complexity results have been obtained for these semantics. Schlipf [Sch95] and Eiter and Gottlob [EG95] summarize complexity results known for alternative semantics.

[6] For normal databases (without disjunctions), the M–stable models coincide with the regular models.

Some of these results, taken from [EG93a,EG95], are listed in Table 1. Further results are reported in [ELS98]. A user may wish to determine the semantics to be used based upon the complexity expected to find answers to queries.

Ben–Eliyahu and Dechter [BED94] investigate tractable cases of disjunctive theories. They introduced the concept of a *head–cycle free (HCF)* program as follows. A *dependency graph* $G_{\mathcal{P}}$ is associated with each program $\mathcal{P}$ as follows:

- each rule and each predicate in $\mathcal{P}$ is a node.
- there is a positive (negative) arc from a predicate node p to a rule node r iff p appears positive (negative) in the body of r, and an arc from r to p (resp., and also an arc from p to r) if p appears in the head of r.

The *positive dependency graph* of $\mathcal{P}$ is a subgraph of $G_{\mathcal{P}}$ containing only positive arcs. A directed cycle in $G_{\mathcal{P}}$ is called *negative* if it contains at least one negative arc. $\mathcal{P}$ is *head–cycle free (HCF)* if for every two predicate symbols p and q, if p and q are on a positive directed cycle in the dependency graph $G_{\mathcal{P}}$ then there is no rule in $\mathcal{P}$ in which both p and q appear in the head. They show in [BEPZ96] that answers to queries expressed in this language can be computed in polynomial time. It is shown in [BEP94] that there is an algorithm that performs, in polynomial time, minimal model finding and minimal model checking if the theory is *HCF*. An efficient algorithm to solve the (co–$\mathcal{NP}$–hard) problem of checking if a model is stable in function–free disjunctive logic programs is developed in [LRS96]. The algorithm runs in polynomial time on *HCF* programs and in the case of general disjunctive logic programs, it limits the inefficient part of the computation only to the components of the program which are not *HCF*.

Dix et al. [DGM96] describe *causal* programs, where disjunction is simulated by negation–as–failure. Disjunctive programs are reduced to *stratified nondisjunctive* programs by a series of *shift operations*. They show *causal* semantics belongs to the first level of the polynomial hierarchy unlike minimal model semantics (GCWA), which is Π_2^P-complete for positive disjunctive programs. *Causal semantics* are also *cumulative* and *rational* (see [Dix95]). The class of *positive causal* programs extends the class of *positive HCF* programs [BED94].

Consideration has been given to approximate reasoning. In such reasoning, one may give up soundness or completeness of answers. Efforts have been developed both for deductive and disjunctive deductive databases by Selman and Kautz [SK91,KS92,SK96], who developed lower and upper bounds for Horn (DATALOG) databases and compilation methods, by Cadoli [Cad93], who developed computational and semantical approximations, and by del Val [deVa95], who developed techniques for approximating and compiling databases. See Cadoli [Cad96] for references on compilation, approximation and tractability of knowledge bases.

The complexity results in Table 2 refer to worst case analysis for skeptical reasoning, i.e. to determining if a given literal is *true* in every canonical model (with respect to a particular semantics) of the program. For logic programs with no function symbols, the data complexity over an EDB E is presented. The notation used is the following: $|\mathcal{P}|$ denotes the length of the program $\mathcal{P}$; $|\mathcal{A}|$

Semantics	Propositional		First Order over Herbrand models		First Order no function symb. over Herbrand models	
	Complexity	Ref.	Complexity	Ref.	Data Complexity	Ref.
Positive Consequences						
Minimal Model	$\mathcal{O}(\lvert\mathcal{P}\rvert)$	[DG84], [IM82]	r.e.–complete	[Smu56], [AN78]	polynomial in $\lvert E\rvert$	[CH82]
Negation						
CWA	$\mathcal{O}(\lvert\mathcal{P}\rvert)$	[DG84], [IM82]	co–r.e.–complete	[Smu56], [AN78]	co–r.e.–complete	[CH82]
Stratified Programs						
Perfect	$\mathcal{O}(\lvert\mathcal{P}\rvert)$		complete arithmetic	[AB90]	polynomial in $\lvert E\rvert$	[CH85]
Locally Stratified Programs						
Perfect	$\mathcal{O}(\lvert\mathcal{P}\rvert)$		Δ^1_1–complete over ω	[BMS92]* ([Min99])	N/A	
Normal Programs						
2–valued completion	co–$\mathcal{NP}$–complete	[KP91]	Π^1_1–complete over ω	[KP91]	co–$\mathcal{NP}$–complete	[KP91]
3–valued completion	$\mathcal{O}(\lvert\mathcal{P}\rvert)$	folklore	Π^1_1–complete over ω	[Fit85]	polynomial in $\lvert E\rvert$	[Fit85]
Stable	co–$\mathcal{NP}$–complete	[MT91]	Π^1_1–complete over ω	[MNR92], [Sch90]	co–$\mathcal{NP}$–complete	[MT91]
Well–Founded	$\mathcal{O}(\lvert\mathcal{A}\rvert\lvert\mathcal{P}\rvert)$	folklore	Π^1_1–complete over ω	[VaGe89], [Sch90]	polynomial in $\lvert E\rvert$	[VGRS91], [VaGe89]
Extended Programs						
Stable	co–$\mathcal{NP}$–complete	[MR94]	Π^1_1–complete over ω	[MR94]	co–$\mathcal{NP}$–complete	[MR94]
Well–Founded	$\mathcal{O}(\lvert\mathcal{A}\rvert\lvert\mathcal{P}\rvert)$	[MR94]	Π^1_1–complete over ω	[MR94]	polynomial in $\lvert E\rvert$	[MR94]

Table 2. (Adapted from [Sch95]) Complexity of Cautious Reasoning with Horn Logic Programs

denotes the number of ground atoms in the language of $\mathcal{P}$; $|E|$ denotes the total number of symbols that occur in the *EDB* E.

Properties of Semantics

A second way to determine the semantics to be used for an application is through their properties. Dix in [Dix92a] proposed criteria useful in determining the appropriate semantics to be used. He developed semantics both for normal *DDBs* [Dix95] and normal *DDDBs* [BD95b] that satisfy some of the properties that he describes. While some properties are adaptations and extensions to those developed by Kraus et al. [KLM90] to compare nonmonotonic theories, *relevance*, *partial evaluation* and *modularity* were newly developed.

A property an arbitrary semantics, *SEM*, might have is that its semantics should not be changed if a tautology is eliminated from its database. Table 3 summarizes other useful properties of semantics of *DDDBs* and specifies for alternative semantics the properties that they satisfy. This table is adapted from tables in [Dix95d,BD98,BD97]. Although complexity results and properties that a semantics satisfy are extremely useful, no generally accepted criteria exist as to why one semantics should be used over another. A semantics may have all the properties one may desire, be computationally tractable and yet not provide answers that a user expected. If for the normal logic program $\mathcal{P} = \{\, a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, a,\ c \leftarrow a,\ c \leftarrow b \,\}$ of Section 3 the user expected an answer *yes* in response to a query "$\leftarrow c$", and the semantics were the *WFS*, the user would receive the answer *unknown*. However, if the *stable model semantics* had been used, the answer returned would be *yes*. Perhaps the best that can be expected is to provide users with complexity results and criteria so they may decide which semantics meets the needs of their problems.

Relationship to More General Forms of Nonmonotonic Reasoning

Understanding the semantics of disjunctive theories is related to nonmonotonic reasoning. The field of nonmonotonic reasoning has resulted in several alternative approaches to perform default reasoning [McCa80,Rei80,MD80,Moo84], [Moo85]. The articles [Min93,EG95,CS93] cite results where alternative theories of nonmonotonic reasoning can be mapped into extended disjunctive logic programs and databases. Hence, *DDDBs* may be used to compute answers to queries in such theories. In [BE98] priority information on extended logic programs and principles that an approach to handling priorities should satisfy are discussed. The expressive power of a query language over a disjunctive ground database is studied in [BE96]. They show there exist simple queries that cannot be expressed by any preferential semantics (including minimal model semantics and various forms of circumscription), while they can be expressed in default and autoepistemic logic. Default logic, autoepistemic logic and some of their fragments are shown to express the same class of Boolean queries, which turns out to be a strict subclass of the Σ_2^p-recognizable Boolean queries. They prove that under the assumption that the database consists of clauses whose length is

Property	Condition on a Semantics *SEM* to satisfy the Property						
	Clark's Compl.	GCWA	WGCWA	Perfect	Stable	WFS, D-WFS	Static
Elimination of Tautologies	If a rule $\alpha \leftarrow \beta \wedge not \cdot \gamma$ with $\alpha \cap \beta \neq \emptyset$ is eliminated from a program $\mathcal{P}$, then the resulting program is *SEM*–equivalent to $\mathcal{P}$.						
	No	Yes	No	Yes	Yes	Yes	Yes
Generalized Principle of Partial Evaluation (GPPE)	If a rule $\alpha \leftarrow \beta \wedge not \cdot \gamma$, where β contains an atom B, is replaced in a program $\mathcal{P}$ by the n rules $\alpha \cup (\alpha^i - \{B\}) \leftarrow ((\beta - \{B\}) \cup \beta^i) \wedge not \cdot (\gamma \cup \gamma^i)$, where $\alpha^i \leftarrow \beta^i \wedge not \cdot \gamma^i$ $(i = 1, \ldots, n)$ are all rules for which $B \in \alpha^i$, then the resulting program is *SEM*–equivalent to $\mathcal{P}$.						
	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Positive and Negative Reduction	If (1) a rule $\alpha \leftarrow \beta \wedge not \cdot \gamma$ is replaced in a program $\mathcal{P}$ by $\alpha \leftarrow \beta \wedge not \cdot (\gamma - C)$ where C appears in no rule head, or (2) a rule $\alpha \leftarrow \beta \wedge not \cdot \gamma$ is deleted from $\mathcal{P}$ if there is a fact $\alpha' \leftarrow$ in $\mathcal{P}$ such that $\alpha' \subseteq \gamma$, then the resulting program is *SEM*–equivalent to $\mathcal{P}$.						
	Yes	N/A	N/A	Yes	Yes	Yes	Yes
Elimination of Non–Minimal Rules	If a rule $\alpha \leftarrow \beta \wedge not \cdot \gamma$ is deleted from a program $\mathcal{P}$ if there is another rule $\alpha' \leftarrow \beta' \wedge not \cdot \gamma'$ such that $\alpha' \subseteq \alpha$, $\beta' \subseteq \beta$, and $\gamma' \subseteq \gamma$, where at least one $\subseteq$ is proper, then the resulting program is *SEM*–equivalent to $\mathcal{P}$.						
	Yes	Yes	No	Yes	Yes	Yes	Yes
Consistency	$SEM(\mathcal{P}) \neq \emptyset$ for all disjunctive deductive databases $\mathcal{P}$.						
	No	Yes	Yes	Yes	No	Yes	Yes
Independence	For every literal L, L is true in every $M \in SEM(\mathcal{P})$ iff L is true in every $M \in SEM(\mathcal{P} \cup \mathcal{P}')$ provided that the language of $\mathcal{P}$ and $\mathcal{P}'$ are disjoint and L belongs to the language of $\mathcal{P}$.						
	No	Yes	Yes	Yes	No	Yes	Yes

Table 3. (Adapted from [Dix95a]* ([Min99])) Properties of the semantics of disjunctive deductive databases.

bounded by some constant, default logic and autoepistemic logic express *all* of the Σ_2^p-recognizable Boolean queries, while preference–based logics cannot. Eiter and Gottlob [EG97] show that over the standard infinite Herbrand universe, disjunctive logic programming and normal logic programming under the (cautious) stable model semantics coincide. See Cadoli and Lenzerini for complexity results concerning circumscription and closed world reasoning [Cad92,CL94]. See Yuan and You [YY93] for relationships between autoepistemic circumscription and logic programming. They use two different belief constraints to define two semantics, the *stable circumscriptive semantics* and *the well–founded circumscriptive semantics*, for autoepistemic theories. The work in [YY93] and on *static semantics* developed by Przymusinski [Prz95] appear to be related. As shown in [BDP96,BDNP98], these approaches, though differently defined, are also related to the D-WFS approach [BD95a,BD99]

DDDBs have also contributed to the *null value* problem. If an attribute of a relation may have a null value, where this value is part of a known set, then one can represent this as a disjunction of relations, where, in each disjunction a different value is given to the argument. For papers on the null value problem both in relational and *DDBs*, see [Cod79,GM86,Lip81,Rei86,Zan84] and [Vas79]* ([Min99]).

Extended–Disjunctive Deductive Databases

Gelfond and Lifschitz ([GL90,GL91]) pointed out the need for another form of negation next to default negation. They called called it *classical negation* and they extended stable model semantics to work with classical negation, i.e., they defined answer set semantics. Alferes, Pereira and Przymusinski ([APP98]) introduced two other definitions of negation, which they summarize as symmetric negations: *strong negation* and *explicit negation*. For logic programs with stable model semantics both coincide with classical negation. Symmetric negation can be used to provide natural solutions to various knowledge representation problems, such as theory and interpretation update, and belief revision. Minker and Ruiz ([MR94]) describe general techniques for extending semantics to extended–disjunctive logic programs, and they apply these techniques to stable models, disjunctive well–founded and stationary semantics. In [RM97,RM98] they study the semantics of extended–disjunctive logic programs that simultaneously contain multiple kinds of default negation.

Extensions of Disjunctive Logic Programming

There are several effective implementations of disjunctive logic programming systems, some of which will be described in more detail in Section 5. These systems will have to be enhanced to be able to handle more sophisticated applications. Several new concepts, introduced for deductive databases have to be extended and incorporated into disjunctive system implementations.

Extensions of disjunctive programs to more general belief programs are considered in [BDP96,BDNP98]. Buccafurri et al. extend Ordered Logic programming to disjunctive theories. They relate the work to knowledge base systems, show the language $\mathcal{DOL}$ *(Disjunctive Ordered Logic)* to be useful for diagnostic processes based on stepwise refinements and study the expressive power and complexity of $\mathcal{DOL}$. Buccafurri et al. [BLR97] also extended disjunctive logic programs by *weak constraints* and defined a semantics which tends to minimize the number of violated instances.

Practical applications of disjunctive logic programs often require the use of extended features like *aggregation operators*. In [Sei99] it has been shown how one can deal with aggregation in the presense of *recursion* and *disjunction*. A suitable, intuitive semantics is defined based on stable model semantics, which is applied to a special *program transformation*, which replaces an aggregation by a suitable construct using *default negation* and the function symbol for lists. Another approach for dealing with aggregation in normal logic programs based on the choice construct was given by [ZW99].

Kifer, Lausen, and Wu, cf. [KLW95] proposed *F–Logic* (Frame Logic) as a database language that accounts in a clean declarative fashion for many object–oriented features such as object identity, complex objects, inheritance, polymorphic types, methods, and encapsulation. F–Logic has a formal semantics and a sound and complete resolution–based proof procedure. Buccafurri, Faber, and Leone [BFL99] have proposed a new knowledge representation language $DLP^{<}$, which extends disjunctive logic programming with strong negation by inheritance.

5 Implementations of DLP–Systems

In this section we discuss four systems that are relevant to disjunctive logic programming: Smodels developed by Niemela and Simons [NS97], DLV developed by Eiter et al. [ELM+98], XSB developed by Warren et al. [RSS+97,War99], and DeReS developed by Marek and Truszczyński [CMT99].[7]

The system Smodels ([NS97]) was designed to handle range–restricted normal logic programs $\mathcal{P}$ without function symbols, where additionally every variable occurring in a rule must appear in a *domain literal* (i.e., a literal with a non–recursive predicate symbol). Smodels can compute the well–founded semantics as well as the stable model semantics. The inference algorithm for computing stable models is based on bottom–up backtracking search, where a powerful pruning method – related to the WFS – is employed. One of the advantages of Smodels is that it can be implemented to work in linear space. This makes it possible to apply the stable model semantics also in areas where the programs are highly non–stratified and can possess a large number of stable models. In

[7] The systems XSB and DeReS are not disjunctive. However, they can simulate disjunctive systems by shifting all but one literal in the head of a disjunctive clause as negated literals to the body. If this is done for all literals in the head, the systems can partially simulate disjunctive approaches.

[JNSY00] it is shown how Smodels can be used as a core engine for computing stable models of disjunctive logic programs without function symbols as well. Smodels has been tested using examples from the logic programming literature, combinatorial graph problems, circuit diagnosis, and propositional satisfiability.

The DLV system of Eiter et al. ([ELM+98]) can handle range–restricted disjunctive deductive databases, i.e. disjunctive logic programs without function symbols. DLV takes a new approach to intractability. Instead of attempting to guarantee performance under restrictive assumptions, it relies on the fact that it is often easy to find solutions to instances of hard problem classes, and employs a so–called "guess and check" inference algorithm. The model generator (MG) of DLV exploits heuristic information to construct (guess) one candidate for a stable model at a time, which then is verified by the model checker (MC) against the given problem constraints. In the event that the constraints turn out not to be satisfied, another guess must be made. Considerable success was achieved with DLV in quickly solving NP–complete problems such as finding Hamiltonian paths. Other applications where DLV was applied are planning under incomplete knowledge ([EFLPP00]) and abductive diagnosis ([EFLP98]* ([Min99])). DLV appears to indicate that the effectiveness of "iterative sampling" approaches ([CMT99]) (following a single, possibly heuristically–guided "probe" through the search tree) discovered in the constraint satisfaction literature carries over to logic–based approaches such as disjunctive logic programming.

Both DLV and Smodels have been implemented in C++. They both use intelligent grounding modules which generate a subset $\mathcal{P}'$ of the grounded input program, such that $\mathcal{P}'$ is much smaller than $gnd(\mathcal{P})$ but has the same set of stable models.[8]. Also, both systems use a program simplification method that is based on the WFS for normal logic programs. For DLV, which works with disjunctive deductive databases, a generalization of the notion of unfounded sets, which is fundamental in WFS, is used ([LRS95,LRS96]). The algorithms for computing stable models are then applied to the grounded and simplified versions of the original programs.

The system XSB ([RSS+97]) can compute most cases of the WFS for normal logic programs with function symbols. The inference engine, which is called the SLG–WAM, consists of an efficient tabling engine for definite logic programs, which is extended by mechanisms for handling cycles through negation. These mechanisms are negative loop detection, delay and simplification. They serve for detecting, breaking and resolving cycles through negation. XSB is the only nonmonotonic reasoning system that is a fully–fledged PROLOG–system as well. XSB has been used in systems that contain large sets of rules and data.

The system DeReS ([CMT99]) supports basic automated reasoning tasks for default logic and for logic programming with the stable model semantics. It is shown that a normal logic program $\mathcal{P}$ can be represented by a suitable default theory $\mathcal{D}$, such that the stable models of $\mathcal{P}$ correspond to the so–called *extensions* of $\mathcal{D}$. DeReS uses *relaxed stratification* as a primary mechanism for pruning the search–space. A default theory $\mathcal{D}$ is partitioned into several smaller subtheories,

[8] For Smodels, the intelligent grounding module is called lparse.

called *strata*, and the extensions of $\mathcal{D}$ are constructed from the extensions of its strata. The approach taken by DeReS is somehow orthogonal to the one taken by Smodels, and it is argued in [CMT99] that next–generation implementations of nonmonotonic systems must combine techniques developed in both projects in order to be effective in a large range of different applications. DeReS has been tested extensively on the domain of *combinatorial problems*. A benchmarking environment called TheoryBase has been implemented, which can systematically generate parameterized families of default theories that encode graph problems taken from the Stanford GraphBase developed by Knuth.

Other systems that implement disjunctive semantics are the following: The system *near Horn* was implemented in PROLOG by a group headed by Loveland [Llo87,Lov87], [SL88,RL90,RLS91]. Seipel [Sei94] developed a system DISLOG that incorporates different disjunctive theories and strategies including the semantics introduced in [LMR92]; DISLOG tries to eliminate redundant computations by using a breadth–first approach. The system DISLOP, headed by Dix and Furbach [ADN97b], aims at extending the *restart model elimination* and *hyper tableau calculi* for DLPs under the D-WFS and stable semantics.

6 Applications

As discussed in Section 4, a problem with disjunctive theories is the complexity of the theory, in general. There was also a sense that there were few applications for disjunctive theories. This impression has changed during the past few years. Below, we discuss several applications that have arisen that require disjunctive theories: data integration, abductive reasoning, knowledge representation, diagnosis, and graph coloring.

6.1 Data Integration

An important part of data integration involves answering queries using various resources rather than by accessing database relations. The process of transforming a query from database relations to resources is often referred to as query folding. For instance, a database of interest to a user may be distributed over a network. It is necessary to bring data distributed over a network to a user's machine so that the data may be manipulated to answer user queries. In a distributed environment it is likely that one will want to save answers to queries in the local user's machine so that if the same or a related query is posed to the distributed database, one can look in the local machine's cached database for answers, rather than have to access data over the network to answer the query. In this situation the resources are the cached relations and the use of these resources is an important aspect of query optimization. In some data integration systems the database relations are themselves virtual and the data must be obtained from the resources. Resources may also be materialized views.

Several researchers have considered various aspects of this problem. Grant and Minker (see [GM00] for references) take a logic–based approach to the problem. We briefly sketch where disjunctions enter into seemingly relational and Datalog databases. Consider a database that consists of an extensional database (EDB), an intensional database (IDB), a set of integrity constraints (ICs), and a set resources ($\mathsf{Res}_{\mathsf{DB}}$), where the resources have been obtained by using resource rules. These resources are referred to as materialized views. That is, they have been made explicit in a local computer as a result, for example, of an answer to a conjunctive query. The EDB, IDB, ICs are part of a conventional Datalog database. A resource r may be defined by several Horn rules as, e.g.,

$$r(X,Y) \leftarrow p(X,Y),$$
$$r(X,Y) \leftarrow q(X,Y) \wedge s(X,Y).$$

Thus, the relation r has been obtained by retrievals from the relations p, q, and s. The assumption is made that these and only these definitions are how r has been formed. This is effectively an iff assumption, that is, r is defined as:

$$r(X,Y) \leftrightarrow (\, p(X,Y) \vee (\, q(X,Y) \wedge s(X,Y)\,)\,).$$

This effectively is the *Clark completion axiom* for the resource r. This then yields the two rules,

$$p(X,Y) \vee q(X,Y) \leftarrow r(X,Y),$$
$$p(X,Y) \vee s(X,Y) \leftarrow r(X,Y).$$

Given the integrity constraint,

$$\leftarrow q(X,a) \wedge s(X,a)$$

the query, $\leftarrow p(X,a)$ can be answered by querying the resource $r(X,Y)$ with $\leftarrow r(X,a)$. Thus, in this instant if the s, p, and q relations are not local to the original query, but the resource r is local, answering the query may be done in an optimized manner.

6.2 Abductive Reasoning

The idea of performing abductive diagnosis based on disjunctive logic programming has been studied by several researchers, e.g. Sakama and Inoue [SaIn00] or Eiter et al. [EFLP98]* ([Min99]). An abductive diagnosis explains a set $\mathcal{O}$ of *observations* based on background knowledge and a set $\mathcal{H}$ of *hypotheses*. In [SaIn00] and [EFLP98]* ([Min99]) the background knowledge is given by an extended disjunctive logic program $\mathcal{P}$, and $\mathcal{H} \subseteq H_{L\mathcal{P}}$ is a set of ground literals. In [SaIn00], $\Pi = \langle \mathcal{P}, \mathcal{H} \rangle$ is called an abductive logic program, and the elements of $\mathcal{H}$ are called *abducibles*. In [EFLP98]* ([Min99]) an abductive diagnosis problem $\langle \mathcal{P}, \mathcal{H}, \mathcal{O} \rangle$ consists of an abductive logic program $\Pi = \langle \mathcal{P}, \mathcal{H} \rangle$ and a set $\mathcal{O} \subseteq H_{L\mathcal{P}}$ of ground literals, called *observations*. An *abductive diagnosis* is given by a set $\mathcal{H}' \subseteq \mathcal{H}$ of hypotheses, such that $\mathcal{P}' = \mathcal{P} \cup \mathcal{H}' \models \mathcal{O}$, where $\mathcal{P}' \models \mathcal{O}$, iff for some answer set S of $\mathcal{P}$ it holds $\mathcal{O} \subseteq S$. Then, $\mathcal{E} = S \cap \mathcal{H}$ is called an *explanation* in [SaIn00]. E.g. if the logic program $\mathcal{P}$ consists of the two rules

$$r_1 = \mathit{wet\_grass} \leftarrow \mathit{rained},$$
$$r_2 = \mathit{wet\_grass} \leftarrow \mathit{sprinkler\_on},$$

and $\mathcal{H} = \{\, \mathit{rained}, \mathit{sprinkler\_on} \,\}$, then the observation $\mathcal{O} = \{\, \mathit{wet\_grass} \,\}$ can be explained by $\mathcal{E} = \{\, \mathit{rained} \,\}$, since the answer set $S = \{\, \mathit{wet\_grass}, \mathit{rained} \,\}$ of $\mathcal{P} \cup \mathcal{E}$ contains $\mathcal{O}$. The abductive diagnosis $\mathcal{H}' = \mathcal{E} = \{\, \mathit{rained} \,\}$[9] is called a single–error diagnosis, since $|\mathcal{H}'| = 1$.

In [EFLP98]* ([Min99]) a simple transformation to extended–disjunctive logic programs is given which allows for computing explanations based on the answer set semantics of Gelfond and Lifschitz:

$$\mathit{dlp}(\Pi, \mathcal{O}) = \mathcal{P} \cup \{\, A \vee \epsilon_A \,|\, A \in \mathcal{H} \,\} \cup \{\, \leftarrow \neg o \,|\, o \in \mathcal{O} \,\} \cup \{\, \leftarrow o \,|\, \neg o \in \mathcal{O} \,\}.$$

If S is an answer set of $\mathit{dlp}(\Pi, \mathcal{O})$, then $\mathcal{E} = S \cap \mathcal{H}$ is an explanation. The disjunctive facts $A \vee \epsilon_A$ allow for assuming arbitrary hypotheses, and the denial rules enforce that all observations from $\mathcal{O}$ will hold in S.

Refinements for computing minimal abductive diagnoses or for computing abducibles that should be deleted from the logic program $\mathcal{P}$[10] have also been investigated.

Deduction vs. Abduction. Abductive reasoning with normal logic programs can be done based on deduction with disjunctive logic programs, cf. Figure 4. Disjunction is introduced by *program completion* which turns normal rules into into disjunctive rules. From the normal rules $r_i = A \leftarrow \beta_i$ defining an atom A

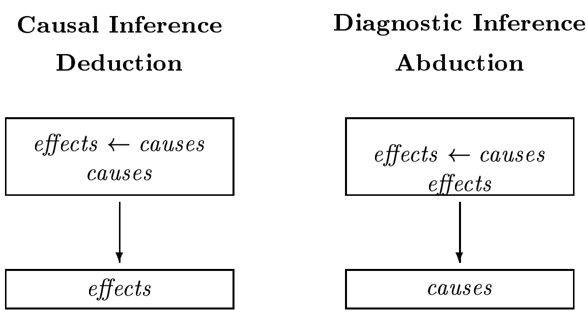

Fig. 4. Deduction vs. Abduction

[9] Here it holds that $\mathcal{H}' = \mathcal{E}$, which need not be true in general.
[10] in addition to the set $\mathcal{H}'$ of abducibles that is added to $\mathcal{P}$

we conclude the disjunctive rule $r = \beta_1 \vee \ldots \vee \beta_k \leftarrow A$ by program completion.[11] This diagnostic rule r allows one to deduce one of $\beta_1, \ldots, \beta_k$ as an abductive diagnosis (cause) for the observation (effect) A.

6.3 Knowledge Representation

As noted by Baral and Gelfond [BG94],

> *Knowledge representation is one of the most important subareas of artificial intelligence. If we want to design an entity (a machine or a program) capable of behaving intelligently in some environment then we need to supply this entity with sufficient knowledge about this environment. To do that, we need an unambiguous language capable of expressing this knowledge, together with some precise and well–understood way of manipulating sets of sentences of the language which will draw inferences, answer queries and update both the knowledge base and the desired program behavior.*

They note that McCarthy [McCa59] first proposed the use of *logical formulas* as a basis of a knowledge representation language of this type. In their paper, they provide many examples of the use of a logic programming formalism to represent and manipulate knowledge. The following example is used in [Poo89] to demonstrate the difficulties with representing disjunctive information in Reiter's default logic. They note that the example has a natural representation in the language of disjunctive logic programs.

> Normally, a person's left arm is usable, but a person with a broken left arm is an exception. and similarly for the right arm. Suppose also that we remember seeing Matt with a broken left arm or a broken right arm, but we do not remember which.

Baral and Gelfond show how to represent the information in the language of disjunctive logic programs, where the predicate names are self–explanatory, except that $ab(X,Y)$ denotes that it is 'abnormal that person Y's arm is X'. They represent the first statement as

$$\begin{aligned} &lh\_usable(X) \leftarrow not\, ab(l,X) \\ &ab(l,X) \leftarrow lh\_broken(X) \\ &rh\_usable(x) \leftarrow not\, ab(r,X) \\ &ab(r,X) \leftarrow rh\_broken(X). \end{aligned}$$

The second statement is represented as

$$lh\_broken \vee rh\_broken(X).$$

[11] If one of the β_i is a conjunction of more than one atom, then r is not a disjunctive rule of the form (1), but it can be normalized to an equivalent set of disjunctive rules by applying the distributivity law to $\beta_1 \vee \ldots \vee \beta_k$.

They then represent the *closed world assumption* for the broken predicates by the following two rules:

$$\neg lh\_broken(X) \leftarrow not\, lh\_broken(X)$$
$$\neg rh\_broken(X) \leftarrow not\, rh\_broken(X)$$

The last two rules state that if an arm is considered broken by default (*not*), then is is logically considered broken ($\neg$). The disjunctive logic program consisting of the above seven rules has two answer sets:

$$\{\, lh\_broken(matt),\ ab(l, matt), rh\_usable(matt),\ \neg rh_b roken(matt)\,\},$$
$$\{\, rh\_broken(matt),\ ab(r, matt),\ lh\_usable(matt),\ \neg lh\_broken(matt)\,\}$$

and therefore infers

$$rh\_usable(matt) \vee lh\_usable(matt)$$

which correspond to the intended specification.

Baral and Gelfond also show how the logical representation of knowledge can be used for more complicated examples.

6.4 Medical Diagnosis

As an example we will consider the following disjunctive deductive database for *medical diagnosis*, which deals with the pair of genes that determine a persons blood–group, cf. Figure 5. Every person has two genotypes, which may be identical. One is inherited from the mother, the other is inherited from the father.

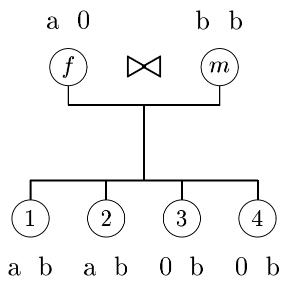

Fig. 5. Inheritance of Blood–Groups

A statement *genotype*(P, T) means that the person P has the genotype T. The predicates *homozygot* and *heterozygot*, which are defined in the first three rules, below, tell if a person has two identical or two different genotypes, respectively. The rules r_4 and r_5 express that a person must have at least one of the genotypes

"a", "b", or "o" and can have at most two different genotypes. r_5 is an integrity constraint, which forbids that a single person has three different genotypes: it has an empty rule head denoting a contradiction, which is derived in that situation. The rules r_6 and r_7 relate the genotype of a person to the genotypes of the parents. Finally, the *phenotype*, i.e. the person's actual blood–group, is a result of the two genotypes. According to the rule r_8 for *phenotype*, four different phenotypes are possible: "a", "b", "ab", and "o". The rules for determining phenotype "a" are given by r_9 and r_{10}.

$$
\begin{aligned}
r_1 &= \mathit{heterozygot}(P, T_1, T_2) \leftarrow \\
&\qquad\qquad \mathit{genotype}(P, T_1) \wedge \mathit{genotype}(P, T_2) \wedge \mathit{not\ equal}(T_1, T_2). \\
r_2 &= \mathit{heterozygot}(P) \leftarrow \mathit{heterozygot}(P, T_1, T_2). \\
r_3 &= \mathit{homozygot}(P, T) \leftarrow \mathit{genotype}(P, T) \wedge \mathit{not\ heterozygot}(P). \\[6pt]
r_4 &= \mathit{genotype}(P, a) \vee \mathit{genotype}(P, b) \vee \mathit{genotype}(P, o) \leftarrow \mathit{person}(P). \\
r_5 &= \ \leftarrow \mathit{genotype}(P, T_1) \wedge \mathit{genotype}(P, T_2) \wedge \mathit{genotype}(P, T_3) \wedge \\
&\qquad\qquad \mathit{not\ equal}(T_1, T_2) \wedge \mathit{not\ equal}(T_1, T_3) \wedge \mathit{not\ equal}(T_2, T_3). \\[6pt]
r_6 &= \mathit{genotype}(\mathit{Child}, T_1) \vee \mathit{genotype}(\mathit{Child}, T_2) \leftarrow \\
&\qquad\qquad \mathit{parent}(\mathit{Child}, \mathit{Parent}) \wedge \mathit{heterozygot}(\mathit{Parent}, T_1, T_2). \\
r_7 &= \mathit{genotype}(\mathit{Child}, T) \leftarrow \\
&\qquad\qquad \mathit{parent}(\mathit{Child}, \mathit{Parent}) \wedge \mathit{homozygot}(\mathit{Parent}, T). \\[6pt]
r_8 &= \mathit{phenotype}(P, a) \vee \mathit{phenotype}(P, b) \vee \mathit{phenotype}(P, ab) \vee \\
&\qquad\qquad \mathit{phenotype}(P, o) \leftarrow \mathit{person}(P). \\
r_9 &= \mathit{phenotype}(P, a) \leftarrow \mathit{heterozygot}(P, a, o). \\
r_{10} &= \mathit{phenotype}(P, a) \leftarrow \mathit{homozygot}(P, a).
\end{aligned}
$$

Expanding the above rules to include those for phenotypes "b", "ab", "o", and a database of facts, one can then use the stable model semantics to compute answers to queries. This disjunctive logic program is non–stratified: *genotype* depends recursively on itself, and there is a negative dependency between *homozygot* and *heterozygot* involved in the recursion. Therefore it needs to be evaluated by stable model semantics or a semantics that handles default negation with recursion.

6.5 Graph Problems

Eiter et al. [EFLP00] have shown how complex graph problems can be encoded easily using disjunctive logic programming. A directed graph $G = \langle V, E \rangle$ is encoded using an unary predicate *node* for the set V of nodes and a binary predicate *edge* for the set E of edges.

In the following we will describe the problems of *3–colorability* and of *Hamiltonian paths*, which are classical NP–complete problems. Also other graph problems like independent sets and kernels in directed graphs or maximal independent sets and matchings in undirected graphs have been investigated using disjunctive logic programming.

3–Colorability. The problem is to color the edges of a directed graph $G = \langle V, E \rangle$ by three colors (say red, green and blue), such that adjacent nodes always have different colors.

$$\begin{aligned} g &= \mathit{color}(X, \mathit{red}) \vee \mathit{color}(X, \mathit{green}) \vee \mathit{color}(X, \mathit{blue}) \leftarrow \mathit{node}(X), \\ c &= \ \leftarrow \mathit{edge}(X, Y) \wedge \mathit{color}(X, C) \wedge \mathit{color}(Y, C). \end{aligned}$$

In [EFLP00] this program is called a *guess&check* program: the rule g nondeterministically guesses color assignments for the nodes in the graph, and the integrity constraint c checks that these choices are legal, i.e. that no two nodes which are connected by an edge have the same color.

Hamiltonian Path. The problem is to find a path in a directed graph $G = \langle V, E \rangle$ starting at a distinguished node "a" and passing through each node in V exactly once.

$$\begin{aligned} g &= \mathit{in\_path}(X, Y) \vee \mathit{out\_path}(X, Y) \leftarrow \mathit{edge}(X, Y), \\ a_1 &= \mathit{reached}(a), \\ a_2 &= \mathit{reached}(X) \leftarrow \mathit{reached}(Y) \wedge \mathit{in\_path}(Y, X), \\ c_1 &= \ \leftarrow \mathit{node}(X) \wedge \mathit{not\ reached}(X), \\ c_2 &= \ \leftarrow \mathit{in\_path}(X, Y) \wedge \mathit{in\_path}(X, Y') \wedge Y \neq Y', \\ c_3 &= \ \leftarrow \mathit{in\_path}(X, Y) \wedge \mathit{in\_path}(X', Y) \wedge X \neq X'. \end{aligned}$$

The rule g guesses a subset $E' \subseteq E$ of all given edges to be in the path. The auxiliary rules a_1 and a_2 compute all nodes that are reachable from the starting node "a" and the integrity constraint c_1 checks that there is no node that is not reachable from "a". Finally, the integrity constraints c_2 and c_3 check that there is no two edges in the path that start or end at the same node.

7 Summary and Assessment

During the almost 20 years of its history, the field of disjunctive logic programming and disjunctive deductive databases has made significant progress in the development of theories, implementation and applications. In the area of theory, two significant developments have taken place, the development of semantics, and the development of complexity results. In the area of implementation, we have seen several powerful systems developed: XSB, Smodels, DLV, and DeReS.

In the area of applications, we have seen many new applications arise, when initially it was difficult to imagine realistic problems that would need disjunctive theories.

We believe that the following significant developments have taken place in the development of theories for disjunctive logic programming. For disjunctive theories, the work of Minker [Min82] on default negation and that of Minker and Rajasekar [MR90] laid the groundwork for further developments. They showed that a corresponding fixpoint theory for pure disjunctive theories, based on mapping disjuncts to disjuncts, generalized the van Emden/Kowalski fixpoint operator and led to the concept of a minimal state, where a state is a set of disjuncts. The least minimal state for disjunctive theories corresponds to the least minimal model of Horn theories. The work on stratified deductive theories by Przymusinski [Prz88] and by Rajasekar and Minker [RM90] extended the work to stratified theories. The work by Lifschitz and Gelfond who developed answer set semantics for disjunctive theories that include both default and logical negation is also important. This work generalizes the stable model semantics to disjunctive theories. The work by Brass, Dix and Przymusinski on "super logic programs" ([BDP96]) generalizes the well–founded semantics to the disjunctive case. It is clear that the two dominant semantics of logic and disjunctive logic programming are the well–founded and the stable model semantics and their generalizations. The work of Eiter and Gottlob in the development of complexity results is also a significant accomplishment. They show that disjunctive theories capture computationally hard problems that cannot be handled by extended logic programs. This makes it possible to implement extremely complex problems from the complexity class Σ_2^p, such as the *strategic companies* problem ([EFLP00])[12].

The most significant accomplishment in the past 5 years in logic programming and disjunctive logic programming has been the implementation of large scale systems which can handle thousands of rules. Since, as noted in Section 4, most nonmonotonic reasoning systems can be mapped to DLPs, one now has a mechanism to implement nonmonotonic systems. Those systems that have been implemented are: XSB, Smodels, DLV, and DeReS. Although XSB can handle only extended logic programs, it can modify disjunctive theories by writing, for example $a \vee b$ as two clauses, $a \leftarrow not\, b$ and $b \leftarrow not\, a$. In this case, it can obtain disjunctive answers by looking at the minimal models. XSB contains all of the features that are available in PROLOG and extends PROLOG to extended logic programs that can handle the stable model semantics and some aspects of the well–founded. Smodels handles stable model semantics. DLV is a disjunctive system and is being extended to handle large systems. DeReS implements default logic and is being extended for large systems. These systems make it possible to handle large application problems of substantial complexity.

There was a sense that disjunctive theories would not be needed for applications of interest. As noted in Section 6, several recent papers show that this need not be the case. In data integration problems, when one has a resource defined by multiple rules, to determine whether a query can be answered by resource

[12] which is Σ_2^p–complete as a decision problem

rules only, reduces to a disjunctive set of clauses. As noted by Baral and Gelfond [BG94], large classes of knowledge base system applications are formulated using disjunctive clauses. Abductive reasoning, useful in diagnosis, reduces in many instances to handling disjunctive data. In addition, some problems in cognitive robotics need to be represented by disjunctive data. Many of the disjuncts can be disambiguated by testing sensors to determine the disjunctive condition that may apply, and hence reduce the planning problem to a Horn theory. Problems that relate to nonmonotonic reasoning may be transformed to disjunctive logic programs, and hence, solved by the existing implementations. These applications show the relevance of disjunctive logic programming to real world problems.

The field of disjunctive logic programming has made large strides in the past several years so that there are now significant semantics available, implementations of large systems, and a wide range of practical applications that need such systems.

Tribute to Bob Kowalski

We are very pleased to have been invited to contribute a chapter in this collection to honor Bob Kowalski. The field of *logic programming* is deeply indebted to Bob's pioneering work. Bob made many contributions to logic programming. He provided the first formalization of logic programming in terms of Horn clauses, a computable subset of first–order logic. Bob's research provided the theoretical framework for logic programming: an inference mechanism and three different, but equivalent semantics (the semantics were developed with Maarten van Emden). His interaction with Alain Colmerauer during the development of PROLOG led to an efficient implementation and an effective inference mechanism. Kowalski's influential dictum "Algorithm = Logic + Control" provided fundamental direction for increasing clarity and scope in the description of algorithms. He showed the relevance of logic programming in legal reasoning, meta–level and commonsense reasoning, representation of temporal knowledge, verification of integrity constraints for databases and abductive reasoning for medical reasoning.

The first author of this paper, Jack Minker, was introduced to logic programming by listening to Bob Kowalski [Kow74] at the IFIP Congress in Stockholm, Sweden. As Minker stated in several papers, he was skeptical about its effectiveness to compete with conventional languages because of his experience with solving problems using a theorem prover. However, after reading Kowalski's paper and realizing that he was talking about a subset of clausal form, namely Horn clauses, he became convinced that logic programming was a viable way to do programming. It has been Minker's privilege to be both a friend and a colleague of Bob Kowalski.

The second author, Dietmar Seipel, started working with PROLOG in 1987 for implementing certain methods for query evaluation in deductive databases. The dual nature of PROLOG– as a declarative specification–style language (e.g. DATALOG for databases) and as a programming language – and the possibility of rapidly prototyping systems became very appealing to Seipel. In 1992 he became interested in disjunctive deductive databases and non–monotonic reasoning. At the ICLP'93 conference in Budapest for the first time he attended a lecture given by Bob Kowalski, and he was fascinated by the convincing style in which Bob Kowalski presented his ideas. At subsequent ICLP conferences Seipel was also able to get to know Bob Kowalski personally.

Abbreviations

AAAI: American Association for Artificial Intelligence
AMAI: Annals of Mathematics and Artificial Intelligence
CADE: International Conference on Automated Deduction
DOOD: Intl. Conf. on Deductive and Object–Oriented Databases
ICLP: International Conference on Logic Programming
ILPS: International Logic Programming Symposium
IJCAI: Intl. Joint Conf. on Artificial Intelligence
KR: Intl. Conf. on Principles of Knowledge Representation and Reasoning
LPNMR: Intl. Conf. on Logic Programming and Nonmonotonic Reasoning
PODS: ACM Symposium on Principles of Database Systems

References

[AB90] K.R. Apt and H.A. Blair. Arithmetic classification of perfect models of stratified programs. *Fundamenta Informaticae*, XIII: pp. 1–18, 1990. With addendum in vol. XIV: pp. 339–343. 1991.

[AB94] K.R. Apt and R.N. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 19/20: pp. 9–71, 1994.

[ABW88] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In [Min88b], pp. 89–148. 1988.

[ADN97b] C. Aravindan, J. Dix, and I. Niemelä. DisLoP: Towards a Disjunctive Logic Programming System. Proc. LPNMR'97, Springer, LNAI 1265, pp. 342–353, 1997.

[AN78] H. Andreka and I. Nemeti. The generalized completeness of Horn predicate logic as a programming language. *Acta Cybernetica*, 4(1): pp. 3–10, 1978.

[APP98] J.J. Alferes, L.M. Pereira, and T.C. Przymusinski. "Classical" negation in nonmonotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20, pp. 107–142, 1998.

[Apt90] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pp. 493–574. Elsevier, 1990.

[AvE82] K.R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3): pp. 841–862, 1982.

[BD95a] S. Brass and J. Dix. A general approach to bottom–up computation of disjunctive semantics. In *Nonmonotonic Extensions of Logic Programming.* Springer, LNCS 927, pp. 127–155, 1995.

[BD95b] S. Brass and J. Dix. Disjunctive semantics based upon partial and bottom–up evaluation. Proc. ICLP'95, pp. 199–213. MIT Press, 1995.

[BD97] S. Brass and J. Dix. Characterizations of the disjunctive Stable semantics by partial evaluation. *J. of Logic Programming*, 32(3): pp. 207–228, 1997.

[BD98] S. Brass and J. Dix. Characterizations of the disjunctive well-founded semantics: confluent calculi and iterated GCWA. *Journal of Automated Reasoning*, 20(1): pp. 143–165, 1998.

[BD99] S. Brass and J. Dix. Semantics of (disjunctive) logic programs based on partial evaluation. *J. of Logic Prog.*, 38(3): pp. 167–213, 1999.

[BDNP01] S. Brass, J. Dix, I. Niemelae, and T.C. Przymusinski: Comparison of the STATIC and the disjunctive well–founded semantics and their computation. *Theoretical Computer Science*, 251, 2001.

[BDFZ01] S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation–based bottom–up computation of the well–founded model. accepted for publication in *Theory and Practice of Logic Programming*, to appear 2001.

[BDNP98] S. Brass, J. Dix, I. Niemelae, and T.C. Przymusinski: A Comparison of the static and the disjunctive well–founded semantics and its implementation. Proc. KR'98, pp. 74–85. Morgan Kaufmann, 1998.

[BDP96] S. Brass, J. Dix, and T.C. Przymusinski: Super logic programs. Proc. KR'96, pp. 529–541. Morgan Kaufmann, 1996.

[BE96] P.A. Bonatti and T. Eiter. Querying disjunctive databases through nonmonotonic logics. *Theoretical Computer Science*, 160: pp. 321–363, 1996.

[BE98] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Journal of Artificial Intelligence* 109(1–2): pp. 297–356, 1999.

[BED94] R. Ben–Eliyahu and R. Dichter. Propositional semantics for disjunctive logic programs. AMAI 12: pp. 53–87, 1994.

[BEP94] R. Ben–Eliyahu and L. Palopoli. Reasoning with minimal models: efficient algorithms and applications. Proc. KR'94, pp. 39–50, 1994.

[BEPZ96] R. Ben–Eliyahu, L. Palopoli, and V. Zemlyanker. The expressive power of tractable disjunction. In *Proc. of the 12th European Conference on Artificial Intelligence (ECAI'96)*, pp. 345–349, 1996.

[BFL99] F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. Proc. ICLP'99, pp. 79–93, 1999.

[BG94] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20: pp. 73–148, 1994.

[BLR97] F. Buccafurri, N. Leone, and P. Rullo. Strong and weak constraints in disjunctive datalog. Proc. LPNMR'97. Springer, LNAI 1265, pp. 2–17, 1997.

[BNNS94] C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian. Mixed integer programming methods for computing nonmonotonic deductive databases. *Journal of the ACM*, 41(6): pp. 1178–1215, 1994.

[Bry90] F. Bry. Query evaluation in deductive databases: bottom-up and top-down reconciled. Journal of Data & Knowledge Engineering, 5(4): pp. 289–312, 1990.

[Cad92] M. Cadoli. The complexity for model checking for circumscriptive formulae. *Information Processing Letters*, 44: pp. 113–118, 1992.

[Cad93] M. Cadoli. Semantical and computational aspects of Horn approximations. Proc. IJCAI'93, pp. 39–44, 1993.

[Cad96] M. Cadoli. Panel on "Knowledge compilation and approximation": terminology, questions, references. In *Proc. of the 4th Int. Symp. on Artificial Intelligence and Math. (AI/Math'96)*, pp. 183–186, 1996.

[CH82] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer System Sci.*, 25: pp. 99–128, 1982.

[CH85] A. Chandra and D. Harel. Horn clause queries and generalizations. *J. of Logic Programming*, 2(1): pp. 1–15, 1985.

[Cha93] E. Chan. A possible world semantics for disjunctive databases. *IEEE Trans. Data and Knowledge Eng.*, 5(2): pp. 282–292, 1993.

[CL94] M. Cadoli and M. Lenzerini. The complexity of closed world reasoning and circumscription. *J. of Computer System Sciences*, 43: pp. 165–211, 1994.

[Cla78] K. L. Clark. Negation as failure. In [GMN78], pp. 293–322. 1978.

[CMT99] P. Cholewiński, A. Marek, V. W. Mikitiuk, and M. Truszczyński. Computing with default logic. *J. of Art. Intelli.*, 112(1–2): pp. 105–146, 1999.

[Cod79] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4): pp. 397–434, 1979.

[CS90] J. Chomicki and V.S. Subrahmanian. Generalized closed world assumption is π_2^0–complete. *Inf. Processing Letters*, 34: pp. 289–291, 1990.

[CS93] M. Cadoli and M. Schaerf. A survey of complexity results for non–monotonic logics. *Journal of Logic Programming*, 13: pp. 127–160, 1993.

[CW93] W. Chen and D.S. Warren. A goal–oriented approach to computing the well–founded semantics. *J. of Logic Progr.*, 17(2–4): pp. 279–300, 1993.

[Dec91] H. Decker. On the declarative, operational and procedural semantics of disjunctive computational theories. In *Proc. of the Second Intl. Workshop on the Deductive Approach to Inf. Syst. and Databases (DAISD'91)*, 1991.

[DEGV97] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proc. of the 12th IEEE International Conference on Computational Complexity (CCC'97)*, pp. 82–101, 1997.

[DFN99] J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic reasoning: towards efficient calculi and implementations. In A. Voronkov and A. Robinson, editors, *Handbook of Automated Reasoning*. Elsevier–Science–Press, 1999.

[DG84] W.F. Dowling and J.H. Gallier. Linear time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1: pp. 267–284, 1984.

[DGM96] J. Dix, G. Gottlob, and V. Marek. Reducing disjunctive to nondisjunctive semantics by shift operations. *Fundamenta Informaticae*, 28(1,2): pp. 87–100, 1996.

[Dix92a] J. Dix. A framework for representing and characterizing semantics of logic programs. Proc. KR'92, pp. 591–602. Morgan Kaufmann, 1992.

[Dix95] J. Dix. A classification–theory of semantics of normal logic programs: I. Strong properties and II. Weak properties. *Fund. Informaticae*, XXII(3): pp. 227–255 and 257–288, 1995.

[Dix95d] J. Dix. Semantics of logic programs: their intuitions and formal properties. An overview. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, DeGruyter, pp. 241–327. 1995.

[DS98] J. Dix and F. Stolzenburg. A framework to incorporate nonmonotonic reasoning into constraint logic programming. *Journal of Logic Programming*, 37(1,2,3): pp. 47—76, 1998.

[deVa95] A. del Val. An analysis of approximate knowledge compilation. Proc. IJCAI'95, 1995.

[EFLP00] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem solving using the DLV system. In J. Minker, editor, Book on *Logic–Based Artificial Intelligence*, Kluwer, 2000.

[EFLPP00] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete knowledge. in *Proc. of the First International Conference on Computational Logic (CL'2000)*, Springer, LNAI 1861, pp. 807–821, 2000.

[EG93a] T. Eiter and G. Gottlob. Complexity aspects of various semantics for disjunctive databases. Proc. PODS'93, pp. 158–167. ACM Press, 1993.

[EG95] T. Eiter and G. Gottlob. On the computation cost of disjunctive logic programming: Propositional case. AMAI 15(3-4): pp. 289–323, 1995.

[EG97] T. Eiter and G. Gottlob. Expressiveness of stable model semantics for disjunctive logic programs with functions. *Journal of Logic Programming*, 33(2): pp. 167–178, 1997.

[EGM94] T. Eiter, G. Gottlob, and H. Mannila. Adding disjunction to DATALOG. Proc. PODS'94, pp. 267–278, 1994.

[EK89] K. Eshghi, R.A. Kowalski. Abduction compared with Negation by Failure. Proc. ICLP'89, pp. 234–254. MIT Press, 1989.

[ELM$^+$98] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The kr system DLV: Progress report, comparisons, and benchmarks. Proc. KR'98, pp. 406–417, 1998.

[ELS97] T. Eiter, N. Leone, and D. Saccà. On the partial semantics for disjunctive deductive databases. AMAI 19(1–2): pp. 59–96, 1997.

[ELS98] T. Eiter, N. Leone, and D. Saccà. Expressive power and complexity of partial models for disjunctive deductive databases. *Theoretical Computer Science*, 206(1–2): pp. 181–218, 1998.

[FGM96] J.A. Fernández, J. Grant, and J. Minker. Model theoretic approach to view updates in deductive databases. *Journal of Automated Reasoning*, 17(2): pp. 171–197, 1996.

[Fit85] M. Fitting. A Kripke–Kleene semantics for logic programs. *Journal of Logic Programming*, 2: pp. 295–312, 1985.

[Fi91] M. Fitting. Bilattices and the semantics of logic programs, *Journal of Logic Programming*, 11: pp. 91–116, 1991.

[FLMS93] J.A. Fernández, J. Lobo, J. Minker, and V.S. Subrahmanian. Disjunctive LP + integrity constraints = stable model semantics. AMAI 8(3–4): pp. 449–474, 1993.

[FLP99] W. Faber, N. Leone, and G. Pfeifer. Pushing goal derivation in DLP computations. Proc. LPNMR'97, Springer, LNAI 1265, pp. 177–191, 1997.

[FM91a] J.A. Fernández and J. Minker. Computing perfect models of disjunctive stratified databases. In *ILPS'91 Workshop on Disjunctive Logic Programming*, 1991.

[FM91b] J.A. Fernández, J. Minker. Bottom–up evaluation of hierarchical disjunctive deductive databases. Proc. ICLP'91, pp. 660–675. MIT Press, 1991.

[FM92] J.A. Fernández and J. Minker. Semantics of disjunctive deductive databases. In *Proc. of the International Conference on Database Theory (ICDT'92)*, pp. 332–356, 1992. (Invited Paper).

[FM95] J.A. Fernández and J. Minker. Bottom–up computation of perfect models for disjunctive theories. *J. of Logic Programming*, 25(1): pp. 33–51, 1995.

[GMN78] H. Gallaire and J. Minker, editors, Logic and Data Bases, Plenum Press, New York, 1978.

[GHLM93] J. Grant, J. Horty, J. Lobo, and J. Minker. View updates in stratified disjunctive databases. *J. Automated Reasoning*, 11: pp. 249–267, 1993.

[Gin87] M.L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, 1987.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th Intl. Conf. and Symp. on Logic Programming*, pp. 1070–1080, MIT Press, 1988.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. Proc. ICLP'90, pp. 579–597, MIT Press, 1990.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9: pp. 365–385, 1991.

[GM86] J. Grant and J. Minker. Answering queries in indefinite databases and the null value problem. In P. Kanellakis, editor, *Advances in Computing Research: The Theory of Databases*, pp. 247–267. 1986.

[GM00] J. Grant and J. Minker. A logic–based approach to data integration. 2000. Submitted for publication.

[GPP86] M. Gelfond, H. Przymusinska, and T.C. Przymusinski. The extended closed world assumption and its relation to parallel circumscription. Proc. PODS'86, pp. 133–139, 1986.

[GPZ01] F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and expressive power of non–deterministic constructs in deductive databases. *Journal of Computer and System Sciences*, 62(1), pp. 15–42, 2001.

[Gre98] S. Greco. Binding propagation in disjunctive databases. In *Proc. of the Intl. Conf. on Very Large Databases (VLDB'98)*, pp. 287–298, 1998.

[Gre99] S. Greco. Optimization of disjunctive queries. Proc. ICLP'99, pp. 441–455, 1999.

[Gre99b] S. Greco. Minimal founded semantics for disjunctive logic programming. Proc. LPNMR'99, Springer, LNAI 1730, pp. 221–235, 1999.

[HeWa97] H. Herre and G. Wagner. Stable models are generated by a stable chain. *Journal of Logic Programming*, 30(2): pp. 165–177, 1997.

[Hil74] R. Hill. Lush resolution and its completeness. Technical Report DCL Memo 78, Department of Artificial Intelligence, Univ. of Edinburgh, 1974.

[HIOK97] R. Hasegawa, K. Inoue, Y. Ohta, and M. Koshimura. Non–Horn magic sets to incorporate top–down inference into bottom–up theorem proving. Proc. CADE'97, pp. 176–190, 1997.

[IM82] A. Itai and J.A. Makowsky. On the complexity of Herbrand's theorem. Technical report, Dept. of Computer Science, Israel Institute of Technology, Haifa, 1982.

[Imi91] T. Imielinski. Incomplete deductive databases. AMAI 3: pp. 259–293, 1991.

[IS93] K. Inoue and C. Sakama. Transforming abductive logic programs to disjunctive programs. Proc. ICLP'93, pp. 335–353, 1993.

[IV89] T. Imielinski and K. Vadaparty. Complexity of query processing in databases with OR–objects. Proc. PODS'89, pp. 51–65, 1989.

[JLL83] J. Jaffar, J.-L. Lassez, and J.W. Lloyd. Completeness of the Negation as Failure rule. Proc. IJCAI'83, pp. 500–506, 1983.

[JNSY00] T. Janhunen, I. Niemelä, P. Simons, and J.-H. You: Unfolding partiality and disjunctions in stable model semantics. Proc. KR'2000, Morgan Kaufmann, 2000.

[KLM90] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Journal of Artificial Intelligence*, 44(1): pp. 167–207, 1990.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object–oriented and frame–based languages. *J. of the ACM*, 42(4), pp. 741–843, 1995.

[Kow74] R.A. Kowalski. Predicate logic as a programming language. *Proc. of IFIP 4*, pp. 569–574, 1974.

[Kow78] R.A. Kowalski. Logic for data description. In [GMN78], pp. 77–102. 1978.

[Kow88] R.A. Kowalski. The early years of logic programming. *Communications of the ACM*, 31(1): pp. 38–43, 1988.

[KP91] P. Kolaitis and C. Papadimitriou. Why not negation by fixpoint? *Journal of Computer and System Sciences*, 43: pp. 125, 1991.

[KS92] H.A. Kautz and B. Selman. Forming concepts for fast inference. Proc. AAAI'92, pp. 786–793, 1992.

[Lip81] W. Lipski. On databases with incomplete information. volume 28, pp. 41–70. ACM, New York, 1981.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd ed., 1987.

[LM84] J.-L. Lassez and M.J. Maher. Closure and fairness in the semantics of programming logic. *Theoretical Computer Science*, 29: pp. 167–184, 1984.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.

[Lov87] D.W. Loveland. Near–Horn PROLOG. Proc. ICLP'87, pp. 456–459, 1987.

[LRS95] N. Leone, P. Rullo, F. Scarcello. Declarative and fixpoint characterizations of disjunctive stable models. Proc. ILPS'95, pp. 399–413. MIT Press, 1995.

[LRS96] N. Leone, P. Rullo, and F. Scarcello. Stable model checking for disjunctive programs. In *Prof. of Logic in Databases (LID'96)*, pp. 281–294, 1996.

[LRS97] N. Leone, P. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135: pp. 69–112, 1997.

[LRW93] D. Loveland, D. Reed, and D. Wilson. Satchmore: Satchmo with relevancy. Technical report, Duke Univ., Durham, North Carolina, USA, 1993.

[LS90b] K.-C. Liu and R. Sunderraman. On representing indefinite and maybe information in relational databases: A generalization. *Proc. of IEEE Data Engineering*, pp. 495–502, 1990.

[MB88] R. Manthey and F. Bry. Satchmo: A theorem prover implemented in PROLOG. Proc. CADE'88, 1988.

[McCa59] J. McCarthy. Programs with common sense. In *Proc. Teddington Conf. on the Mechanisation of Thought Processes*, pp. 75–91, London, 1959. Her Majesty's Stationery Office.

[McCa80] J. McCarthy. Circumscription – a form of non–monotonic reasoning. *Journal of Artificial Intelligence*, 13(1,2): pp. 27–39, 1980.

[MD80] D. McDermott and J. Doyle. Non–monotonic logic I. *Journal of Artificial Intelligence*, 13: pp. 41–72, 1980.

[Min82] J. Minker. On indefinite databases and the closed world assumption. Proc. CADE'82, Also in: Springer, LNCS 138, pp. 292–308, 1982.

[Min88a] J. Minker. Perspectives in deductive databases. *Journal of Logic Programming*, 5: pp. 33–60, 1988.

[Min88b] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.

[Min89] J. Minker. Toward a foundation of disjunctive logic programming. In *Proc. of the North American Conference on Logic Programming*, pp. 121–125. MIT Press, 1989. Invited Banquet Address.

[Min93] J. Minker. An overview of nonmonotonic reasoning and logic programming. *Journal of Logic Programming*, 17(2, 3 and 4): pp. 95–126, 1993.

[Min94] J. Minker. Overview of disjunctive logic programming. *Journal of Artificial Intelligence & Mathematics*, 12(1-2): pp. 1–24, 1994.

[Min99] J. Minker. Logic and databases: a 20 year retrospective - updated in honor of Ray Reiter. In H.J. Levesque and F. Pirri, editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pp. 234–299. Springer, 1999.

[MNR92] V.W. Marek, A. Nerode, and J.B. Remmel. The stable models of a predicate logic program. In *Proc. of the Joint Intl. Conf. and Symposium on Logic Programming (JICSLP'92)*, pp. 446–460, MIT Press, 1992.

[Moo84] R.C. Moore. Possible–world semantics for autoepistemic logic. In *Proc. of AAAI Workshop on Non–Mon. Reasoning*, pp. 396–401, New Paltz, 1984.

[Moo85] R.C. Moore. Semantical considerations on nonmonotonic logic. *Journal of Artificial Intelligence*, 25(1): pp. 75–94, 1985.

[MR90] J. Minker and A. Rajasekar. A fixpoint semantics for disjunctive logic programs. *Journal of Logic Programming*, 9(1): pp. 45–74, 1990.

[MR94] J. Minker and C. Ruiz. Semantics for disjunctive logic programs with explicit and default negation. *Fundamenta Informaticae*, 20(3/4): pp. 145–192, 1994. Anniversary Issue edited by H. Rasiowa.

[MT91] V.W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3): pp. 588–619, 1991.

[MT93] V.W. Marek and M. Truszczyński. *Nonmonotonic logic: Context–dependent reasoning.* Springer, 1993.

[NS97] I. Niemela and P. Simons. Smodels – an implementation of the stable model and well–founded semantics for normal logic programs. Proc. LPNMR'97. Springer, LNAI 1265, pp. 420–429, 1997.

[OIH98] Y. Ohta, K. Inoue, R. Hasegawa. On the relationship between non–horn magic sets and relevancy testing. Proc. CADE'98, pp. 333–348, 1998.

[Poo89] D. Poole. What the lottery paradox tells us about default reasoning. Proc. KR'89, pp. 333–340, 1989.

[Prz88] T.C. Przymusinski. On the declarative semantics of deductive databases and logic programming. In [Min88b], Chapter 5, pp. 193–216. 1988.

[Prz89] T.C. Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5: pp. 167–205, 1989.

[Prz90a] T.C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In *Proc. of the North Amer. Conf. on Logic Programming*, pp. 40–62, 1990.

[Prz90c] T.C. Przymusinski. Extended stable semantics for normal and disjunctive programs. Proc. ICLP'90, pp. 459–477, MIT Press, 1990.

[Prz91] T.C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9: pp. 401–424, 1991.

[Prz95] T.C. Przymusinski. Static semantics for normal and disjunctive logic programs. AMAI 14 (Festschrift in honor of Jack Minker): pp. 323–357, 1995.

[Rei78] R. Reiter. On closed world data bases. In [GMN78], pp. 55–76. 1978.

[Rei80] R. Reiter. A logic for default reasoning. *Journal of Artificial Intelligence*, 13: pp. 81–132, 1980.

[Rei86] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM*, 33(2): pp. 349–370, 1986.

[Rei87] R. Reiter. Nonmonotonic reasoning. *Annual Reviews of Comp. Sci.*, 1987.

[RL90] David W. Reed and Donald W. Loveland. A comparison of three Prolog extensions. *Journal of Logic Programming*, 12(1&2): pp. 25–50, 1992.

[RLM89] A. Rajasekar, J. Lobo, and J. Minker. Weak generalized closed world assumption. *Journal of Automated Reasonig*, 5(3): pp. 293–307, 1989.

[RLS91] D.W. Reed, D.W. Loveland, and B.T. Smith. An alternative characterization of disjunctive logic programs. In *Proc. of the Intl. Symposium on Logic Programming (ISLP'91)*, pp. 54–68, MIT Press, 1991.

[RM90] A. Rajasekar and J. Minker. On stratified disjunctive programs. AMAI 1(1–4): pp. 339–357, 1990.

[RM97] C. Ruiz and J. Minker. Combining closed world assumptions with stable negation. *Fundamenta Informaticae*, 32(2): pp. 163–181, 1997.

[RM98] C. Ruiz and J. Minker. Logic knowledge bases with two default rules. AMAI 22(3-4): pp. 333–361, 1998.

[Ros89] K. Ross. Well–founded semantics for disjunctive logic programs. Proc. DOOD'89, pp. 352–369, 1989.

[RSS+97] P. Rao, K. Sagonas, T. Swift, D.S. Warren, and J. Friere. XSB: A system for efficiently computing well–founded semantics. Proc. LPNMR'97. Springer, LNAI 1265, pp. 430–440, 1997.

[RT88] K.A. Ross and R.W. Topor. Inferring negative information from disjunctive databases. *J. of Automated Reasoning*, 4(2): pp. 397–424, 1988.

[Sac96] D. Saccà. The expressive power of stable models for bound and unbound DATALOG queries. *J. of Comp. and System Sci.*, 54(3): pp. 441–464, 1997.

[SaIn00] C. Sakama and K. Inoue. Abductive logic programming and disjunctive logic programming: their relationship and transferability, *Journal of Logic Programming*, 44(1–3): pp. 75–100, 2000.

[Sak89] C. Sakama. Possible model semantics for disjunctive databases. Proc. DOOD'89, pp. 337–351, 1989.

[Sch90] L. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyburg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pp. 23–67. Kluwer, 1990.

[Sch95] J.S. Schlipf. Complexity and undecideability results for logic programming. AMAI 15(3-4): pp. 257–288, 1995.

[Sei94] D. Seipel. An efficient computation of the extended generalized closed world assumption by support–for–negation sets. In *Proc. of the International Conference on Logic Programming and Automated Reasoning (LPAR'94)*, Springer, LNAI 822, pp. 245–259, 1994.

[Sei97] D. Seipel. Partial evidential stable models for disjunctive deductive databases. In *Proc. of the Workshop on Logic Prog. and Knowledge Representation LPKR'97 at ILPS'97*, Springer, LNAI 1471, pp. 66–84, 1998.

[Sei99] D. Seipel. Aggregation in disjunctive deductive databases. In *International Conference on Applications of* PROLOG *(INAP'99)*, 1999.

[Sei00] D. Seipel. Clausal deductive databases and a general framework for semantics in disjunctive deductive databases. In *Proc. of the International Symposium on Foundations of Information and Knowledge Systems (FoIKS'2000)*, Springer, LNCS 1762, pp. 241–259, 2000.

[She88] J.C. Shepherdson. Negation in logic prog. In [Min88b], pp. 19–88. 1988.

[SK91] B. Selman and H.A. Kautz. Knowledge compilation using Horn approximations. Proc. AAAI'91, pp. 904–909, 1991.

[SK96] B. Selman and H.A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2): pp. 193–224, 1996.

[SMR97] D. Seipel, J. Minker, and C. Ruiz. Model generation and state generation for disjunctive logic programs, *J. of Logic Prog.*, 32(1): pp. 48–69, 1997.

[SMR97b] D. Seipel, J. Minker, and C. Ruiz. A characterization of partial stable models for disjunctive deductive databases, Proc. ILPS'97, pp. 245–259, MIT Press, 1997.

[SL88] B.T. Smith and D. Loveland. A simple near–Horn PROLOG interpreter. In *Proc. of the 5th Intl. Conf. and Symp. on Logic Progr.*, pp. 794–809, 1988.

[Smu56] R.M. Smullyan. *Bull, AMS62*, 1956. page 600: Elementary formal system (abstract). page 601: On definability by recursion (abstract).

[SS86] L.S. Sterling and E.Y. Shapiro. *The Art of* PROLOG. MIT Press, 1986.

[Sti88] M. Stickel. A PROLOG technology theorem prover: Implementation by an extended PROLOG compiler. *J. of Aut. Reas.*, 4(4): pp. 353–380, 1988.

[Var82] M.Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM Symp. on Theory of Comp. (STOC'82)*, pp. 137–146, 1982.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *J. of the ACM*, 23(4): pp. 733–742, 1976.

[VaGe88] A. Van Gelder. Negation as Failure using tight derivations for general logic programs. In [Min88b], pp. 1149–176. 1988.

[VaGe89] A. Van Gelder. The alternating fixpoint of logic programs with negation. Proc. PODS'89, pp. 1–10, 1989.

[VGRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well–founded semantics for general logic programs. *J. of the ACM*, 38(3): pp. 620–650, 1991.

[War99] D.S. Warren, et al. The XSB programming system. Technical report, State University of New York at Stony Brook, 1999. http://www.cs. sunysb.edu/ sbprolog/xsb-page.html.

[WB93] C. Witteveen and G. Brewka. Skeptical reason maintenance and belief revision, *Journal of Artificial Intelligence*, 61, pp. 1–36, 1993.

[WVH97] C. Witteveen and W. van der Hoek. A general framework for revising nonmonotonic theories, Proc. LPNMR'97, Springer, LNAI 1265, pp. 258–272, 1997.

[Yah97] A.H. Yahya. Generalized query answering in disjunctive deductive databases: Procedural and nonmonotonic aspects. Proc. LPNMR'97. Springer, LNAI 1265, pp. 325–341, 1997.

[Yah00] A.H. Yahya: Minimal model generation for refined answering of generalized queries in disjunctive deductive databases. *Journal of Data and Knowledge Engineering*, 34(3), pp. 219–249, 2000.

[YC89] L.Y. Yuan and D.-A. Chiang. A sound and complete query evaluation algorithm for relational databases with disjunctive information. Proc. PODS'89, pp. 66–74. ACM Press, 1989.

[YH85] A. Yahya and L.J. Henschen. Deduction in non–Horn databases. *Journal of Automated Reasoning*, 1(2): pp. 141–160, 1985.

[YY93] L.Y. Yuan and J.-H. You. Autoepistemic circumscription and logic programming. *Journal of Automated Reasoning*, 10: pp. 143–160, 1993.

[YY94] J.-H. You and L.Y. Yuan. A three–valued semantics for deductive databases and logic programs. *J. of Comp. and System Sci.*, 49: pp. 334–361, 1994.

[YYG00] J.-H. You, L.Y. Yuan, and R. Goebel. An abductive approach to disjunctive logic programming. *J. of Logic Programming*, 44(1–3): pp. 101–128, 2000.

[YWY97] J.-H. You, X. Wang, and L.Y. Yuan. Disjunctive logic programming as constrained inferences. Proc. ICLP'97, pp. 361–375, 1997.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences*, 28: pp. 142–166, 1984.

[ZBF97] U. Zukowski, S. Brass, and B. Freitag. Improving the alternating fixpoint: The transformation approach, Proc. LPNMR'97, Springer, LNAI 1265, pp. 40–59, 1997.

[ZF99] U. Zukowski and B. Freitag. Well–founded semantics by transformation: The non–ground case, Proc. ICLP'99, pp. 456–470, 1999.

[ZW99] C. Zaniolo and H. Wang: Logic–based user–defined aggregates for the next generation of database systems. In K.R. Apt, V.W. Marek, M. Truszczyński, and D.S. Warren, editors, *The Logic Prog. Paradigm: Current Trends and Future Directions*, Springer, pp. 401–426, 1999.