TLP **4** (1 & 2): 75–93, 2004. © Cambridge University Press
DOI: 10.1017/S1471068403001704 Printed in the United Kingdom

Minimal founded semantics for disjunctive logic programs and deductive databases

FILIPPO FURFARO, GIANLUIGI GRECO and SERGIO GRECO
DEIS, Università della Calabria, 87030 Rende, Italy
(*e-mail:* {filippo.furfaro,gianluigi.greco,greco}@deis.unical.it)

Abstract

In this paper, we propose a variant of stable model semantics for disjunctive logic programming and deductive databases. The semantics, called *minimal founded*, generalizes stable model semantics for normal (i.e. non-disjunctive) programs, but differs from disjunctive stable model semantics (the extension of stable model semantics for disjunctive programs). Compared with disjunctive stable model semantics, minimal founded semantics seems to be more intuitive, it gives meaning to programs which are meaningless under stable model semantics and is no harder to compute. More specifically, minimal founded semantics differs from stable model semantics only for disjunctive programs having constraint rules or rules working as constraints. We study the expressive power of the semantics, and show that for general disjunctive datalog programs it has the same power as disjunctive stable model semantics.

KEYWORDS: disjunctive logic programs, disjunctive deductive databases, semantics, minimal models, stable models

1 Introduction

Several different semantics have been proposed for normal and disjunctive logic programs. Stable model semantics, first proposed for normal (i.e. disjunction free) programs, has been subsequently extended to disjunctive programs. For normal programs, stable model semantics has been widely accepted since it captures the intuitive meaning of programs and, for stratified programs it coincides with perfect model semantics which is the standard semantics for this class of programs (Apt *et al*., 1988; Przymusinski, 1988; Przymusinska & Przymusinski, 1988; VanGelder *et al*., 1991). For positive programs, stable model semantics coincides with minimal model semantics which is the standard semantics for positive disjunctive programs.

For general disjunctive programs several semantics have been proposed. We mention here the *generalized closed world assumption* (GCWA) (Minker, 1982), the *weak generalized closed world assumption* (WGCWA) (Rajasekar *et al*., 1989; Lobo *et al*., 1992), the *possible model semantics* (Sakama & Inoue, 1994), the *perfect model semantics* (Przymusinski, 1991), particularly suited to stratified programs, the *disjunctive well-founded semantics* (Ross, 1989), the *disjunctive stable model semantics* (Gelfond & Lifschitz, 1991; Przymusinski, 1991) and the *partial stable model semantics* (Przymusinski, 1991; Eiter *et al*., 1998).

Disjunctive stable model semantics is widely accepted since i) it gives a good intuition of the meaning of programs, ii) for normal programs it coincides with stable model semantics and for stratified (resp. positive) programs it coincides with the perfect (resp. minimal) model semantics. However, disjunctive stable model semantics has some drawbacks. It is defined for a restricted class of programs and there are several reasonable programs which are meaningless, i.e. they do not have stable models.

Motivating examples

The following examples present some programs whose intuitive meaning is not captured by disjunctive stable model semantics.

Example 1
Consider the following simple disjunctive program P_1

$$a \vee b \vee c \leftarrow$$
$$\leftarrow \neg a$$
$$\leftarrow \neg b$$

where the second and third rules are constraints, i.e. rules which are satisfied only if the body is false, which can be rewritten into equivalent normal rules.[1] P_1 has a unique minimal model $M_1 = \{a, b\}$ but M_1 is not stable. □

Thus, under stable model semantics the above program is meaningless. However, the intuitive meaning is captured by the unique minimal model since the constraints force more than one atom to be inferred from the disjunctive rule. The next example presents a real life situation that can be easily modeled by means of a disjunctive program.

Example 2
Consider the Internet structure where every computer in the network makes use of a primary DNS (Domain Name Server) for resolving names associated to IP addresses; moreover if the primary server fails, a secondary (supplementary) DNS is searched. So, an address cannot be resolved if both primary and secondary DNSs are not reachable. An interesting task could be the identification of a minimal set of servers that ensures the connectivity of a set of computers. This task can be formalized by the following disjunctive program:

$$active(D_1) \vee active(D_2) \leftarrow dns(C, D_1, D_2)$$

where $active(D)$ means that D is a working DNS, $dns(C, D_1, D_2)$ means that C is a computer with D_1 and D_2 as primary and secondary DNSs. Assuming that *dns* is a relation of our database, it is easy to see that this program has minimal (stable)

[1] A constraint rule of the form $\leftarrow b_1, \ldots, b_k$ can be rewritten under total semantics (i.e. a two value semantics where every atom is either true or false) as $p(X) \leftarrow b_1, \ldots, b_k, \neg p(X)$ where p is a new predicate symbol and X is the list of all distinct variables appearing in the source rule.

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

models (under the disjunctive stable model semantics) and that each stable model corresponds to the set of working DNSs.

Now suppose that we are looking for a set of active DNSs containing both d_1 and d_2; this situation can be modeled by adding to the program the following constraint:

$$\leftarrow \neg active(d_1)$$
$$\leftarrow \neg active(d_2)$$

Under this hypothesis, if there is a computer c with d_1 and d_2 as primary and secondary DNSs (i.e. there is a fact $dns(c, d_1, d_2)$ in the database), the program has a minimal model containing $active(d_1)$ and $active(d_2)$; but this model is not stable. Thus, under stable model semantics this program is meaningless, even though its intuitive meaning is captured by the minimal model. □

For a better understanding of this problem, consider now the formalization in terms of logic programming of the 3SAT problem.

Example 3
The 3SAT problem in which clauses consist of exactly 3 literals can be expressed by the following three rules:

$$val(X, true) \vee val(X, false) \leftarrow var(X)$$
$$\leftarrow val(X, true), val(X, false)$$
$$val(X, Vx) \vee val(Y, Vy) \vee val(Z, Vz) \leftarrow occur(C, X, Vx), occur(C, Y, Vy), occur(C, Z, Vz)$$

The first two rules state that the value of each literal must be either *true* or *false*. In the third rule a predicate $occur(C, X, Vx)$ checks if the literal X occurs in the clause C; the value of Vx is *true* (resp. *false*) if X occurs positively (resp. negatively) in C. The set of clauses is described by means of the database predicate *occur*. For instance, the clause $c_1 = x_1 \vee x_2 \vee \neg x_3$ is defined by the three facts $occur(c_1, x_1, true)$, $occur(c_1, x_2, true)$ and $occur(c_1, x_3, false)$. For the sake of simplicity, we are assuming that all clauses consist of exactly three literals. Thus, the third rule above states that for each clause, at least one of its literals must be satisfied.

The above program, for an assigned set of input clauses, has a number of models corresponding to all the truth assignments that satisfy all the clauses; so asking for one model is equivalent to solving the 3SAT problem.

Now suppose that one wants to find a solution in which two variables x_1 and x_2 are both true: this situation is modeled as usual by means of the following two constraints:

$$\leftarrow \neg val(x_1, true)$$
$$\leftarrow \neg val(x_2, true)$$

If there is no clause in which both x_1 and x_2 appear positively, the program still solves the 3SAT problem with constraint; but if there is such a clause then the program has no minimal stable model because the constraint forces more than one atom to be inferred from a disjunctive rule, and the minimal model becomes not stable. □

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

Observe that the first two clauses in the program of the above example can be rewritten into the following normal rules:

$$val(X, true) \leftarrow var(X),\ \neg val(X, false)$$
$$val(X, false) \leftarrow var(X),\ \neg val(X, true)$$

since they are used to define a partition of the relation *var* and the constraint defined by the second rule is used to force exclusive disjunction. Observe also that the constraints $\leftarrow \neg val(x_1, true)$ and $\leftarrow \neg val(x_2, true)$ are used to infer, if possible, the atoms $val(x_1, true)$ and $val(x_2, true)$. These constraints cannot be replaced by the two facts $val(x_1, true) \leftarrow$ and $val(x_2, true) \leftarrow$ since by doing so we assert that x_1 and x_2 are true whereas the constraints are used to force the semantics to infer, if possible, that x_1 and x_2 are true.

Intuitively, the problem with stable model semantics is that in some cases the inclusive disjunction is interpreted as exclusive disjunction. This is an old problem first noticed in (Ross & Topor, 1988) who proposed an alternative rule, called disjunctive database rule (DDR), to infer negative information. DDR is equivalent to the weak generalized closed world assumption (Rajasekar *et al.*, 1989), an extension of the generalized closed world assumption proposed in (Minker, 1982).

In this paper, we try to conjugate minimality of models and inclusive disjunction by presenting a new semantics, called *minimal founded*, which overcomes some drawbacks of disjunctive stable model semantics and gives meaning to a larger class of programs by interpreting disjunction in a more liberal way.

Contributions

The main contributions of the paper are the following:

- We introduce a semantics for disjunctive programs. The proposed semantics seems to be more intuitive than stable model semantics and it gives meaning to programs which are meaningless under disjunctive stable model semantics.
- We show that the new semantics coincides with disjunctive stable model semantics for normal and positive programs.
- We formally define the expressive power and complexity of the proposed semantics for datalog programs and we show that it has the same expressive power and complexity of disjunctive stable model semantics.

As a consequence, the proposed semantics differs from stable model semantics only for programs containing both disjunctive rules and negation.

Although the full expressive power of disjunctive datalog can be reached by only considering stratified programs, the natural way to express NP problems and problems in the second level of the polynomial hierarchy (Σ_p^2 and Π_p^2 problems) is to use the *guess-and-check* technique, where the *guess* part is expressed by means of disjunctive rules and the *check* part is expressed by means of constraints (i.e. unstratified rules) (Eiter *et al.*, 1998). However, as shown by the previous examples, there are several interesting programs whose intuitive semantics is not captured by

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

stable models. Thus, the problem of defining an intuitive semantics for disjunctive datalog is still an interesting topic.

We point out that the aim of this paper is not the introduction of a more powerful semantics but only the definition of a semantics which gives an intuitive meaning to a larger class of programs. In the same way, disjunctive stable models do not increase the expressive power of stratified disjunctive datalog under the perfect model semantics, but just give semantics to a larger class of programs.

Organization of the paper

The rest of the paper is organized as follows. Section 2 presents preliminaries on disjunctive datalog, minimal and stable model semantics. Section 3 introduces the *minimal founded* semantics. Its relation with minimal model semantics and stable model semantics is investigated. Section 4 presents results on the expressive power and complexity of minimal founded semantics. Finally, section 5 presents our conclusions.

2 Preliminaries

A *(disjunctive datalog) rule* r is a clause of the form

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n, \qquad k + m + n > 0.$$

where $A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n$ are atoms of the form $p(t_1, \ldots, t_h)$, p is a *predicate* of arity h and the terms $t_1, \ldots, t_h$ are either constants or variables. The disjunction $A_1 \vee \cdots \vee A_k$ is the *head* of r, while the conjunction $B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$ is the *body* of r. Moreover, if $k = 1$ we say that the rule is normal, i.e. not disjunctive.

We denote by $Head(r)$ the set $\{A_1, \ldots, A_k\}$ of the head atoms, and by $Body(r)$ the set $\{B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n\}$ of the body literals. We often use upper-case letters, for example L, to denote literals. As usual, a literal is an atom A or a negated atom $\neg A$; in the former case, it is positive, and in the latter negative. Two literals L_1 and L_2 are *complementary* if $L_1 = A$ and $L_2 = \neg A$, for some atom A. For a literal L, $\neg L$ denotes its complementary literal, and for a set S of literals, $\neg S = \{\neg L \mid L \in S\}$. Moreover, $Body^+(r)$ and $Body^-(r)$ denote the set of positive and negative literals occurring in $Body(r)$, respectively.

A *(disjunctive) logic program* is a finite set of rules. A $\neg$-free (resp. $\vee$-free) program is called *positive* (resp. *normal*). A term, (resp. an atom, a literal, a rule or a program) is *ground* if no variables occur in it. In the following we also assume the existence of rules with empty head, called *denials*, which define constraints[2], i.e. rules which are satisfied only if the body is false.

The *Herbrand Universe* U_P of a program P is the set of all constants appearing in P, and its *Herbrand Base* B_P is the set of all ground atoms constructed from the predicates appearing in P and the constants from U_P. A rule r' is a *ground*

[2] Under total semantics.

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

instance of a rule r, if r' is obtained from r by replacing every variable in r with some constant in U_P. We denote by $ground(P)$ the set of all ground instances of the rules in P.

Given a program P and two predicate symbols (resp. ground atoms) p and q, we write $p \rightarrow q$ if there exists a rule where q occurs in the head and p in the body or there exists a predicate (resp. ground atom) s such that $p \rightarrow s$ and $s \rightarrow q$. If $p \rightarrow q$ then we say that q *depends on* p; also we say that q *depends on* any rule where p occurs in the head. A predicate (resp. ground atom) p is said to be recursive if $p \rightarrow p$.

An interpretation of P is any subset of B_P. The value of a ground atom L w.r.t. an interpretation I, $value_I(L)$, is *true* if $L \in I$ and *false* otherwise. The value of a ground negated literal $\neg L$ is $\neg value_I(L)$. The truth value of a conjunction of ground literals $C = L_1, \ldots, L_n$ is the minimum over the values of the L_i, i.e., $value_I(C) = min(\{value_I(L_i) \mid 1 \leqslant i \leqslant n\})$, while the value $value_I(D)$ of a disjunction $D = L_1 \vee \cdots \vee L_n$ is their maximum, i.e., $value_I(D) = max(\{value_I(L_i) \mid 1 \leqslant i \leqslant n\})$; if $n = 0$, then $value_I(C) = true$ and $value_I(D) = false$. Finally, a ground rule r is *satisfied* by I if $value_I(Head(r)) \geqslant value_I(Body(r))$. Thus, a rule r with empty body is satisfied by I if $value_I(Head(r)) = true$ whereas a rule r' with empty head is satisfied by I if $value_I(Body(r')) = false$. An interpretation M for P is a model of P if M satisfies each rule in $ground(P)$. The set of all models of P will be denoted by $\mathcal{M}(P)$.

Minker (1982) proposed a model-theoretic semantics for a positive program P, which assigns to P the set of its *minimal models* $\mathcal{MM}(P)$, where a model M for P is minimal, if no proper subset of M is a model for P. Accordingly, the program $P = \{a \vee b \leftarrow\}$ has the two minimal models $\{a\}$ and $\{b\}$, i.e. $\mathcal{MM}(P) = \{\{a\}, \{b\}\}$. The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation (Gelfond & Lifschitz, 1991; Przymusinski, 1991). Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs (Gelfond & Lifschitz, 1988).

Definition 1

Let P be a logic program and let I be an interpretation for P, $\frac{P}{I}$ denotes the ground positive program derived from $ground(P)$

1. by removing all rules that contain a negative literal $\neg a$ in the body and $a \in I$, and
2. by removing all negative literals from the remaining rules.

An interpretation M is a (disjunctive) stable model of P if and only if $M \in \mathcal{MM}(\frac{P}{M})$. □

For general P, the stable model semantics assigns to P the set $\mathcal{SM}(P)$ of its *stable models*. It is well known that stable models are minimal models (i.e. $\mathcal{SM}(P) \subseteq \mathcal{MM}(P)$) and that for negation-free programs minimal and stable model semantics coincide (i.e. $\mathcal{SM}(P) = \mathcal{MM}(P)$).

An extension of the perfect model semantics for stratified datalog programs to disjunctive programs has been proposed in (Przymusinski, 1991).

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

A disjunctive datalog program P is said to be *locally stratified* if there exists a decomposition $S_1, \ldots, S_\omega$ of the Herbrand base such that for every (ground instance of a) clause

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$$

in P, there exists an l, called level of the clause, so that:

1. $\forall i \leqslant k$ $stratum(A_i) = l$,
2. $\forall i \leqslant m$ $stratum(B_i) \leqslant l$, and
3. $\forall i \leqslant n$ $stratum(C_i) < l$.

where $stratum(A) = i$ iff $A \in S_i$.

The set of clauses in $ground(P)$ having level i (resp. $\leqslant i$) is denoted by P_i (resp. P_i^*). Any decomposition of the ground instantiation of a program P is called local stratification of P.

The preference order on the models of P is defined as follows: $M \prec N$ iff $M \neq N$ and for each $a \in M - N$ there exists a $b \in N - M$ such that $stratum(a) > stratum(b)$. Intuitively, $stratum(a) > stratum(b)$ means that a has higher priority than b.

Definition 2
Let P be a locally stratified disjunctive datalog program. A model M for P is *perfect* if there is no model N such that $N \prec M$. The collection of all perfect models of P is denoted by $\mathscr{PM}(P)$.

Consider, for instance, the program consisting of the clause $a \vee b \leftarrow \neg c$. The minimal models are $M_1 = \{a\}$, $M_2 = \{b\}$ and $M_3 = \{c\}$. Since $stratum(a) > stratum(c)$ and $stratum(b) > stratum(c)$, we have that $M_1 \prec M_3$ and $M_2 \prec M_3$. Therefore, only M_1 and M_2 are perfect models.

Notice that $M \subset N$ implies $M \prec N$; thus, for locally stratified P, $\mathscr{PM}(P) \subseteq \mathscr{MM}(P)$. For positive P, $\mathscr{MM}(P) = \mathscr{PM}(P)$ and for stratified P, $\mathscr{PM}(P) = \mathscr{SM}(P) \subseteq \mathscr{MM}(P)$. The computation of the perfect model semantics of a program P can be done by considering a decomposition $(P_1, \ldots, P_\omega)$ of $ground(P)$ and computing the minimal models of all subprograms, one at time, following the linear order (Fernandez & Minker, 1991; Greco, 1998; Greco, 1999). In the decomposition $(P_1, \ldots, P_\omega)$, for each P_i and for each rule r of P_i, if $A \in Head(r)$ and $B \in Body^+(r)$ (resp. $B \in Body^-(r)$) then B does not appear in the head of any rule of P_j with $j > i$ (resp. $j \geqslant i$).

3 Minimal founded semantics

In this section we introduce a new semantics for disjunctive programs.

Definition 3
Let P be a positive disjunctive program and let M be an interpretation. Then,

$$S_P(M) = \{a \in B_P \mid \exists r \in ground(P) \wedge a \in Head(r) \wedge Body(r) \subseteq M\}$$

$S_P^\omega(\emptyset)$ denotes the least fixpoint of the operator S_P.

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

The operator S_P extends the classical immediate consequence operator T_P to disjunctive programs by replacing head disjunctions with conjunctions. It is obvious that the operator S_P, for positive P, is monotonic and continuous and, therefore, it admits a least fixpoint.

Definition 4 (*Minimal Founded Semantics*)
Let P be a disjunctive program and let M be a model for P. Then, M is a *founded* model if it is contained in $S^{\omega}_{\frac{P}{M}}(\emptyset)$. M is said to be *minimal founded* if it is a minimal model of P and it is also founded. The collection of all minimal founded models of P is denoted by $\mathscr{MF}(P)$.

For any program P, the set of founded models of P will be denoted by $\mathscr{F}(P)$.

Example 4
The program P_1 of Example 1 has a unique minimal model $M_1 = \{a, b\}$ which is also founded since it is the fixpoint of $S_{\frac{P_1}{M_1}}$. Observe that the interpretation $N_1 = \{a, b, c\}$ is a founded model for P_1 but it is not minimal since $M_1 \subset N_1$. □

Fact 1
Let P be a disjunctive datalog program. Then, $\mathscr{MF}(P) \subseteq \mathscr{MM}(P)$.

Proof
By definition of minimal founded model. □

The following example presents a disjunctive program where stable and minimal founded semantics coincide.

Example 5
Consider the following simple disjunctive program P_5

$$a \vee b \vee c \leftarrow$$
$$a \leftarrow \neg b, \neg c$$
$$b \leftarrow \neg a$$
$$c \leftarrow \neg a$$

This program has two stable models $M_5 = \{a\}$ and $N_5 = \{b, c\}$ which are also minimal founded. □

Moreover, for general programs containing both disjunction and negation, stable and minimal founded semantics do not coincide. The relation between the two semantics is given by the following result.

Theorem 1
Let P be a disjunctive program. Then, $\mathscr{SM}(P) \subseteq \mathscr{MF}(P)$.

Proof
Since stable models are minimal models, we have to show that any stable model M of P is founded, i.e. $M \subseteq S^{\omega}_{\frac{P}{M}}(\emptyset)$. Since $\frac{P}{M}$ is negation-free, every minimal model of $\frac{P}{M}$ is contained in $S^{\omega}_{\frac{P}{M}}(\emptyset)$. Thus, M is founded and, consequently, $\mathscr{SM}(P) \subseteq \mathscr{MF}(P)$.
□

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

Therefore, for every disjunctive program P, $\mathcal{SM}(P) \subseteq \mathcal{MF}(P) \subseteq \mathcal{MM}(P)$. Moreover, as shown by the previous examples, there are programs where the containment is strict, i.e. there are programs, such as the ones presented in the Introduction, having minimal founded models which are not stable.

Corollary 1
Let P be a positive disjunctive datalog program. Then, $\mathcal{MM}(P) = \mathcal{MF}(P)$.

Proof
From Theorem 1 $\mathcal{SM}(P) \subseteq \mathcal{MF}(P)$. Moreover, by definition $\mathcal{MF}(P) \subseteq \mathcal{MM}(P)$. Since for positive programs $\mathcal{SM}(P) = \mathcal{MM}(P)$, we conclude that $\mathcal{MF}(P) = \mathcal{MM}(P)$. □

The following result states that for disjunction-free programs, stable model semantics and minimal founded semantics coincide.

Proposition 1
Let P be a normal datalog program. Then, $\mathcal{SM}(P) = \mathcal{MF}(P)$.

Proof
Generally, $\mathcal{SM}(P) \subseteq \mathcal{MM}(P)$. Thus we have to show that every minimal founded model is also stable. Since for every normal program P and any interpretation M of P, the operators $T_{\frac{P}{M}}$ and $S_{\frac{P}{M}}$ coincide, we have that every minimal founded model M of P is equal to $T^{\omega}_{\frac{P}{M}}(\emptyset)$. □

The following example presents another case of a program which is meaningless under stable model semantics but has minimal founded models.

Example 6
Consider the program P_6

$$a \vee b \vee c \leftarrow$$
$$a \leftarrow \neg b$$
$$b \leftarrow \neg c$$
$$c \leftarrow \neg a$$

From the first rule we have that a subset of $\{a, b, c\}$ must be selected whereas the last three rules state that at least two atoms among a, b and c must be true. The program has three minimal founded models, $M_6 = \{a, b\}$, $N_6 = \{b, c\}$ and $H_6 = \{a, c\}$, but none of them is stable. □

It is worth noting that a disjunctive program P may have no, one or several minimal founded models. In the previous examples, we have presented programs which are meaningless under the stable model semantics, but have minimal founded models (those presented in the Introduction), and a program where stable and minimal founded semantics coincide. The following example presents a program which has stable models, but the stable and minimal founded semantics do not coincide.

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

Example 7
Consider the program P_7

$$\begin{array}{lll} eat \vee drink & \leftarrow & \\ eat & \leftarrow & \\ thirsty & \leftarrow & \neg drink \end{array}$$

This program has two minimal founded models $M_7 = \{eat, thirsty\}$ and $N_7 = \{eat, drink\}$, but only M_7 is stable. □

We now introduce a different characterization of the minimal founded semantics which permits us to better understand the relationship between stable and minimal founded semantics.

Definition 5
Let P be a disjunctive program and let M be an interpretation. Then, P^M denotes the program derived from $ground(P)$ by deleting for each rule

$$r : A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$$

every $A_i \notin M$.

Proposition 2
Let P be a disjunctive program and let M be an interpretation. Then $M \in \mathcal{MF}(P)$ if and only if $M \in \mathcal{MF}(P^M)$.

Proof
M is a minimal founded model of P iff it is a minimal founded model of $P' = ground(P)$. M is a minimal model for P' if and only if it is a minimal model of P^M because P^M is obtained by deleting head atoms which are false in M from P'. Moreover, if an atom can be inferred in $\frac{P'}{M}$ it can also be inferred in $\frac{P^M}{M}$ and vice-versa, i.e. $\mathcal{F}(\frac{P'}{M}) = \mathcal{F}(\frac{P^M}{M})$. Therefore, M is a minimal founded model for P' iff it is a minimal founded model for P^M. □

Observe that the program P^M consists of standard rules whose head is not empty and denials (rules with empty head). Thus, in the following we shall denote with P_S^M the set of standard rules of P^M whose head is not empty and with P_D^M the set of denial rules of P^M.

Theorem 2
Let P be a disjunctive datalog program and M a minimal model for P. Then, $M \in \mathcal{MF}(P)$ if and only $M \in \mathcal{F}(P_S^M)$ and $M \models P_D^M$.

Proof
Clearly, M is a minimal founded model for P iff it is a minimal founded model for $P' = ground(P)$.

We first prove that $M \in \mathcal{MF}(P')$ implies that $M \in \mathcal{F}(P_S^M)$ and $M \models P_D^M$. Let P'' be the subset of rules in P' from which the rules in P_D^M are derived. Every denial $r : \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$, derived from a rule $r'' : A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$, is satisfied in M if and only if r'' is also satisfied in M

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

because all atoms $A_1, \ldots, A_k$ are false in M. As $P_S^M = P' - P''$, if M is a (minimal) founded model for P' it is also a founded model for P_S^M since from the rules in P'' it is not possible to infer any atom.

We now prove that if M is a minimal model of P such that $M \in \mathscr{F}(P_S^M)$, then $M \in \mathscr{MF}(P')$. As $P_S^M \subseteq P'$, if M is a founded model for P_S^M and is a minimal model for P' it will be a minimal founded model for P'. It is obvious that if M is a minimal model of P' every rule of P' is satisfied. □

It is important to note that in the ground program there are rules which with respect to a given model act as constraints forcing atoms to be true or false. In the following example we reconsider the program P_6 of Example 6 containing rules which force the selection of two atoms from the disjunctive rule.

Example 8
The program P_6 of Example 6 admits three minimal founded models: $M_6 = \{a, b\}$, $H_6 = \{a, c\}$ and $N_6 = \{b, c\}$. The program $P^{M_6} = (P_S^{M_6}, P_D^{M_6})$ is

$$a \vee b \leftarrow$$
$$b \leftarrow$$
$$\leftarrow \neg a$$

where $P_S^{M_6}$ consists of the first two rules and $P_D^{M_6}$ contains the last rule. The only minimal model for P^{M_6} is M_6; this model satisfies $P_D^{M_6}$ and is a founded model of $P_S^{M_6}$.

As the program P_6 is symmetric, we have that also H_6 and N_6 are minimal founded model of P_6. □

Theorem 2 shows the difference between minimal founded and stable model semantics. In particular, given a program P and a minimal model M for P, M is stable if M is a minimal model of $\frac{P_S^M}{M}$ and M satisfies P_D^M whereas M is a minimal founded model if M is a model of $\frac{P_S^M}{M}$ and M satisfies P_D^M. Thus, the main difference between the two semantics is that the stable model semantics asks for minimal models of $ground(P)$ which satisfy the constraints P_D^M and are also minimal for the subset of standard rules P_S^M, whereas the minimal founded model asks for minimal models of $ground(P)$ which satisfy the constraints P_D^M and are founded, i.e. their atoms are derivable from the rules in P_S^M.

It is worth noting that the above result can be very useful in the computation of the semantics of programs. Indeed, during the computation of a model, from the assumption of the falsity of atoms we derive constraints which further restrict the search strategy (Leone *et al.*, 1997; Eiter *et al.*, 1998).

4 Expressive power and complexity

In this section we present some results about the expressive power and the data complexity of minimal founded semantics for disjunctive datalog programs (Eiter *et al.*, 1997; Eiter *et al.*, 1998; Saccà, 1997). We first introduce some preliminary definitions and notation, and then present our results.

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

Predicate symbols are partitioned into the two sets of *base* (*EDB*) and *derived* (*IDB*) predicates. Base predicate symbols correspond to database relations on a countable domain U and do not occur in the rule heads. Derived predicate symbols appear in the head of rules. Possible constants in a program are taken from the domain U.

A program P has associated a relational database scheme $DS_P = \{r |\ r$ is an EDB predicate symbol of $P\}$, thus EDB predicate symbols are seen as relation symbols. A database D on DS_P is a set of finite relations, one for each r in DS_P, denoted by $D(r)$. The set of all databases on DS_P is denoted by $\mathbf{D}_P$.

Given a database $D \in \mathbf{D}_P$, P_D denotes the following logic program:

$$P_D = P \cup \{r(t) \leftarrow\ \mid r \in DS_P \wedge t \in D(r)\}.$$

The Herbrand universe U_{P_D} is a finite subset of U and consists of all constants occurring in P or in D (*active domain*). If D is empty and no constant occurs in P, then U_{P_D} is assumed to be equal to $\{a\}$, where a is an arbitrary constant in U.

Definition 6
A *bound query* Q is a pair $\langle P, g\rangle$, where P is a disjunctive program and g is a ground literal (the *query goal*).

We use XF as generic notation for a generic semantics. The result of a query $Q = \langle P, g\rangle$ on an input database D is defined in terms of the XF models of P_D, by taking either the union of all models (*brave or possible inference*, $\exists_{XF}$) or the intersection (*cautious or certain inference*, $\forall_{XF}$).

Definition 7
Given a program P and a database D, a ground atom g is true, under the brave version of the XF semantics, if there exists an XF model M for P_D such that $g \in M$. Analogously, g is true, under the cautious version of the XF semantics, if g is true in every XF model. The set of all queries is denoted by $\mathbf{Q}$.

Definition 8
Let $Q = \langle P, g\rangle$ be a bound query. Then the *database collection* of Q w.r.t. the set of XF models is:

(a) *under the brave version of semantics*, the set of all databases D in $\mathbf{D}_P$ such that g is *true* in P_D under the brave version of the XF semantics; this set is denoted by $EXP^{\exists}_{XF}(Q)$;
(b) *under the cautious version of semantics*, the set of all databases D in $\mathbf{D}_P$ such that g is *true* in P_D under the cautious version of the XF semantics; this set is denoted by $EXP^{\forall}_{XF}(Q)$.

The *expressive power* of a given version (either brave or cautious) of the XF semantics is given by the family of the database collections of all possible queries, i.e. $EXP^{\exists}_{XF}[\mathbf{Q}] = \{EXP^{\exists}_{XF}(Q) | Q \in \mathbf{Q}\}$ and $EXP^{\forall}_{XF}[\mathbf{Q}] = \{EXP^{\forall}_{XF}(Q) | Q \in \mathbf{Q}\}$. □

The database collection of every query is indeed a generic set of databases. A set $\mathbf{D}$ of databases on a database scheme DS with domain U is (W-)*generic* if there exists a finite subset W of U such that for any D in $\mathbf{D}$ and for any isomorphism

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

θ on relations extending a permutation on $U - W$, $\theta(D)$ is in $\mathbf{D}$ as well (Chandra & Harel, 1982; Abiteboul *et al.*, 1994) – informally, all constants not in W are not interpreted, and relationships among them are only those explicitly provided by the databases. Note that for a query $Q = \langle P, g \rangle$, W consists of all constants occurring in P and in g. From now on, any generic set of databases will be called a *database collection.*

Following the *data complexity* approach of Chandra and Harel (1982) and Vardi (1982), for which the query is assumed to be a constant while the database is the input variable, the expressive power coincides with the complexity class of the problem of recognizing the database collection of each query. The expressive power of each semantics will be compared with database complexity classes, defined as follows. Given a Turing machine complexity class C (for instance P or NP), a relational database scheme DS, and a database collection $\mathbf{D}$ on DS, $\mathbf{D}$ is *C-recognizable* if the problem of deciding whether D is in $\mathbf{D}$ is in C. The *database complexity class DB-C* is the family of all C-recognizable database collections (for instance, *DB-P* is the family of all database collections that are recognizable in polynomial time). If the expressive power of a given semantics coincides with a complexity class *DB-C*, we say that the given semantics captures (or expresses all queries in) *DB-C*.

Recall that the classes Σ_k^P, Π_k^P of the polynomial hierarchy (Stockmeyer, 1976) are defined by $\Sigma_0^P = \mathrm{P}$, $\Sigma_{i+1}^P = \mathrm{NP}^{\Sigma_i^P}$, and $\Pi_i^P = \text{co-}\Sigma_i^P$, for all $i \geqslant 0$. In particular, $\Pi_0^P = \mathrm{P}$, $\Sigma_1^P = \mathrm{NP}$, and $\Pi_1^P = \text{co-NP}$. Using Fagin's Theorem (Fagin, 1974) and its generalization in Stockmeyer (1976), complexity and second-order definability are linked as follows.

Fact 2
(Fagin, 1974; Stockmeyer, 1976) A database collection $\mathbf{D}$ over a scheme DS is in $DB\text{-}\Sigma_k^P$ (resp. $DB\text{-}\Pi_k^P$), $k \geqslant 1$, iff it is definable by a second-order formula $(\exists A_1)(\forall A_2)\cdots(Q_k A_k)\Phi$ (resp. $(\forall A_1)(\exists A_2)\cdots(Q_k A_k)\Phi$) on DS, where the A_i are lists of predicate variables preceded by alternating quantifiers and Φ is first-order.

The following example shows how a NP problem can be expressed by means of a second order formula and how the formula can be translated into a disjunctive datalog program under minimal founded or stable model semantics.

Example 9
Consider the *graph kernel* problem defined as: *given a directed graph* $G = \langle V, E \rangle$, *does there exist a kernel for* G, i.e. is there a set $S \subseteq V$ of vertices such that both (i) for each i in $V - S$, there exists j in S for which the edge (j, i) is in E, and (ii) for each i, j in S, (i, j) is not in E?

We denote the set of all (finite) directed graphs with $\mathbf{D}_G$, the set of all graphs in $\mathbf{D}_G$ for which a kernel exists with $\mathbf{D}_G^K$, and $\overline{\mathbf{D}}_G^K = \mathbf{D}_G - \mathbf{D}_G^K$. Any graph is represented by a database on the database scheme $BD = \{V, E\}$, where V and E store its vertices and edges, respectively.

Consider the following second-order formula over BD:

$$\exists S\, \forall x\, \{[\neg S(x) \wedge \exists y(S(y) \wedge E(y, x))] \vee [S(x) \wedge \forall y(S(y) \Rightarrow \neg E(y, x))]\}$$

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

Note that V supplies the interpretation domain of the formula. It is easy to see that a graph G is in $\mathbf{D}_G^K$ iff the formula is satisfied by G. The above formula can be rewritten in the following equivalent *Skolem normal format* for existential second order formulas:

$$\exists S\, \forall x_1, x_2\, \exists y \{ [\neg S(x_1) \wedge S(y) \wedge E(y, x_1)] \vee [S(x_1) \wedge \neg S(x_2)] \vee [S(x_1) \wedge S(x_2) \wedge \neg E(x_2, x_1)] \}$$

This formula is then used to construct the following datalog program:

$$\begin{aligned}
&\mathtt{r}_1 : \ \mathtt{s(W)} \vee \hat{\mathtt{s}}\mathtt{(W)} \leftarrow \\
&\mathtt{r}_2 : \leftarrow \mathtt{s(W)},\ \hat{\mathtt{s}}\mathtt{(W)}. \\
&\mathtt{r}_3 : \ \mathtt{q(X_1, X_2)} \leftarrow \hat{\mathtt{s}}\mathtt{(X_1)},\ \mathtt{s(Y)},\ \mathtt{e(Y, X_1)}. \\
&\mathtt{r}_4 : \ \mathtt{q(X_1, X_2)} \leftarrow \mathtt{s(X_1)},\ \hat{\mathtt{s}}\mathtt{(X_2)}. \\
&\mathtt{r}_5 : \ \mathtt{q(X_1, X_2)} \leftarrow \mathtt{s(X_1)},\ \mathtt{s(X_2)},\ \neg\mathtt{e(X_2, X_1)}. \\
&\mathtt{r}_6 : \ \mathtt{g} \leftarrow \neg\mathtt{q(X_1, X_2)}.
\end{aligned}$$

where v and e are EDB predicate symbols and s and $\hat{s}$ are used to define a partition of the database domain (the Herbrand universe). Note that the rules (r_3)-(r_5) implement the three conjunctions in the above Skolem normal form formula.

Let $G = \langle V, E \rangle$ be a directed graph. A minimal founded (or stable model) is constructed as follows. The first two rules non-deterministically select two disjoint subsets of V, say S and $\hat{S}$, respectively. For each x_1 in $\hat{S}$, if there exists a vertex y in S for which (y, x_1) is in G (i.e. x_1 is connected to some vertex in S) then the third rule makes $q(x_1, x_2)$ true for every x_2 in V. The fourth rule makes $q(x_1, x_2)$ true for each x_1 in S and for each x_2 in $\hat{S}$, and the fifth rule makes $q(x_1, x_2)$ true if both x_1 and x_2 are in S and the edge from x_2 to x_1 is not in G. Note that $q(x_1, x_2)$ is derived to be true for every x_1, x_2 in V iff S and $\hat{S}$ cover V and S is a kernel. But g is false iff for every x_1, x_2 in V, $q(x_1, x_2)$ is true; so g is false iff S and $\hat{S}$ cover V and S is a kernel.

For a graph for which a kernel exists, g may be either true or false. Moreover there exists at least one stable model which selects a kernel and, therefore, makes g false. For a graph without kernels, g is always true in every stable model. □

It is well known that, under total stable model semantics, disjunctive datalog captures the complexity classes Σ_2^P and Π_2^P, respectively, under brave and cautious semantics (Eiter *et al.*, 1997), whereas plain datalog (i.e. datalog with negation and without disjunction) captures the complexity classes NP and *co*NP, respectively, under brave and cautious semantics (Marek & Truszczyński, 1991; Schlipf, 1995).

We now present some results on the expressive power and data complexity of the minimal founded semantics.

Theorem 3
Given a disjunctive program P, a database D on DS_P, and an interpretation M for P_D, deciding whether M is a minimal founded model for P_D is *co*NP-complete.

Proof
Let M be an interpretation and consider the complementary problem $\overline{\Pi}$: is it true that M is not a strongly founded model? $\overline{\Pi}$ is in NP since we can guess an

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

interpretation N and verify in polynomial time that either (i) M is not a founded model for P_D or (ii) N is a model for P_D and $N \subset M$. Hence the problem Π is in *co*NP.

Moreover, deciding whether an interpretation M for a positive disjunctive program P_D is a minimal model is *co*NP-complete. Since for positive programs minimal models are also founded, then, deciding whether M is minimal founded is *co*NP-hard. Therefore, deciding whether M is a minimal founded model for P_D is *co*NP-complete. □

Observe that, deciding whether an interpretation M is a stable model for P_D is also *co*NP-complete.

Theorem 4
$EXP^{\forall}_{\mathcal{MF}}[\mathbf{Q}] = DB\text{-}\Pi^P_2$.

Proof
We first prove that for any query $Q = \langle P, g\rangle$ in $\mathbf{Q}$, recognizing whether a database D is in $EXP^{\forall}_{\mathcal{MF}}(Q)$ is in Π^P_2. To this end, we consider the complementary problem: is it true that D *is not* in $EXP^{\forall}_{\mathcal{MF}}(Q)$? Now, D is not in $EXP^{\forall}_{\mathcal{MF}}(Q)$ iff there exists a minimal founded model M of P_D such that $g \notin M$. Following the line of the proof of Theorem 5, we can easily see that the latter problem is in Σ^P_2. Hence, recognizing whether a database D is in $EXP^{\forall}_{\mathcal{MF}}(Q)$ is in Π^P_2.

Let us now prove that every Π^p_2 recognizable database collection $\mathbf{D}$ on a database scheme DS is in $EXP^{\forall}_{\mathcal{MF}}[\mathbf{Q}]$. By Fact 2, $\mathbf{D}$ is defined by a second order formula of the form $\forall\mathbf{R}^1\exists\mathbf{R}^2\Phi(\mathbf{R}^1,\mathbf{R}^2)$. Using the usual transformation technique, the above formula is equivalent to a second order Skolem form formula $(\forall\mathbf{S}^1)(\exists\mathbf{S}^2)\Gamma(\mathbf{S}^1,\mathbf{S}^2)$, where

$$\Gamma(\mathbf{S}^1,\mathbf{S}^2) = (\forall\mathbf{X})(\exists\mathbf{Y})(\Theta_1(\mathbf{S}^1,\mathbf{S}^2,\mathbf{X},\mathbf{Y}) \vee \ldots \vee \Theta_k(\mathbf{S}^1,\mathbf{S}^2,\mathbf{X},\mathbf{Y})),$$

$\mathbf{S}^1$ and $\mathbf{S}^2$ are two lists of, respectively, m_1 and m_2 predicate symbols, containing all symbols in $\mathbf{R}^1$ and $\mathbf{R}^2$, respectively. Consider the following program P:

$$\begin{array}{lll}
r_1: & s^1_j(\mathbf{W}^1_j) \vee \hat{s}^1_j(\mathbf{W}^1_j) \leftarrow & (1 \leqslant j \leqslant m_1)\\
r_2: & s^2_j(\mathbf{W}^2_j) \vee \hat{s}^2_j(\mathbf{W}^2_j) \leftarrow & (1 \leqslant j \leqslant m_2)\\
r_3: & q(\mathbf{X}) \leftarrow \Theta_i(\mathbf{S}^1,\mathbf{S}^2,\mathbf{X},\mathbf{Y}) & (1 \leqslant i \leqslant k)\\
r_4: & g \leftarrow \neg q(\mathbf{X}). &\\
r_5: & g \leftarrow s^2_j(\mathbf{W}^2_j), \hat{s}^2_j(\mathbf{W}^2_j) & (1 \leqslant j \leqslant m_2)\\
r_6: & \hat{s}^2_j(\mathbf{W}^2_j) \leftarrow g. & (1 \leqslant j \leqslant m_2)\\
r_7: & s^2_j(\mathbf{W}^2_j) \leftarrow g & (1 \leqslant j \leqslant m_2)
\end{array}$$

where, intuitively, $\hat{s}^1_j(\mathbf{W}^1_j)$ corresponds to $\neg s^1_j(\mathbf{W}^1_j)$, $\hat{s}^2_j(\mathbf{W}^2_j)$ corresponds to $\neg s^2_j(\mathbf{W}^2_j)$ and the rules of group r_3 defining q are used to implement the disjunction of the above second order formula. Observe that the guesses defined by the rules in the groups r_1 and r_2 are used in the rules in the group r_3 defining the predicate q and that the rules in the groups r_5, r_6 and r_7 force g to be false. Now it is easy to show that the formula $(\forall\mathbf{S}^1)(\exists\mathbf{S}^2)\Gamma(\mathbf{S}^1,\mathbf{S}^2)$ is valid if g is false in all minimal founded

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

models of P (if g is true the last two sets of rules make the second group of rules false). □

Theorem 5
$EXP^{\exists}_{\mathcal{MF}}[\mathbf{Q}] = DB\text{-}\Sigma^P_2$.

Proof
We first prove that for any query $Q = \langle P, g\rangle$ in $\mathbf{Q}$, recognizing whether a database D is in $EXP^{\exists}_{\mathcal{MF}}(Q)$ is in Σ^P_2. D is in $EXP^{\exists}_{\mathcal{MF}}(Q)$ iff there exists a minimal founded model M of P_D such that $g \in M$. To check this, we may guess an interpretation M of P_D and verify that M is a minimal founded model of P_D. The guess of the interpretation M is polynomial time. To check that M is minimal founded we can ask an NP oracle. Therefore, recognizing whether a database D is in $EXP^{\exists}_{\mathcal{MF}}(Q)$ is in Σ^P_2.

Let us now prove that every Σ^p_2 recognizable database collection $\mathbf{D}'$ on a database scheme BD is in $EXP^{\exists}_{\mathcal{MF}}[\mathbf{Q}]$. We have that $\mathbf{D}'$ is defined by a second order formula of the form $\exists\mathbf{R}^1\forall\mathbf{R}^2\Phi'(\mathbf{R}^1, \mathbf{R}^2)$. By setting $\Phi(\mathbf{R}^1, \mathbf{R}^2) = \neg\Phi'(\mathbf{R}^1, \mathbf{R}^2)$, we have that the formula $\forall\mathbf{R}^1\exists\mathbf{R}^2\Phi(\mathbf{R}^1, \mathbf{R}^2)$ defines the database collection $\mathbf{D}$, where $\mathbf{D} = \mathbf{D}_{BD} - \mathbf{D}'$ and $\mathbf{D}_{BD}$ is the set of all databases on BD. Consider the program P and the query $Q = \langle P, \neg g\rangle$ in the proof of Theorem 4. In it we have shown that a database D in $\mathbf{D}_{BD}$ is in $\mathbf{D}$ iff D is in $EXP^{\forall}_{\mathcal{MF}}(Q)$; hence a database D in $\mathbf{D}_{BD}$ is in $\mathbf{D}'$ iff D is not in $EXP^{\forall}_{\mathcal{MF}}(Q)$. But D is not in $EXP^{\forall}_{\mathcal{MF}}(Q)$ iff there exists some stable model M for which g is in M. It follows that $\mathbf{D}' = EXP^{\exists}_{\mathcal{MF}}(Q')$ where $Q' = \langle P, g\rangle$. □

Therefore, the expressive power of disjunctive datalog under minimal founded and stable model semantics is the same.

Data complexity is usually closely tied to expressive power and, in particular, it provides an upper bound for the expressive power (Eiter & Gottlob, 1993).

In this section we have shown that minimal founded semantics is complete for the second level of the polynomial hierarchy. For the stable model semantics it has been shown that for the class of head-cycle-free (hcf) the computation of a model selected nondeterministically can be done in polynomial time and checking if a ground atom belongs to a minimal model (resp. all minimal models) is complete for the first level of the polynomial hierarchy, i.e. NP-complete (resp. coNP-complete) (Ben-Eliyahu and Dechter, 1994). This result does not immediately apply to the minimal founded semantics since there could be rules which could force the selection of more than one atom appearing in the head of a rule. We conjecture that we have the same results for the class of head-cycle-free programs where constraints do not force the selection of more than one atom from the head of disjunctive rules. It is possible to identify a syntactic class consisting of hcf programs where after the rewriting of every ground constraint $\leftarrow B(X)$ in P with a rule $p(X) \leftarrow B(X), \neg p(X)$, there is no recursive atom A in $ground(P)$ depending on itself through an odd number of negations. The formal proof of this is outside the scope of this paper, and it could be investigated in some future work. Another interesting problem to be investigated in the future could be the syntactic characterization of programs for which stable and strongly

https://doi.org/10.1017/S1471068403001704 Published online by Cambridge University Press

founded models coincide. Clearly, this class contains positive and normal programs and programs where head disjunctions are forced to be exclusive by constraints.

5 Conclusion

The semantics proposed in this paper is essentially a variant of stable model semantics for normal programs. The aim of our proposal is the solution of some drawbacks of disjunctive stable model semantics which, in some cases, interprets inclusive disjunction as exclusive disjunction.

As disjunction is not interpreted as exclusive, the proposed semantics is not invariant if rules which are subsumed by other rules (under stable model semantics) are removed from the program; for instance, the first rule in the program of Example 7 can be deleted under stable model semantics as it is subsumed by the second rule, whereas under the minimal founded model semantics it cannot be deleted.

Several questions which need further investigation have been left open. For instance, further research could be devoted to (i) the identification of fragments of disjunctive datalog for which one minimal founded model can be computed in polynomial time; (ii) the use of two different types of disjunctive rule (inclusive disjunction and exclusive disjunction), and (iii) the investigation of abstract properties for disjunctive datalog under minimal founded semantics (Brass & Dix, 1992).

Acknowledgements

A preliminary version of this paper was presented at the LPNMR'99 conference (Greco, 1999). The work was partially supported by the Murst projects "DataX" and "D2I". The third author is also supported by ISI-CNR.

The authors are grateful to the anonymous referees for their useful comments and suggestions.

References

Abiteboul, S., Hull, R. and Vianu, V. (1994) *Foundations of Databases*. Addison-Wesley.

Apt, K. R., Blair, H. A. and Walker, A. (1988) Towards a theory of declarative knowledge. *Foundations of Deductive Databases and Logic Programming (J. Minker ed.)*. Morgan-Kaufman, pp. 89–148.

Ben-Eliyahu, R. and Dechter, R. (1994) Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence 12*, 53–87.

Brass, S. and Dix, J. (1992) Classifying semantics of disjunctive logic programs. In *Proceedings Joint International Conference and Symposium on Logic Programming*. MIT Press, pp. 798–812.

Chandra, A. and Harel, H. (1982) Structure and complexity of relational queries. *Journal of Computer and System Sciences*, **25**(1), 99–128.

Eiter, T. and Gottlob, G. (1993) Complexity aspects of various semantics of disjunctive databases. *Proceedings International Conference on Principles of Database Systems*. ACM Press, pp. 158–166.

Eiter, T., Gottlob, G. and Mannila, H. (1997) Disjunctive datalog. *ACM Transactions on Database Systems*, **22**(3), 364–418.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G. and Scarcello, F. (1998) The kr system dlv: Progress report, comparisons and benchmarks. *Proceedings of 6th International Conference on Principles of Knowledge Representation*. Morgan Kaufmann, pp. 406–417.

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

Fagin, R. (1974) Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation, Proceedings SIAM-AMS Conference (R. Karp ed.)* 7, pp. 43–73.

Fernandez, J. A. and Minker, J. (1991) Bottom-up evaluation of hierarchical disjunctive deductive databases. In *Proceedings of the Eigth International Conference on Logic Progrmming*. MIT Press, pp. 660–675.

Gelfond, M. and Lifschitz, V. (1988) The stable model semantics for logic programming. *Proceedings of Fifth International Conference on Logic Programming*. MIT Press, pp. 1070–1080.

Gelfond, M. and Lifschitz, V. (1991) Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9**(3/4), 365–385.

Greco, S. (1998) Binding propagation in disjunctive databases. *Proceedings of 24rd International Conference on Very Large Data Bases*. Morgan Kaufmann, pp. 287–298.

Greco, S. (1999) Optimization of disjunctive queries. *Proceedings International Conference on Logic Programming*. MIT Press, pp. 441–455.

Leone, N., Rullo, P. and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, **135**(2), 69–112.

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

Marek, W. and Truszczyński, M. (1991) Autoepistemic logic. *Journal of the ACM*, **38**(3), 518–619.

Minker, J. (1982) On indefinite data bases and the closed world assumption. *Proceedings of the 6-th Conference on Automated Deduction*. Lecture Notes in Computer Science, vol. 138. Springer, pp. 292–308.

Przymusinska, A. and Przymusinski, T. (1988) Weakly perfect model semantics for logic programs. *Proceedings of the Fifth International Conference and Symposium on Logic Programming*. MIT Press, pp. 1106–1120.

Przymusinski, T. (1988) On the declarative semantics of deductive databases and logic programming. *Foundations of deductive databases and logic programming (J. Minker ed.)*. Morgan-Kaufman, pp. 193–216.

Przymusinski, T. (1991) Stable semantics for disjunctive programs. *New Generation Computing*, **9**(3/4), 401–424.

Rajasekar, A., Lobo, J. and Minker, J. (1989) Weak generalized closed world assumption. *Journal of Automated Reasoning*, **5**(3), 293–307.

Ross, K. and Topor, R. W. (1988) Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, **4**(2), 397–424.

Ross, K. A. (1989) The well founded semantics for disjunctive logic programs. *Proceedings First International Conference on Deductive and Object-Oriented Databases*. North-Holland/Elsevier Science, pp. 385–402.

Saccà, D. (1997) The expressive powers of stable models for bound and unbound datalog queries. *Journal of Computer and System Sciences*, **54**(3), 441–464.

Sakama, C. and Inoue, K. (1994) An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, **13**(1), 145–172.

Schlipf, J. S. (1995) The expressive power of the logic programming semantics. *Journal of Computer and System Sciences*, **51**(1), 64–86.

Stockmeyer, L. J. (1976) The polynomial-time hierarchy. *Theoretical Computer Science*, **3**(1), 1–22.

VanGelder, A., Ross, K. A. and Schlipf, J. S. (1991) The polynomial-time hierarchy. *The Well-Founded Semantics for General Logic Programs*, **38**(3), 620–650.

Vardi, M. Y. (1982) The complexity of relational query languages. *Proceedings ACM Symposyum on Theory of Computing*. ACM Press, pp. 137–146.