BINDING PROPAGATION IN DISJUNCTIVE DATABASES*

Sergio Greco

DEIS
Università della Calabria
87030 Rende, Italy
greco@si.deis.unical.it

Abstract

In this paper we present a technique for the propagation of bindings into disjunctive deductive databases. The optimization is based on the rewriting of the source program into a program which is equivalent to the original one under the possible semantics. In particular, the rewriting technique generates a program which is disjunctive with nested rules in the head, i.e., elements in the head may also be (special) rules. The proposed optimization reduces the size of the data relevant to answer the query and, consequently, (i) reduces the complexity of computing a single model and, more importantly, (ii) greatly reduces the number of models to be considered to answer the query. Although in this paper we consider negation free and stratified linear programs, the technique can easily be extended to the full class of programs with stratified negation.

*Work partially supported by a MURST grant under the project "Interdata". The author is also supported by ISI-CNR.

Proceedings of the 24th VLDB Conference
New York, USA, 1998

1 Introduction

Recent research on databases has been concerned with situations where the knowledge of the world is incomplete. Two classic cases of incomplete knowledge are the presence of null values — i.e., a value of some attribute is unknown, and by the definition of probabilistic knowledge [13]. Another interesting area arises in the presence of incomplete data, i.e., it is unknown among several facts which one is true, but it is known that one or more are true. A natural way to extend databases to include incomplete data is to permit disjunctive statements as part of the language. This leads to deductive databases which permit clauses with disjunctions in their heads [15].

The presence of disjunctions in the head of rules makes the computation of queries very difficult. This is because no efficient techniques, such as the ones defined for standard Datalog queries (e.g., magic-set), have been defined, and for the presence of multiple models (generally the number of models

can be exponential with respect to the size of the input [1]).

Computation algorithms for disjunctive queries are based on the evaluation of the ground instantiation of programs and the only significant technique so far presented, known as intelligent grounding, is mainly based on the elimination of ground rules whose head cannot be derived from the program [6]. However, in many cases it is not necessary to compute all the models of the program. Take for instance a query asking if, given a graph G, there exists a simple path from the node a to the node b. In this case it is not necessary to check all models but just the ones containing paths with source node a and end node b. Although intelligent grounding reduces the number of ground rules, by eliminating useless rules (or heads of rules), it does not reduce the number of models to be checked.

Therefore, techniques which reduce the number of models, by eliminating the ones which are not useful to answer the query, should be exploited. The following example, presents a program were only a strict subset of the minimal models needs to be considered to answer the query.

Example 1.1 Consider the disjunctive program P consisting of the following rule

$$p(X) \vee q(X) \leftarrow a(X)$$

and a database D consisting of a set of the facts $a(1), a(2), ..., a(n)$.

Consider now a query asking if there is some model for $P \cup D$ containing the atom $p(m)$. A 'brute force' approach, based on an exhaustive search of the minimal models of $P \cup D$, would consider 2^n minimal models.

However, to answer the query (under 'brave reasoning') we could consider only the ground rule

$$p(m) \vee q(m) \leftarrow a(m)$$

and, therefore, consider only two minimal models: $M_1 = \{p(m)\} \cup D$ and $M_2 = \{q(m)\} \cup D$. □

The main result of this paper is the introduction of a technique which permits us to exploit binding propagation into disjunctive Datalog programs. The proposed technique extends binding propagation methods, previously defined for Datalog queries.

Although, for the sake of presentation, we consider only the extension of the magic-set method [2, 21], other methods such as supplementary magic-set, factorization techniques and special techniques for linear and chain queries [3, 11, 17, 18, 20, 21], can be applied as well. To the best of our knowledge, this is the first attempt to use the well known magic-set optimization for normal Datalog for the disjunctive case.

The rest of the paper is organized as follows. In Section 2 we recall basic concepts of Datalog and magic-set optimization. In Section 3 we present basic concepts of disjunctive Datalog and disjunctive nested rules. We also introduce a restricted class of disjunctive Datalog with nested rules whose semantics can be given in terms of minimal models (this is not true generally). In Section 4 we present our rewriting method for disjunctive queries. More specifically, in Section 4.1 we consider only negation free programs and in Section 4.2 the extension for stratified programs. Finally, in Section 5 we present an improvement of this technique for a subclass of disjunctive queries. Due to space limitations the proofs of our results are omitted. They can be found in the extended version of the paper.

2 Binding propagation in Datalog

2.1 Datalog

We assume familiarity with the Datalog language and only recall basic concepts on binding propagation and magic-set rewriting [21, 3].

Database predicates are divided into two parts: extensional predicates consisting of ground tuples and intentional predicates consisting of rules.

Extensional predicates define the input database whereas the rules define the program. As usual, we assume that rules are *safe*, i.e. each variable occurring in a rule also has to occur in a positive body literal of that rule. We denote by $U_{\mathcal{P}}$, $B_{\mathcal{P}}$ and $ground(\mathcal{P})$, the database domain (Herbrand universe), the set of all possible ground atoms (Herbrand base) and the ground instantiation of $\mathcal{P}$, respectively, defined as usual. Total (Herbrand) interpretations and models of $\mathcal{P}$ are also defined in the usual way.

Let $\mathcal{P}$ be a program, D a database and r an extensional predicate symbol, $D(r)$ denotes the set of tuples in the relation r. Given a program $\mathcal{P}$ and a database D, $\mathcal{P}_D$ denotes the program $\mathcal{P} \cup \{r(t) | r$ is an extensional predicate and $t \in D(r)\}$. A Datalog *query* is a pair $\langle G, \mathcal{P} \rangle$ where G is an atom called *query-goal* and $\mathcal{P}$ is a Datalog program. The *answer* to a query $\langle G, \mathcal{P} \rangle$ and a database D is the set of substitutions for the variables in G such that G is true with respect to $\mathcal{P}_D$. Two queries $(G, \mathcal{P})$ and $(G', \mathcal{P}')$ are *equivalent* if they have the same answer for all possible databases.

Given a program P and two predicate symbols 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 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 program is *stratified* if there is no rule r where a predicate p occurs in a negative literal in the body, q occurs in the head and $q \rightarrow p$, i.e. there is no recursion through negation.

We assume that programs are partitioned according to a topological order $(P_1, ..., P_n)$ such that each two predicates p and q, defined in the same component P_i, are mutually recursive. This means that each predicate appearing in P_i depends only on predicates belonging to P_j such that $j \leq i$. We assume also that the computation follows the topological order and that when we compute the component P_i the components $P_1, ..., P_{i-1}$ have already been computed. When we compute the component P_i all the facts obtained from the computation of the components $P_1, ..., P_{i-1}$ are basically treated the same as database facts. A rule in a component P_i is called *exit rule* if each predicate in the body belongs to a component P_j such that $j < i$. All the other rules are *recursive rules*.

2.2 Magic-set rewriting

We recall now the magic-set rewriting techniques for Datalog queries. The technique here presented applies only to negation free linear programs where bindings are propagated only through predicates which are not mutually recursive with the head predicate.

The magic-set method consists of three separate steps

1. An *Adornment step* in which the relationship between a bound argument in the rule head and the bindings in the rule body is made explicit.
2. A *Generation step* in which the adorned program is used to generate the *magic rules* which simulate the top-down evaluation scheme.
3. A *Modification step* in which the adorned rules are modified by the magic rules generated in step (2); these rules will be called *modified rules*.

We now informally recall the above steps for the case of linear programs, i.e. programs containing at most one predicate mutually recursive with the head predicate in the body of rules.

An *adorned program* is a program whose predicate symbols have associated a string α, defined on the alphabet $\{b, f\}$, of length equal to the arity of the predicate. A character b (resp. f) in the i-th position of the adornment associated with a predicate p means that the i-th argument of p is bound (resp. free).

The adornment step consists in generating a new program whose predicates are adorned. Given a rule r and an adornment α of the rule head, the adorned version of r is derived as follows:

1. Identify the distinguished arguments of the rules as follows: an argument is distinguished if it is bound in the adornment α, is a constant or appears in a base predicate of the rule-body which includes an adornment argument;
2. Assume that the distinguished arguments are bound and use this information in the adornment of the derived predicates in the rule body.

Adornments containing only f symbols can be omitted.

Given a query $Q = \langle q(T), P\rangle$ and let α be the adornment associated with $q(T)$. The set of adorned rules for Q is generated by 1) first computing the adorned version of the rules defining q and 2) next generating, for each new adorned predicate p^β introduced in the previous step, the adorned version of the rules defining p w.r.t. β; Step 2 i repeated until no new adorned predicate is generated.

The second step in the process is to use the adorned program for the generation of the magic rules. For each of the adorned predicates in the body of the adorned rule:

1. Eliminate all the derived predicates in the rule body which are not mutually recursive with the rule head;
2. Replace the derived predicates symbol p^α with $magic\_p^\alpha$ and eliminate the variables which are free w.r.t. α;
3. Replace the head predicates symbol q^β with $magic\_q^\beta$ and eliminate the variables which are free w.r.t. β;
4. Interchange the transformed head and derived predicate in the body.

Finally, the modification step of an adorned rule is performed as follows. For each adorned rule whose head is $p^\alpha(X)$, where X is a list of variables, append the rule body with $magic\_p^\alpha(X')$ where X' is the list of variables in X which are bound w.r.t. α.

The final program will contains only the rules which are useful to answer the query.

Example 2.1
Consider the query $Q = \langle p(1, C), P\rangle$ where P is defined as follows:

```
p(X,C) ← q(X,2,C).
q(X,Y,C) ← a(X,Y,C).
q(X,Y,C) ← b(X,Y,Z,W), q(Z,W,D), c(D,C).
```

The adorned program P' is

```
p^bf(X,Y) ← q^bbf(X,2,C).
q^bbf(X,Y,C) ← a(X,Y,C).
q^bbf(X,Y,C) ← b(X,Y,Z,W), q^bbf(Z,W,D), c(D,C).
```

The rewritten query is $Q' = \langle p^{bf}(1, Y), P'\rangle$ where P' is as follows:

```
magic_p^bf(1).
magic_q^bbf(X,2) ← magic_p^bf(X).
magic_q^bbf(Z,W) ← magic_q^bf(X,Y), b(X,Y,Z,W).

p^bf(X,Y) ← magic_p^bf(X), q^bbf(X,2,C).
q^bbf(X,Y,C) ← magic_q^bbf(X,Y), a(X,Y,C).
q^bbf(X,Y,C) ← magic_q^bbf(X,Y), b(X,Y,Z,W),
               q^bbf(Z,W,D), c(D,C).   □
```

Observe that, although the technique here presented applies only to negation free linear programs, it is general and can also be applied to non-linear programs with some form of negation (e.g., stratified negation) where bindings are also propagated through derived predicates [3].

Let $\mathcal{Q} = \langle G, \mathcal{P}\rangle$ be a query, then $Magic(\mathcal{Q})$ denotes the query derived from $\mathcal{Q}$ by applying the magic-set method. The query $Magic(\mathcal{Q})$ will be denoted also as $\langle magic(G), magic(G, \mathcal{P})\rangle$ where $magic(G, \mathcal{P})$ denotes the rewriting of $\mathcal{P}$ w.r.t. the goal G.

3 Disjunctive Deductive Databases

3.1 Disjunctive Datalog

For a background and unexplained concepts, see [15]. A *disjunctive Datalog rule* r is a clause of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_{k+m}$$

where $n \geq 1$, $k, m \geq 0$ and $a_1, \cdots, a_n, b_1, \cdots, b_{k+m}$ are function-free atoms.

We denote by $Head(r)$ (resp. $Body(r)$) the set of head atoms (resp. body literals) of r. If $n = 1$, then r is *normal* (i.e. $\vee$-free); if $m = 0$, then r is *positive* (or $\neg$-free). A *disjunctive Datalog program* $\mathcal{P}$, also called *disjunctive deductive database*, is a finite set of rules; it is *normal* (resp. *positive*) if all its rules are normal (resp. positive). The definition of stratified program defined for standard programs also applies to disjunctive programs. In the following we shall first consider positive disjunctive deductive databases and next we consider also disjunctive programs with stratified negation.

Minker proposed in [16] a model-theoretic semantics for positive $\mathcal{P}$, which assigns to $\mathcal{P}$ the set $\mathrm{MM}(\mathcal{P})$ of its *minimal models*, where a model M for $\mathcal{P}$ is minimal, if no proper subset of M is a model for $\mathcal{P}$. Accordingly, the program $\mathcal{P} = \{a \vee b \leftarrow\}$ has the two minimal models $\{a\}$ and $\{b\}$, i.e. $\mathrm{MM}(\mathcal{P}) = \{\ \{a\},\ \{b\}\ \}$.

The more general stable model semantics also applies to programs with (unstratified) negation. For general $\mathcal{P}$, the stable model semantics assigns to $\mathcal{P}$ the set $\mathrm{SM}(\mathcal{P})$ of its *stable models*. For positive $\mathcal{P}$, stable model and minimal model semantics coincide, i.e. $\mathrm{SM}(\mathcal{P}) = \mathrm{MM}(\mathcal{P})$.

The result of a query $\mathcal{Q} = \langle G, \mathcal{P} \rangle$ on an input database D is defined in terms of the minimal models of $\mathcal{P}_D$, by taking either the union of all models (*possible inference*) or the intersection (*certain inference*). Thus, given a program $\mathcal{P}$ and a database D, a ground atom G is true, under possible (brave) semantics, if there exists a minimal model M for $\mathcal{P}_D$ such that $G \in M$. Analogously, G is true, under certain (cautious) semantics, if G is true in every minimal model for $\mathcal{P}_D$.

3.2 Disjunctive Datalog with nested rules

In this section, we recall the extension of disjunctive Datalog by nested rules first proposed in [10].

A *nested rule* is of the form:

$$A \hookleftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_{k+m}, \qquad k, m \geq 0$$

where $A, b_1, \cdots, b_{k+m}$ are atoms. If $m = 0$, then the implication symbol "$\hookleftarrow$" can be omitted.

A *disjunctive nested rule* r is of the form

$$A_1 \vee \cdots \vee A_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_{k+m},$$

where $n \geq 1$, $k, m \geq 0$, $b_1, \cdots, b_{k+m}$ are atoms, and $A_1, \cdots, A_n$ are nested rules. If $A_1, \cdots, A_n$ are atoms, then r is *flat*.

Example 3.1 A rule may appear in the head of another rule. For instance, the rule $r_1 : a \vee (b \hookleftarrow c) \leftarrow d$ is an allowed disjunctive nested rule, while the rule $r_2 : a \vee b \leftarrow d$ is a flat disjunctive rule □

The definition of stratified programs can be also extended to disjunctive programs with nested rules. Given a program P and two predicate symbols p and q, we write $p \rightarrow q$ if i) there exists a rule r such that q occurs in head of some nested rule of r, say r', and p appears either in the body of r' or in the body of r, ii) there exists a predicate s such that $p \rightarrow s$ and $s \rightarrow q$. A program is *stratified* if there exists no rule r where a predicate p occurs in a negative literal and q occurs in the head of some nested rule appearing in the head of r, i.e. there is no recursion through negation.

Let r be a ground nested rule. We say that r is *applied* in the interpretation I if (i) every literal in $Body(r)$ is true w.r.t. I, and (ii) the atom in the head of r is true w.r.t. I. A rule $r \in ground(\mathcal{P})$ is *satisfied* (or *true*) w.r.t. I if its body is false (i.e., some body literal is false) w.r.t. I or an element

of its head is applied. (Note that for flat rules this notion coincides with the classical notion of truth).

Example 3.2 The nested rule $b \hookleftarrow \neg c$ is applied in the interpretation $I = \{b, d\}$, as its body is true w.r.t. I and the head atom b is in I. Therefore, rule $r_1 : a \vee (b \hookleftarrow \neg c) \leftarrow d$ is satisfied w.r.t. I. r_1 is true also in the interpretation $I = \{a, d\}$; while it is not satisfied w.r.t. the interpretation $I = \{c, d\}$.

Observe that the two implication symbols have different semantics. In the interpretation $I = \{c\}$, the rule $b \hookleftarrow \neg c$ is not true whereas the rule $b \leftarrow \neg c$ is true. □

A *model* for $\mathcal{P}$ is an interpretation M for $\mathcal{P}$ which satisfies every rule $r \in ground(\mathcal{P})$.

Example 3.3 For the flat program $\mathcal{P} = \{a \vee b \leftarrow\}$ the interpretations $\{a\}$, $\{b\}$ and $\{a, b\}$ are its models.

For the program $\mathcal{P} = \{a \vee b \leftarrow;\ \ c \vee (d \hookleftarrow a) \leftarrow\}$ the interpretations $\{a, d\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, d\}$, $\{a, b, c\}$, $\{a, c, d\}$, $\{a, b, c, d\}$ are models. $\{b, d\}$ is not a model, as rule $c \vee (d \hookleftarrow a) \leftarrow$ has a true body but neither c nor $d \hookleftarrow a$ are applied w.r.t. $\{b, d\}$ (the latter is not applied because a is not true). □

In the presence of negation and nested rules, not all minimal models represent an intuitive meaning for the programs at hand. A proper semantics for Disjunctive Datalog with nested rules and (possible unstratified) negation has been defined in [10] by extending the notion of *unfounded set* given for normal and disjunctive logic programs in [22] and [14], respectively.

We present here a subclass of Disjunctive Datalog with nested rules whose semantics is given by the set of minimal models.

Definition 3.4 A nested disjunctive program $\mathcal{P}$ is said to be *weakly nested* if all nested rules in $\mathcal{P}$ are not recursive. □

Thus, in weakly nested programs, predicates appearing in the head of nested rules are not mutually recursive with predicates appearing in the body of the nested rules. For instance, the program of Example 3.1 consisting of the single rule r_1 is weakly nested since the nested rule $b \hookleftarrow c$ is not recursive.

Theorem 3.5 *Let $\mathcal{P}$ be a positive weakly nested disjunctive program. Then,* $\mathrm{SM}(\mathcal{P}) = \mathrm{MM}(\mathcal{P})$. □

The above result implies that for weakly nested programs the set of minimal and stable models coincide. Thus, for this class of programs we can consider the global set of minimal models, whereas for general nested disjunctive programs we must consider only minimal models which are also stable.

4 Binding Propagation in Disjunctive Programs

In this section we present the propagation of bindings into disjunctive programs. Before presenting how disjunctive queries are rewritten to propagate bindings into the bodies of rules, let us first define the equivalence of queries for disjunctive programs. A (nested) disjunctive Datalog *query* over a database defines a mapping from the database to a finite (possibly empty) set of finite (possibly empty) relations for the goal.

Given an atom G and an interpretation M, $A(G, M)$ denotes the set of substitution for the variables in G such that G is true in M. The answer to a query $\mathcal{Q} = \langle G, \mathcal{P} \rangle$ over a database D under *brave* (resp. *cautious*) semantics, denoted $Ans_b(\mathcal{Q}, D)$ (resp., $Ans_c(\mathcal{Q}, D)$) is the relation $\cup_M A(G, M)$ such that $M \in \mathrm{MM}(\mathcal{P}, D)$ (resp., $\cap_M A(G, M)$ such that $M \in \mathrm{MM}(\mathcal{P}, D)$). Two queries $\mathcal{Q}_1 = \langle G_1, \mathcal{P}_1 \rangle$ and $\mathcal{Q}_2 = \langle G_2, \mathcal{P}_2 \rangle$ are said to be *equivalent* under semantic s ($\mathcal{Q}_1 \equiv_s \mathcal{Q}_2$) if for every database D on a fixed schema is $Ans_s(\mathcal{Q}_1) = Ans_s(\mathcal{Q}_2)$. Moreover, for stratified disjunction free programs, since two semantics coincide, we will simply write $\mathcal{Q}_1 \equiv \mathcal{Q}_2$.

We next present how bingings are propagated in the body of disjunctive rules. We consider first the case of positive programs and next we extend the method to programs with stratified negation.

4.1 Positive Programs

The main problem in propagating bindings in disjunctive rules is that, generally, we cannot apply standard techniques since by propagating bindings from some atom in the head into the body, we restrict all head atoms. This behaviour can be better explained by means of an example. Consider the query $Q = \langle q(3), P\rangle$ where P is as follows:

$$\begin{array}{rcl} p(1). & & \\ p(Y) \vee q(Y) & \leftarrow & p(X),\ a(X,Y) \end{array}$$

Assuming that the database D consists of the tuples $a(1,2)$ and $a(2,3)$, the program P_D has three minimal models: $M_1 = \{p(1), p(2), p(3)\} \cup D$, $M_2 = \{p(1), p(2), q(3)\} \cup D$ and $M_3 = \{p(1), q(2)\} \cup D$. However, if we apply the standard magic-set technique we get the following program P'

$$\begin{array}{rcl} mq(3). & & \\ p(1). & & \\ p(Y) \vee q(Y) & \leftarrow & mq(Y),\ p(X),\ a(X,Y) \end{array}$$

where adornments have been omitted. P'_D has the unique minimal model $M = \{mq(3), p(1)\} \cup D$. Therefore, the queries Q and $Q' = \langle q(3), P'\rangle$ are not equivalent.

We now present an algorithm for the optimization of disjunctive Datalog queries. The algorithm uses nested rules as a vehicle to propagate bindings. We will use the following running example to explain our rewriting method.

Example 4.1 We are given the query $\langle$`anc_father(john,Y)`, `ANC`$\rangle$ where the program `ANC` consists of the following rules:

```
father(X,Y) ∨ mother(X,Y) ←  parent(X,Y).
anc_father(X,Y) ←  father(X,Y).
anc_father(X,Y) ←  father(X,Z), anc_father(Z,Y).
```

Definition 4.2 Let $\mathcal{P}$ be a disjunctive Datalog program. The *standard* version of $\mathcal{P}$, denoted $sv(\mathcal{P})$, is the Datalog program derived from $\mathcal{P}$ by replacing each disjunctive rule $A_1 \vee ... \vee A_m \leftarrow B$ with the m rules of the form $A_i \leftarrow B$ for $1 \leq i \leq m$. Moreover, we denote with $sv(\mathcal{Q})$ the query $\langle G, sv(\mathcal{P})\rangle$. □

Example 4.3 The standard version of the program `ANC` of Example 4.1, denoted sv(`ANC`), is as follows

```
father(X,Y) ←      parent(X,Y).
mother(X,Y ←       parent(X,Y).
anc_father(X,Y) ←  father(X,Y).
anc_father(X,Y) ←  father(X,Z),  anc_father(Z,Y).
```

Observe that we are considering negation free programs and, therefore, the standard version of a disjunctive program has a unique minimal model.

Proposition 4.4 *Let $\mathcal{P}$ be a positive disjunctive Datalog program and let N be the minimal model of $sv(P)$. Then, every minimal model for $\mathcal{P}$ is contained in N.* □

Given a disjunctive Datalog program $\mathcal{P}$ (query $\mathcal{Q}$), we denote with $SV(\mathcal{P})$ (resp., $SV(\mathcal{Q})$) the standard program (resp., query) derived from $sv(\mathcal{P})$ (resp., $sv(\mathcal{Q})$) by replacing each derived predicate symbol p appearing in the head of some disjunctive rule with a new predicate symbol P. For instance, the standard version SV(`ANC`) of the program `ANC` of Example 4.1 is derived from the program sv(`ANC`) of Example 4.3 by replacing the predicate symbols `father` and `mother`, respectively, with the new predicate symbols `FATHER` and `MOTHER`.

The application of the magic-set method to $SV(\mathcal{Q})$ gives an equivalent query which can be evaluated more efficiently [3, 21].

Example 4.5 Consider the program of Example 4.3 and the query goal anc_father(john, Y). The adorned program with respect to the query goal is

$$\begin{array}{ll} \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{parent}(\texttt{X},\texttt{Y}). \\ \texttt{MOTHER}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{parent}(\texttt{X},\texttt{Y}). \\ \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}). \\ \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Z}), \\ & \texttt{anc\_father}^{\texttt{bf}}(\texttt{Z},\texttt{Y}). \end{array}$$

The query obtained by applying the magic-set method is $\langle \texttt{anc\_father}^{\texttt{bf}}(\texttt{john},\texttt{Y}), \mathit{Magic}(\texttt{SV(ANC)})\rangle$ where the rewritten program $\mathit{Magic}(\texttt{SV(ANC)})$ is as follows:

$$\begin{array}{ll} \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{john}). & \\ \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{Z}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Z}). \\ \texttt{magic\_FATHER}^{\texttt{bf}}(\texttt{X}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}). \\ \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_FATHER}^{\texttt{bf}}(\texttt{X}),\ \texttt{parent}(\texttt{X},\texttt{Y}). \\ \texttt{MOTHER}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{parent}(\texttt{X},\texttt{Y}). \\ \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}). \\ \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Z}), \\ & \texttt{anc\_father}^{\texttt{bf}}(\texttt{Z},\texttt{Y}). \end{array}$$ □

Fact 4.6 *Let Q be a positive disjunctive Datalog query. Then, $Magic(SV(Q)) \equiv SV(Q)$.* □

We are now in the position to present our rewriting algorithm. The algorithm is based on the rewriting of each atom a in the head of a disjunctive rule into a nested rule $a \hookleftarrow A$, where the atom A is used to make a restriction on the values of a. Predicates in the body of nested rules are defined by standard Datalog rules which can be optimized by means of classical rewriting techniques such as magic-set. The algorithm implementing the rewriting of disjunctive queries is reported in Fig. 1.

Observe that the nested disjunctive program generated by Algorithm 1 is a (positive) weakly nested program.

Example 4.7 The complete program obtained from the application of Algorithm 1 to the program of Example 4.1 with the query goal $\texttt{anc\_father}^{\texttt{bf}}(\texttt{john},\texttt{Y})$ is as follows:

$$\begin{array}{ll} r_1: \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{john}). & \\ r_2: \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{Z}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Z}). \\ r_3: \texttt{magic\_FATHER}^{\texttt{bf}}(\texttt{X}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}). \\ r_4: \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_FATHER}^{\texttt{bf}}(\texttt{X}), \texttt{parent}(\texttt{X},\texttt{Y}). \\ r_5: \texttt{MOTHER}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{parent}(\texttt{X},\texttt{Y}). \\ r_6: \texttt{FATHER}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{FATHER}^{\texttt{bf}}(\texttt{X},\texttt{Y}). \\ r_7: (\texttt{father}(\texttt{X},\texttt{Y}) \hookleftarrow \texttt{FATHER}(\texttt{X},\texttt{Y})) \vee & \\ \quad (\texttt{mother}(\texttt{X},\texttt{Y}) \hookleftarrow \texttt{MOTHER}(\texttt{X},\texttt{Y})) \leftarrow & \texttt{parent}(\texttt{X},\texttt{Y}). \\ r_8: \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{father}(\texttt{X},\texttt{Y}). \\ r_9: \texttt{anc\_father}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow & \texttt{magic\_anc\_father}^{\texttt{bf}}(\texttt{X}), \\ & \texttt{father}(\texttt{X},\texttt{Z}), \\ & \texttt{anc\_father}^{\texttt{bf}}(\texttt{Z},\texttt{Y}). \end{array}$$

Where rule r_6 is added in Step 4 whereas the body of the last two rules is modified in Step 3. □

Algorithm 1 *Magic-set for disjunctive queries*
Input: $\mathcal{Q} = \langle g, \mathcal{P}\rangle$
Output: $\mathcal{Q}' = \langle g, \mathcal{P}'\rangle$
begin
(1) Generate $sv(\mathcal{Q})$;
(2) Generate $SV(\mathcal{Q})$;
(3) $\mathcal{P}' = Magic(\langle G, SV(\mathcal{P})\rangle)$;
(4) **for each** *pred. R^α generated in Step (2)* **do**
add to $\mathcal{P}'$ the following rule with $k = arity(R)$
$R(X_1, ..., X_k) \leftarrow R^\alpha(X_1, ..., X_k)$;
(5) **for each** *rule $r = a_1(X_1) \vee ... \vee a_m(X_m) \leftarrow B$* **do**
replace r with the rule
$(a_1(X_1) \hookleftarrow A_1(X_1)) \vee ... \vee (a_m(X_m) \hookleftarrow A_m(X_m)) \leftarrow B$
end.

Figure 1: Rewriting of disjunctive queries

Theorem 4.8 *Let $\mathcal{Q} = \langle G, \mathcal{P}\rangle$ be a disjunctive Datalog query and let $\mathcal{Q}'$ be the query derived by applying Algorithm 1 to $\mathcal{Q}$. Then, $\mathcal{Q} \equiv_b \mathcal{Q}'$.* □

The computation of positive weakly nested disjunctive queries can be carried out by any algorithm computing minimal (or even stable) models for disjunctive queries [12, 4], just by introducing a minor change. More specifically, it is sufficient, in the

construction of a minimal model, to consider rules whose body is true (as usual), and nested rules in the head whose bodies are also true.

4.2 Stratified Programs

The minimal model semantics is not able to capture the intuitive meaning of programs with negation. Stable and perfect models have been proposed to capture the semantics of general disjunctive programs. The disjunctive stable models of a program $\mathcal{P}$ are defined as follows. For any interpretation I, denote with $\mathcal{P}^I$ the ground positive program derived from $ground(\mathcal{P})$ (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 $\mathcal{P}$ if and only if $M \in MM(\mathcal{P}^M)$. The set of stable models will be denoted $SM(\mathcal{P})$. Since each stable model is minimal we have that $SM(\mathcal{P}) \subseteq MM(\mathcal{P})$.

The perfect model semantics has been defined for general disjunctive programs but is particularly suited for stratified disjunctive programs. In this paper we do not consider propagation of bindings into programs with unstratified negation since stratified negation in disjunctive programs is sufficient to gain the full expressive power of unstratified negation [5] and also because the combination of disjunction and unstratified negation may result in programs with unclear meaning.

An interpretation M is a perfect model for a stratified program $\mathcal{P}$, iff for some stratification $(\mathcal{P}_0, \mathcal{P}_1, ..., \mathcal{P}_k)$ of $\mathcal{P}$ $M \cap B_{\mathcal{P}_i} \in MM(\mathcal{P}_i^{*M_i})$, for $i = 0, ..., k$, where $\mathcal{P}_i^* = ground(P_0 \cup ... \cup P_i)$, $M_0 = \emptyset$ and $M_i = M \cap B_{\mathcal{P}_{i-1}^*}$ [5]. The set of perfect models of $\mathcal{P}$ is denoted by $PM(\mathcal{P})$. Since each perfect model is minimal we have that $PM(\mathcal{P}) \subseteq MM(\mathcal{P})$.

Moreover, for stratified programs we have that $PM(\mathcal{P}) = SM(\mathcal{P})$ whereas for positive programs $MM(\mathcal{P}) = PM(\mathcal{P}) = SM(\mathcal{P})$.

Fact 4.9 *Let $\mathcal{P}$ be a stratified disjunctive Datalog program, then, $sv(\mathcal{P})$ is also stratified.* □

The application of the magic-set rewriting to stratified non-linear programs may result in programs which are, syntactically, not stratified. Thus, for the sake of simplicity, here we consider only linear programs, and defer the rewriting of general programs to the extended version of the paper.

Proposition 4.10 *Let $\mathcal{P}$ be a stratified linear disjunctive Datalog program and let N be the perfect model of $sv(\mathcal{P})$. Then, every perfect model for $\mathcal{P}$ is contained in N.* □

Now, let $Magic^{\neg s}(\mathcal{Q})$ be the function rewriting a stratified linear query $\mathcal{Q}$. The algorithm for the rewriting of $\mathcal{Q}$ can be derived from Algorithm 1 by simply replacing the rewriting of the positive program in Step (2) with the function $Magic^{\neg s}(\mathcal{Q})$. We will denote the modified algorithm as *Algorithm 1.1.*

Corollary 4.11 *Let $\mathcal{Q} = \langle g, \mathcal{P} \rangle$ be a stratified disjunctive linear Datalog query and let $\mathcal{Q}'$ be the query derived by applying Algorithm 1.1 to $\mathcal{Q}$. Then, $\mathcal{Q} \equiv_b \mathcal{Q}'$.* □

4.3 Special subclasses of queries

When bindings in non-linear rules are passed through recursive predicates, the only method at hand is the *generalized* magic-set method [3]. However, there are several programs such as *left-linear*, *right-linear*, *counting linear*, *factorized* and *chain* programs for which specialized methods exist that are much more efficient than the magic-set method. Given the importance and frequency of these special situations in actual applications, deductive systems detect these special cases and compile them using special techniques, such as those proposed in [17, 18, 20, 11].

The rewriting of disjunctive queries is orthogonal with respect to the specific rewriting technique used

to optimize the standard version of the query (step (2) in Algorithm 1). Thus, for special classes of queries, it is possible to apply specialized optimization techniques. The following example presents a disjunctive program which can be optimized by using the classical right-linear optimization technique [18, 21].

Example 4.12 Consider the query goal $\mathtt{sp(X,b)}$ over the following program P.

$$
\begin{array}{ll}
r_1 : \mathtt{sp(X,Y)} \leftarrow & \mathtt{g(X,Y)}. \\
r_2 : \mathtt{sp(X,Y)} \vee \mathtt{no\_sp(X,Y)} \leftarrow & \mathtt{sp(X,Z),\ g(Z,Y)}. \\
r_3 : \mathtt{no\_sp(X,Z)} \leftarrow & \mathtt{sp(X,Y), node(Z), Z \neq Y}. \\
r_4 : \mathtt{no\_sp(Z,Y)} \leftarrow & \mathtt{sp(X,Y), node(Z), Z \neq X}.
\end{array}
$$

The query, under brave reasoning, computes the simple path in the graph having as end node b. The standard version of $SV(P)$ is as follows

$$
\begin{array}{ll}
r_5 : \mathtt{SP(X,Y)} \leftarrow & \mathtt{g(X,Y)}. \\
r_6 : \mathtt{SP(X,Y)} \leftarrow & \mathtt{SP(X,Z),\ g(Z,Y)}. \\
r_7 : \mathtt{NO\_SP(X,Y)} \leftarrow & \mathtt{SP(X,Z),\ g(Z,Y)}. \\
r_8 : \mathtt{NO\_SP(X,Z)} \leftarrow & \mathtt{SP(X,Y), node(Z), Z \neq Y}. \\
r_9 : \mathtt{NO\_SP(Z,Y)} \leftarrow & \mathtt{SP(X,Y), node(Z), Z \neq X}.
\end{array}
$$

The definition of the predicate SP, w.r.t. the goal $SP(X,b)$, is right-linear and, therefore, can be optimized by using the special technique defined for this class of queries. The definition of SP consists of one single set of recursive rules.

$$
\begin{array}{ll}
r_{10} : \mathtt{magic\_SP(b)}. \leftarrow & \\
r_{11} : \mathtt{magic\_SP(Z)} \leftarrow & \mathtt{magic\_SP(Y),\ g(Z,Y)}. \\
r_{12} : \mathtt{SP'(X)} \leftarrow & \mathtt{magic\_SP(Y), g(X,Y)}. \\
r_{13} : \mathtt{SP(X,b)} \leftarrow & \mathtt{SP'(X)}.
\end{array}
$$

The final program consists of the following rules

$$
\begin{array}{l}
\mathtt{sp(X,Y)} \leftarrow \mathtt{g(X,Y)}. \\
(\mathtt{sp(X,Y)} \hookleftarrow \mathtt{SP(X,Y)}) \vee \\
(\mathtt{no\_sp(X,Y)} \hookleftarrow \mathtt{NO\_SP(X,Y)}) \leftarrow \quad \mathtt{sp(X,Z),\ g(Z,Y)}. \\
\mathtt{no\_sp(X,Z)} \leftarrow \quad \mathtt{sp(X,Y), node(Z), Z \neq Y}. \\
\mathtt{no\_sp(Z,Y)} \leftarrow \quad \mathtt{sp(X,Y), node(Z), Z \neq X}.
\end{array}
$$

plus the definition of SP and NO_SP consisting of the rules $r_7 - r_{13}$. □

5 Extending binding propagation

In this section we present an extension of the binding propagation for disjunctive queries to propagate bindings among heads of the same rule. Our algorithm generates a nested program which makes a stronger restriction on the data, but it applies to a restricted set of queries.

Consider the query of Example 1.1. From the application of Algorithm 1, we have that the values of q depend on the binding in the query goal ($q(m)$). However, from the disjunctive rule, the atom $q(m)$ does not depend on atoms $p(x)$ with $x \neq m$ (i.e. it depends only on the atom $p(m)$). This means that we can consider only the atoms $p(m)$ and $q(m)$ to answer the query. Therefore, bindings can also be propagated among atoms in the head of disjunctive rules.

Before presenting our algorithm, we need to introduce a restricted class of programs denoted as *strongly safe*.

Definition 5.1 Let $\mathcal{Q} = \langle G, \mathcal{P} \rangle$ be a disjunctive query and let $sv(\mathcal{P})$ be the standard version of $\mathcal{P}$. Then, the extended standard version of $\mathcal{P}$, denoted $esv(\mathcal{P})$, is the program derived from $sv(\mathcal{P})$ by adding for each disjunctive rule r and for each pair of atoms a_1 and a_2 in the head of r the two rules $a_1 \leftarrow a_2$ and $a_2 \leftarrow a_1$. □

As in the standard version of programs, we shall denote with $ESV(\mathcal{P})$ the program derived from $esv(\mathcal{P})$ by replacing every derived predicate symbol r, appearing in a disjunctive head, with a new predicate symbol R.

For instance the extended standard version of the program of Example 1.1 consists of the following rules

$$
\begin{array}{lll}
p(X) & \leftarrow a(X) & \\
q(X) & \leftarrow a(X) p(X) & \leftarrow q(X) \\
q(X) & \leftarrow p(X) &
\end{array}
$$

Observe that, the last two rules added to the extended standard version are used only to propagate bindings among heads and they will be deleted from the final version. Moreover, the extended standard version could contain unsafe rules even in the case where the source program is safe. Thus, we consider here a restricted class of disjunctive programs.

Definition 5.2 A disjunctive program is *strongly safe* if all its rules are safe and for each disjunctive rule all atoms in the head have the same variables. □

Observe that all programs presented in this paper are strongly safe. Thus, we have

Fact 5.3 *Let $\mathcal{P}$ be a strongly safe disjunctive Datalog program. Then, esv($\mathcal{P}$) is safe.* □

The algorithm we present uses the extended standard version to propagate adornments also among heads of rules. Rules added in the extended version are used only to propagate adornments and not to restrict our predicates and, therefore, they are deleted (in Step (4) of Algorithm 2) after the magic rules are generated. The new algorithm which is a variant of Algorithm 1 for the class of strongly safe programs and is reported in Figure 2.

Theorem 5.4 *Let $\mathcal{Q} = \langle g, \mathcal{P} \rangle$ be a strongly safe disjunctive Datalog query and let $\mathcal{Q}'$ be the query derived by applying Algorithm 2 to $\mathcal{Q}$. Then, $\mathcal{Q} \equiv_b \mathcal{Q}'$.* □

We conclude by presenting an application example of Algorithm 2

Example 5.5 Consider the following program P with the query goal $p(a, Y)$.

$$\mathtt{p(X,Y)} \vee \mathtt{q(X,Y)} \leftarrow \mathtt{b(X,Y)}$$

Algorithm 2 *Magic rewriting for strongly safe disjunctive Datalog queries*
Input: $\mathcal{Q} = \langle g, \mathcal{P} \rangle$
Output: $\mathcal{Q}' = \langle g, \mathcal{P}' \rangle$
begin
(1) Generate $sv(\mathcal{P})$ and $esv(\mathcal{P})$;
(2) Generate $\mathcal{P}_1 = ESV(\mathcal{P})$ and $\mathcal{P}_2 = SV(\mathcal{P})$
(3) $\Delta\mathcal{P} = \mathcal{P}_1 - \mathcal{P}_2$; $P' = Magic(\langle G, \mathcal{P}_1 \rangle)$;
(4) delete from $\mathcal{P}'$ modified rules derived from rules in $\Delta\mathcal{P}$;
(5) **for each** *pred. R^α generated in Step (2)* **do**
add to $\mathcal{P}'$ the following rule with $k = arity(R)$
$R(X_1, ..., X_k) \leftarrow R^\alpha(X_1, ..., X_k)$;
(6) **for each** *rule $r = a_1(X_1) \vee ... \vee a_m(X_m) \leftarrow B$* **do**
replace r with the rule
$(a_1(X_1) \hookleftarrow A_1(X_1)) \vee ... \vee (a_m(X_m) \hookleftarrow A_m(X_m)) \leftarrow B$
end.

Figure 2: Rewriting of strongly safe queries

The Datalog program $ESV(P)$ consists of the rules

$$\mathtt{P(X,Y)} \leftarrow \mathtt{b(X,Y)}$$
$$\mathtt{Q(X,Y)} \leftarrow \mathtt{b(X,Y)}$$
$$\mathtt{P(X,Y)} \leftarrow \mathtt{Q(X,Y)}$$
$$\mathtt{Q(X,Y)} \leftarrow \mathtt{P(X,Y)}$$

The program derived from the application of the magic-set method is

r_1 : $\mathtt{magic\_P(a)}$.
r_2 : $\mathtt{magic\_Q(X)} \leftarrow \mathtt{magic\_P(X)}$.
r_3 : $\mathtt{magic\_P(X)} \leftarrow \mathtt{magic\_Q(X)}$.
r_4 : $\mathtt{P(X,Y)} \leftarrow \mathtt{magic\_P(X)},\ \mathtt{b(X,Y)}$
r_5 : $\mathtt{Q(X,Y)} \leftarrow \mathtt{magic\_P(X)},\ \mathtt{b(X,Y)}$
r_6 : $\mathtt{P(X,Y)} \leftarrow \mathtt{magic\_P(X)},\ \mathtt{Q(X,Y)}$
r_7 : $\mathtt{Q(X,Y)} \leftarrow \mathtt{magic\_P(X)},\ \mathtt{P(X,Y)}$

The final program consists of the above rules $r_1 - r_5$ (rules r_6 and r_7 are deleted because they are modified rules derived from rules in $ESV(P) - SV(P)$) plus the rule

$$(\mathtt{p(X,Y)} \hookleftarrow \mathtt{P(X,Y)}) \vee (\mathtt{q(X,Y)} \hookleftarrow \mathtt{Q(X,Y)}) \leftarrow \mathtt{b(X,Y)}$$

where we have omitted adornments and consequently, rules of the form $P(X) \leftarrow P^\alpha(X)$. □

6 Conclusions

In this paper we have presented a technique for the application of classical optimization methods for Datalog queries to disjunctive Datalog. The application of binding propagation techniques has been carried out by rewriting disjunctive rules into nested disjunctive rules (i.e. disjunctive rules whose heads could contain special Datalog rules). The computation of nested disjunctive queries is no more difficult than for disjunctive queries. Although in this paper we have considered only negation-free and stratified linear programs, our technique can be easily extended to the global class of stratified disjunctive queries.

Acknowledgements: The author would like to thank the anonymous referees for many stimulating suggestions.

References

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

[2] F. Bancilhon, D. Mayer, Y. Sagiv, and J.F. Ullman. Magic sets and other strange ways to implement logic programs. *Proc. PODS Conf.*, 1986.

[3] Beeri, C. and R. Ramakrisnhan. (1991) On the power of magic. *Journal of Logic Programming.* (prel. version in *PODS* 1987).

[4] Dix J., U. Furbach and A. Nerode (eds.) *Proc. Int. Conf. Logic Programming and Nonmonotonic Reasoning* (System Descriptions). 1997.

[5] Eiter, T., Gottlob, G. and Mannila, Disjunctive Datalog, *in ACM Trans. on Database Systems*, Sept. 1997, Vol 22, N. 3 (Prel. vers. in PODS-94).

[6] Eiter T., N. Leone, C. Mateis, G. Pfeifer and F. Scarcello. A Deductive System for Non-monotonic Reasoning. *Proc. LPNMR Conf.*, 1997. 363-374.

[7] Fernández, J.A. and Minker, J., Semantics of Disjunctive Deductive Databases, *in Proc. 4th ICDT Conference*. 1992. pp. 21–50.

[8] Gelfond, M., Lifschitz, V., The Stable Model Semantics for Logic Programming, *in Proc. Fifth Conf. on Logic Progr.*, 1988, pp. 1070–1080.

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

[10] Greco, S., N. Leone, and F. Scarcello, Disjunctive Datalog with Nested Rules, *Proc. LPKR Workshop*, Port Jefferson, NY, 1997.

[11] S. Greco and C. Zaniolo, The PushDown Method to Optimize Chain Logic Programs. In *Proc. ICALP Conference*, 1995.

[12] IFIP-GI Workshop (1994), "Disjunctive Logic Programming and Disjunctive Databases," 13-th IFIP World Computer Congress.

[13] Lakshmanan, Laks V.S. and F. Sadri. (1994) Probabilistic deductive databases. In *Proc. Int. Logic Programming Symposium*, 254-268.

[14] Leone, N., Rullo, P., Scarcello, F. (1997) Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation, *Information and Computation*, Forthcoming.

[15] Lobo, J., Minker, J. and Rajasekar, A. (1992) *Foundations of Disjunctive Logic Programming* MIT Press, Cambridge, MA.

[16] Minker, J., On Indefinite Data Bases and the Closed World Assumption, *in "Proc. 6^{th} CADE Conference*, 1982, pp. 292–308.

[17] J. Naughton, R. Ramakrisnhan, Y. Sagiv, and J.F. Ullman. Argument Reduction by Factoring. *Proc. 15th VLDB Conference*, 1989, pp. 173–182.

[18] J. Naughton, R. Ramakrisnhan, Y. Sagiv, and J.F. Ullman. Efficient evaluation of right-, left-, and multi-linear rules. *Proc. SIGMOD Conf.*, 1989. pp. 235–242.

[19] Przymusinski, T. (1991), Stable Semantics for Disjunctive Programs, *New Generation Computing*, **9**, pp. 401–424.

[20] R. Ramakrisnhan, Y. Sagiv, J.F. Ullman, and M.Y. Vardi. Logical Query Optimization by Proof-Tree Transformation. *JCSS*, 47, 1993, pp. 222-248.

[21] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, Rockville, Maryland (USA), 1989.

[22] Van Gelder, A., Ross, K. A. and Schlipf, J. S. (1991), The Well-Founded Semantics for General Logic Programs, *J. of ACM*, **38**(3), pp. 620–650.