DISJUNCTIVE DATALOG WITH NESTED RULES *

(EXTENDED ABSTRACT)

Sergio Greco
DEIS
Universitá della Calabria
87030 Rende, Italy
greco@si.deis.unical.it

Nicola Leone
Information Systems Dept.
Technical University of Vienna
A-1040 Vienna, Austria
leone@dbai.tuwien.ac.at

Francesco Scarcello
ISI-CNR
c/o DEIS–UNICAL
87030 Rende, Italy
frank@si.deis.unical.it

Abstract

This paper presents an extension of disjunctive datalog (Datalog$^{\vee}$) by nested rules. Nested rules are (disjunctive) rules where elements of the head may be also rules. Nested rules increase the knowledge representation power of Datalog$^{\vee}$ both from a theoretical and from a practical viewpoint. A number of examples show that nested rules allow to naturally model several real world situations that cannot be represented in Datalog$^{\vee}$. An in depth analysis of complexity and expressive power of the language shows that nested rules do increase the expressiveness of Datalog$^{\vee}$ without implying any increase in its computational complexity.

Keywords. Deductive Databases, Disjunctive Datalog, Knowledge Representation, Non-Monotonic Reasoning, Computational Complexity, Expressive Power.

1 Introduction

In this paper, we propose an extension of Datalog$^{\vee}$ by nested rules that we call Datalog$^{\vee,\hookleftarrow}$. Informally, a Datalog$^{\vee,\hookleftarrow}$ rule is a (disjunctive) rule where rules may occur in the head. For instance, $r: \quad A \vee (B \hookleftarrow C) \leftarrow D$, where A and B are atoms and C and D are conjunctions of atoms is a Datalog$^{\vee,\hookleftarrow}$ rule. The intuitive meaning of r is the following: if D is true, then A or B could be derived from r; however, B can be derived from r *only if* C is also true.

*This work has been supported in part by FWF (Austrian Science Funds) under the *project P11580-MAT "A Query System for Disjunctive Deductive Databases"*; by the *Istituto per la Sistemistica e l'Informatica, ISI-CNR*; and by a MURST grant (40% share) under the project "Sistemi formali e strumenti per basi di dati evolute."

Example 1.1 The organizer of a party wants to invite either *susan* or *john* and, in addition, either *mary* or *paul*. This situation can be expressed by means of the following disjunctive Datalog program

$$susan \vee john \leftarrow$$
$$mary \vee paul \leftarrow$$

This program has four stable models giving all possible solutions: $M_1 = \{susan, mary\}$, $M_2 = \{susan, paul\}$, $M_3 = \{john, mary\}$ and $M_4 = \{john, paul\}$.

Suppose now that you know that *john* will attend the party only if *mary* will attend the party too (therefore, inviting *john* makes sense only if also *mary* has been invited). This situation cannot be naturally expressed in disjunctive Datalog whereas can be naturally expressed by means of nested rules.

$$susan \vee (john \hookleftarrow mary) \leftarrow$$
$$mary \vee paul \leftarrow$$

The new program has only three stable models, namely M_1, M_2 and M_3 (see Section 2), that represent the three reasonable alternative sets of persons to be invited. □

Thus, the addition of nested rules allows us to represent real world situations that cannot be represented in plain Datalog$^{\vee}$ programs.

Remark. It is worth noting that nested rules could be simulated by using (possibly unstratified) negation; however, in cases like that above, a nested rule allows us a more direct representation of the reality and it is therefore preferable.

In this paper we will contrast disjunctive Datalog with nested rules (Datalog$^{\vee,\hookleftarrow}$) mainly against plain (i.e., negation free) disjunctive Datalog (Datalog$^{\vee}$), in order to evidentiate the types of disjunctive information that become expressible thanks to the introduction of nested rules. □

The main contributions of the paper are the following:

- We add nested rules to disjunctive Datalog and define an elegant declarative semantics for the resulting language. We show that our semantics generalizes the stable model semantics [12, 7] of disjunctive Datalog programs. Moreover, we show how nested rules can be used for knowledge representation and commonsense reasoning.

- We analyze the complexity and the expressive power of Datalog$^{\vee,\hookleftarrow}$. It appears that, while nested rules do not affect the complexity of the language, they do increase its expressive power. Indeed, as for Datalog$^{\vee}$, brave reasoning is Σ_2^P-complete for Datalog$^{\vee,\hookleftarrow}$ (that is, the complexity is the same). However, Datalog$^{\vee}$ allows to express only a strict subset of Σ_2^P (e.g., even the simple *even* query,[1] asking whether

[1] See example 4.4.

a relation has an even number of elements, is not expressible) [4], while Datalog$^{\vee,\hookleftarrow}$ expresses exactly Σ_2^P (that is, it allows to represent all and only the properties that are computable in polynomial time by a nondeterministic Turing machine endowed with an NP oracle).

2 The Datalog$^{\vee,\neg,\hookleftarrow}$ Language

In this section, we extend disjunctive Datalog by nested rules. For the sake of generality, we will consider also negation in the rules' bodies (defining the language Datalog$^{\vee,\neg,\hookleftarrow}$).

2.1 Syntax

A *term* is either a constant or a variable[2]. An *atom* is $a(t_1, ..., t_n)$, where a is a *predicate* of arity n and $t_1, ..., t_n$ are terms. A *literal* is either a *positive literal* p or a *negative literal* $\neg p$, where p is an atom.

A *nested rule* is of the form:

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

where $A, b_1, \cdots, b_m$ are atoms. If $m = 0$, then the implication symbol "$\hookleftarrow$" can be omitted.

A *rule* r is of the form

$$A_1 \vee \cdots \vee A_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_m, \qquad n > 0, m \geq 0$$

where $b_1, \cdots, b_m$ are atoms, and $A_1, \cdots, A_n$ are nested rules. The disjunction $A_1 \vee \cdots \vee A_n$ is the *head* of r, while the conjunction $b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m$ is the *body* of r; we denote the sets $\{A_1, \cdots, A_n\}$ and $\{b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m\}$ by $Head(r)$ and $Body(r)$, respectively; moreover, we denote $\{b_1, ..., b_k\}$ and $\{\neg b_{k+1}, ..., \neg b_m\}$ by $Body^+(r)$ and $Body^-(r)$, respectively. Notice that atoms occurring in $Head(r)$ stand for nested rules with an empty body. If $n = 1$ (i.e., the head is $\vee$-free), then r is *normal*; if no negative literal appear in r (r is $\neg$-free), then r is *positive*; if $A_1, \cdots, A_n$ are atoms, then r is *flat*. We will use the notation $Body(r)$ and $Head(r)$ also if r is a nested rule. A Datalog$^{\vee,\neg,\hookleftarrow}$ program $\mathcal{P}$ is a set of rules; $\mathcal{P}$ is *normal* (resp., *positive*, *flat*) if all rules in $\mathcal{P}$ are normal (resp. positive, flat). We denote by: (i) Datalog$^{\vee,\hookleftarrow}$, (ii) Datalog$^{\vee,\neg}$, and (iii) Datalog$^{\vee}$, the fragments of Datalog$^{\vee,\neg,\hookleftarrow}$ where we disallow: (i) negation in the body, (ii) nested implication in the head, and (iii) both negation in the body and nested implication in the head, respectively. Moreover, if negation is constrained to be stratified [11], then we will use the symbol $\neg_s$ instead of $\neg$ (e.g., Datalog$^{\vee,\neg_s}$ will denote disjunctive Datalog with stratified negation).

[2]Note that function symbols are not considered in this paper.

Example 2.1 A rule may appear in the head of another rule. For instance,

$$r_1: \qquad a \vee (b \hookleftarrow \neg c) \leftarrow d$$

is an allowed Datalog$^{\vee,\neg,\hookleftarrow}$ rule. Moreover,

$$r_2: \qquad a \vee (b \hookleftarrow c) \leftarrow d$$

is a Datalog$^{\vee,\hookleftarrow}$ rule as well. Neither, r_1 nor r_2 belong to Datalog$^{\vee}$; while

$$r_3: \qquad a \vee b \leftarrow d$$

is in Datalog$^{\vee}$. □

2.2 Semantics

Let $\mathcal{P}$ be a Datalog$^{\vee,\neg,\hookleftarrow}$ program. The *Herbrand universe* $U_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all constants appearing in $\mathcal{P}$. The *Herbrand base* $B_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all possible ground atoms constructible from the predicates appearing in $\mathcal{P}$ and the constants occurring in $U_{\mathcal{P}}$ (clearly, both $U_{\mathcal{P}}$ and $B_{\mathcal{P}}$ are finite). The instantiation of the rules in $\mathcal{P}$ is defined in the obvious way over the constants in $U_{\mathcal{P}}$, and is denoted by $ground(\mathcal{P})$.

A *(total) interpretation* for $\mathcal{P}$ is a subset I of $B_{\mathcal{P}}$. A ground positive literal a is *true* (resp., *false*) w.r.t. I if $a \in I$ (resp., $a \notin I$). A ground negative literal $\neg a$ is *true* (resp., *false*) w.r.t. I if $a \notin I$ (resp., $a \in I$).

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 to the classical notion of truth).

Example 2.2 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 $I = \{c, d\}$. □

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

Example 2.3 For the flat program $\mathcal{P} = \{a \vee b \leftarrow\}$ the interpretations $\{a\}$, $\{b\}$ and $\{a, b\}$ are its models.

For the program $\mathcal{P} = \{a \vee b \leftarrow; \quad c \vee (d \hookleftarrow a) \leftarrow\}$ 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). □

As shown in [10], the intuitive meaning of positive (disjunctive) programs (i.e., Datalog$^{\vee}$ programs) is captured by the set of its minimal models (a model M is minimal if no proper subset of M is a model). However, in presence of negation and nested rules, not all minimal models represent an intuitive meaning for the programs at hand.

To define a proper semantics of Datalog$^{\vee,\neg,\hookleftarrow}$ programs, we define next a suitable notion of *unfounded sets* for disjunctive logic programs with nested rules which extends in a very natural way the analogous notion of unfounded sets given for normal and disjunctive logic programs in [14] and [8, 9], respectively.

Unfounded sets with respect to an interpretation I are essentially set of atoms that are definitely not derivable from the program (assuming I), and, as a consequence, they can be declared false according to the given interpretation.

Definition 2.4 Let $\mathcal{P}$ be a Datalog$^{\vee,\neg,\hookleftarrow}$ program and $I \subseteq B_{\mathcal{P}}$ an interpretation for $\mathcal{P}$. $X \subseteq B_{\mathcal{P}}$ is an *unfounded set* for $\mathcal{P}$ w.r.t. I if, for each $a \in X$, every rule r with a nested rule $r' : a \hookleftarrow Body(r')$ in $Head(r)$,[3] satisfies at least one of the following conditions (we also say r has a witness of unfoundness):

1. $Body(r) \cup Body(r')$ is false w.r.t. I, i.e., at least one literal in $Body(r) \cup Body(r')$ is false w.r.t. I;
2. $(Body^{+}(r) \cup Body^{+}(r')) \cap X \neq \emptyset$;
3. some nested rule in $Head(r)$ is *applied* w.r.t. $I - X$. □

Informally, if a model M includes any unfounded set, say X, then, in a sense, we can get a better model, according to the closed world principle, by declaring false all the atoms in the set X. Therefore, a "supported" model must contain no unfounded set. This intuition is formalized by the following definition of stable models.

Definition 2.5 Let $\mathcal{P}$ be a Datalog$^{\vee,\neg,\hookleftarrow}$ program and $M \subseteq B_{\mathcal{P}}$ be a model for $\mathcal{P}$. M is a *stable model* for $\mathcal{P}$ if it does not contain any non empty unfounded set w.r.t. M (i.e., if both $X \subseteq M$ and $X \neq \emptyset$ hold, then X is not an unfounded set for $\mathcal{P}$ w.r.t. M). □

Example 2.6 Let $\mathcal{P} = \{a \vee b \leftarrow c, \quad b \leftarrow \neg a, \neg c, \quad a \vee c \leftarrow \neg b\}$. Consider $I = \{b\}$. It is easy to verify that $\{b\}$ is not an unfounded set for $\mathcal{P}$ w.r.t. I. Indeed, rule $b \leftarrow \neg a, \neg c$ has no witness of unfoundedness w.r.t. I. Thus, as I is a model for $\mathcal{P}$, then I is a stable model for $\mathcal{P}$ according to Definition 2.4.

Let $\mathcal{P} = \{a \vee (b \hookleftarrow \neg c) \leftarrow d, \quad d \vee c \leftarrow\}$. Consider the model $I = \{b, d\}$. It is easy to verify that $\{b, d\}$ is not an unfounded set w.r.t. I and neither $\{a\}$ nor $\{b\}$ is an unfounded set for $\mathcal{P}$ w.r.t. I. Therefore, I is a stable model of $\mathcal{P}$.

[3] An atom A in $Head(r)$ is seen as a nested rule with empty body $a \hookleftarrow$.

It is easy to see that the stable models of the program $\mathcal{P} = \{susan \vee (john \hookleftarrow mary) \leftarrow, mary \vee paul \leftarrow\}$ of example 1.1 are: $M_1 = \{susan, mary\}$, $M_2 = \{susan, paul\}$, and $M_3 = \{john, mary\}$. $\Box$

We conclude this section by showing that the above definitions of unfounded sets and stable models extend the analogous notions given for normal and disjunctive logic programs.

Proposition 2.7 *Let I be an interpretation for a flat program $\mathcal{P}$. $X \subseteq B_{\mathcal{P}}$ is an unfounded set for $\mathcal{P}$ w.r.t. I according to [8, 9] if and only if X is an unfounded set for $\mathcal{P}$ w.r.t. I according to Definition 2.4.*

Proof. For a flat program $\mathcal{P}$, every nested rule r' is of the form $a \hookleftarrow$. Consequently, Condition 1 and Condition 2 of Definition 2.4 correspond exactly to the analogous conditions of the definition of unfounded set given in [8, 9] (as $Body(r') = \emptyset$). Moreover, in absence of nested rules with nonempty bodies, Condition 3 of Definition 2.4 just says that some head atom is true w.r.t. $I - X$ (which corresponds to Condition 3 of the definition of unfounded set given in [8, 9]). $\Box$

As a consequence, if $\mathcal{P}$ is a non disjunctive flat program, then the notion of unfounded set does coincide with the original one given in [14].

Corollary 2.8 *Let I be an interpretation for a normal flat program $\mathcal{P}$. $X \subseteq B_{\mathcal{P}}$ is an unfounded set for $\mathcal{P}$ w.r.t. I according to [14] if and only if X is an unfounded set for $\mathcal{P}$ w.r.t. I according to Definition 2.4.*

Proof. In [8, 9], it is shown that the Definition of unfounded sets given there, coincides on normal programs with the classical definition of unfounded sets of [14]. The result therefore follows from Proposition 2.7. $\Box$

Theorem 2.9 *Let $\mathcal{P}$ be a flat program and M a model for $\mathcal{P}$. Then, M is a stable model for $\mathcal{P}$ according to [12, 7] if and only if M is a stable model for $\mathcal{P}$ according to Definition 2.5.*

Proof. It follows from Proposition 2.7 and the results in [8, 9]. $\Box$

Moreover, if $\mathcal{P}$ is a positive flat program, then the set of its stable models coincide with the set of its minimal models. Hence, for positive flat programs our stable models semantics coincide with minimal model semantics proposed for such programs in [10].

In fact the stable model semantics defined above, is a very natural extension of the widely accepted semantics for the various (less general) classes of logic programs, since it is based on the same concepts of minimality and supportedness, which follow from the closed world assumption.

3 Complexity and Expressiveness

3.1 Preliminaries

In the context of deductive databases, some of the predicate symbols correspond to database relations (the *extensional (EDB) predicates*), and are not allowed to occur in rule heads; the other predicate symbols are called *intensional (IDB) predicates*. Actual database relations are formed on a fixed countable domain U, from which also possible constants in a Datalog$^{\vee,\neg,\hookleftarrow}$ program are taken.

More formally, a Datalog$^{\vee,\neg,\hookleftarrow}$ program $\mathcal{P}$ has associated a relational database scheme $\mathcal{DB}_{\mathcal{P}} = \{r \mid r$ is an EDB predicate symbol of $\mathcal{P}\}$; thus EDB predicate symbols are seen as relation symbols. A database D on $\mathcal{DB}_{\mathcal{P}}$ is a set of finite relations on U, one for each r in $\mathcal{DB}_{\mathcal{P}}$, denoted by $D(r)$; note that D can be seen as a first-order structure whose universe consists of the constants occurring in D (the *active domain* of D).[4] The set of all databases on $\mathcal{DB}_{\mathcal{P}}$ is denoted by $\mathbf{D}_{\mathcal{P}}$.

Given a database $D \in \mathbf{D}_{\mathcal{P}}$, $\mathcal{P}_D$ denotes the following program:

$$\mathcal{P}_D = \mathcal{P} \cup \{r(t) \leftarrow \ \mid r \in \mathcal{DB}_{\mathcal{P}} \wedge t \in D(r)\}.$$

Definition 3.1 A (*bound* Datalog$^{\vee,\neg,\hookleftarrow}$) *query* $\mathcal{Q}$ is a pair $\langle \mathcal{P}, G\rangle$, where $\mathcal{P}$ is a Datalog$^{\vee,\neg,\hookleftarrow}$ program and G is a ground literal (the *query goal*). Given a database D in $\mathbf{D}_{\mathcal{P}}$, the *answer* of $\mathcal{Q}$ on D is *true* if there exists a stable model M of $\mathcal{P}_D$ such that G is true w.r.t. M, and *false* otherwise. [5] $\Box$

Constraining $\mathcal{P}$ on fragments of Datalog$^{\vee,\neg,\hookleftarrow}$, we obtain smaller sets of queries. More precisely, we say that $\mathcal{Q} = \langle \mathcal{P}, G\rangle$ is a Datalog^X query, where $X \subseteq \{\vee, \hookleftarrow, \neg\}$, if $\mathcal{P}$ is a Datalog^X program (and G is a ground literal). Clearly, $\neg$ could also be replaced by $\neg_s$ to obtain queries of stratified fragments of Datalog$^{\vee,\neg,\hookleftarrow}$.

The constants occurring in $\mathcal{P}_D$ and G define the active domain of query $\mathcal{Q} = \langle \mathcal{P}, G\rangle$ on the database D.

Observe that, in general, two queries $\langle \mathcal{P}, G\rangle$ and $\langle \mathcal{P}, \neg G\rangle$ on the same database need not give symmetric answers. That is, if e.g. $\langle \mathcal{P}, G\rangle$ answers yes for D, it may be possible that also $\langle \mathcal{P}, \neg G\rangle$ answers yes for D.

A bound query defines a Boolean C-generic query of [1], i.e., a mapping from $\mathbf{D}_{\mathcal{P}}$ to $\{true, false\}$. As common, we focus in our analysis of the expressive power of a query language on generic queries, which are those mappings whose result is invariant under

[4]We use here active domain semantics (cf. [1]), rather then a setting in which a (finite) universe of D is explicitly provided [5, 3, 15]. Note that Fagin's Theorem and all other results to which we refer remain valid in this (narrower) context; conversely, the results of this paper can be extended to that setting.

[5]We consider brave (also called possibility) semantics in this paper; however, complexity and expressiveness of cautious (also called skeptical) semantics can be easily derived from it.

renaming the constants in D with constants from U. Genericity of a bound query $\langle \mathcal{P}, G\rangle$ is assured by excluding constants in $\mathcal{P}$ and G. As discussed in [1, p. 421], this issue is not central, since constants can be provided by designated input relations; moreover, any query goal $G = (\neg)p(\cdots)$ can be easily replaced by a new goal $G' = (\neg)q$ and the rule $q \leftarrow p(\cdots)$, where q is a propositional letter. In the rest of this paper, we thus implicitly assume that constants do not occur in queries.

Definition 3.2 Let $\mathcal{Q} = \langle \mathcal{P}, G\rangle$ be a (constant-free) query. Then the *database collection* of $\mathcal{Q}$, denoted by $\mathcal{EXP}(\mathcal{Q})$, is the set of all databases D in $\mathbf{D}_{\mathcal{P}}$ for which the answer of $\mathcal{Q}$ is true.

The *expressive power* of Datalog^X ($X \subseteq \{\vee, \hookleftarrow, \neg\}$), denoted $\mathcal{EXP}(\text{Datalog}^X)$, is the family of the database collections of all Datalog^X queries, i.e.,

$$\mathcal{EXP}[\text{Datalog}^X] = \{\mathcal{EXP}(\mathcal{Q}) \mid \mathcal{Q} \text{ is a constant-free Datalog}^X \text{ query}\}. \qquad \Box$$

The expressive power will be related to database complexity classes, which are as follows. Let C be a Turing machine complexity class (e.g., P or NP), $\mathbf{R}$ be a relational database scheme, and $\mathbf{D}$ be a set of databases on $\mathbf{R}$.[6] Then, $\mathbf{D}$ is C*-recognizable* if the problem of deciding whether $D \in \mathbf{D}$ for a given database D on $\mathbf{R}$ 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 language (fragment of Datalog$^{\vee,\neg,\hookleftarrow}$) L coincides with some class *DB*-C, we say that the given language *captures* C, and denote this fact by $\mathcal{EXP}[\text{L}] = C$.

Recall that the classes Σ_k^P, Π_k^P of the polynomial hierarchy [13] are defined by $\Sigma_0^P = \text{P}$, $\Sigma_{i+1}^P = \text{NP}^{\Sigma_i^P}$, and $\Pi_i^P = \text{co-}\Sigma_i^P$, for all $i \geq 0$. In particular, $\Pi_0^P = \text{P}$, $\Sigma_1^P = \text{NP}$, and $\Pi_1^P = \text{co-NP}$.

3.2 Results

Theorem 3.3 $\mathcal{EXP}[\mathit{Datalog}^{\vee,\neg_s}] \subseteq \mathcal{EXP}[\mathit{Datalog}^{\vee,\hookleftarrow}]$

Proof. *(Sketch)* It is sufficient to show that every Datalog$^{\vee,\neg_s}$ query can be rewritten into an equivalent Datalog$^{\vee,\hookleftarrow}$ query.

Every Datalog$^{\vee,\neg_s}$ program (i.e., disjunctive Datalog program with stratified negation) can be rewritten in a program where negative literals appear only in the body of rules of the form

$$r: \; p(X) \leftarrow q(X), \; \neg s(X)$$

where p and s are not mutually recursive and r is the only rule having p as head predicate symbol.

[6] As usual, adopting the data independence principle, it is assumed that $\mathbf{D}$ is generic, i.e., it is closed under renamings of the constants in U.

Now, a stratified rule like r can be equivalently represented by the following Datalog$^{\vee,\hookleftarrow}$ rule:

$$p(X) \vee (s(X) \hookleftarrow s(X)) \leftarrow q(X) \qquad \square$$

Corollary 3.4 $\Sigma_2^P \subseteq \mathcal{EXP}[Datalog^{\vee,\hookleftarrow}]$

Proof. From [4], $\Sigma_2^P \subseteq \mathcal{EXP}[\text{Datalog}^{\vee,\neg_s}]$. Therefore, the result follows from Theorem 3.3. $\square$

Corollary 3.5 $\mathcal{EXP}[Datalog^{\vee}] \subset \mathcal{EXP}[Datalog^{\vee,\hookleftarrow}]$

Proof. From [4], Datalog$^{\vee}$ can express only a strict subset of Σ_2^P (e.g., the simple *even* query, deciding whether the number of tuples of a relation is even or odd, is not expressible in Datalog$^{\vee}$ [4]). Therefore, the result follows from Corollary 3.4. $\square$

We next prove that the inclusion of Corollary 3.4 is not proper.

Theorem 3.6 $\mathcal{EXP}[Datalog^{\vee,\neg,\hookleftarrow}] \subseteq \Sigma_2^P$.

Proof. To prove the theorem, we have to show that for any Datalog$^{\vee,\neg,\hookleftarrow}$ query $\mathcal{Q} = \langle \mathcal{P}, G\rangle$, recognizing whether a database D is in $\mathcal{EXP}(\mathcal{Q})$ is in Σ_2^P.

Observe first that recognizing whether a given model M of a Datalog$^{\vee,\neg,\hookleftarrow}$ program is stable can be done in co-NP. Indeed, to prove that M is not stable, it is sufficient to guess a subset X of M and check that it is an unfounded set. (Note that, since $\mathcal{Q}$ is fixed, $ground(\mathcal{P}_D)$ has size polynomial in D, and can be constructed in polynomial time.)

Now, D is in $\mathcal{EXP}(\mathcal{Q})$ iff there exists a stable model M of $\mathcal{P}_D$ such that $G \in M$. To check this, we may guess an interpretation M of $\mathcal{P}_D$ and verify that: (i) M is a stable model of $\mathcal{P}_D$, and (ii) $G \in M$. From the observation above, (i) is done by a single call to an NP oracle; moreover, (ii) is clearly polynomial. Hence, this problem is in Σ_2^P. Consequently, recognizing whether a database D is in $\mathcal{EXP}(\mathcal{Q})$ is in Σ_2^P. $\square$

Corollary 3.7 $\mathcal{EXP}[Datalog^{\vee,\neg,\hookleftarrow}] = \mathcal{EXP}[Datalog^{\vee,\hookleftarrow}] = \mathcal{EXP}[Datalog^{\vee,\neg}] = \Sigma_2^P$ $\square$

4 Some Examples

In this section we present some examples to show that classical graph problems can be expressed in Datalog$^{\vee,\hookleftarrow}$. For the sake of presentation we shall use the predicate $\neq$ which can be easily emulated by Datalog$^{\vee,\hookleftarrow}$. In the following examples we assume to have the graph $G = (V, E)$ stored by means of the unary relation v and the binary relation e.

Example 4.1 *Spanning tree.* The following program computes a spanning tree rooted in the node a for a graph $G = (V, E)$. The set of arcs in the spanning tree are collected by means of the predicate st.

$$\begin{array}{lcl} st(root, a). & & \\ st(X,Y) \vee (no\_st(X,Y) \hookleftarrow no\_st(X,Y)) & \leftarrow & st(\_, X),\ e(X,Y). \\ no\_st(X,Y) & \leftarrow & st(X',Y), X \neq X'. \end{array}$$

Observe that the nested rule forces to select for each value of Y a unique tuple for $st(X,Y)$. Indeed, if some stable model M contains two tuples of the form $t_1 = st(x_1, y)$ and $t_2 = st(x_2, y)$, from the last rule, M must contain also the tuples $no\_st(x_1, y)$ and $no\_st(x_2, y)$. But this implies that also the interpretation $N \subseteq M - \{t_i\}$ for $t_i \in \{t_1, t_2\}$ is a stable model and, therefore, M is not minimal. On the other side, assume now that there is some stable model M containing a tuple $no\_st(x', y)$ but not containing tuples of the form $st(x, y)$ for $x \neq x'$. This means that the tuple $no\_st(x', y)$ cannot be derived from the last rule and, therefore, it must belong to some unfounded set w.r.t. M.

Thus, there is a one-to-one correspondence between the stable models of the program and the spanning trees rooted in a of the graph. □

Example 4.2 *Simple path.* In this example we compute a simple path in a graph G, i.e., a path passing through every node just once (if any). The set of tuples in the simple path are collected by means of the predicate sp below defined:

$$\begin{array}{lcl} sp(root, X) \vee (no\_sp(root, X) \hookleftarrow no\_sp(root, X)) & \leftarrow & e(X, \_). \\ sp(X,Y) \vee (no\_sp(X,Y) \hookleftarrow no\_sp(X,Y)) & \leftarrow & sp(W,X),\ e(X,Y). \end{array}$$

$$\begin{array}{lcl} no\_sp(X,Y) & \leftarrow & sp(X',Y),\ X' \neq X. \\ no\_sp(X,Y) & \leftarrow & sp(X,Y'),\ Y' \neq Y. \end{array}$$

As for the program computing a spanning tree, the nested rule forces to select for each value of X a unique tuple for $sp(X,Y)$ and for each value of Y a unique tuple for $sp(X,Y)$. The nested rules impose the constraint that the set of tuples for sp defines a chain. Thus, the first nested rule is used to select the starting node of the simple path, whereas the second nested rule is used to select the set of arcs belonging to the simple path.

The above program can be used to define the hamiltonian path problem checking if a graph G has simple path passing through all nodes (hamiltonian path). Therefore, the hamiltonian graph problem can be defined by adding the check that all nodes in G are in the simple path. □

Example 4.3 *Shortest path.* In this example we assume to have a weighted directed graph $G = (V, E)$. We assume that the database domain contains a finite subset of the integer numbers and that the weight argument of the arcs takes values from this domain. We assume also that the minimum weight of all paths between two nodes takes values from this domain. The arcs of the graph are stored by means of tuples of the form $e(x,y,c)$ where c is the weight of the arc from x to y. The minimum weights of the paths from a source node a to every node in the graph can be defined as follows:

$$\begin{array}{lll} mp(a,0) & \leftarrow & \\ mp(Y,C) \vee (no\_mp(Y,C) \hookleftarrow no\_mp(Y,C)) & \leftarrow & mp(X,C_1), e(X,Y,C_2), C = C_1 + C_2. \\ no\_mp(Y,C) & \leftarrow & mp(Y,C'),\ C' < C. \end{array}$$

The predicate mp computes, for each node x, the minimum distance from the source node a to the node x. A stable model M contains for each tuple $mp(y,c')$ in M all tuples of the form $no\_mp(y,c)$ with $c > c'$. Thus, a tuple $mp(y,c)$ is in M iff there is no tuple $no\_mp(y,c)$ in M, i.e., if all tuples in no_mp with first argument y have cost greather than c. □

Example 4.4 *Even query.* We are given a relation d and we want to check whether its cardinality is even or not. This can be done by first defining a linear order on the elements of the relation and, then, checking whether the number of elements in the ordering is even.

$$\begin{array}{l} succ(root, root). \\ succ(X,Y) \vee (no\_succ(X,Y) \hookleftarrow no\_succ(X,Y)) \ \leftarrow \ succ(\_,X),\ d(Y). \end{array}$$

$$\begin{array}{lll} no\_succ(X,Y) & \leftarrow & succ(X,Y'),\ Y' \neq Y,\ Y' \neq root,\ d(Y). \\ no\_succ(X,Y) & \leftarrow & succ(X',Y),\ X' \neq X,\ d(X). \\ & & \\ odd(X) & \leftarrow & succ(root,X), X \neq root. \\ even(X) & \leftarrow & odd(Z), succ(Z,X). \\ odd(X) & \leftarrow & even(Z), succ(Z,Y). \\ even\_rel & \leftarrow & even(X), \neg has\_a\_succ(X). \\ has\_a\_succ(X) & \leftarrow & d(X),\ succ(X,\_). \end{array}$$

The first four rules define a linear order on the elements of the relation d (by using a nested implication). Once a linear order has been defined on the domain it is easy to check, by a simple stratified program, whether the cardinality is even. Thus, the predicate $even\_rel$ is true iff the relation d has an even number of elements.

Therefore, Datalog$^{\vee,\hookleftarrow}$ expresses the even query,[7] while it cannot be expressed in Datalog$^{\vee}$ [4]. □

We conclude by observing that the problems of the above examples could be expressed by means of disjunctive datalog with (unstratified) negation. However, programs with unstratified negation are neither intuitive nor efficiently computable (while Datalog$^{\vee,\hookleftarrow}$ has nice computational properties – see Section 5).

[7] Recall that both $\neq$ and stratified negation are used for simplicity, bu they can be easily emulated in Datalog$^{\vee,\hookleftarrow}$.

	Datalog$^{\vee,\hookleftarrow}$	**Datalog**$^{\vee}$	**Datalog**$^{\vee,\neg}$	**Datalog**$^{\vee,\neg,\hookleftarrow}$
Expressive Power	$= \Sigma_2^P$	$\subset \Sigma_2^P$	$= \Sigma_2^P$	$= \Sigma_2^P$
Data Complexity	Σ_2^P-complete	Σ_2^P-complete	Σ_2^P-complete	Σ_2^P-complete

Table 1: *Expressibility and complexity results on Datalog*$^{\vee,\neg,\hookleftarrow}$

5 Conclusion

We have presented an extension of Disjunctive Datalog by nested rules. We have shown the suitability of the language to naturally express complex knowledge-based problems, which are not expressible by Datalog$^{\vee}$. A formal definition of the semantics of Datalog$^{\vee,\neg,\hookleftarrow}$ programs has been provided, and we have shown that it is a generalization of the classical stable model semantics. Finally, we have carefully analyzed both data-complexity and expressiveness of Datalog$^{\vee,\neg,\hookleftarrow}$ under the possibility (brave) semantics.

The results on the data-complexity and the expressiveness of Datalog$^{\vee,\neg,\hookleftarrow}$ are compactly represented in Table 1. [8]

Each column in Table 1 refers to a specific fragment of Datalog$^{\vee,\neg,\hookleftarrow}$. The table clearly shows that the addition of nested rules does not increase the complexity of disjunctive Datalog; indeed, brave reasoning for Datalog$^{\vee,\hookleftarrow}$ is Σ_2^P-complete as for Datalog$^{\vee}$. Nevertheless, nested rules do increase the expressive power, as Datalog$^{\vee,\hookleftarrow}$ allows to express all Σ_2^P database properties; while, Datalog$^{\vee}$ expresses only a strict subset of them (e.g., the simple *even* query, that decides whether a relation has an even number of tuples, cannot be expressed in Datalog$^{\vee}$).

Clearly, the power of Datalog$^{\vee,\hookleftarrow}$ does not exceed that of Datalog$^{\vee,\neg}$, as nested rules could be simulated by means of unstratified negation. However, the increase of expressiveness w.r.t. Datalog$^{\vee}$ confirms that nested rule allow to express some useful forms of disjunctive information which are not expressible in plain disjunctive Datalog.

Ongoing work concerns the definition of a fragment of Datalog$^{\vee,\hookleftarrow}$ for which one stable model can be computed in polynomial time; this fragment, under nondeterministic semantics, allows to express *all* polynomial time properties. Moreover, the investigation of abstract properties of Datalog$^{\vee,\hookleftarrow}$ would also be interesting to see whether this language can be characterized as for the stable model semantics [2].

References

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

[8] Note that the results on data-complexity are immediately derived from the expressibility results of Section 3.2.

[2] S. Brass and J. Dix. Characterizations of the Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, 1997, forthcoming.

[3] Chandra, A., Harel, D. (1982), Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99–128.

[4] Eiter, T., Gottlob, G. and Mannila, H. (1994), Adding Disjunction to Datalog, *in* "Proc. ACM PODS-94," pp. 267–278.

[5] Fagin R. (1974), Generalized First-Order Spectra and Polynomial-Time Recognizable Sets, in *Complexity of Computation (R. Karp, Ed.)*, SIAM-AMS Proc., Vol. 7, pp. 43-73.

[6] Gelfond, M., Lifschitz, V. (1988), The Stable Model Semantics for Logic Programming, *in Proc. of Fifth Conf. on Logic Programming,* pp. 1070–1080, MIT Press.

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

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

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

[10] Minker, J. (1982), On Indefinite Data Bases and the Closed World Assumption, *in* "Proc. of the 6^{th} Conference on Automated Deduction (CADE-82)," pp. 292–308.

[11] Przymusinski, T. (1988), On the Declarative Semantics of Deductive Databases and Logic Programming, *in* "Foundations of deductive databases and logic programming," Minker, J. ed., ch. 5, pp.193–216, Morgan Kaufman, Washington, D.C.

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

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

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

[15] Vardi, M. (1982), Complexity of relational query languages, *in* "Proceedings 14th ACM STOC," pp. 137–146.