Journal of Logic and Computation, 7(6): 805–824, 1997.

On the Indiscernibility of Individuals in Logic Programming*

Thomas Eiter† Georg Gottlob‡ Nicola Leone‡

Abstract

According to Leibniz' principle, two individuals a and b are indiscernible, if they share the same properties. Indiscernibility of objects provides a potential for optimization in deductive systems, and has e.g. been exploited in the area of active database systems. In this paper, we address the issue of indiscernibility in logic programs and outline possible benefits for computation. After a formal definition of the notion of indiscernibility, we investigate some basic properties. The main contribution is then an analysis of the computational cost of checking indiscernibility of individuals (i.e. constants) in logic programs without function symbols, which we pursue in detail for ground logic programs. For the concern of query optimization, they show that online computation of indiscernibility is expensive, and thus suggest to adopt an offline strategy, which may pay off for certain computational tasks.

1 Introduction

Logic programming, extended with negation and disjunction, is nowadays widely recognized as a powerful tool for knowledge representation and commonsense reasoning [5].

After quite some efforts on identifying the most intuitive semantics [46, 19, 36, 47, 38, 39], as well as useful linguistic and epistemic extensions of logic programs [26, 25, 18, 20, 24, 11], the logic programming community is recently paying more attention to discovering both semantical and computational properties of logic programs, cf. [31, 32, 27, 43, 13, 9, 28, 7, 44], which may strengthen the theoretical foundations of the field, provide a better understanding of the various semantics, and help in the design of efficient algorithm for the computation.

This paper addresses the property of indiscernibility of individuals (constants) in logic programs.

*This work was supported in part by FWF (Austrian Science Foundation) under *project P11580-MAT*, and by the *Istituto per la Sistemistica e l'Informatica, ISI-CNR.*

†Institut für Informatik, Universität Gießen, D-35392 Arndtstraße 2, Germany. Email: `eiter@informatik.uni-giessen.de`

‡Christian Doppler Laboratory for Expert Systems, Information Systems Department, TU Vienna, A-1040 Wien, Paniglgasse 16, Austria. Email: `(gottlob|leone)@dbai.tuwien.ac.at`

Intuitively, for semantics based on the "canonical model" approach, which give a unique and always existing model M as the meaning of the logic program (like, e.g., the *perfect* [36] or the *well-founded* [47] semantics), two constants a and b are globally indiscernible if replacing both a by b and b by a in M does leave the canonical model M unchanged. In this case, global indiscernibility of a from b implies that an atom, say $p(a,t)$, is true in the canonical model M if and only if $p(b,t)$ is true in M as well.

Indiscernibility can be also limited to a set Q of predicates (local indiscernibility). In this case, we limit the a/b replacement to the atoms with predicate Q. Thus, we say that a and b are Q-indiscernible if replacing both a by b and b by a in the atoms with predicate Q of M does leave the canonical model M unchanged.

For instance, consider a positive logic program with predicates v, e and $path$, where v and e specify the vertices and the edges of a directed graph $G = (V, E)$, respectively, and $path$ defines the reflexive and transitive closure of e in the usual manner. Then, two vertices a and b are $\{path\}$-indiscernible precisely if they belong to the same strongly connected component of G. Moreover, an equivalence class of all constants that is $\{path\}$-indiscernible from a vertex, say a, is a strongly connected component of G. In particular, the graph G is strongly connected if and only if there is a single such class, i.e., all vertices are $\{path\}$-indiscernible (see Example 10 in Section 3).

For semantics based on "multiple models" (like, e.g., the stable model semantics [19, 38, 20]), the notion of indiscernibility is slightly more involved, as the non-determinism coming from the multiplicity of the (preferred) models must be taken into account. For instance, consider indiscernibility under stable model semantics. Then, a is Q-indiscernible from b if, for each stable model M of P, replacing both a by b and b by a in the atoms with predicate Q of M generates an interpretation which coincides with some stable model of P on the atoms from Q. Global indiscernibility is then defined in the obvious way. (We refer the reader to Section 3 for formal definitions.)

Indiscernibility of constants in a logic program can be used for the following purposes.

For example, it can be useful to know about indiscernibility if one is concerned with finding a model of a program that contains a particular fact. Such a model finding task arises e.g. in the in course of diagnosis, where one wants to find a model that amounts to a diagnosis.

Suppose we want to find a model of a program P in which an observed fact $p(a)$ is true, and we know that a and b are globally-indiscernible.

If we succeed to find a model M of P in which $p(b)$ is true, then we can obtain the desired model by replacing b by a in M and vice versa.

Indiscernibility may also be useful in reasoning. If we know that a and b are $\{p\}$-indiscernible, and we succeed to prove $p(a)$ (under cautious or brave semantics), then we can infer that also $p(b)$ must be provable. For example, consider

$$P = \{p(a) \leftarrow \neg p(b); \quad p(b) \leftarrow \neg p(a)\}$$

This program has the stable models $M_1 = \{p(a), \neg p(b)\}$, $M_2 = \{\neg p(a), p(b)\}$; thus, a and b are $\{p\}$-indiscernible. If we succeed to prove $p(a)$ credulously (e.g., by generating the stable model M_1), then we can infer that also $p(b)$ is credulously provable. On the other hand, if we want to infer $p(a)$ under cautious semantics, and we generate M_1, then we can immediately infer from indiscernibility that $p(a)$ is not provable from P, since $p(b)$ is false in M_1, and hence there exists a model of P in which $p(a)$ is false.

Since a and b are globally indiscernible in this program, the stable model M_2 can be immediately generated from M_1 by replacing a by b and b by a.

Unfortunately, as will be clear from the next sections, local indiscernibility comes at a high computational cost, and so it is not useful to determine indiscernibility on-line; it may be useful, however, to determine it off-line and use it when reasoning is done.

The contribution of this paper is mainly twofold. Firstly, we define precisely the notion of indiscernibility in the framework of logic programming, demonstrate some basic properties of indiscernibility and provide a number of examples which explain the intuitive meaning of indiscernibility.

Secondly, we provide an in depth analysis of the complexity of both local and global indiscernibility in ground logic programming. We pay also attention to the impact of syntactical restrictions on the complexity of deciding indiscernibility. In fact, we determine the complexity of deciding indiscernibility for normal positive, normal stratified, normal general, disjunctive positive, and disjunctive general logic programs.

The complexity analysis of ground logic programs highlights some important points:

- Deciding whether two given constants a, b are indiscernible in a given ground normal or disjunctive logic program P, is intractable and complete for a class Π_i^P of the polynomial hierarchy, where $i \leq 3$. This applies to both global and local indiscernibility (where the local predicates Q are part of the input),[1] with the exception of normal logic programs that use stratified negation.

- The complexity of global indiscernibility coincides with the complexity of cautious reasoning under the stable model semantics, and is P-complete for stratified normal logic programs, co-NP-complete for normal logic programs, and Π_2^P-complete for disjunctive logic programs.

- In case of intractability, local indiscernibility is always one level higher up in the polynomial hierarchy than global indiscernibility.

- In case of disjunctive programs, hard cases of indiscernibility checking are already present with positive (i.e., negation-free) programs.

In case of nonground logic programs, the complexities increase by an exponential (from P to EXPTIME, co-NP to co-NEXP, Π_2^P to co-NEXP$^{\mathrm{NP}}$ and so on), and thus indiscernibility checking has provably an exponential lower bound.

The complexity analysis carried out in this paper refers to indiscernibility under *total* model semantics. It is worth noting that several interesting *partial* model and disjunctive state semantics have been also defined for logic programs [6, 10, 16, 29, 38, 39, 40, 42, 47, 49]; we will discuss the complexity of deciding indiscernibility for some partial model semantics in Section 5.

[1] As follows immediately from the results we derive, the same complexity results hold if the local predicates Q are fixed.

To our knowledge, no other notion of indiscernibility has been defined in the framework of logic programming. Different notions of indiscernibility have been defined in other contexts [1, 2, 8]. In particular, in [2] indiscernibility is used to optimize active databases computations. In [1] indiscernibility is employed to determine some expressibility results. Finally, in [8] indiscernibility allows to identify the extensions of a default theory which holds some nice properties and should therefore be preferred over the others.

The remainder of the paper is organized as follows. Section 2 contains preliminaries on both Logic Programming and Complexity Theory. Section 3 defines formally the notion of indiscernibility. Section 4 is devoted to the analysis of the complexity of indiscernibility on ground programs. Finally, we briefly consider nonground programs and draw our conclusions in Section 5. The appendix overviews some problems on abductive logic programming which are utilized to derive hardness results in the complexity section.

2 Preliminaries

2.1 Logic Programming

This section recalls the basic concepts of logic programming. The syntax of logic programs is given first; then, stable semantics [19, 47] is provided. Stratified programs [3] and their properties are finally presented.

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

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_m, \qquad n \geq 1,\ m \geq 0$$

where $a_1, \cdots, a_n, b_1, \cdots, b_m$ are atoms. 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. If $n = 1$ (the head is $\vee$-free), then r is *normal*; if $m = 0$ (the body is $\neg$-free), then r is *positive.* A *(disjunctive) program* P is a finite set of rules. Program P is *normal* (resp., *positive*) if all rules in P are normal (resp. positive).

A term, an atom, a literal, a rule or program is *ground* if no variable appears in it. A ground program is also called *propositional* program.

Let P be a program. The *Herbrand universe* U_P of P is the set of all constants appearing in P. The *Herbrand base* B_P of P is the set of all possible ground atoms constructible from the predicates appearing in the rules of P and the constants occurring in U_P (clearly, both U_P and B_P are finite). Given a rule r occurring in a program P, a *ground instance* of r is a rule obtained from r by replacing every variable X in r by $\sigma(X)$, where σ is a mapping from the variables occurring in r to the constants in U_P. We denote by $ground(P)$ the (finite) set of the ground instances of the rules occurring in P.

An *(total) interpretation* for P is a subset I of B_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., $A \in I$).

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

Let r be a ground rule in $ground(P)$. Rule r is *satisfied* (or *true*) w.r.t. I if its head is true w.r.t. I (i.e., some head atom is true) or its body is false (i.e., some body literal is false) w.r.t. I.

A *model* for P is an interpretation M for P such that every rule $r \in ground(P)$ is true w.r.t. M. A model M for P is *minimal* if no proper subset of M is a model for P. The set of all minimal models for P is denoted by $\mathrm{MM}(LP)$.

The first proposal for assigning a semantics to a disjunctive logic program is due to Minker, who presented in [33] a model-theoretic semantics for positive programs. According to [33], the semantics of a program LP is given by the set $\mathrm{MM}(P)$ of the minimal models for P. Observe that every program admits at least one minimal model, that is, for every program P, $\mathrm{MM}(P) \neq \emptyset$ holds.

Example 1 For the positive program P_1:

$$a \vee b \leftarrow,$$

the (total) interpretations $\{a\}$ and $\{b\}$ are its minimal models (i.e., $\mathrm{MM}(P) = \{\ \{a\},\ \{b\}\ \}$). As another example, for the program P_2:

$$\begin{aligned} a \vee b &\leftarrow\ , \\ b &\leftarrow a, \\ a &\leftarrow b, \end{aligned}$$

$\{a, b\}$ is the only minimal model. □

As far as general programs (i.e., programs where negation may appear) are concerned, a number of semantics has been recently proposed [9, 20, 33, 37, 38, 39, 40, 41] (see [4, 12, 29] for comprehensive surveys).

A generally acknowledged proposal is the extension of the stable model semantics [19] to take into account disjunction [20, 38].

Definition 2 [38] *Let I be an interpretation for a program P. The* Gelfond-Lifschitz transformation *("GL transformation", for short) of P with respect to I, denoted by P^I, is the positive program defined as follows:*

$$\begin{aligned} P^I = \{a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k \mid\ & a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_m \\ & \textit{is in } ground(P) \textit{ and } b_i \notin I, \textit{ for all } k < i \leq m\}. \end{aligned}$$

I is a stable model *for P if $I \in \mathrm{MM}(P^I)$ (that is I is a minimal model of the positive program P^I). The set of all stable models for P is denoted by $STM(P)$.* □

Example 3 Let P be the following program:

$$\begin{aligned} a \vee b &\leftarrow c, \\ b &\leftarrow \neg a, \neg c, \\ a \vee c &\leftarrow \neg b. \end{aligned}$$

Consider $I = \{b\}$. Then, the program P^I is:

$$\begin{aligned} a \vee b &\leftarrow c, \\ b &\leftarrow . \end{aligned}$$

It is easy to verify that I is a minimal model for P^I; thus, I is a stable model for P. □

Clearly, if P is positive then P^I coincides with $ground(P)$. It turns out that, for a positive program, minimal and stable models coincide.

A normal positive program P has exactly one stable model which coincide with the least model of P (i.e., it is the unique minimal model of P). When negation is allowed, however, even a normal program can admit several stable models. In this respect, an interesting class of normal programs is the following. A normal program P is *(locally) stratified* if each atom $p \in B_P$ can be associated a positive integer $l(p)$ such that, for each rule $r \in ground(P)$ with head p and each ground atom q, the following holds: (i) if $\neg q$ occurs in the body of r, then $l(p) > l(q)$; (ii) if q occurs in the body of r, then $l(p) \geq l(q)$. In particular, if all ground atoms with the same predicate have the same number associated, then P is globally stratified.

Every stratified normal program P has the nice property of admitting precisely one stable model.

Proposition 4 Let P be a normal logic program. If P is stratified, then P has precisely one stable model.

Concluding we recall the complexity of recognizing stable models, for both normal and disjunctive programs.

Proposition 5 (cf. [31]) Given a normal propositional logic program P and an interpretation M, deciding whether M is a stable model of P is possible in polynomial time.

Proposition 6 (cf. [13]) Given a disjunctive propositional logic program P and an interpretation M, deciding whether M is a stable model of P is co-NP-complete. Hardness holds even if P is positive.

2.2 Complexity Theory

For NP-completeness and complexity theory, cf. [35]. The classes Σ_k^P, Π_k^P and Δ_k^P of the *Polynomial Hierarchy (PH)* (cf. [45]) are defined as follows:

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathrm{P} \quad \text{and for all } k \geq 1, \quad \Delta_k^P = \mathrm{P}^{\Sigma_{k-1}^P},\ \Sigma_k^P = \mathrm{NP}^{\Sigma_{k-1}^P},\ \Pi_k^P = \text{co-}\Sigma_k^P.$$

In particular, NP $= \Sigma_1^P$, co-NP $= \Pi_1^P$, and $\Delta_2^P = \mathrm{P}^{\mathrm{NP}}$. Here P^C and NP^C denote the classes of problems that are solvable in polynomial time on a deterministic (resp. nondeterministic) Turing machine with an oracle for any problem π in the class C. The oracle replies to a query in unit time, and thus, roughly speaking, models a call to a subroutine for π that is evaluated in unit time. If C has complete problems, then instances of any problem π' in C can be solved in polynomial time using an oracle for any C-complete problem π, by transforming them into instances of π; we refer to this by stating that an oracle for C is used. Notice that all classes C considered here have complete problems.

Observe that for all $k \geq 1$,

$$\Sigma_k^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P \subseteq \mathrm{PSPACE};$$

each inclusion is widely conjectured to be strict. Note that, by the rightmost inclusion, all these classes contain only problems that are solvable in polynomial space. They allow, however, a finer grained distinction between NP-hard problems that are in PSPACE.

There is an exponential analog of PH, called the *Weak EXP Hierarchy (WEXPH)* [22], whose classes ExpΔk, ExpΣ_k^P, and ExpΠ_k^P are defined like the classes Δ_k^P, Σ_k^P, and Π_k^P, with the only difference that Turing machines have an exponential bound on the execution time rather than a polynomial. In particular, ExpΔ_0^P = ExpP = EXPTIME, ExpΣ_1^P = ExpNP = NEXP (nondeterministic exponential time), ExpΠ_1^P = Expco-NP = co-NEXP, ExpΠ_2^P = co-NEXP$^{\mathrm{NP}}$, and so on. Intuitively, the classes of WEXPH comprise problems whose time complexity is within an exponential factor to the analog classes in PH. Every problem which is complete for a class of WEXPH has provably exponential worst case complexity, and is thus provably intractable; for complete problems within PH, this is currently not known.

3 Indiscernibility

The semantics of a program P is a set Sem_P of models (e.g., the set MM(P) of minimal models or the set $STM(P)$ of stable models). In this section we assume that a program P and its semantics Sem_P is given.

Let Q be a set of predicates and I a set of ground literals. We denote by I_Q the restriction of I to the predicates from Q, that is, the set of the literals of I whose predicate is in Q. Moreover, given two constants $a, b \in U_P$, we denote by $I[a,b]$ the set of literals obtained by replacing all occurrences of a and b in I by b and a, respectively.

Definition 7 *Given a set Q of predicates of P. Two constants $a, b \in U_P$ are Q-*indiscernible*, denoted $a \equiv_Q b$, if, for each model $M \in Sem_P$, there exists a model $M' \in Sem_P$ such that $M_Q[a,b] = M'_Q$.*
Constants $a, b \in U_P$ are (globally) indiscernible*, denoted $a \equiv b$, if $a \equiv_Q b$, where Q is the set of all predicates of P.* □

It is worth noting that in the literature the semantics of a logic program is not always defined as a set of total models (that we consider here). There are proposals where the semantics is a set of *partial* models (see, e.g., [6, 16, 38, 42, 47, 49]), and other proposals where the semantics is not even represented by models (see, e.g., [10, 29, 39, 40]). Definition 7 of indiscernibility applies to partial model semantics [6, 16, 38, 42, 47, 49] without any change. For semantics which are not based on models, the extension of Definition 7 would be more involved.

Proposition 8 For any program P, the relations $\equiv$ and $\equiv_Q$ are equivalence relations on its Herbrand universe U_P.

Proof We have to check that $\equiv$ and $\equiv_Q$ satisfy reflexivity, symmetry, and transitivity.

We consider here the case where Q contains all predicates, i.e., global indiscernibility ($\equiv$); the arguments for local indiscernibility are similar. Reflexivity holds, since obviously for every interpretation I and constant a, $I[a,a] = I$. Also symmetry holds, since $I[a,b] = I[b,a]$ for every interpretation I and constants a, b. Finally, transitivity holds, since $I[a,c] = I[a,b][b,c][a,b]$ for every interpretation I and constants a, b, and c, and if $a \equiv b$, $b \equiv c$ and M is a model of program P, then each of $M[a,b]$, $M[a,b][b,c]$, and $M[a,b][b,c][a,b]$ is a model of P as well.

Given a constant a, we denote by $[a]_{\equiv}$ and $[a]_{\equiv_Q}$ the equivalence class of a under $\equiv$ and $\equiv_Q$, respectively (e.g., $[a]_{\equiv_Q}$ is the set of all constants Q-indiscernible from a). For convenience, we also write "p" for "$\{p\}$" in case Q is a singleton $\{p\}$.

For semantics based on the canonical model approach, which give a unique and always existing model as the intended meaning of the program (e.g., well-founded semantics [47] for normal programs, or the least model semantics for normal positive programs), two constants a and b are indiscernible if and only if their substitution is the identity on the canonical model. For instance, a is indiscernible from b on a normal positive program P, if $M_Q[a,b] = M_Q$, where M is the least model of P.

Proposition 9 Let a and b be two constants in U_P and Q be a set of predicates of P. If $|Sem_P| = 1$, then

$$(a \equiv_Q b) \Longleftrightarrow (M_Q[a,b] = M_Q)$$

where $\{M\} = Sem_P$.

Proof Immediate from Definition 7.

Example 10 Consider a positive logic program with predicates v, e and $path$, where v and e specify the vertices and the edges of a directed graph $G = (V, E)$, respectively, and $path$ defines the reflexive and transitive closure of e. Then, two vertices a and b are $path$-indiscernible precisely if they belong to the same strongly connected component of G. The classes $[a]_{\equiv_{path}}$ are the strongly connected components of G. In particular, G is strongly connected if and only if there is a single such class, i.e., all vertices are $path$-indiscernible.

For instance, given the program P:

$$\begin{aligned} path(X,X) &\leftarrow v(X), \\ path(X,Y) &\leftarrow e(X,Z) \wedge path(Z,Y), \\ v(X) &\leftarrow e(X,Y), \\ v(Y) &\leftarrow e(X,Y), \\ e(a,b) &\leftarrow, \\ e(b,c) &\leftarrow, \\ e(c,a) &\leftarrow, \end{aligned}$$

representing the graph G of Figure 1.a, we have that all vertices are $path$-indiscernible, that is, $[a]_{\equiv_{path}} = \{a,b,c\}$ $(= [b]_{\equiv_{path}} = [c]_{\equiv_{path}})$; notice that G is strongly connected.

For instance, a and b are $path$-indiscernible, since the least model (which is also the unique stable model) of the program is

$$\begin{aligned} M = \{ \; & v(a),\ v(b),\ v(c),\ e(a,b),\ e(b,c),\ e(c,a),\ path(a,a),\ path(b,b),\ path(c,c), \\ & path(a,b),\ path(b,c),\ path(a,c)\ path(c,a),\ path(c,b),\ path(b,a)\} \end{aligned}$$

and

$$\begin{aligned} M_{path} &= \{path(a,a),\ path(b,b),\ path(c,c),\ path(a,b),\ path(b,c),\ path(a,c) \\ & \quad path(c,a),\ path(c,b),\ path(b,a)\,\} \\ &= M_{path}[a,b]. \end{aligned}$$

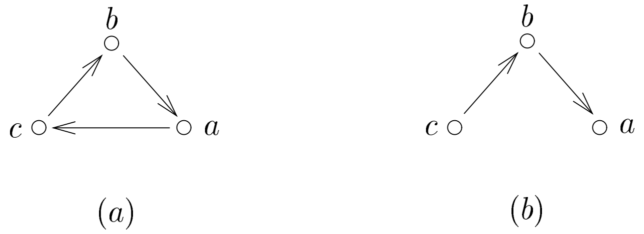

Figure 1: Graphs G and G'

Observe that a and b are also $\{v\}$-indiscernible; but they are not $\{e\}$-indiscernible. Thus, they are not globally indiscernible.

For program $P' = P - \{e(c,a)\}$, no pair of vertices is *path*-indiscernible. Indeed, the least model of the program is

$$M = \{ \ v(a),\ v(b),\ v(c),\ e(a,b),\ e(b,c),\ path(a,a),\ path(b,b),\ path(c,c),\ path(a,b),\ path(b,c),\ path(a,c)\ \}.$$

Thus,

$$M_{path} = \{\ path(a,a),\ path(b,b),\ path(c,c),\ path(a,b),\ path(b,c),\ path(a,c)\ \}$$

and it is different from

$$M_{path}[a,b] = \{\ path(a,a),\ path(b,b),\ path(c,c),\ path(b,a),\ path(a,c),\ path(b,c)\ \}$$

P' describes the graph G' of Figure 1.b, whose strongly connected components are $\{a\}$, $\{b\}$, and $\{c\}$, respectively. □

It is worth noting that, for multiple model semantics (like, e.g., the minimal model semantics), condition $(M_Q[a,b] = M_Q)$ is sufficient to guarantee that $(a \equiv_Q b)$, but, it is not necessary.

Example 11 Let P be the program $\{q(a) \vee q(b)\}$. If we consider the minimal model semantics, then we have $Sem_P = \{M, N\}$, where $M = \{q(a)\}$ and $M = \{q(b)\}$. Therefore, $(a \equiv_q b)$, even if $(M_q[a,b] \neq M_q)$ (and also $(N_q[a,b] \neq N_q)$ holds). □

We next provide an example of indiscernibility for the case of multiple stable models.

Example 12 Consider the following program P

$$\begin{array}{ll}
(1) & color(X, red) \vee color(X, blue) \vee color(X, green) \leftarrow v(X), \\
(2) & bad\text{-}coloring \leftarrow e(X,Y), color(X,C), color(Y,C), \\
(3) & abs \leftarrow bad\text{-}coloring, \neg abs.
\end{array}$$

Program P represents the 3-colorability of a graph. Intuitively, rule (1) assigns a color to each vertex of the graph. Rule (2) derives *bad* iff the coloring is not admissible (as two connected vertices have the same color). Then, rule (3) acts as an integrity constraint and discards the models representing inadmissible colorings (i.e., the models where *bad-coloring* is true).

Thus, given a set F of facts for predicates v and e (encoding the vertices and the edges of a graph G, respectively), there is a one-to-one correspondence between the stable model of $P \cup F$ and the good coloring of the graph G.

Now, suppose that the color of vertex a is fixed to *red* (by adding the fact $color(a, red)$ to the program). Then, $[a]_{color}$ is the set of vertices which are colored red in every admissible coloring of the graph where a is red. That is, a vertex b is *color*-indiscernible from a, if and only if its color is *red* in every admissible coloring where a is red. As a consequence, $[a]_{color}$ contains all vertices that have always the same color as a, in every admissible coloring of the graph. □

4 The Complexity of Indiscernibility

The main decisional problems arising in the context of indiscernibility are as follows.

Let a program P, two constants a and b in U_P, and a set of predicates Q of P be given.

- Is a Q-indiscernible from b ? (*Local Indiscernibility*)
- Is a globally indiscernible from b ? (*Global Indiscernibility*)

In this section we analyze the complexity of the above decisional problems on indiscernibility for the case of ground (i.e., propositional) programs. In particular, we shall study the complexity of both local and global indiscernibility under stable model semantics for general disjunctive programs and syntactical subclasses of them.

4.1 Normal Logic Programming

Throughout this section we assume that the logic programs are normal (i.e., disjunction-free).

We start by analyzing the complexity of deciding indiscernibility for the syntactically simplest classes of logic programs.

Theorem 13 If P is either a positive or a stratified program, then both global and local indiscernibility are P-complete.

Proof From Proposition 4, P has precisely one stable model M. As a consequence, by virtue of Proposition 9, $a \equiv_Q b$ iff $M_Q[a, b] = M_Q$. As a consequence, Q-indiscernibility check for a, b can be done by first computing the unique stable model M of P, and then checking if $M[a, b] = M$. Both these tasks are polynomial (for positive or stratified normal programs) and, as a consequence, Q-indiscernibility can be recognized in polynomial time. This strategy is correct regardless whether Q is the set of all predicates of P or Q is a subset of it. Therefore, both local and global indiscernibility are polynomial. P-completeness is an easy consequence of the fact that inference of a ground atom A from a ground datalog program is P-time complete [48] (add a fact $p(b, a)$ and a rule $p(a, b) \leftarrow A$, where p is a new predicate and a, b are fresh constants, and ask whether a and b are indiscernible).

We next prove that global indiscernibility for normal logic programs with negation is a co-NP-complete decisional problem. The proof of co-NP membership is based on an alternative characterization of global indiscernibility, which is shown below.

Theorem 14 Let a and b be two constants in U_P.

$$(a \equiv b) \Longleftrightarrow (\forall M \in Sem_P \quad M[a,b] \in Sem_P)$$

Proof Sufficiency is immediate from Definition 7. Concerning necessity, recall that global indiscernibility is indiscernibility under the set of *all* predicates of P. Now, if Q is the set of all predicates of the program, then, given a model $M \in Sem_P$, either $M[a,b] \in Sem_P$ or no model $M' \in Sem_P$ can exist such that $M_Q[a,b] = M'_Q$.

Thus, two constants are globally indiscernible if and only if $(\cdot)[a,b]$ is an automorphism on the set Sem_P.

Theorem 15 Global indiscernibility is co-NP-complete.

Proof Membership in co-NP is an immediate consequence of Theorem 14. To show hardness, we reduce the problem of deciding whether a ground logic program P has no stable model, which is co-NP-complete [31, 32], to this problem. Let a and b be fresh constants not occurring in P, and let p a new predicate. Then, consider the program

$$P' = P \cup \{\, p(a) \leftarrow , \quad p(b) \leftarrow p(b) \,\}.$$

Clearly, a, and b are globally indiscernible in P' if and only if P has no stable model. This proves the result.

We now turn our attention to the complexity of local indiscernibility.

Theorem 16 Let P be a ground logic program and Q be a set of predicates of P. Deciding if two given constants a and b are Q-indiscernible is Π_2^P-complete. Hardness holds even if P has only two monadic predicates.

Proof *Π_2^P-Membership.* To decide that a and b are not Q-indiscernible, we proceed as follows. Guess $M \subseteq B_{LP}$, verify that: (i) M is a stable model of P, (ii) there exists no stable model M' of P such that $M_Q[a,b] = M'_Q$. By Proposition 5, (i) is done in polynomial time; while (ii) is done by a single call to a NP oracle (to check the complement of (ii) we may guess M' and polynomially verify the condition $M_Q[a,b] = M'_Q$). The problem is therefore in Π_2^P.
Π_2^P-Hardness. We give a transformation from abductive logic programming. In particular, we consider the necessity problem [14], which is as follows. An abduction problem $\mathcal{P} = \langle Hyp, Man, LP\rangle$ is a set of propositional atoms Hyp, a set of propositional literals Man, and a propositional logic program LP. An explanation of $\mathcal{P}$ is a subset $S \subseteq Hyp$ such that $LP \cup S$ has a stable model and $LP \cup S$ infers each literal $L \in Man$ under cautious stable inference. Now, the problem is the following one. Given $\mathcal{P}$ and $h \in Hyp$, decide whether h is necessary for $\mathcal{P}$, i.e., h belongs to every explanation of $\mathcal{P}$. As shown in [14], this problem is Π_2^P-complete (see the appendix for more details).

We reduce this problem to local indiscernibility as follows. Define a program P which includes the following clauses.

1. For each clause $b_0 \leftarrow (\neg)b_1, \ldots (\neg)b_m$ in LP, include the clause

$$(1) \quad p(b_0) \leftarrow (\neg)p(b_1), \ldots, (\neg)p(b_m).$$

   The propositional atoms in LP are viewed as constants, and $p(c)$ tells that c is a proposition.

2. For each $h_i \in Hyp$, include the rules

$$\begin{array}{ll} (2) & s(h_i) \leftarrow \neg p(h_i^*) \\ (3) & p(h_i^*) \leftarrow \neg s(h_i) \\ (4) & p(h_i) \leftarrow s(h_i) \end{array}$$

   where h_i^* is a new constant. Here $s(h_i)$ intuitively states that h_i is included in an explanation for $\mathcal{P}$. The first two rules choose then a subset S of Hyp; the third rule expresses that the chosen S is added to LP, by stating that the atoms in S are true.

3. for each positive literal $b_i \in Man$, include the rule

$$(5) \quad p(not\_exp) \leftarrow \neg p(b_i)$$

   and for each negative literal $\neg b_i \in Man$ the rule

$$(6) \quad p(not\_exp) \leftarrow p(b_i),$$

   where not_exp is a new constant. Intuitively, $p(not\_exp)$ will be true in a stable model of P if it witnesses that the choice of S made in this model is not an explanation.

4. Finally, include the rules

$$\begin{array}{ll} (7) & p(c) \leftarrow \neg p(d) \\ (8) & p(d) \leftarrow \neg p(c) \\ (9) & s(a) \leftarrow p(c) \\ (10) & s(b) \leftarrow p(d), p(not\_exp) \\ (11) & s(b) \leftarrow p(d), s(h) \end{array}$$

   where a, b, c and d are new constants. The first two rules choose between $p(c)$ and $p(d)$; the third rule generates a stable model containing $s(a)$; upon every choice for S in 2., there is always a possibility to apply this rule, and thus a stable model containing $s(a)$ exists. The last two rules alternatively generate on the choice for S a stable model containing $s(b)$, but only if either $p(not\_exp)$ or $s(h)$ (i.e., the choice for S includes h) is true.

We note the following modularity lemma on the stable models of a logic program [15, 28], which is useful to argue about P:

Lemma 17 Suppose P_1^* and P_2^* are two ground disjunctive logic programs, such that no atom in the head of a rule in P_2^* occurs in P_1^*. Let $P^* = P_1^* \cup P_2^*$. Then, If M is a stable model of P^*, then the restriction of M to the ground atoms of P_1^* is a stable model of P_1^*; on the other hand, if N is a stable model of P_1^*, then every stable model of $P_2^* \cup N$ is a stable model of P.

In what follows, let P_1 be the clauses of groups 1. and 2. and let P_2 be the clauses of groups 3. and 4.

We claim that a and b are s-indiscernible in P if and only if h is necessary for $\mathcal{P}$. ($\Rightarrow$) To prove the *only if* direction, it suffices to show that if h is not necessary for $\mathcal{P}$, then $a \not\equiv_s b$. To prove this, we show that assuming simultaneously that h is not necessary for $\mathcal{P}$ and that $a \equiv_s b$ raises a contradiction.

As by hypothesis h is not necessary for $\mathcal{P}$, there exists an explanation S of $\mathcal{P}$ such that $h \notin S$. Take any stable model N of $LP \cup S$ (a stable model must exist, by definition of explanation), and define an interpretation M of programs P as follows:

$$M = \{p(b) \mid b \in N\} \cup \{s(h_i) \mid h_i \in S\} \cup \{p(h_i^*) \mid h_i \in H \setminus S\} \cup \{p(c), s(a)\}.$$

It is easy to see that M is a stable model of P (note that $N \models Man$). Since, by hypothesis, a and b are s-indiscernible in P, there must exist a stable model M' of P such that $M'_s = M_s[a,b]$. Notice that $M'_s = (M_s \setminus \{s(a)\}) \cup \{s(b)\}$. Let in the following N' be the restriction of M' to the set of all ground atoms of P_1. Notice that, by the above fact, N' is a stable model of P_1.

Since $s(h) \notin M'$ and $s(b) \in M'$, it follows from the clauses (10) and (11) defining $s(b)$, that $p(d), p(not\_exp) \in M'$. Since $p(not\_exp)$ is defined by clauses (5) and (6) only, it follows that for some positive literal $b_i \in Man$ (resp. negative literal $\neg b_i \in Man$), we have $p(b_i) \notin M'$ (resp. $p(b_i) \in M'$). Since the only rules with head $p(b_i)$ are in program P_1, we infer that $p(b_i) \notin N'$ (resp. $p(b_i) \in N'$) must hold.

The stable model N' of P_1 trivially corresponds to a stable model

$$N'' = \{b \mid p(b) \in N', b \text{ occurs in } LP \cup S\}$$

of $LP \cup S$. Since $p(b_i) \notin N'$ (resp. $p(b_i) \in N'$), it follows $b_i \notin N''$. Thus, N'' is a stable model of $LP \cup S$ such that $N'' \not\models Man$. Therefore, S is not an explanation of $\mathcal{P}$. This is a contradiction, however, as S was supposed to be an explanation. This proves the *only if*-direction.

($\Leftarrow$) We show that $a \not\equiv_s b$ implies that h is not necessary for $\mathcal{P}$, i.e., there exists an explanation S such that $h \notin S$.

Suppose that $a \not\equiv_s b$ holds. Hence, there exists a stable model M of P such that for every other stable model M' of P, $M'_s \neq M_s[a,b]$. We observe the following properties of M:

(a) $s(a) \in M$, $s(b) \notin M$: Indeed, it is easy to see that either $s(a) \in M$ or $s(b) \in M$ must hold thanks to modularity properties. Suppose $p(b) \in M$ would hold. Then, $p(d) \in M$, and the set $M' = (M \setminus \{p(d), s(b)\}) \cup \{p(c), s(a)\}$ would be a stable model of P. Since $M'_s = M_s[a,b]$, a contradiction arises. Thus, $s(a) \in M$, $s(b) \notin M$ must hold.

(b) $s(h) \notin M$, $p(not\_exp) \notin M$: Indeed, if $s(h) \in M$ or $p(not\_exp) \in M$ would hold, then thanks to rules (10) and (11), either $s(b) \in M$, which contradicts (a), or $M' = (M \setminus \{p(c), s(a)\}) \cup \{p(d), s(b)\}$ is a stable model of P such that $M'_s = M_s[a,b]$, which is again a contradiction.

(c) $p(b_i) \in M$ (resp. $p(b_i) \notin M$) for every positive literal $b_i \in M$ (resp. negative literal $\neg b_i \in M$). This is immediate from (b) and rules (5), (6).

Define now

$$S = \{h_i \in H \mid s(h_i) \in M\}.$$

We show that S is an explanation of $\mathcal{P}$ such that $h \notin S$. Thus, we have to show that (i) $LP \cup S$ has a stable model, (ii) $LP \cup S \models Man$, and (iii) $h \notin S$.

For (i), let N_0 be the restriction of M to the ground atoms of P_1. From Lemma 17, it follows that N_0 is a stable model of P_1. Consequently, the corresponding set of atoms

$$N = \{b \mid p(b) \in M, b \text{ occurs in } LP \cup S\}$$

is a stable model of the program $LP \cup S$.

For (ii), assume towards a contradiction that there exists a stable model N' of $LP \cup S$ such that $N \not\models Man$. From N', it is straightforward to build a stable model M_0' of the program consisting of the clauses in 1.–3. in which $p(not\_exp)$ is true; from Lemma 17, it follows that the set $M' = M_0' \cup \{p(d), s(b)\}$ is then a stable model of P. As easily checked, $M_s' = M_s[a,b]$. This is a contradiction. Consequently, an N' as hypothesized does not exist. It follows that $LP \cup S \models Man$.

For (iii), observe that (a) from above states that $s(h) \notin M$; hence, by definition of S, $h \notin S$.

Consequently, S is an explanation of $\mathcal{P}$ such that $h \notin S$. This proves the *if*-direction.

Since the program P is easily constructed from $\mathcal{P}$, the result follows.

Observe that Theorem 16 grasps the simplest case where local indiscernibility is Π_2^P-hard, namely, that there are two monadic predicates. Indeed, whenever the program P contains only one predicate, local indiscernibility coincides with global indiscernibility, which is in NP by virtue of Theorem 15.

4.2 Disjunctive Logic Programming

Theorem 18 Deciding global indiscernibility of two constants a and b for a disjunctive logic program P under stable models is Π_2^P-complete, and hard already for positive programs.

Proof Membership is a consequence of Theorem 14. To prove that a and b are not indiscernible, we can guess a stable model M of P such that $M[a,b]$ is not a stable model of P. Verifying whether an interpretation N is a stable model of a ground disjunctive program P is in co-NP; thus, the guess for M can be verified in polynomial time with an oracle for NP.

To prove hardness, we reduce the problem of cautious inference from a positive ground disjunctive logic program P to our problem. As shown in [13], cautious inference of a literal $\neg w$ from a given propositional program P is Π_2^P-hard. Let p be a new predicate and a and b be fresh constants, and define the program P' by

$$P' = P \cup \{\ p(a) \vee p(b) \leftarrow;\ \ p(a) \leftarrow w\ \}$$

Then, a and b are globally indiscernible in P' if and only if $\neg w$ is cautiously inferred from P. Indeed, observe that the minimal models of P' are the minimal models of $P \cup M$, for all minimal model M of P, and each minimal model of P', restricted to the language of P, is a minimal model of P. (This follows from modularity properties of disjunctive logic programs, see [15, 28].)

Suppose $\neg w$ is not inferred from the minimal models of P. Then, P has a minimal model M in which w is contained. As a consequence, $M \cup \{p(a)\}$ is a minimal model of P', while $M \cup \{p(b)\}$ is not. As a consequence, a and b are not indiscernible in P'. On the other hand, if a and b are not indiscernible in P', there is a minimal model M of P' such that $M[a,b]$ is not a minimal model of P'. Since clearly exactly one of $p(a)$ and $p(b)$ is in M, it follows that $p(a) \in M$ and $w \in M$. Since M is, restricted to the language of P, a minimal model of P, a minimal model of P exists which contains w. Thus, $\neg w$ is not inferred from the minimal models of P. This proves the result.

Theorem 19 Deciding local indiscernibility of constants a and b for a set of predicates Q in a ground disjunctive logic program P is Π_3^P-complete. Hardness holds even for positive logic programs with two monadic predicates.

Proof Π_3^P*-Membership.* The proof of membership is similar as in the case of normal logic programs, with the only difference that an co-NP-oracle is used to check whether an interpretation M is a stable model (for normal logic programs, that problem is polynomial), which is by the Proposition 6 co-NP-complete.
Π_3^P*-Hardness.* For the general case, Π_3^P-hardness is shown using the same reduction as in the proof of Theorem 16, but applied to a disjunctive logic programming abduction problem $\langle Hyp, Man, LP\rangle$, i.e., the program LP is a disjunctive logic program. Indeed, the necessity problem (deciding whether a given hypothesis $h \in H$ belongs to all explanations) is Π_3^P-complete [15] (see also appendix). The proof that the reduction is correct is analogous.

In the rest of the proof, we show that Π_3^P-hardness holds under the restriction to positive programs P. For that, we use the fact that the necessity problem remains Π_3^P-hard even if the disjunctive logic program LP in $\mathcal{P}$ is positive. Thus, all rules in the group 1. of the reduction are positive. We show how the nonpositive rules in the other groups 2.–4. can be be replaced by positive rules.

- The choice rules
$$s(h_i) \leftarrow \neg p(h_i^*)$$
$$p(h_i^*) \leftarrow \neg s(h_i)$$
for each $h_i \in H$ from 2. are replaced by the equivalent single disjunctive rule
$$s(h_i) \vee p(h_i^*) \leftarrow;$$
Similarly, the choice
$$p(c) \leftarrow \neg p(d)$$
$$p(d) \leftarrow \neg p(c)$$
from 4. is implemented by the single disjunctive rule
$$p(c) \vee p(d) \leftarrow$$

- for each rule from 3. that has a negative literal in the body, i.e., rule
$$p(not\_exp) \leftarrow \neg p(b_i)$$

introduce the following positive rules:

$$\begin{aligned} p(c_i) \vee p(d_i) &\leftarrow \\ p(c_i) &\leftarrow p(b_i) \\ p(not\_exp) &\leftarrow p(d_i), \end{aligned}$$

where c_i and d_i are fresh constants. The effect of these rules is as follows. By the first rule, each stable model M of the program must contain either $p(c_i)$ or $p(d_i)$. If M includes $p(b_i)$, it includes $p(c_i)$ (as it is a model), and hence, by the minimality of a stable model, $p(d_i)$ is not contained in M. Therefore, the last clause cannot be applied to include $p(not\_exp)$ in M. If $p(b_i)$ is not in the model, then either $p(c_i)$ or $p(d_i)$ is in M, but not both; in the latter case, $p(not\_exp)$ is included, and hence some stable model M containing $p(not\_exp)$ exists.

It holds that in the obtained positive program P, the constants a and b are s-indiscernible if and only if h is necessary for $\mathcal{P}$. This shows that deciding local indiscernibility is Π_3^P-hard, even if the program P is positive and and there are only two monadic predicates. The result follows.

5 Discussion and Conclusion

The results on indiscernibility checking for normal and disjunctive ground logic programs are summarized in Table 1. They show that this problem is polynomial only in case where the underlying program has no disjunction and negation is stratified. If we allow either for disjunction or for arbitrary negation, then indiscernibility checking becomes intractable, where local indiscernibility is always one level higher in the polynomial hierarchy than global indiscernibility.

	Deciding $a \equiv b$	Deciding $a \equiv_Q b$
normal positive	P-complete	P-complete
normal stratified	P-complete	P-complete
normal general	co-NP-complete	Π_2^P-complete
disjunctive positive	Π_2^P-complete	Π_3^P-complete
disjunctive general	Π_2^P-complete	Π_3^P-complete

Table 1: Complexity of indiscernibility for ground logic programming

It appears that global indiscernibility checking has the same complexity as cautious inference of a literal from the stable models of a normal resp. disjunctive logic program, which is co-NP-complete and Π_2^P-complete, respectively; the same holds in case of a positive disjunctive logic program.

The results for the ground case suggest that in the nonground case, the complexity of indiscernibility checking increases by an exponential, similar as for reasoning problems [14]. Indeed, by first grounding a given program P and then applying the algorithm for the ground case, we obtain that the exponential analog of the respective complexity class in Table 1 (EXPTIME for P and $\mathrm{Exp}\Pi_k^P$ for Π_k^P, in particular co-NEXP for co-NP) is an upper bound of the complexity of indiscernibility checking. Moreover, by utilizing results through complexity upgrading techniques [15, 21], it is possible to derive matching hardness results; we do not pursue this here. Notice that by our reductions in the previous section, the EXPTIME-hardness results follow immediately from the result that inference of a ground atom from a nonground datalog program is EXPTIME-complete (cf. [48]), and the $\mathrm{Exp}\Pi_2^P$-hardness results for disjunctive programs from the result that brave inference of a ground literal from a (positive) disjunctive datalog program is $\mathrm{Exp}\Sigma_2^P$-complete [15].

As pointed out, indiscernibility may be useful in the process of model finding as well as reasoning. From the computational side, it is not attractive (and in case of local indiscernibility even disadvantageous) to determine indiscernibility online. However, the indiscernibility relation can be determined off-line for later online use. In particular, indiscernibility information can be quite useful for model finding; notice that it is not viable to precompute all stable models of a logic program, since there may be exponentially many and hence they occupy exponential space. On the other hand, the indiscernibility relation occupies only linear space.

The results of this paper give a clear picture of the complexity of indiscernibility of logic programs in case of no function symbols. In the case where function symbols are present, global and local indiscernibility are clearly undecidable, since it is undecidable whether a normal logic program has a stable model [30]. We leave the study of indiscernibility on logic programs with function symbols for other research.

As mentioned in Section 3, the notion of indiscernibility can be extended naturally to partial model semantics as well, viewing each partial model as a set of ground literals. Let us briefly address some of the well-known proposals of such semantics [47, 38, 49], and comment on the complexity of indiscernibility checking for them.

The well-founded semantics for normal logic programs [47] fosters a unique partial model of a normal logic program, which can be computed in polynomial time. Therefore, both local and global indiscernibility of constants a and b can be decided in polynomial time; since the well-founded model coincides with the least model on normal positive programs, both tasks are P-complete.

The partial stable model semantics has been proposed as a natural generalization of the stable semantics from a two-valued to a three-valued framework [38]. However, on positive and stratified programs, total stable models and partial stable models coincide. On the other hand, the complexity of verifying partial stable models and total stable models is the same for both normal and disjunctive logic programs [42, 13]. Therefore, all complexity results in Table 1 except for the hardness results in the normal general case immediately carry over to partial stable models as well by the proofs in this paper. The hardness results for the normal general case can be established by different transformations. (Notice that the respective transformations for total stable models do not work.)

The regular semantics has been proposed as an attempt to minimize undefinedness in partial models [49]. Roughly, a regular model is a three-valued model of the program which is founded, i.e., the positive part regenerates itself, and which is maximal, i.e., it is not properly contained in any other such model. Regular semantics is similar in spirit to the maximal stable semantics [43, 16], which selects the partial stable models that are not contained properly in any other partial stable model. However, while regular and maximal partial stable models coincide on normal logic programs [50], the semantics are yet different on disjunctive general programs. On positive and stratified programs, both semantics coincide with the stable semantics [16, 49]. Hence, the complexity results in Table 1 for normal positive, normal stratified, and disjunctive positive programs apply to these semantics as well. Utilizing results on the complexity of inference under these semantics [17], for both global indiscernibility on normal and disjunctive general logic programs, respectively, can be easily shown to be complete for Π_2^P and Π_3^P, respectively, employing a reduction similar as in the proof of Theorem 18. In case of local indiscernibility, the complexity can be shown to be one level higher up in the polynomial hierarchy than global indiscernibility and thus complete for Π_3^P and Π_4^P, respectively.

Acknowledgements

The authors would like to thank the anonymous referees for their comments and useful suggestions to improve this paper; in particular, the discussion of partial model semantics was suggested by a referee.

Appendix

Abductive logic programming deals with the problem of finding an explanation for observations, based on a theory represented by a logic program [23]. Roughly, abduction is an inverse of modus ponens: Given the clause $a \leftarrow b$ and the observation a, abduction concludes b as a possible explanation.

A number of variants of abductive logic programmings have been introduced so far; the paper [14] presents a basic framework of abduction, discusses different modes of abduction, and presents a detailed analysis of the complexity of the main abductive reasoning tasks.

An *propositional abductive logic programming problem* (*LPAP*) is formally described as a tuple $\mathcal{P} = \langle Hyp, Man, LP \rangle$, where Hyp is a set of propositional atoms (called *hypotheses*), Man is a set of propositional literals (called *manifestations* or *observations*), and LP is a propositional logic program (with negation allowed). In case LP is a disjunctive logic program, we have a disjunctive *LPAP*. An *explanation* for $\mathcal{P}$ is a subset $S \subseteq Hyp$ which satisfies the following properties:

1. The program $LP \cup S$ has some stable model; and
2. $LP \cup S \models Man$, i.e., each literal in Man is a cautious consequence of the stable models of the program $LP \cup S$.

(In fact in [14] abduction based on different semantics and reasoning modalities are considered.)

In general, an *LPAP* may have no, a single, or several explanations. In account of the latter, necessity of a hypothesis has been identified as an important property. A hypothesis is *necessary* for an *LPAP* $\mathcal{P}$, if it occurs in every explanation of $\mathcal{P}$. The problem of deciding whether a hypothesis is necessary is one of the main decisional abductive reasoning tasks, since it is important for computing the core of abductive explanations. Thus, the necessity problem of logic programming abduction is as follows: Given a *LPAP* $\mathcal{P} = \langle Hyp, Man, LP \rangle$ and a hypothesis $h \in H$, decide whether h is necessary for $\mathcal{P}$.

Concerning the computational complexity of the necessity problem, the following results have been established in [14]:

[**14, Theorem 11**] Deciding if a given hypothesis is necessary for a given *LPAP* $\mathcal{P} = \langle Hyp, Man, LP \rangle$ is Π_2^P-complete.

[**14, Theorem 18,24**] Deciding if a given hypothesis is necessary for a given disjunctive *LPAP* $\mathcal{P} = \langle Hyp, Man, LP \rangle$ is Π_3^P-complete; moreover, the problem is Π_3^P-hard already on instances where LP is positive, i.e., $\neg$-free.

For a formal proof of these results, we refer the reader to [14]; we here only describe the intuition why the problems have this complexity.

In order to decide whether hypothesis h is necessary, all explanations of $\mathcal{P}$ must be inspected (in the worst case) to see whether h this is the case; in order to show that h is not necessary, we have to exhibit an explanation S for $\mathcal{P}$ in which h is not contained. There is an exponential space of candidates $S \subseteq Hyp$, and in fact there are cases where exponentially many explanations exist. An explanation S such that $h \notin S$ might be nondeterministically guessed; we are left with proving then that the guess is proper, i.e., is in fact an explanation. Deciding whether a given S is an explanation turns out difficult; first, we must decide whether $LP \cup S$ has some stable model and second, whether $LP \cup S \models Man$. Both tests are inference problems that are well-known NP- and co-NP-complete problems [32]. Thus, we can verify the guess with an oracle for NP in polynomial time. Overall, we have a Σ_2^P algorithm for deciding whether h is *not* necessary; as a consequence, the necessity problem is in Π_2^P.

Hardness is shown in [14] encoding the test of validity of quantified Boolean formulas of form $\forall X \exists Y E$, where X, Y are sets of propositional variables and E is a Boolean formula whose atoms are from $X \cup Y$.

For the case of disjunctive *LPAP*s, the situation is similar as for normal *LPAP*s. The only difference is that the complexity of deciding whether $LP \cup S$ has a (disjunctive) stable model and $LP \cup S \models Man$ is Σ_2^P-complete and Π_2^P-complete, respectively [13]. Consequently, we need an oracle for Σ_2^P problems in order to decide whether a given S is an explanation. Hardness for Π_3^P is shown in [14] again by an encoding of validity testing of quantified Boolean formulas, which in this case have form $\forall X \exists Y \forall Z E$.

That deciding necessity is already Π_3^P-hard on positive disjunctive programs LP is somewhat unexpected, but not really surprising. In fact, cautious inference of literals from the minimal models of a positive disjunctive program is Π_2^P-hard [13]. This causes that the full power of the Σ_2^P-oracle is needed for checking whether a set S is an explanation.

References

[1] A. Abiteboul, K. Compton, and V. Vianu. Queries are Easier than You Thought (Probably). In *Proceedings ACM PODS '92*, pages 218–229, 1992.

[2] S. Abiteboul and A. van Gelder. Optimizing Active Databases using the Split Technique. In J. Biskup and R. Hull, editors, *Proceedings 4th Intl. Conference on Database Theory (ICDT '92)*, LNCS 646, pages 171–187, 1992.

[3] K. Apt, H. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In Minker [34], pages 89–148.

[4] K. Apt and N. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19/20:9–71, 1994.

[5] C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[6] C. Baral and V. Subrahmanian. Stable and Extension Class Theory for Logic Programs and Default Logic. *Journal of Automated Reasoning*, 8:345–366, 1992.

[7] R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[8] R. Ben Eliyahu, N. Francez, M. Kaminski. Similarity Preservation in Default Logic. Technical Report 96-04, Department of mathematics and computer science, Ben-Gurion university of the Negev, Beer Sheva, Israel.

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

[10] S. Brass and J. Dix. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. In L. Sterling Ed., *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo* (June 1995), pp. 199–213. MIT Press.

[11] F. Buccafurri, N. Leone, and P. Rullo. Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *Journal of Logic Programming*, 27(1):5–43, 1996.

[12] J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In *Logic, Action and Information. Proc. of the Konstanz Colloquium in Logic and Information (LogIn'92)*, pages 241–329. DeGruyter, 1995.

[13] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[14] T. Eiter, G. Gottlob, and N. Leone. Abduction From Logic Programs: Semantics and Complexity. *Theoretical Computer Science*, forthcoming. Extended abstract in Proc. LPNMR'95, LNCS 982, 1–14, 1995.

[15] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Trans. on Database Syst.*, September 1997. Extended abstract *Proc. ACM PODS '94*, pages 267–278, 1994.

[16] T. Eiter, N. Leone, and D. Saccà. On the Partial Semantics for Disjunctive Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 17(1/2): 59–96, 1997.

[17] T. Eiter, N. Leone, and D. Saccà. Expressive Power of Partial Models for Disjunctive Deductive Databases. *Theoretical Computer Science*, forthcoming. Extended abstract in Proc. International Workshop on Logic in Databases (LID '96), LNCS 1154, pp. 245–264. Springer, 1996.

[18] M. Gelfond. Logic Programming and Reasoning with Incomplete Information. *Annals of Mathematics and Artificial Intelligence*, 12:89–116, 1994.

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

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

[21] G. Gottlob, N. Leone, and H. Veith. Second-Order Logic and the Weak Exponential Hierarchies. In J. Wiedermann and P. Hajek, editors, *Proc. 20th Conf. on Mathematical Foundations of Computer Science (MFCS '95)*, LNCS 969, pages 66–81, Prague, 1995. Full paper CD/TR 95/80, Christian Doppler Lab for Expert Systems, TU Vienna.

[22] L. Hemachandra. The Strong Exponential Hierarchy Collapses. *Journal of Computer and System Sciences*, 39:299–322, 1989.

[23] A. Kakas, R. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 1993.

[24] R. Kowalski and F.Sadri. Logic Programs with Exceptions. In *Proceedings ICLP-90*, pages 598–616. MIT Press, 1990.

[25] E. Laenens, D. Saccá, and D. Vermeir. Extending Logic Programming. In *Proceedings ACM SIGMOD Int'l Conference on Management of Data*, pages 184–13, 1990.

[26] E. Laenens and D. Vermeir. A Fixpoint Semantics for Ordered Logic. *Journal of Logic Programming*, 1:159–185, 1990.

[27] N. Leone, P. Rullo, F. Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, forthcoming. Extended abstract *in* Proc. of International Logic Programming Symposium–ILPS'95, pp.399-413, MIT Press.

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

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

[30] W. Marek, A. Nerode, and J. Remmel. The Stable Models of a Predicate Logic Program. *Journal of Logic Programming*, 21(3):129–153, 1994.

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

[32] W. Marek and M. Truszczyński. Computing Intersection of Autoepistemic Expansions. In A. Nerode, W. Marek, and V. Subrahmanian, editors, *Proc. LPNMR-91*, pages 37–50, Washington DC, July 1991. MIT Press.

[33] J. Minker. On Indefinite Data Bases and the Closed World Assumption. In D. Loveland, editor, *Proc.* 6^{th} *Conference on Automated Deduction (CADE '82)*, LNCS 138, pages 292–308, New York, 1982.

[34] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufman, Washington DC, 1988.

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

[36] T. Przymusinski. On the Declarative and Procedural Semantics of Stratified Deductive Databases. In Minker [34], pages 193–216.

[37] T. Przymusinski. Well-founded Semantics Coincides with 3-Valued Semantics. *Fundamenta Informaticae*, 13:445–463, 1990.

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

[39] T. Przymusinski. Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 14(2-4), 1995.

[40] K. Ross. The Well-Founded Semantics for Disjunctive Logic Programs. In *Proceedings First Intl. Conf. on Deductive and Object-Oriented Databases (DOOD-89)*, pages 352–369, 1989.

[41] C. Sakama. Possible Model Semantics for Disjunctive Databases. In *Proceedings First Intl. Conf. on Deductive and Object-Oriented Databases (DOOD-89)*, pages 337–351, Kyoto Japan, 1989.

[42] D. Saccà. The Expressive Powers of Stable Models for Bound and Unbound DATALOG Queries. *Journal of Computer and System Sciences*, 1997, forthcoming.

[43] D. Saccá, C. Zaniolo. Deterministic and Non-Deterministic Stable Models. *Journal of Logic and Computation*, 1997, forthcoming.

[44] J. Schlipf. The Expressive Powers of Logic Programming Semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995. Abstract in Proc. ACM PODS '90, pp. 196–204.

[45] L. Stockmeyer. Classifying the Computational Complexity of Problems. *Journal of Symbolic Logic*, 52(1):1–43, 1987.

[46] M. H. van Emden and R. Kowalski. The Semantics of Logic as a Programming Language. *Journal of the ACM*, 3:733–742, 1976.

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

[48] M. Vardi. Complexity of relational query languages. In *Proceedings 14th ACM STOC*, pages 137–146, San Francisco, 1982.

[49] J.-H. You and L. Yuan. A Three-Valued Semantics for Deductive Databases and Logic Programs. *Journal of Computer and System Sciences*, 49:334–361, 1994.

[50] J.-H. You and L. Yuan. On the Equivalence of Semantics for Normal Logic Programming. *Journal of Logic Programming*, pp. 211–222, 1995.