**INFSYS**
**RESEARCH**
**REPORT**

INSTITUT FÜR INFORMATIONSSYSTEME
ABTEILUNG WISSENSBASIERTE SYSTEME

# ENHANCING DISJUNCTIVE DATALOG BY CONSTRAINTS

F. Buccafurri N. Leone P. Rullo

INFSYS RESEARCH REPORT 1843-99-03
FEBRUARY 1999

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Treitlstraße 3
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-99-03, FEBRUARY 1999

# ENHANCING DISJUNCTIVE DATALOG BY CONSTRAINTS

Francesco Buccafurri[1] Nicola Leone[2] Pasquale Rullo[3]

**Abstract.** This paper presents an extension of Disjunctive Datalog (DATALOG$^{\vee,\neg}$) by integrity constraints. These are of two types: *strong*, that is, classical integrity constraints, and *weak*, that is, constraints that are satisfied *if possible.* While strong constraints must be satisfied, weak constraints express desiderata, that is, they may be violated - actually, their semantics tends to minimize the number of violated instances of weak constraints. Weak constraints may be ordered according to their importance to express different priority levels. As a result, the proposed language (call it DATALOG$^{\vee,\neg,c}$) is well-suited to represent commonsense reasoning and knowledge-based problems arising in different areas of computer science such as planning, graph theory optimizations and abductive reasoning.

The formal definition of the language is first given. The declarative semantics of DATALOG$^{\vee,\neg,c}$ is defined in a general way that allows us to put constraints on top of any existing (model-theoretic) semantics for DATALOG$^{\vee,\neg}$ programs. Knowledge representation issues are then addressed, and the complexity of reasoning on DATALOG$^{\vee,\neg,c}$ programs is carefully determined. An in-depth discussion on complexity and expressiveness of DATALOG$^{\vee,\neg,c}$ is finally reported. The discussion contrasts DATALOG$^{\vee,\neg,c}$ to DATALOG$^{\vee,\neg}$ and highlights the significant increase in knowledge modeling ability carried out by constraints.

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

---

[1]DIMET Dep., University of Reggio Calabria, 89100 Reggio Cal., Italy, e-mail: `bucca@ns.ing.unirc.it`

[2]Information Systems Dept., Technical University of Vienna, A-1040 Vienna, Austria, e-mail: `leone@dbai.tuwien.ac.at`

[3]Pasquale Rullo is with the Dip. di Matematica, University of Calabria, 87030 Rende, Italy, e-mail: `rullo@si.deis.unical.it`

**Acknowledgements**: This work has been supported in part by *FWF* (Austrian Science Funds) under project P11580-MAT "A Query System for Disjunctive Deductive Databases".

A short abridged version of this paper appeared in the Proc. of the conference LPNMR '97, pp. 2–17, Springer

# 1 Introduction

*Disjunctive Datalog* (or DATALOG$^{\vee,\neg}$) programs [18, 15] are nowadays widely recognized as a valuable tool for knowledge representation and commonsense reasoning [2, 34, 25, 20]. An important merit of DATALOG$^{\vee,\neg}$ programs over normal (that is, disjunction-free) Datalog programs is their capability to model incomplete knowledge [2, 34]. Much research work has been done so far on the semantics of disjunctive programs and several alternative proposals have been formulated [5, 20, 40, 46, 47, 48, 51, 59]. One which is widely accepted is the extension to the disjunctive case of the stable model semantics of Gelfond and Lifschitz. According to this semantics [20, 46], a disjunctive program may have several alternative models (possibly none), each corresponding to a possible view of the reality. In [11, 15], Eiter, Gottlob and Mannila show that DATALOG$^{\vee,\neg}$ has a very high expressive power, as (under stable model semantics) the language captures the complexity class $\Sigma_2^P$ (i.e., it allows us to express *every* property which is decidable in non-deterministic polynomial time with an oracle in NP).

In this paper, we propose an extension of DATALOG$^{\vee,\neg}$ by constraints. In particular, besides classical integrity constraints (that we call ßtrongconstraints), we introduce the notion of *weak constraints*, that is, constraints that *should possibly be satisfied*. Contrary to strong constraints, that express conditions that must be satisfied, weak constraints allow us to express desiderata. The addition of (both weak and strong) constraints to DATALOG$^{\vee,\neg}$ makes the language (call it DATALOG$^{\vee,\neg,c}$) well-suited to represent a wide class of knowledge-based problems (including, e.g., planning problems, NP optimization problems, and abductive reasoning) in a very natural and compact way.

As an example, consider the problem SCHEDULING which consists in the scheduling of examinations for courses. That is, we want to assign course exams to time slots in such a way that no two exams are assigned with the same time slot if the respective courses have a student in common (we call such courses ¨ ncompatible"). Supposing that there are three time slots available, namely, $ts_1$, $ts_2$ and $ts_3$, we express the problem in DATALOG$^{\vee,\neg,c}$ by the following program $\mathcal{P}_{sch}$:

$$\begin{array}{ll} r_1: & assign(X, ts_1) \vee assign(X, ts_2) \vee assign(X, ts_3) \leftarrow course(X) \\ s_1: & \leftarrow assign(X, S), assign(Y, S), incompatible(X, Y) \end{array}$$

Here we assumed that the courses and the pair of incompatible courses are specified by a number of input facts with predicate *course* and *incompatible*, respectively. Rule $r_1$ says that every course is assigned to either one of the three time slots $ts_1$, $ts_2$ or $ts_3$; the strong constraint $s_1$ (a rule with empty head) expresses that no two incompatible courses can be overlapped, that is, they cannot be assigned to the same time slot. In general, the presence of strong constraints modifies the semantics of a program by discarding all models which do not satisfy some of them. Clearly, it may happen that no model satisfies all constraints. For instance, in a specific instance of problem above, there could be no way to assign courses to time slots without having some overlapping between incompatible courses. In this case, the problem does not admit any solution. However, in real life, one is often satisfied with an approximate solution, that is, one in which constraints are satisfied as much as possible. In this light, the problem at hand can be restated as follows (APPROX SCHEDULING): ässign courses to time slots trying to not overlap incompatible courses”. To express this problem we resort to the notion of *weak* constraint, as shown by the following program $\mathcal{P}_{a\_sch}$:

$$\begin{array}{ll} r_1: & assign(X,ts_1) \vee assign(X,ts_2) \vee assign(X,ts_3) \leftarrow course(X) \\ w_1: & \Leftarrow assign(X,S), assign(Y,S), incompatible(X,Y) \end{array}$$

From a syntactical point of view, a weak constraint is like a strong one where the implication symbol $\leftarrow$ is replaced by $\Leftarrow$. The semantics of weak constraints minimizes the number of violated instances of constraints. An informal reading of the above weak constraint $w_1$ is: ”preferably, do not assign the courses $X$ and $Y$ to the same time slot if they are incompatible”. Note that the above two programs $\mathcal{P}_{sch}$ and $\mathcal{P}_{a\_sch}$ have exactly the same models if all incompatible courses can be assigned to different time slots (i.e., if the problem admits an “exact” solution).

In general, the informal meaning of a weak constraint, say, $\Leftarrow B$, is try to falsify $B$ör ”$B$ is preferably false”, etc. (thus, weak constraints reveal to be very powerful for capturing the concept of preference¨ n commonsense reasoning).

Since preferences may have, in real life, different priorities, weak constraints in DATALOG$^{\vee,\neg,c}$ can be assigned with different priorities too, according to their ¨ mportance”. [1] For example, assume that incompatibilities among courses may be either *strong* or *weak* (e.g., basic courses with common students can be considered strongly incompatible, while complementary courses give weak incompatibilities). Consider the following problem (SCHEDULING WITH PRIORITIES): ßchedule courses by trying to avoid overlapping between strongly incompatible courses first, and by trying to avoid overlapping between weakly incompatible courses then”(i.e., privilege the elimination of overlapping between strongly incompatible courses). If strong and weak incompatibilities are specified through input facts with predicates *strongly_incompatible* and *weakly_incompatible*, respectively, we can represent SCHEDULING WITH PRIORITIES by the following program $\mathcal{P}_{p\_sch}$:

$$\begin{array}{ll} r_1: & assign(X,ts_1) \vee assign(X,ts_2) \vee assign(X,ts_3) \leftarrow course(X) \\ w_2: & \Leftarrow assign(X,S), assign(Y,S), strongly\_incompatible(X,Y) \\ w_3: & \Leftarrow assign(X,S), assign(Y,S), weakly\_incompatible(X,Y) \end{array}$$

where the weak constraint $w_2$ is defined ßtronger than”$w_3$. The models of the above program are the assignments of courses to time slots that minimize the number of overlappings between strongly incompatible courses and, among these, those which minimize the number of overlappings between weakly incompatible courses.

The main contributions of the paper are the following:

- We add weak constraints to DATALOG$^{\vee,\neg}$ and provide a formal definition of the language DATALOG$^{\vee,\neg,c}$. The semantics of constraints is given in a general way that

[1]Note that priorities are meaningless among strong constraints, as all of them *must* be satisfied.

allows us to define them on top of any existing model-theoretic semantics for Disjunctive Datalog.

- We show how constraints can be profitably used for knowledge representation and reasoning, presenting several examples of DATALOG$^{\vee,\neg,c}$ encoding. DATALOG$^{\vee,\neg,c}$ turns out to be both general and powerful, as it is able to represent in a simple and coincise way hard problems arising in different domains ranging from planning, graph theory, and abduction.

- We analyze the computational complexity of reasoning with DATALOG$^{\vee,\neg,c}$ (propositional case under stable model semantics). The analysis pays particular attention to the impact of syntactical restrictions on DATALOG$^{\vee,\neg,c}$ programs in the form of limited use of weak constraints, strong constraints, disjunction, and negation. It appears that, while strong constraints do not affect the complexity of the language, weak constraints "mildly" increase the computational complexity. Indeed, we show that brave reasoning is $\Delta_3^P$-complete (resp., $\Delta_2^P$-complete) for DATALOG$^{\vee,\neg,c}$ (resp., DATALOG$^{\neg,c}$) programs. Interestingly, priorities among constraints affect the complexity, which decreases to $\Delta_3^P[O(\log n)]$ ($\Delta_2^P[O(\log n)]$ for DATALOG$^{\neg,c}$) if priorities are disallowed. The complexity results may support in choosing an appropriate fragment of the language, which fits the needs in practice (see Section 5).

- We carry out an in-depth discussion on expressiveness and complexity of DATALOG$^{\vee,\neg,c}$, contrasting the language to DATALOG$^{\vee,\neg}$. Besides adding expressive power from the theoretical view point (as DATALOG$^{\vee,\neg,c}$ can encode problems which cannot be represented at all in DATALOG$^{\vee,\neg}$), it turns out that constraints improve also usability and knowledge modeling features of the language. Indeed, well-known problems can be encoded in a simple and easy-to-understand way in DATALOG$^{\vee,\neg,c}$; while their DATALOG$^{\vee,\neg}$ encoding is unusable (long and difficult to understand).

Even if strong constraints (traditionally called integrity constraints) are well-known in the logic programming community (see, e.g., [15, 10, 17, 33, 12]), to our knowledge this is the first paper which formally studies weak constraints and proposes their use for knowledge representation and reasoning. Related works can be considered the interesting papers of Greco and Saccà [24, 23] which analyze other extensions of Datalog to express NP optimizations problems. Related studies on complexity of knowledge representation languages have been carried out in [15, 50, 8, 14, 21, 39, 52, 9, 12]. Priority levels have been used also in the context of theory update and revision [16] and [22], in prioritized circumscription [31], in the preferred subtheories approach for default reasoning [6], and in the preferred answer set semantics for extended logic programs [7].

The paper is organized as follows. In Section 2 we provide both the syntax and the semantics of the language. Then, in Section 3 we describe the knowledge representation capabilities of DATALOG$^{\vee,\neg,c}$ by a number of examples. In particular, we show how the language can be used to easily formulate complex knowledge-based problems arising in various areas of computer science, including planning, graph theory and abduction. In Section 4 we analyze the complexity of the language under the possibility version of the stable model semantics. Finally, Section 5 reports an in-depth discussion on the language and draws our conclusions. Appendix A shows the encoding of a number of classical optimization problems in DATALOG$^{\vee,\neg,c}$; while Appendix B reports the proofs of the complexity results for the fragments of DATALOG$^{\vee,\neg,c}$ where disjunction is either disallowed or constrained to the head cycle free case.

# 2 The DATALOG$^{\vee,\neg,c}$ Language

We assume that the reader is familiar with the basic concepts of deductive databases [55, 34] and logic programming [32]. We describe next the extension of Disjunctive Datalog by constraints.

## 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 *(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 $n = 1$ (i.e., the head is $\vee$-free), then $r$ is *normal*; if $m = 0$ (the body is $\neg$-free), then $r$ is *positive*. A *DATALOG*$^{\vee,\neg}$ *program* $LP$ is a finite set of rules; $LP$ is *normal* (resp., *positive*) if all rules in $LP$ are normal (resp. positive).

A *strong constraint* is a syntactic of the form $\leftarrow L_1, \cdots, L_k$, where $L_i$, $1 \leq\ i \leq k$, is a literal (i.e., it is a rule with empty head).

A *weak constraint* is a syntactic of the form $\Leftarrow L_1, \cdots, L_k$, where $L_i$, $1 \leq\ i \leq k$, is a literal.

**Definition 1** A *DATALOG*$^{\vee,\neg,c}$*program* (often simply program") is a triple $\mathcal{P} = (LP, S, W)$, where $LP$ is a DATALOG$^{\vee,\neg}$ program, $S$ a (possibly empty) finite set of strong constraints and $W = \langle W_1, ..., W_n\rangle$, is a (possibly empty) finite list of *components*, each consisting of a finite set of weak constraints. If $w \in W_i$, $w' \in W_j$ and $i < j$, then we say that $w'$ is ßtronger thanör more important than"$w$ (hence, the last component $W_n$ is the strongest).

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

**Example 2** *Consider the program $\mathcal{P}_{p\_sch}$ of Section 1. Here, $LP_{p\_sch} = \{r_1\}$, $S_{p\_sch} = \emptyset$ and $W_{p\_sch} = \langle W_1, W_2 \rangle$, where $W_1 = \{w_3\}$ and $W_2 = \{w_2\}$ (that is, $w_2$ is stronger than $w_3$).* $\triangle$

## 2.2 General Semantics

Let $\mathcal{P} = (LP, S, W)$ be a 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 rules, strong and weak constraints is defined in the obvious way over the constants in $U_{\mathcal{P}}$, and are denoted by $ground(LP)$, $ground(S)$ and $ground(W)$, respectively; we denote the instantiation of $\mathcal{P}$ by $ground(\mathcal{P}) = (ground(LP), ground(S), ground(W))$.

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 rule in $ground(LP)$. 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 ground (strong or weak) constraint $c$ in $(ground(S) \cup ground(W))$ is *satisfied* w.r.t. $I$ if (at least) one literal appearing in $c$ is false w.r.t. $I$; otherwise, $c$ is *violated*.

We next define the semantics of DATALOG$^{\vee,\neg,c}$ in a general way which does not rely on a specific semantical proposal for DATALOG$^{\vee,\neg}$; but, rather, it can be applied to any semantics of Disjunctive Datalog. [3] To this end, we define a *candidate model* for $\mathcal{P} = (LP, S, W)$ as an interpretation $M$ for $\mathcal{P}$ which satisfies every rule $r \in ground(LP)$. A *ground semantics for* $\mathcal{P}$ is a finite set of candidate models for $\mathcal{P}$. Clearly, every classical semantics for $LP$ provides a ground semantics for $\mathcal{P}$. For example the semantics presented in [20, 40, 46, 47, 48, 51, 59] can be taken as ground semantics.

Now, to define the meaning of a program $\mathcal{P} = (LP, S, W)$ in the context of a given ground semantics, we need to take into account the presence of constraints. To this end, since weak constraints may have different priorities, we associate with each component $W_i \in W$, $1 \leq i \leq n$, a positive weight $f(W_i)$ inductively defined as follows:

$$f(W_1) = 1$$

$$f(W_i) = f(W_{i-1}) \cdot |ground(W)| + 1, \ 1 < i \leq n$$

where $|ground(W)|$ denotes the total number of ground instances of the weak constraints appearing in $W$. Further, given an interpretation $M$ for a program $\mathcal{P}$, we denote by $H_{\mathcal{P},M}$ the following sum of products:

$$H_{\mathcal{P},M} = f(W_1) \cdot N_1^M + ... + f(W_n) \cdot N_n^M$$

[3] Note that, although we consider only total model semantics in this paper, the semantics of weak constraints simply extends to the case of partial model semantics.

where $N_i^M$ $(1 \leq i \leq n)$ is the number of ground instances of weak constraints in the component $W_i$ which are violated in $M$.

Now, we are ready to define the notion of model for $\mathcal{P} = (LP, S, W)$ w.r.t. a given ground semantics.

**Definition 3** Given a ground semantics $\Gamma$ for $\mathcal{P} = (LP, S, W)$, a $\Gamma$*-model of* $\mathcal{P}$ is a candidate model $M \in \Gamma$ such that: (1) every strong constraint $s \in ground(S)$ is satisfied w.r.t. $M$, and (2) $H_{\mathcal{P},M}$ is minimum, that is, there is no candidate model $N \in \Gamma$ verifying Point (1) such that $H_{\mathcal{P},N} < H_{\mathcal{P},M}$

Thus, a $\Gamma$-model $M$ of $\mathcal{P}$, by minimizing $H_{\mathcal{P},M}$ selects those candidate models that, besides satisfying strong constraints, minimize the number of unsatisfied (instances of) weak constraints according to their importance - that is, those with the minimal number of violated constraints in $W_n$ are chosen, and, among these, those with the minimal number of violated constraints in $W_{n-1}$, and so on and so forth.

## 2.3 Stable Model Semantics

Using the notion of candidate model we have parametrized the semantics of DATALOG$^{\vee,\neg,c}$ programs, that is, the actual semantics of a program $\mathcal{P}$ relies on the semantics we choose for $LP$. Several proposals can be found in the literature for disjunctive logic programs [5, 20, 40, 46, 47, 48, 51]. One which is generally acknowledged is the extension of the stable model semantics to take into account disjunction [20, 46]. We next report a brief discussion on this semantics.

In [40], Minker proposed a model-theoretic semantics for positive (disjunctive) programs, whereby a positive program $LP$ is assigned with a set $\mathrm{MM}(LP)$ of minimal models, each representing a possible meaning of $LP$ (recall that a model $M$ for $LP$ is minimal if no proper subset of $M$ is a model for $LP$).

**Example 4** *For the positive program* $LP = \{a \vee b \leftarrow\}$ *the (total) interpretations* $\{a\}$ *and* $\{b\}$ *are its minimal models (i.e.,* $\mathrm{MM}(LP) = \{\ \{a\},\ \{b\}\ \}$*).* $\triangle$

The stable model semantics generalises the above approach to programs with negation. Given a program $LP$ and a total interpretation $I$, the *Gelfond-Lifschitz transformation* of $LP$ with respect to $I$, denoted by $LP^I$, is the positive program defined as follows:

$$\begin{array}{l} LP^I = \{a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k \mid \\ \quad a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_m \in ground(LP) \quad and\ b_i \notin I,\ k < i \leq m\} \end{array}$$

Now, let $I$ be an interpretation for a program $LP$. $I$ is a *stable model* for $LP$ if $I \in \mathrm{MM}(LP^I)$ (i.e., $I$ is a minimal model of the positive program $LP^I$). We denote the set of all stable models for $LP$ by $SM(LP)$.

**Example 5** *Let* $LP = \{a \vee b \leftarrow c, \quad b \leftarrow \neg a, \neg c, \quad a \vee c \leftarrow \neg b\}$. *Consider* $I = \{b\}$. *Then,* $LP^I = \{a \vee b \leftarrow c, \quad b \leftarrow\}$.
*It is easy to verify that* $I$ *is a minimal model for* $LP^I$*; thus,* $I$ *is a stable model for* $LP$. $\triangle$

Clearly, if $LP$ is positive then $LP^I$ coincides with $ground(LP)$. It turns out that, for a positive program, minimal and stable models coincide. A normal positive programs $LP$ has exactly one stable model which coincides with the least model of $LP$ (i.e., it is the unique minimal model of $LP$). When negation is allowed, however, even a normal program can admit several stable models.

We conclude this section observing that Definition 3 provides the stable model semantics of a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ if the choosen ground semantics is $SM(LP)$, the set of the stable models of $LP$.

**Definition 6** A $\Gamma$-*model* for a program $\mathcal{P} = (LP, S, W)$ is a *stable model for* $\mathcal{P}$ if $\Gamma = SM(LP)$.

**Example 7** *Consider the program* $\mathcal{P}_{sch} = (LP_{sch}, S_{sch}, W_{sch})$ *of Section 1; here,* $LP_{sch} = \{r_1\}$, $S_{sch} = \{s_1\}$ *and* $W_{sch} = \emptyset$. *According to the stable model semantics,* $LP_{sch}$ *has as many stable models as the possibilities of assigning all courses, say* $n$*, to 3 time slots (namely,* $3^n$*). The stable models of* $\mathcal{P}_{sch}$ *are the stable models of* $LP_{sch}$ *satisfying the strong constraint* $s_1$*, that is, those (if any) for which no two incompatible courses are assigned with the same time slot.*

*The program* $\mathcal{P}_{a\_sch} = (LP_{a\_sch}, S_{a\_sch}, W_{a\_sch})$ *of Section 1, where* $LP_{a\_sch} = \{r_1\}$, $S_{a\_sch} = \emptyset$ *and* $W_{a\_sch} = \langle\{w_1\}\rangle$*, is obtained from* $LP_{sch}$ *by replacing* $s_1$ *by* $w_1$ *(note that* $LP_{a\_sch} = LP_{sch}$*). The stable models of* $\mathcal{P}_{a\_sch}$ *are the stable models of* $LP_{a\_sch}$ *which minimize the number of violated instances of* $w_1$*, that is, the number of incompatible courses assigned to the same time slots. Thus,* $\mathcal{P}_{a\_sch}$ *provides a different solution from* $\mathcal{P}_{a\_sch}$ <u>*only if*</u> *the latter does not admit any stable model, i.e., the problem has no "exact" solution - there is no way to assign different time slots to all incompatible courses. Otherwise, the two programs have exactly the same stable models.*

*Finally, consider the program* $\mathcal{P}_{p\_sch} = (LP_{p\_sch}, S_{p\_sch}, W_{p\_sch})$ *of Section 1, where* $LP_{p\_sch} = \{r_1\}$, $S_{p\_sch} = \emptyset$ *and* $W_{p\_sch} = \langle\{w_3\}, \{w_2\}\rangle$ *(read "*$w_3$ *is weaker than* $w_2$*"). Each stable model of* $LP_{p\_sch}$ *is a possible assignment of courses to time slots. The stable models of* $\mathcal{P}_{p\_sch}$ *are the stable models of* $LP_{p\_sch}$ *that, first of all, minimize the number of violated instance of* $w_2$ *and, in second order, minimize the number of violated instances of* $w_3$. $\triangle$

It is worth noting that a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P}$ may have several stable models (there can also be no one). The modalities of brave and cautious reasoning are used to handle this.
**Brave reasoning** (or *credulous reasoning*) infers that a ground literal $Q$ is true in $\mathcal{P}$ (denoted $\mathcal{P} \models_{brave} Q$) iff $Q$ is true w.r.t. $M$ for *some* stable model $M$ of $\mathcal{P}$.
**Cautious reasoning** (or *skeptical reasoning*) infers that a ground literal $Q$ is true in $\mathcal{P}$ (denoted $\mathcal{P} \models_{cautious} Q$) iff $Q$ is true w.r.t. $M$ for *every* stable model $M$ of $\mathcal{P}$.

The inferences $\models_{brave}$ and $\models_{cautious}$ extend to sets of literals as usual.

# 3 Knowledge Representation in DATALOG$^{\vee,\neg,c}$

In this section we provide a number of examples that show how DATALOG$^{\vee,\neg,c}$ can be used to easily formulate many interesting and difficult knowledge-based problems. Among those, we consider planning problems, classical optimization problems from Graph Theory and various forms of abductive reasoning. We show how the language provides natural support for their representation. A number of further examples are reported in AppendixA.

As we shall see, most programs have a common structure of the form guess, check, choose-best. Candidate solutions are first nondeterministically generated (through disjunctive rules); then, mandatory properties are checked (through strong constraints), finally, solutions that best satisfy desiderata are selected (through weak constraints). Such a modular structure shows quite natural for expressing complex problems and, further, makes programs easy to understand.

## 3.1 Planning

A first example of planning, namely the APPROX SCHEDULING problem, has been presented in the introduction. A further example is next reported.

**Example 8** *PROJECT GROUPS. Consider the problem of organizing a given set of employees into two (disjoint) project groups* $p1$ *and* $p2$*. We wish each group to be possibly heterogeneous as far as skills are concerned. Further, it is preferable that couple of persons married to each other do not work in the same group. And, finally, we would like that the members of the same group already know each other. We consider the former two requirements more important than the latter one. Supposing that the information about employees, skills, known and married persons, are specified through a number of input facts a simple way of solving this problem is given by the following program* $\mathcal{P}_{proj}$*:*[4]

$$
\begin{array}{ll}
r_1: & member(X,p1) \vee member(X,p2) \leftarrow employee(X) \\
w_1: & \Leftarrow member(X,P), member(Y,P), X \neq Y, \neg know(X,Y) \\
w_2: & \Leftarrow member(X,P), member(Y,P), married(X,Y) \\
w_3: & \Leftarrow member(X,P), member(Y,P), same\_skill(X,Y), X \neq Y
\end{array}
$$

*where* $w_2$ *and* $w_3$ *are stronger than* $w_1$ *(i.e.,* $W = \langle\{w_1\},\{w_2,w_3\}\rangle$*). The first rule* $r_1$ *above assigns each employee to either one of the two projects* $p1$ *and* $p2$ *(recall that, by minimality of a stable model, exactly one of* $member(X,p1)$ *and* $member(X,p2)$ *is true in it, for each employee* $X$*). The weak constraint* $w_1$ *expresses the aim of forming groups with persons that possibly already know each other;* $w_2$*, in turn, expresses the preference of having groups with no persons married each other; and, finally, the weak constraint* $w_3$ *tries avoid that persons*

[4]For notational simplicity, in the examples we represent programs just as sets of rules and constraints. Observe also that we use the inequality predicate $\neq$ as a built-in, this is legal as inequality can be easily simulated in DATALOG$^{\vee,\neg,c}$.

*with the same skill work in same project. It is easy to recognize that each stable model of the DATALOG*$^{\vee,\neg}$ *program* $LP = \{r_1\}$ *is a possible assignment of employees to projects. Due to the priority of weak constraints (recall that* $w_2$ *and* $w_3$ *are both ßtronger than"*$w_1$*) the stable models of* $\mathcal{P}_{proj}$ *are those of* $LP$ *that minimize the overall number of violated instances of both* $w_2$ *and* $w_3$ *and, among these, those minimizing the number of violated instances of* $w_1$*. For an instance, suppose that the employees are* $a$, $b$, $c$, $d$, $e$, *of which* $a$ *and* $b$ *have the same skill,* $c$ *and* $d$ *are married to each other and, finally, the following pairs know reciprocally:* $(b,c)$, $(c,d)$, $(a,e)$, $(d,e)$. *In this case,* $\mathcal{P}_{proj}$ *has two stable models* $M_1$ *and* $M_2$*, which correspond to the following division in project teams (we list only the atoms with predicate member of the two models).*

$$M_1: \quad \{\, member(a,p1), member(d,p1), member(e,p1), member(b,p2), member(c,p2)\}$$

$$M_2: \quad \{\, member(a,p2), member(d,p2), member(e,p2), member(b,p1), member(c,p1)\}$$

*Observe that both* $M_1$ *and* $M_2$ *violate only two instances of* $w_1$ *(as* $a$ *and* $b$ *do not know each other), and they satisfy all instances of* $w_2$ *and* $w_3$*.* △

## 3.2 Optimization from Graph Theory

In the above example we have regarded weak constraints essentially as desiderata - that is, conditions to be possibly satisfied. However, a useful way of regarding weak constraints is as öbjective functionsöf optimization problems. That is, a program with a weak constraint of the form $\Leftarrow B$ can be regarded as modeling a minimization problem whose objective function is the cardinality of the relation $B$ (from this perspective, a strong constraint $\leftarrow B$ can then be seen as a particular case in which the cardinality of the relation $B$ is required to be zero). This suggests us to use DATALOG$^{\vee,\neg,c}$ to model optimization problems. Next we show some classical $NP$ optimization problems from graph theory formulated in our language. A number of further examples can be found in AppendixA.

**Example 9** *MAX CLIQUE : Given a graph* $G = \langle V, E\rangle$*, find a maximum clique, that is, a subset of maximum cardinality* $C$ *of* $V$ *such that every two vertices in* $C$ *are joined by an edge in* $E$*. To this end, we define the following program* $\mathcal{P}_{cliq}$*:*

$$\begin{array}{ll} r_1: & c(X) \vee not\_c(X) \leftarrow vertex(X) \\ s_1: & \leftarrow c(X), c(Y), X \neq Y, \neg edge(X,Y) \\ w_1: & \Leftarrow not\_c(X) \end{array}$$

*Here,* $r_1$ *partitions the set of vertices into two subsets, namely,* $c$ *and* $not\_c$ *- i.e., it guesses a clique* $c$*. The strong constraint* $s_1$ *checks the guess, that is, every pair of vertices in* $c$ *(the clique) must be joined by an edge. Finally, the weak constraint* $w_1$ *minimizes the number of vertices that are not in the clique* $c$ *(thus, maximizing the size of* $c$*, by discarding all cliques whose size is not maximum). It holds that each stable model of* $\mathcal{P}_{cliq}$ *is a maximum clique of* $G$*.* △

**Example 10** *MIN COLORING. Given a graph $G = \langle V, E \rangle$, a coloring of $G$ is an assignment of colors to vertices, in such a way that every pair of vertices joined by an edge have different colors. A coloring is minimum if it uses a minimum number of colors. We determine a minimum coloring by the following program $\mathcal{P}_{col}$:*

$$
\begin{array}{ll}
r_1: & col(X,I) \vee not\_col(X,I) \leftarrow vertex(X), color(I) \\
s_1: & \leftarrow col(X,I), col(Y,I), edge(X,Y) \\
s_2: & \leftarrow col(X,I), col(X,J), I \neq J \\
s_3: & \leftarrow vertex(X), \neg colored(X) \\
r_2: & colored(X) \leftarrow col(X,I) \\
r_3: & used\_col(I) \leftarrow col(X,I) \\
w_1: & \Leftarrow used\_col(I)
\end{array}
$$

*The first rule $r_1$ guesses a graph coloring; $col(X,I)$ says that vertex $X$ is assigned to color $I$ and $not\_col(X,I)$ that it is not. Strong constraints $s_1$-$s_3$ check the guess: two joined vertices cannot have the same color ($s_1$), and each vertex is assigned to exactly one color ($s_2$ and $s_3$). Finally, the weak constraint $w_1$ requires the cardinality of the relation used_col to be minimum (i.e., the number of used colors is minimum). It holds that there is a one-to-one correspondence between the stable models of $\mathcal{P}_{col}$ and the minimum colorings of the graph $G$.*

$\triangle$

## 3.3 Abduction

We next show that some important forms of abduction over (disjunctive) logic programs, namely abduction with prioritization and abduction under minimum-cardinality solution preference [14], can be represented in a very easy and natural way in DATALOG$^{\vee,\neg,c}$ (while, in general, they cannot be encoded in DATALOG$^{\vee,\neg}$).

Abduction, first studied by Peirce [43], is an important kind of reasoning, having wide applicability in different areas of computer science; in particular, it has been recognized as an important principle of common-sense reasoning. For these reasons, abduction plays a central role in Artificial Intelligence, and has recently received growing attention also in the field of Logic Programming.

Abductive logic programming deals with the problem of finding an explanation for observations, based on a theory represented by a logic program [35, 36]. Roughly speaking, abduction is an inverse of modus ponens: Given the clause $a \leftarrow b$ and the observation $a$, abduction concludes $b$ as a possible explanation. Following [13, 35, 36], an *abductive logic programming problem* (*LPAP*) can be formally described as a tuple $\mathbf{P} = \langle Hyp, Obs, LP \rangle$, where $Hyp$ is a set of ground atoms (called *hypotheses*), $Obs$ is a set of ground literals (called *manifestations* or *observations*), and $LP$ is a logic program (with disjunction and negation allowed). An *explanation* for $\mathbf{P}$ is a subset $E \subseteq Hyp$ which satisfies the following property:

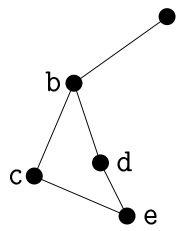


Figure 1: A computer network

$LP \cup E \models_{brave} Obs$, i.e., there exists a stable model $M$ of $LP \cup E$, where all literals in $Obs$ are true.

In general, an *LPAP* may have no, a single, or several explanations. In accordance with Occam's principle of parsimony [42], which states that from two explanations the simpler explanation is preferable, some minimality criterion is usually imposed on abductive explanations. Two important minimality criterions are *Minimum Cardinality* and *Prioritization* [14]. The *minimum cardinality* criterion states that a solution $A \subseteq Hyp$ is preferable to a solution $B \subseteq Hyp$ if $|A| < |B|$ (where $|X|$ denotes the cardinality of set $X$). According to this criterion, the acceptable solutions of $\mathbf{P}$ are restricted to the explanations of minimum cardinality (that are considered the most likely).

**Example 11** *ABDUCTION Consider the computer network depicted in Figure 1. We make the observation that, sitting at machine a, which is online, we cannot reach machine e. Which machines are offline?*

*This can be easily modeled as the LPAP* $\mathbf{P}_{net} = \langle Hyp_{net}, Obs_{net}, LP_{net} \rangle$*, where the theory* $LP_{net}$ *is*

$$
\begin{array}{ll}
LP_{net} = \{ & reaches(X,X) \leftarrow\ node(X),\ \neg\ offline(X); \\
& reaches(X,Z) \leftarrow\ reaches(X,Y),\ connected(Y,Z),\ \neg\ offline(Z); \\
& connected(a,b);\ connected(b,c);\ connected(b,d);\ connected(c,e); \\
& connected(d,e);\ node(a);\ node(b);\ node(c);\ node(d);\ node(e)\}
\end{array}
$$

*and the hypotheses and observations are*

$$Hyp_{net} = \{\ offline(a),\ offline(b),\ offline(c),\ offline(d),\ offline(e)\ \},$$

*and*

$$Obs_{net} = \{\ \neg\ offline(a),\ \neg\ reaches(a,e)\ \}.$$

*respectively.*

*For example, according with the intuition,* $E_1 = \{offline(b)\}$ *is an explanation for* $\mathbf{P}_{net}$*; but also* $E_2 = \{offline(b), offline(c)\}$ *is an explanation and even* $E_3 = \{\ offline(b),$

*$offline(c)$, $offline(d)$, $offline(e)$ } is an explanation. However, under the minimum cardinality criterion, only $E_1 = \{offline(b)\}$, and $E_4 = \{offline(e)\}$ are the solutions of $\mathbf{P}_{net}$ (we can see them as the most probable ones).*

△

Abduction with minimum cardinality can be represented very simply in DATALOG$^{\vee,\neg,c}$. Given a *LPAP* $\mathbf{P} = \langle Hyp, Obs, LP\rangle$, let $pr(\mathbf{P})$ be the DATALOG$^{\vee,\neg,c}$ program $(LP \cup G, S, \langle W_0\rangle)$, where

$$G = \{h \vee not\_h \leftarrow \mid h \in Hyp\}$$

$$S = \{ \leftarrow \sigma(o) \mid o \in Obs \}$$

$$W_0 = \{\Leftarrow h \mid h \in Hyp\}$$

where $\sigma(o) = \neg o$ if $o$ is a positive literal, and $\sigma(o) = a$ if $o = \neg a$ (for each $h \in Hyp$, $not\_h$ denotes a new symbol). It is easy to see that the stable models of $pr(\mathbf{P})$ correspond exactly to the minimum cardinality solutions of $\mathbf{P}$[5]. Intuitively, the clauses in $G$ "guess" a solution (i.e., a set of hypoteses), the strong constraints in $S$ check that the observations are entailed, and the weak constraints in $W_0$ enforce the solution to have minimum cardinality.

**Example 12** *(**cont'd**) The DATALOG$^{\vee,\neg,c}$ program for our network problem is $pr(\mathbf{P}_{net}) = (LP_{net} \cup G, S, \langle W_0\rangle)$, where $LP_{net}$ is the theory of $\mathbf{P}_{net}$ and*

$$G = \{offline(x) \vee not\_offline(x) \leftarrow \mid x \in \{a, b, c, d, e\}\},$$

$$S = \{\leftarrow offline(a);\quad \leftarrow reaches(a, e)\},$$

$$W_0 = \{\Leftarrow offline(x) \mid x \in \{a, b, c, d, e\}\}$$

*The stable models of $LP_{net} \cup G$ that satisfy both strong constraints $\leftarrow offline(a)$ and $\leftarrow reaches(a, e)$ encode one-to-one the abductive explanations of $\mathbf{P}_{net}$; among them, the weak constraints in $W_0$ select the explanations with minimum cardinality. Indeed, $pr(\mathbf{P}_{net})$ has only two stable models: one contains $offline(b)$ and the other contains $offline(e)$.*

△

The method of abduction with *priorities* [14] is a refinement of the above minimality criterion. Roughly speaking, it operates as follows. The set of hypotheses $Hyp$ is partitioned into groups with different priorities, and explanations which are (cardinality) minimal on the lowest priority hypotheses are selected from the solutions. The quality of the selected solutions on the hypotheses of the next priority level is taken into account for further screening;

[5]Without loss of generality, we assume that no hypothesis in $Hyp$ appears in the head of a rule in $LP$.

this process is continued over all priority levels. As pointed out in [14], prioritization is also a qualitative version of probability, where the different priorities levels represent different magnitude of probabilities. It is well suited in case no precise numerical values are known, but the hypotheses can be grouped in clusters such that the probabilities of hypotheses belonging to the same cluster do not (basically) differ compared to the difference between hypotheses from different clusters [14].

**Example 13** *(***cont'd***) Suppose that statistical data about the computers in the network of Figure 1 tell us that $b$ falls down very often, $e$ is always online, while $c$ and $d$ sometimes are offline. Then, we restate the LPAP $\mathbf{P}_{net}$ as an abductive problem with prioritization $\mathbf{P}'_{net} = \langle Hyp^{<}_{net}, Obs_{net}, LP_{net}\rangle$, where*

$$Hyp^{<}_{net} = \langle \{offline(b)\}, \{offline(c), offline(d)\}, \{offline(e)\}\rangle$$

*According with the prioritization, the abductive explainations are ordered as follows:*

$$\{offline(b)\} < \{offline(c), offline(d)\} < \{offline(e)\}$$

*Therefore, $\{offline(b)\}$ becomes the preferred (i.e., most likely) solution. (Note that, differently from the convention we adopted for weak constraints, the hypotheses of the smallest priority level are the most important here).*

△

By using priorities among weak constraints, we obtain a very natural encoding of abduction with prioritization. Given $\mathbf{P} = \langle\langle H_1, ..., H_n\rangle, Obs, LP\rangle$, let $pr(\mathbf{P})$ be the DATALOG$^{\vee,\neg,c}$ program $(LP \cup G, S, \langle W_1, ..., W_n\rangle)$, where

$$W_i = \{\Leftarrow h \mid h \in H_{n-i+1}\} \quad (1 \leq i \leq n)$$

and $S$ and $G$ are defined as above. Then, the stable models of $pr(\mathbf{P})$ correspond exactly to the preferred solutions of $\mathbf{P}$ according with the prioritization.

**Example 14** *(***cont'd***)*

*The DATALOG$^{\vee,\neg,c}$ program simulating abduction with priorities for the network of Example 11 is $pr(\mathbf{P}'_{net}) = (LP_{net} \cup G, S, \langle W_1, W_2, W_3\rangle)$, where $G$ and $S$ are as above, and*

$$W_1 = \{\Leftarrow offline(e)\},$$

$$W_2 = \{\Leftarrow offline(c), \Leftarrow offline(d)\},$$

$$W_3 = \{\Leftarrow offline(b)\}$$

*(recall that $W_3$ is the strongest component). It is easy to see that $pr(\mathbf{P}'_{net})$ has only one stable model $M$ which contains (only) the hypothesys $offline(b)$.*

△

# 4 The Complexity of Constraints: Propositional Case

In this section we analyze the complexity of brave reasoning over disjunctive Datalog programs with constraints. Since the complexity is not independent from the underlying ground semantics, we consider in this section the stable model semantics [20, 46], which is a widely acknowledged semantics for normal and disjunctive Datalog programs.

## 4.1 Preliminaries on Complexity Theory

For NP-completeness and complexity theory, cf. [41]. The classes $\Sigma_k^P, \Pi_k^P$ and $\Delta_k^P$ of the *Polynomial Hierarchy (PH)* (cf. [53]) are defined as follows:

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

In particular, $\mathrm{NP} = \Sigma_1^P$, co-NP $= \Pi_1^P$, and $\Delta_2^P = \mathrm{P}^{\mathrm{NP}}$. Here $\mathrm{P}^C$ (resp. $\mathrm{NP}^C$) denotes the class of problems that are solvable in polynomial time on a deterministic (resp. nondeterministic) Turing machine with an oracle for any problem $\pi$ in the class $C$.

The oracle replies to a query in unit time, and thus, roughly speaking, models a call to a subroutine for $\pi$ that is evaluated in unit time. If $C$ has complete problems, then instances of any problem $\pi'$ in $C$ can be solved in polynomial time using an oracle for any $C$-complete problem $\pi$, by transforming them into instances of $\pi$; we refer to this by stating that an oracle for $C$ is used. Notice that all classes $C$ considered here have complete problems.

The classes $\Delta_k^P$, $k \geq 2$, have been refined by the class $\Delta_k^P[O(\log n)]$, in which the number of calls to the oracle is in each computation bounded by $O(\log n)$, where $n$ is the size of the input.

Observe that for all $k \geq 1$,

$$\Sigma_k^P \subseteq \Delta_{k+1}^P[O(\log n)] \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P \subseteq \mathrm{PSPACE};$$

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.

The above complexity classes have complete problems under polynomial-time transformations involving Quantified Boolean Formulas (QBFs). A QBF is an expression of the form

$$Q_1 X_1 Q_2 X_2 \cdots Q_k X_k\, E, \quad k \geq 1, \tag{1}$$

where $E$ is a Boolean expression whose atoms are from pairwise disjoint nonempty sets of variables $X_1, \ldots, X_k$, and the $Q_i$'s are alternating quantifiers from $\{\exists, \forall\}$, for all $i = 1, \ldots, k$. If $Q_1 = \exists$ then we say the QBF is $k$-existential, otherwise it is $k$-universal. Validity of QBFs is defined in the obvious way by recursion to variable-free Boolean expressions. We denote by $\mathrm{QBF}_{k,\exists}$ (resp., $\mathrm{QBF}_{k,\forall}$) the set of all valid $k$-existential (resp., $k$-universal) QBFs (1).

Given a $k$-existential QBF $\Phi$ (resp. a $k$-universal QBF $\Psi$), deciding whether $\Phi \in \mathrm{QBF}_{k,\exists}$ (resp. $\Psi \in \mathrm{QBF}_{k,\forall}$), is a classical $\Sigma_k^P$-complete (resp. $\Pi_k^P$-complete) problem.

$\Delta_k^P$ has also complete problems for all $k \geq 2$; for example, given a formula $E$ on variables $X_1, ..., X_n, Y_1, ..., Y_r$, $r \geq 0$, and a quantifier pattern $Q_1Y_1, ..., Q_rY_r$, deciding whether the with respect to $\langle X_1, ..., X_n\rangle$ lexicographically minimum truth assignment[6] $\phi$ to $X_1, \ldots, X_n$ such that $Q_1Y_1 \cdots Q_rY_rE_\phi \in \mathrm{QBF}_{k-2,\forall}$ (where such a $\phi$ is known to exist) fulfills $\phi(X_n) = \mathit{true}$ (cf. [37, 58]).[7] Also $\Delta_{k+1}^P[O(\log n)]$ has complete problems for all $k \geq 1$; for example, given QBFs $\Phi_1, \ldots, \Phi_m$, such that $\Phi_i \notin \mathrm{QBF}_{k,\exists}$ implies $\Phi_{i+1} \notin \mathrm{QBF}_{k,\exists}$, for $1 \leq i < m$, decide whether $\max\{i : 1 \leq i \leq m, \Phi_i \in \mathrm{QBF}_{k,\exists}\}$ is odd ([58, 14]).

The problems remain as hard under the following restrictions: (a) $E$ in (1) is in conjunctive normal form and each clause contains three literals (3CNF) if $Q_k = \exists$, and (b) $E$ is in disjunctive normal form and each monom contains three literals (3DNF) if $Q_k = \forall$ [54].

## 4.2 Complexity Results

We analyze the complexity of the propositional case; therefore, throughout this section we assume that the programs and query literals are ground (i.e., variable-free). The complexity results, however, can be easily extended to *data complexity* [57].

Given a program $\mathcal{P} = (LP, S, \langle W_1, \cdots, W_n\rangle)$, we denote by $maxH(\mathcal{P})$ the value that $H_{\mathcal{P},M}$ assumes when the interpretation $M$ violates all the ground instances of weak constraints (in each component). More precisely, $maxH(\mathcal{P}) = \Sigma_{1\leq i\leq n} f(W_i) \times |ground(W_i)|$, where $|ground(W_i)|$ is the cardinality of the set of the ground instances of the constraints in $W_i$. Observe that $maxH(\mathcal{P})$ is exponential in the number of components of $\mathcal{P}$; indeed, from the (inductive) definition of $f(W_i)$ (see Section 2.2), it can be easily seen that $maxH(\mathcal{P}) = \Theta(|ground(W)|^n)$. It turns out that (the value of) $maxH(\mathcal{P})$ can be exponential in the size of the input; however, it is computable in polynomial time (its binary representation has polynomial size).

We next report detailed proofs of all complexity results of DATALOG$^{\vee,\neg,c}$ programs where full disjunction is allowed (these proofs are the most involved and technically interesting). The demonstrations of the results on disjunction-free programs and on head-cycle free disjunctive programs are reported in AppendixB. All complexity results are sumarized in Table 1, and discussed in-depth in Section 5.

First of all, we look for an upper bound to the complexity of brave reasoning on DATALOG$^{\vee,\neg,c}$. To this end, we provide two preliminary lemmata.

**Lemma 15** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, and a positive integer $n$ as input, it is in $\Sigma_2^P$ deciding whether there exists a stable model $M$ of $LP$ satisfying $S$ such that $H_{\mathcal{P},M} \leq n$.*

[6] $\phi$ is lexicographically greater than $\psi$ w.r.t. $\langle X_1, \ldots, X_n\rangle$ iff $\phi(X_j) = \mathit{true}$, $\psi(X_j) = \mathit{false}$ for the least $j$ such that $\phi(X_j) \neq \psi(X_j)$.

[7] $\mathrm{QBF}_{0,\forall} = \mathrm{QBF}_{0,\exists}$ is the set of all variable-free true formulas.

*Proof.* We can decide the problem as follows. Guess $M \subseteq B_{\mathcal{P}}$, and check that: (1) $M$ is a stable model of $LP$, (2) all constraints in $S$ are satisfied, and (3) $H_{\mathcal{P},M} \leq n$. Clearly, property (2) and (3) can be checked in polynomial time; while (1) can be decided by a single call to an NP oracle [39, 15]. The problem is therefore in $\Sigma_2^P$. □

**Lemma 16** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, a positive integer $n$, and a literal $q$ as input, it is in $\Sigma_2^P$ deciding whether there exists a stable model $M$ of $LP$ satisfying $S$ such that $H_{\mathcal{P},M} = n$ and $q$ is true w.r.t. $M$.*

*Proof.* We can decide the problem as follows. Guess $M \subseteq B_{\mathcal{P}}$, and check that: (1) $M$ is a stable model of $LP$, (2) all constraints in $ground(S)$ are satisfied, (3) $H_{\mathcal{P},M} = n$, and (4) $q$ is true w.r.t. $M$. Clearly, property (2), (3), and (4) can be checked in polynomial time; while (1) can be decided by a single call to an NP oracle [39, 15]. The problem is therefore in $\Sigma_2^P$. □

Now, we are in the position to determine the upper bound to brave reasoning on full DATALOG$^{\vee,\neg,c}$ programs.

**Theorem 17** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, and a literal $q$ as input, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is in $\Delta_3^P$.*

*Proof.* We first call a $\Sigma_2^P$ oracle to verify that $\mathcal{P}$ admits some stable model satisfying $S$ (otherwise, $q$ cannot be a brave consequence). We compute then (in polynomial time) $k = maxH(\mathcal{P})$. After that, by binary search on $[0..k]$, we determine the cost $\Sigma$ of the stable models of $\mathcal{P}$, by a polynomial number of calls to an oracle deciding whether there exists a stable model $M$ of $LP$ satisfying $S$ such that $H_{\mathcal{P},M} \leq n$ ($n = k/2$ on the first call; then if the oracle answers "yes", $n = k/4$; otherwise, $n$ is set to $k/2 + k/4$, and so on, according to standard binary search) – the oracle is in $\Sigma_2^P$ by virtue of Lemma 15. (Observe that the number of calls to the oracle is logarithmic in $k$, and, as a consequence, is polynomial in the size of the input, as the value $k$ is $O(2^{|\mathcal{P}|})$.) Finally, a further call to a $\Sigma_2^P$ oracle verifies that $q$ is true in some stable model of $\mathcal{P}$, that is a stable model $M$ of $LP$ satisfying $S$ with $H_{\mathcal{P},M} = \Sigma$ (this is doable in $\Sigma_2^P$ from Lemma 16). □

We will next strenghten the above result on $\Delta_3^P$-membership, to a completeness result in this class. Hardness will be proven by reductions from QBFs into problems related to stable models of DATALOG$^{\vee,\neg,c}$ programs; the utilized disjunctive Datalog programs will be suitable adaptations and extensions of the disjunctive program reported next (which has been first described in [12]).

Let $\Phi$ be a formula of form $\forall Y E$, where $E$ is a Boolean expression over propositional variables from $X \cup Y$, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. We assume that $E$ is in 3DNF, i.e., $E = D_1 \vee \cdots \vee D_r$ and each $D_i = L_{i,1} \wedge L_{i,2} \wedge L_{i,3}$ is a conjunction of literals $L_{i,j}$. Define the following (positive) Disjunctive Datalog program $LP(\Phi)$:

$$
\begin{array}{llll}
x_i \vee x'_i \leftarrow & & & \text{for each } i = 1, \ldots, n \\
y_j \vee y'_j \leftarrow & y_j \leftarrow v & y'_j \leftarrow v & \text{for each } j = 1, \ldots, m \\
v \leftarrow y_j, y'_j & & & \text{for each } j = 1, \ldots, m \\
v \leftarrow \sigma(L_{k,1}), \sigma(L_{k,2}), \sigma(L_{k,3}) & & & \text{for each } k = 1, \ldots, r
\end{array}
$$

where $\sigma$ maps literals to atoms as follows:

$$
\sigma(L) = \begin{cases} x'_i & \text{if } L = \neg x_i \text{ for some } i = 1, \ldots, n \\ y'_j & \text{if } L = \neg y_j \text{ for some } j = 1, \ldots, m \\ L & \text{otherwise} \end{cases}
$$

Intuitively, $x'_i$ corresponds to $\neg x_i$ and $y'_j$ corresponds to $\neg y_j$.

Given a truth assignment $\phi(X)$ to $X = \{x_1, \ldots, x_n\}$, we denote by $M_\phi \subseteq B_{LP(\Phi)}$ the following interpretation

$$M_\phi = \{x_i \mid \phi(x_i) = true\} \cup \{x'_i \mid \phi(x_i) = false\} \cup \{v\} \cup \{y_1, \ldots, y_m\} \cup \{y'_1, \ldots, y'_m\}.$$

Moreover, given an interpretation $M$ of $LP(\Phi)$, we denote by $\phi_M$ the truth assignment to $X = \{x_1, \ldots, x_n\}$:

$$\phi_M(x_i) = true \ iff \ x_i \in M.$$

**Lemma 18** *Let $\Phi = \forall Y\, E$ and $LP(\Phi)$ be the formula and the disjunctive Datalog program defined above, respectively. Then, there is a one-to-one correspondence between the truth assignments $\phi(X)$ to $X = \{x_1, \ldots, x_n\}$ such that $\Phi_\phi = \forall Y\, E_{\phi(X)}$ is valid and the stable models of $LP(\Phi)$ which contain the atom $v$. In particular:*

1. *if $\Phi_\phi = \forall Y\, E_{\phi(X)} \in QBF_{1,\forall}$, then $M_\phi \in SM(LP(\Phi))$, and*
2. *if $M \in SM(LP(\Phi))$ contains $v$, then $\Phi_{\phi_M} \in QBF_{1,\forall}$.*

*Proof.* It follows from Theorems 3.1 of [12]. $\Box$

Note that $LP(\Phi)$ is constructible from $\Phi$ in polynomial time.

We are now ready to determine exactly the complexity of reasoning over programs with constraints.

**Theorem 19** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, and a literal $q$ as input, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_3^P$-complete.*

*Proof.* From Theorem 17 it remains to prove only hardness. $\Delta_3^P$-hardness is shown by exhibiting a DATALOG$^{\vee,\neg,c}$ program that, under brave reasoning, solves the following $\Delta_3^P$-complete problem $\Pi$: let $\phi(X)$, $X = \{x_1, \ldots, x_n\}$, be the lexicographically minimum truth assignment with respect to $\langle x_1, ..., x_n\rangle$ such that $\Phi_\phi = \forall Y\ E_{\phi(X)}$ is valid (we assume such a $\phi$ exists); now, is it true that $\phi(x_n) = true$?". W.l.o.g. we assume that $E$ is in 3DNF of the form defined above. Let $\mathcal{P}' = \{LP(\Phi), \emptyset, W'\}$ be a DATALOG$^{\vee,\neg,c}$ program where $LP(\Phi)$ is the positive disjunctive Datalog program defined above, and $W' = \langle\{\Leftarrow x_n\}, \cdots, \{\Leftarrow x_1\}, \{\Leftarrow \neg v\}\rangle$. We next show that the answer to $\Pi$ is true **iff** $x_n$ is true in some stable model of $\mathcal{P}'$.

Let $SM_v(LP(\Phi))$ denote the set of the stable models of $LP(\Phi)$ which contain $v$. From Lemma 18 we know that there is a one-to-one correspondence between $SM_v(LP(\Phi))$ and the set of truth assignments $\phi$ which make $\Phi_\phi$ valid. Since $\Phi_\phi$ is valid by hypothesis, this implies that $SM_v(LP(\Phi)) \neq \emptyset$. Consequently, $\mathcal{P}'$ has some stable models (as there is no strong constraint in $\mathcal{P}'$), and each stable model of $\mathcal{P}'$ contains $v$ as $\Leftarrow \neg v$ is the strongest constraint in $W'$. The priorities among constraints $\{\{\Leftarrow x_n\}, \cdots, \{\Leftarrow x_1\}\}$ impose a total order on $SM_v(LP(\Phi))$. In particular, given two stable models $M$ and $M'$ in $SM_v(LP(\Phi))$, $H_{\mathcal{P}',M} < H_{\mathcal{P}',M'}$ iff the truth assignment $\phi_{M'}$ is greater than $\phi_M$ in the lexicographically order. Therefore, $\mathcal{P}'$ has a unique stable model $M$ (which is in $SM_v(LP(\Phi))$), corresponding to the lexicographically minimum truth assignment $\phi$ such that $\Phi_\phi = \forall Y\ E_{\phi(X)}$ is valid. Hence, the lexicographically minimum truth assignment $\phi(X)$ making $\Phi_\phi$ valid fulfills $\phi(x_n) = true$ if and only if $x_n$ is true in some stable model of $\mathcal{P}'$. Therefore, brave reasoning is $\Delta_3^P$-hard for DATALOG$^{\vee,\neg,c}$. $\Box$

Next we show that neither strong constraints nor negation affect the complexity of DATALOG$^{\vee,\neg,c}$ that remains unchanged even if we disallow both of them.

**Corollary 20** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, and a literal $q$ as input, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_3^P$-complete even if strong constraints are disallowed ($S = \emptyset$) and $LP$ is stratified or even positive.*

*Proof.* Membership in $\Delta_3^P$ trivially holds from Theorem 17. As far as hardness is concerned, observe that in the reduction of Theorem 19 the DATALOG$^{\vee,\neg,c}$ program $\mathcal{P}'$ has no strong constraints (i.e., $S = \emptyset$) and its logic program $LP(\Phi)$ is positive (and therefore stratified). $\Box$

Thus, the syntactic restrictions imposed in Corollary 20 do not decrease the complexity of DATALOG$^{\vee,\neg,c}$. Interestingly, the complexity of the language decreases if we disallow priorities among weak constraints.

**Theorem 21** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, where $W$ consists of a single component, and a literal $q$ as input, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_3^P[O(\log n)]$-complete.*

*Proof.* $\Delta_3^P[O(\log n)]$-*Membership.* We proceed exactly as in the proof of Theorem 17. However, since $W$ consists of one only component, this time $k = maxH(\mathcal{P})$ is $O(|\mathcal{P}|)$ (while for programs with priorities is $O(2^{|\mathcal{P}|})$), as there is only one component in $W$ and its weight is 1. Consequently, the number of calls to the $\Sigma_2^P$ oracle is logarithmic in the size of the input, as it is logarithmic in $k$ (because we perform a binary search on $[0..k]$).

$\Delta_3^P[O(\log n)]$-*Hardness.* Let $\Phi_1, \ldots, \Phi_m$ be 2-existential QBFs, such that $\Phi_i \notin \mathrm{QBF}_{2,\exists}$ implies $\Phi_{i+1} \notin \mathrm{QBF}_{2,\exists}$, for $1 \leq i < m$. We next reduce the $\Delta_3^P[O(\log n)]$-hard problem of deciding whether $\max\{i : 1 \leq i \leq m, \Phi_i \in \mathrm{QBF}_{2,\exists}\}$ is odd to brave reasoning on DATALOG$^{\vee,\neg,c}$ programs without priorities (i.e., with one only component in $W$). W.l.o.g. we assume that 1) $m$ is even, 2) the same propositional variable does not appear in two distinct QBFs. and 3) the QBFs $\Phi_1, \ldots, \Phi_m$ are as in the proof of Theorem 19.

Now, let $\mathcal{P}' = (LP', S', W')$, where:

a) $LP' = LP(\Phi_1) \cup \cdots \cup LP(\Phi_m) \cup_{1 \leq i \leq m/2} \{odd \leftarrow v_{2i-1}, \neg v_{2i}\}$ and $LP(\Phi_i)$ is defined as in Theorem 19 apart from replacing each occurrence of $v$ by $v_i$;

b) $S' = \emptyset$;

c) $W' = \langle\{\Leftarrow \neg v_1;\ \Leftarrow \neg v_1, \neg v_2;\ \cdots \Leftarrow \neg v_1, ..., \neg v_m\}\rangle$.

The weak constraints in $W'$ enable to select the stable models of $LP(\Phi)$ where the number of valid QBFs is maximum. Then, $\max\{i : 1 \leq i \leq m, \Phi_i \in \mathrm{QBF}_{k,\exists}\}$ is odd if and only if *odd* is true in some stable model of $\mathcal{P}$. $\Box$

As for DATALOG$^{\vee,\neg,c}$ programs with prioritized weak constraints, the complexity is not affected at all by negation and strong constraints.

**Corollary 22** *Let $\mathcal{P} = (LP, S, W)$ be a DATALOG$^{\vee,\neg,c}$ program where $W$ has a single component. Then, brave reasoning is $\Delta_3^P[O(\log n)]$-complete even if $\mathcal{P}$ is subject to the following restrictions:*

*1) $S = \emptyset$, and*

*2) $LP$ is stratified or even positive.*

*Proof.* Membership in $\Delta_3^P[O(\log n)]$ comes from Theorem 21. Hardness under the restriction 1 for stratified programs comes from the hardness proof of Theorem 21, as the reduction there does not make use of both priorities and strong constraints and utilizes only stratified negation. To complete the proof we have only to demonstrate hardness for the case of positive programs. To this end, we just need to eliminate negation from the logic program $LP'$ (of the proof of Theorem 21) as follows:

1. Eliminate from $LP'$ all rules with head *odd* (that are the only rules containing negation).

2. Insert the following weak constraints in $W$:

$$\Leftarrow v_{2i-1}, \neg v_{2i}, even \qquad (1 \leq i \leq m/2)$$

□

# 5 Discussion on Complexity and Expressiveness of DATALOG$^{\vee,\neg}$

In this section we discuss the results on the complexity of DATALOG$^{\vee,\neg,c}$ that we established in the previous section. Interestingly, these results allow us to draw some useful remarks also on the expressive power of our language, which turns out to be strictly more powerful than disjunctive Datalog (without constraints). We show also that the increased expressive power of our language has a relevance even from the practical side, providing some meaningful examples that can be represented very naturally in DATALOG$^{\vee,\neg,c}$, while their disjunctive Datalog encoding is impossible (or so complex to be unusable).

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| | | $\{\}$ | $\{\mathbf{s}\}$ | $\{\mathbf{w}\}$ | $\{\mathbf{s}, \mathbf{w}\}$ | $\{\mathbf{w}^<\}$ | $\{\mathbf{s}, \mathbf{w}^<\}$ |
| 1 | $\{\}$ | P | P | P | P | P | P |
| 2 | $\{\neg^{\mathbf{s}}\}$ | P | P | P | P | P | P |
| 3 | $\{\neg\}$ | NP | NP | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P$ | $\Delta_2^P$ |
| 4 | $\{\vee^{\mathbf{h}}\}$ | NP | NP | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P$ | $\Delta_2^P$ |
| 5 | $\{\vee^{\mathbf{h}}, \neg^{\mathbf{s}}\}$ | NP | NP | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P$ | $\Delta_2^P$ |
| 6 | $\{\vee^{\mathbf{h}}, \neg\}$ | NP | NP | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P[O(\log n)]$ | $\Delta_2^P$ | $\Delta_2^P$ |
| 7 | $\{\vee\}$ | $\Sigma_2^P$ | $\Sigma_2^P$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P$ | $\Delta_3^P$ |
| 8 | $\{\vee, \neg^{\mathbf{s}}\}$ | $\Sigma_2^P$ | $\Sigma_2^P$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P$ | $\Delta_3^P$ |
| 9 | $\{\vee, \neg\}$ | $\Sigma_2^P$ | $\Sigma_2^P$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P[O(\log n)]$ | $\Delta_3^P$ | $\Delta_3^P$ |

Table 1: The Complexity of Brave Reasoning in various Extensions of Datalog with Constraints (Propositional Case under Stable Model Semantics)

Table 1 sumarizes the complexity results of the previous section, complemented with other results (on the complexity of programs without constraints) already known in the literature. Therein, each column refers to a specific form of constraints, namely: $\{\}$ = no constraints, $s$ = strong constraints, $w^<$ = weak constraints with priorities, $w$ = weak constraints without priorities (i.e., $W$ has only one component). The lines of Table 1 specify the allowance of disjunction and negation; in particular, $\neg^s$ stands for stratified negation [45] and $\vee^h$ stands for HCF disjunction [3] (see AppendixB). Each entry of the table provides the complexity class of the corresponding fragment of the language. For an instance, the entry $(8, 5)$ defines the fragment of DATALOG$^{\vee,\neg,c}$ allowing disjunction and stratified negation ($\{\vee, \neg^s\}$) and, as far as constraints is concerned, only weak constraints with priorities ($\{w^<\}$); that is, unstratified negation and strong constraints are disallowed in this fragment. The languages DATALOG$^{\neg}$ and DATALOG$^{\vee,\neg}$ correspond to the fragments (3,1) and (9,1), respectively. Full DATALOG$^{\vee,\neg,c}$ is the fragment (9,6), whose complexity is reported in the lower right corner of the table. The corresponding entry in the table, namely $\Delta_3^P$, expresses that the brave reasoning for DATALOG$^{\vee,\neg,c}$ is $\Delta_3^P$-complete.

The first observation is that DATALOG$^{\vee,\neg,c}$ is strictly more expressive than DATALOG$^{\vee,\neg}$. Indeed, the former language can express all problems that can be represented in the latter, as DATALOG$^{\vee,\neg,c}$ is an extension of DATALOG$^{\vee,\neg}$. Moreover, the $\Delta_3^P$-completeness of DATALOG$^{\vee,\neg,c}$ witnesses that DATALOG$^{\vee,\neg,c}$ can express some $\Delta_3^P$-complete problems, while DATALOG$^{\vee,\neg}$ cannot (unless the Polynomial Hierarchy collapses) [15]. An example of encoding of a $\Delta_3^P$-complete problem has been reported in the proof of Theorem 19. To see another example, consider again abductive reasoning addressed in Section 3.3. A hypothesis is *relevant* for a logic programming abductive problem $\mathbf{P}$ if it occurs in *some* explanation of $\mathbf{P}$; under minimum cardinality (resp. prioritized) abduction, relevance requires membership to a minimum cardinality solution (resp. minimal solution according to the prioritization). From the results of [13, 14], deciding relevance is $\Delta_3^P[O(\log n)]$-complete for abduction with minimum cardinality criterion, while it is $\Delta_3^P$-complete for prioritized abduction. Therefore, relevance of an hypothesis $h$ for a logic programming abduction problem cannot be expressed reasoning on disjunctive Datalog (as $\Sigma_2^P$ is an upper bound to the complexity of brave reasoning on DATALOG$^{\vee,\neg}$ [15]). On the other hand, from the translation we provided in Section 3.3, it is easy to see that $h$ is relevant for $\mathbf{P}$ iff it is a brave consequence of $pr(\mathbf{P})$ (i.e., relevance can be reduced to brave reasoning on DATALOG$^{\vee,\neg,c}$). [8]

Considering that DATALOG$^{\vee,\neg,c}$ is a linguistic extension of DATALOG$^{\vee,\neg}$ by constraints, it turns out that constraints do add expressive power to DATALOG$^{\vee,\neg}$. However, it is not the case of strong constraints, as it can be seen from Table 1. Indeed, if we look at the various fragments of the language that differ only for the presence of strong constraints, we can note that complexity is constant (compare column 1 to 2, or 3 to 4, or 5 to 6). In fact, under stable model semantics, a strong constraint of the form $\leftarrow B$ is actually a shorthand for $p \leftarrow B, \neg p$. As an example, consider the problem SCHEDULING of Section 1. The program $\mathcal{P}_{sch}$ we used to express SCHEDULING relies on $HCF$ disjunction with stratified negation and contains only one strong constraint (no weak constraints occur in it - fragment (5,2)). This problem can be easily formulated in $HCF$ DATALOG$^{\vee,\neg}$ with unstratified negation (fragment (6,1)) simply by replacing the strong constraint $\leftarrow assign(X,S), assign(Y,S), incompatible(X,Y)$ by the DATALOG$^{\neg}$ rule

$$p \leftarrow assign(X,S), assign(Y,S), incompatible(X,Y), \neg p$$

where $p$ is a new atom.[9] It is worth noting that brave reasoning allows us to simulate strong constraints if negation is not allowed in the program: replace each strong constraint $\leftarrow B$ by the rule $p \leftarrow B$ (where $p$ is a fresh atom), and require that $\neg p$ is a brave consequence.

Thus, it is rather clear that strong constraints do not affect at all the complexity and expressiveness of the language. Therefore, the increased expressive power of DATALOG$^{\vee,\neg,c}$ (w.r.t. DATALOG$^{\vee,\neg}$) is due entirely to weak constraints. However, weak constraints do not

---

[8]Note that we have $\Delta_3^P$-hardness only if disjunction is allowed in the abductive theory $LP$. Thus, example $\mathbf{P}_{net}$ is not an hard instance of the problem; but the translation we gave is general and holds also for hard instances with disjunction in the abductive theory $LP$.

[9]actually, the SCHEDULING problem can be even formulated in DATALOG$^{\neg}$, as $HCF$ disjunction can be simulated by unstratified negation. Indeed, the "guess" expressed by the disjunctive rule can be implemented by the following three nondisjunctive rules

$$\begin{array}{ll} assign(X,ts_1) & \leftarrow course(X), \neg assign(X,ts_2), \neg assign(X,ts_3) \\ assign(X,ts_2) & \leftarrow course(X), \neg assign(X,ts_1), \neg assign(X,ts_3) \\ assign(X,ts_3) & \leftarrow course(X), \neg assign(X,ts_1), \neg assign(X,ts_2) \end{array}$$

cause a tremendous increase of the computational cost, they "mildly" increase the complexity, as we can see by comparing column 1 to columns 3 and 5 of Table 1. For example, if we add weak constraints to $HCF$ disjunction (with or without negation - see lines 4-6) we increase the complexity from NP to $\Delta_2^P[O(\log n)]$ or $\Delta_2^P$, depending on whether weak constraints are prioritized or not, respectively. Note that the program $\mathcal{P}_{cliq}$ of Example 9 consists of a weak constraint on top of an HCF program. Likewise, weak constraints added to DATALOG$^{\vee,\neg}$ increase its complexity from $\Sigma_2^P$ to $\Delta_3^P[O(\log n)]$ or $\Delta_3^P$, depending on the possible prioritization of weak constraints (see lines 7-9). Therefore, adding weak constraints, the complexity of the language always remains at the same level of the polynomial hierarchy; for instance, from NP we can reach $\Delta_2^P[O(\log n)]$ or $\Delta_2^P$, but we will never jump to $\Sigma_2^P$ or to other classes of the second level of the polynomial hierarchy.

Clearly, weak constraints do not impact on the complexity of fragments admitting a unique stable model (see lines 1 and 2).

Further remarks on the complexity of the various fragments of the language are the following. First, the presence of negation does not impact on the complexity of the fragments allowing disjunction (see lines 4 to 9). Second, the complexity of the $\vee$-free fragments with full negation (see line 3) coincides with that of $HCF$ disjunction (lines 4 to 6) and lies one level down in the polynomial hierarchy with respect to fragments based on plain DATALOG$^{\vee,\neg}$ (see line 9).

Besides the increase in theoretical expressiveness – the ability to express problems that cannot be represented in DATALOG$^{\vee,\neg}$ – DATALOG$^{\vee,\neg,c}$ provides another even more important advantage: several problems that can be represented also in DATALOG$^{\vee,\neg}$ (whose complexity is thus not greater than $\Sigma_2^P$) admit a much more natural and easy-to-understand encoding in DATALOG$^{\vee,\neg,c}$. This is because coupling both disjunction and constraints, provides a powerful and elegant tool to encode knowledge. To better appreciate the simplicity and readability of DATALOG$^{\vee,\neg,c}$, consider again the problem MAX CLIQUE, that has been represented very simply and elegantly in DATALOG$^{\vee,\neg,c}$ (see Example 9). We next show that it can be encoded also in DATALOG$^{\vee,\neg}$ but, unfortunately, the resulting program is very tricky and difficult to understand. The technique we use to build a DATALOG$^{\vee,\neg}$ version of MAX CLIQUE is that described in [15], based on a modular approach. A first module $LP_1$ generates candidate solutions (cliques in this case) and a second module $LP_2$ discards those that do not satisfy certain criteria (maximal cardinality here).

$r_1:\ c(X) \vee not\_c(X) \leftarrow node(X)$
$r_2:\ p \leftarrow c(X), c(Y), X \neq Y, \neg edge(X,Y), \neg p$
$r_3:\ count(0,0) \leftarrow$
$r_4:\ count(I,J) \leftarrow c(I), count(A,B), succ(A,I), succ(B,J)$
$r_5:\ count(I,B) \leftarrow not\_c(I), count(A,B), succ(A,I)$
$r_6:\ c\_size(K) \leftarrow greatestVertex(X), count(X,K)$
$r_7:\ greatestVertex(X) \leftarrow node(X), \neg greaterThan(X)$
$r_8:\ greaterThan(X) \leftarrow succ(X,Y)$

The above disjunctive program $LP_1$ generates all cliques and their sizes. We have assumed that the nodes are ordered by the relation *succ* that is provided in input (but could also be "guessed" by another module). Rule $r_1$ guesses a clique $c$ and rule $r_2$ enforces the constraint that every two nodes in $c$ must be joined by an edge. By this rule the constraint must be satisfied in every stable model (if the constraint is unsatisfiable, then no stable model exist). The atom $count(I, J)$ expresses that the node $I$ is the $J$-th node of the clique $c$. Thus, rule $r_6$ defines the size (number of nodes) of $c$.

Now, to restrict to only maximum size cliques, we add to $LP_1$ the following DATALOG$^{\vee,\neg}$ program $LP_2$ which discards clique nonmaximal in size.

$$
\begin{array}{ll}
r'_1: & c'(X) \vee not\_c'(X) \leftarrow node(X) \\
r'_2: & notGreater \leftarrow c'(X), c'(Y), X \neq Y, \neg edge(X, Y) \\
r'_3: & count1(0,0) \leftarrow \\
r'_4: & count1(I, J) \leftarrow count1(A, B), succ(A, I), succ(B, J), c'(I) \\
r'_5: & count1(I, B) \leftarrow count1(A, B), succ(A, I), not\_c'(I) \\
r'_6: & c'\_size(K) \leftarrow greatestVertex(N), count1(N, K) \\
 & \\
r'_7: & notGreater \leftarrow c\_size(K), c'\_size(I), I \leq K \\
r'_8: & c'(X) \leftarrow node(X), notGreater \\
r'_9: & not\_c'(X) \leftarrow node(X), notGreater \\
r'_{10}: & q \leftarrow \neg notGreater, \neg q
\end{array}
$$

Here, $r'_1$ guesses another clique $c'$. *notGreater* is true if $c'$ is not a clique. *count*1 counts the elements of $c'$ and $c'\_size$ define the size of $c'$. If the size of $c'$ is less than or equal to that of $c$ then *notGreater* is true. If so, $c'$ and $not\_c'$ are assigned the maximum extension by rules $r'_8$ and $r'_9$. In this way, if all cliques $c'$ are smaller than $c$ then all stable models of $LP_2$ collapse into a single stable model with the maximum extension for both $c$ and $c'$ and containing *notGreater*. If, on the contrary, there is some clique $c'$ bigger than $c$, then no stable model of $LP_2$ contains *notGreater*. Thus, the rule $r'_{10}$ imposes every stable model to contain *notGreater* (if any). It is easy to recognize that, for the program $LP = LP_1 \cup LP_2$, there is a one-to-one correspondence between stable models and maximum cliques of $G$.

Comparing the above DATALOG$^{\vee,\neg}$ program with the DATALOG$^{\vee,\neg,c}$ version of Example 9,[10] it is quite apparent the advantage that weak constraints provide in terms of simplicity and naturalness of programming.

Concluding, we would like to bring reader's attention to the fragment of DATALOG$^{\vee,\neg,c}$ with $HCF$ disjunction and stratified negation ((5,6) in Table 1): it has a very clear and easy-to-understand semantics and, at the same time, allows us to express several hard problems (up to $\Delta_2^P$-complete problems) in a natural and compact fashion. (In our opinion, recursion through disjunction or negation makes programs more difficult to understand).

[10]Perhaps our DATALOG$^{\vee,\neg}$ encoding of MAX CLIQUE is not the best; but we are very skeptical on the existence of a DATALOG$^{\vee,\neg}$ encoding which is simpler than the DATALOG$^{\vee,\neg,c}$ encoding (of example 9).

# References

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

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

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

[4] Ben-Eliyahu, R., Palopoli, L. (1994), Reasoning with Minimal Models: Efficient Algorithms and Applications. In *Proc. Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pp. 39–50.

[5] Brass, S., Dix, J. (1995), Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation, In *Proc. of the 12th Int. Conf. on Logic Programming*, Tokyo, pp. 199–213, MIT Press.

[6] Brewka, G. (1991), *Nonmonotonic Reasoning: Logical Foundations of Commonsense.* Cambridge University Press.

[7] Brewka, G., Eiter, T. (1998), Preferred Answer Sets for Extended Logic Programs, In *Proc. KR'98*. Forthcoming.

[8] Cadoli, M., Schaerf, M. (1993), A Survey of Complexity Results for Non-monotonic Logics, *Journal of Logic Programming*, **17**, pp. 127-160.

[9] Chomicki, J., Subrahmanian, V.S. (1990), Generalized Closed World Assumption is $\Pi^0_2$-Complete. *Information Processing Letters*, **34**, pp. 289–291.

[10] Decker, H., Celma, M. (1994) A slick procedure for integrity checking in deductive databases, In *Proc. of the Eleventh Int. Conference on Logic Programming*, S. Margherita Ligure", Italy, pp. 456 – 469, MIT Press.

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

[12] Eiter, T., Gottlob, G. (1995), On the Computational Cost of Disjunctive Logic Programming: Propositional Case, *Annals of Mathematics and Artificial Intelligence*, J. C. Baltzer AG, Science Publishers, **15**, 289–323.

[13] Eiter, T., Gottlob, G., Leone, N. (1997), Abduction From Logic Programs: Semantics and Complexity. *Theoretical Computer Science – Logic, Semantics and Theory of programming*, Elsevier, **189**(1-2), pp. 129–177.

[14] Eiter, T., Gottlob, G. (1995) The Complexity of Logic-Based Abduction, *Journal of the ACM*, **42**(1), 3–42.

[15] Eiter, T., Gottlob, G., Mannila, H. (1997) Disjunctive Datalog, *ACM Trans. Database Syst.*, **22**(3), 315–363.

[16] Fagin, R., Ullman, J. D., Vardi, M. Y. (1983) On the Semantics of Updates in Databases, In *Proc. PODS-83*, pp 352–365.

[17] Fernández, J.A., Lobo, J., Minker, J., Subrahmanian V.S. (1993), Disjunctive LP + Integrity Constraints = Stable Model Semantics, *Annals of Mathematics and Artificial Intelligence*, **8**(3-4), 449–474.

[18] Fernández, J.A., Minker, J. (1992), Semantics of Disjunctive Deductive Databases, In *Proc. 4th Intl. Conference on Database Theory (ICDT-92)*, Berlin, pp. 21–50.

[19] Garey, M., Johnson, D.S. (1979) *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York.

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

[21] Gottlob, G. (1994), Complexity and Expressive Power of Disjunctive Logic Programming, In *Proc. of the International Logic Programming Symposium (ILPS-'94)*, M. Bruynooghe ed., Ithaca NY, pp. 23–42, MIT Press.

[22] Ginsberg, M. L. (1986) Counterfactuals, *Artificial Intelligence*, **30**, 35–79.

[23] Greco, S. (1996), Extending Datalog with Choice and Weak Constraints, In *Proc. of the Joint Conference on Declarative Programming APPIA-GULP-PRODE'96*, Donostia-San Sebastian, Spain, pp.329–340.

[24] Greco, S., Saccà, D. (1997), NP Optimization Problems in Datalog, In *Proc. of the International Logic Programming Symposium (ILPS '97)*, Port Jefferson, USA.

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

[26] Kadin, J. (1989), $P^{NP[O(\log n)]}$ and Sparse Turing-Complete Sets for $NP$, *Journal of Computer and System Sciences*, **39**, 282–298.

[27] Krentel, M. (1988), The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, **36**, 490–509.

[28] Kowalski, R., Sadri, F. (1988) A theorem-proving approach to database integrity, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 313–362, Morgan Kaufman.

[29] 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.

[30] Leone, N., Rullo, P., Scarcello, F. (1997) Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation, *Information and Computation*, Academic Press, **135**(2), 69–112.

[31] Lifschitz, V. (1985) Computing Circumscription, In *Proc. of IJCAI-85*, pp 121–127.

[32] Lloyd, J. W. (1984) *Foundations of Logic Programming.* Springer, Berlin.

[33] Lloyd, J.W., Sonenberg, E.A., Topor, R.W. (1987) Integrity constraint checking in stratified databases, *Journal of Logic Programming*, **2**, 331–343.

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

[35] Kakas, A., Mancarella, P. (1990) Generalized stable models: a semantics for abduction, In *Proc. of ECAI-90*, pp. 385–391.

[36] Kakas, A., Kowalski, R., Toni, F. (1992) Abductive Logic Programming. *Journal of Logic and Computation*, **2**(6), 719–771.

[37] Krentel, M. (1992), Generalizations of OptP to the Polynomial Hierarchy, *Theoretical Computer Science*, **97**(2) 183–198.

[38] Marek, W., Subrahmanian, V.S. (1989), The Relationship between Logic Program Semantics and Non-Monotonic Reasoning, In *Proc. of the 6th International Conference on Logic Programming – ICLP'89*, pp. 600–617, MIT Press.

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

[40] 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.

[41] Papadimitriou, C.H. (1994), *Computational Complexity*, Addison-Wesley.

[42] Peng, Y., Reggia, J. (1990) *Abductive Inference Models for Diagnostic Problem Solving*, Springer.

[43] Peirce, C.S. (1955) Abduction and induction, In J. Buchler, editor, *Philosophical Writings of Peirce*, chapter 11. Dover, New York.

[44] Przymusinska, H., Przymusinski, T. (1988), Weakly Perfect Model Semantics for Logic Programs, In *Proc. Fifth Int. Conf. and Symp. on Logic Programming*, pp.1106-1120.

[45] 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.

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

[47] Przymusinski, T. (1995), Static Semantics for Normal and Disjunctive Logic Programs, *Annals of Mathematics and Artificial Intelligence*, J. C. Baltzer AG, Science Publishers, **14**, 323–357.

[48] Ross. K.A. (1990), The Well Founded Semantics for Disjunctive Logic Programs, *in* "Deductive and Object–Oriented Databases," W. Kim, J.–M. Nicolas and S. Nishio, ed., pp.385–402, Elsevier Science Publishers B. V.

[49] Ross, K.A. (1990), Modular Stratification and Magic Sets for Datalog Programs with Negation, In *Proc. ACM Symposium on Principles of Database Systems*, pp.161-171.

[50] D. Saccà (1997) The Expressive Powers of Stable Models for Bound and Unbound DATALOG Queries, *Journal of Computer and System Sciences*, **54**(3), 441–464.

[51] Sakama, C. (1989), Possible model semantics for disjunctive databases, In *Proc. of the first international conference on deductive and object oriented databases*, pp.1055–1060.

[52] Schlipf, J.S. (1990) The Expressive Powers of Logic Programming Semantics, In *Proc. ACM Symposium on Principles of Database Systems*, pp. 196-204.

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

[54] Stockmeyer, L., Meyer, A. (1973), Word Problems Requiring Exponential Time, In *Proceedings of the Fifth ACM Symposium on the Theory of Computing*, pp. 1–9.

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

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

[57] Vardi, M. (1982), Complexity of relational query languages, In *Proc. oh the 14th ACM STOC*, pp. 137–146.

[58] Wagner, K. (1990), Bounded query classes; *SIAM J. Comp.*, **19**(5), 833–846.

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

# Appendices

## A Sample Optimization Problems in DATALOG$^{\vee,\neg,c}$

We list below a number of optimization problems [19] and their representation in DATALOG$^{\vee,\neg,c}$.

*MAX SATISFIABILITY.* Let $E$ be a boolean formula in conjunctive normal form; find a truth assignment that satisfy simultaneously the maximum number of clauses of $E$. To this end, we are given two unary relations *clause* and *var* such that a fact $clause(x)$ says that $x$ is a clause and $var(v)$ that $v$ is a variable occurring in some clause. We also have two binary relations *pos* and *neg* such that $pos(x, v)$ and $neg(x, v)$ say that the variable $v$ occurs in the clause $x$ positively or negatively, respectively. Finally, $val(v, t)$ (resp. $val(v, f)$ expresses that the variable $V$ is assigned the truth value $t$(rue) (resp. $f$(alse)).

$$
\begin{array}{l}
val(V,t) \vee val(V,f) \leftarrow var(V) \\
sat(X) \leftarrow clause(X), pos(X,V), val(V,t) \\
sat(X) \leftarrow clause(X), neg(X,V), val(V,f) \\
\Leftarrow clause(X), \neg sat(X)
\end{array}
$$

*MIN NODE COVER.* Let $G = \langle V, E \rangle$ be a graph. Find a minimum node cover, that is, a subset of minimum cardinality $V'$ of $V$ such that for each edge $\{u, v\} \in E$ at least one of $u$ and $v$ belongs to $V'$.

$$
\begin{array}{l}
cover(X) \vee not\_cover(X) \leftarrow node(X) \\
\leftarrow edge(X,Y), \neg cover(X), \neg cover(Y) \\
\Leftarrow cover(X)
\end{array}
$$

*MIN KERNEL.* Let $G = \langle V, E \rangle$ be a directed graph. Find a minimum kernel, that is, a subset of minimum cardinality $V'$ of $V$ such that no two nodes in $V'$ are joined by an edge in $E$ and, further, for every node $u$ in $V - V'$ there is a node $v$ in $V'$ such that $(v, u) \in E$.

$$
\begin{array}{ll}
r_1: & kernel(X) \vee not\_kernel(X) \leftarrow node(X) \\
s_1: & \leftarrow kernel(X), kernel(Y), edge(X,Y), X \neq Y \\
s_2: & \leftarrow not\_kernel(X), \neg connected\_to\_kernel(X) \\
r_2: & connected\_to\_kernel(X) \leftarrow kernel(Y), edge(Y,X) \\
w_1: & \Leftarrow kernel(X)
\end{array}
$$

*MAX CUT.* Let $G = \langle V, E \rangle$ be a graph. A maximal cut is a partition of $V$ into sets $V_1$ and $V_2$ such that the number of edges of $G$ having one endpoint in $V_1$ and one endpoint in $V_2$ is maximum.

$$
\begin{array}{ll}
r_1: & v_1(X) \vee v_2(X) \leftarrow vertex(X) \\
w_1: & \Leftarrow v_1(X), v_1(Y), edge(X,Y) \\
w_2: & \Leftarrow v_2(X), v_2(Y), edge(X,Y)
\end{array}
$$

*MAX BIPARTITE SUBGRAPH.* Let $G = \langle V, E \rangle$ be a planar graph. Find a subset $E' \subseteq E$ of maximum cardinality such that $G' = \langle V, E' \rangle$ is bipartite.

$$\begin{array}{ll} r_1: & e'(X,Y) \vee not\_e'(X,Y) \leftarrow edge(X) \\ r_2: & v'(X) \vee not\_v'(X) \leftarrow vertex(X) \\ s_1: & \leftarrow v'(X), v'(Y), e'(X,Y) \\ s_2: & \leftarrow not\_v'(X), not\_v'(Y), e'(X,Y) \\ w_1: & \Leftarrow not\_e'(X,Y) \end{array}$$

# B Complexity of Head-Cycle Free and Normal DATALOG$^{\vee,\neg,c}$ Programs

## B.1 Preliminaries: HCF and Stratified Programs

A disjunctive Datalog program $LP$ is *(locally) stratified* if each atom $p \in B_{LP}$ can be associated a positive integer $l(p)$ such that, for each rule $r \in ground(LP)$ with $p \in H(r)$ and each ground atom $q$ the following holds: (i) if $q \in H(r)$, then $l(p) = l(q)$; (ii) if $\neg q \in B^-(r)$, then $l(p) > l(q)$; (iii) if $q \in B^+(r)$, then $l(p) \geq l(q)$. In particular, if all ground atoms with the same predicate have the same number associated, then $LP$ is globally stratified.

**Proposition 23** *Let $LP$ be a ground normal logic program. If $LP$ is stratified, then $LP$ has exactly one stable model that can be computed in polynomial time.*

HCF disjunction is a syntactical property of programs that gives no account to negation. A program $LP$ is *(locally) head-cycle-free* if each atom $p \in B_{LP}$ can be associated a positive integer $l(p)$ such that, for each rule $r \in ground(LP)$ with $p \in H(r)$ and each ground atom $q \neq p$ the following holds: (i) if $q \in H(r)$, then $l(p) \neq l(q)$; (ii) if $q \in B^+(r)$, then $l(p) \geq l(q)$. In particular, if all ground atoms with the same predicate have the same number associated, then $LP$ is globally head-cycle-free.

The following proposition, which is implicit in [3], is straightforward from the fact that checking whether a total model of a positive HCF program is minimal is polynomial [4].

**Proposition 24** *(cf. [3, 4]) Let $LP$ be a ground HCF program and let $M$ be an interpretation of it. Then, deciding if $M$ is a stable model of $LP$ is polynomial.*

We say that a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ is *head-cycle-free* (resp., *stratified*) iff $LP$ is head-cycle-free (resp., stratified).

## B.2 Complexity of Head-Cycle-Free Disjunctive Programs

In the case of Disjunctive Datalog without constraints, the complexity of brave reasoning falls back from $\Sigma_2^P$ to NP if disjunction is required to be head-cycle-free (often, simply HCF).

Not surprisingly, we obtain a decrease of one level in the polynomial hierarchy (from $\Delta_3^P$ and $\Delta_3^P[O(\log n)]$ to $\Delta_2^P$ and $\Delta_2^P[O(\log n)]$, respectively) in the complexity of brave reasoning when we impose head-cycle-free disjunction to DATALOG$^{\vee,\neg,c}$.

**Lemma 25** *Let a positive integer $n$ and a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ be given as input, where $LP$ is head-cycle-free. It is in* NP *deciding whether there exists a stable model $M$ of $LP$ satisfying $S$ such that $H_{\mathcal{P},M} \leq n$*

*Proof.* We can decide the problem as follows. Guess $M \subseteq B_{\mathcal{P}}$, and check that: (1) $M$ is a stable model of $LP$, (2) all constraints in $S$ are satisfied, and (3) $H_{\mathcal{P},M} \leq n$. Clearly, property (2) and (3) can be checked in polynomial time. Moreover, since $LP$ is head-cycle-free, stable model checking (i.e., task (1)) is also polynomial [30, 3]. Hence, the problem lies in NP. □

**Lemma 26** *Let a positive integer $n$ and a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ given as input, where $LP$ is head-cycle-free. It is in* NP *deciding whether there exists a stable model $M$ of $LP$ satisfying $S$ such that $H_{\mathcal{P},M} = n$ and $q$ is true w.r.t. $M$.*

*Proof.* We can decide the problem as follows. Guess $M \subseteq B_{\mathcal{P}}$, and check that: (1) $M$ is a stable model of $LP$, (2) all constraints in $ground(S)$ are satisfied, (3) $H_{\mathcal{P},M} = n$, and (4) $q$ is true w.r.t. $M$. Clearly, property (2), (3), and (4) can be checked in polynomial time. Moreover, since $LP$ is head-cycle-free, stable model checking (i.e., task (1)) is also polynomial [30, 3]. Therefore, problem is in NP. □

**Theorem 27** *Let a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, and a literal $q$ be given as input, where $LP$ is an HCF program. Then, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_2^P$-complete.*

*Proof.* The proof of $\Delta_2^P$-membership is exactly like in Theorem 17, where the $\Sigma_2^P$ oracles are replaced by NP oracles, because the respective tasks are one level down in the polynomial hierarchy in this case (just compare the results of Lemma 15 and 16 with those of Lemma 25 and 26, respectively).

We show $\Delta_2^P$-hardness by a polynomial time reduction of the following $\Delta_2^P$-complete problem [26, 27]: Given a satisfiable clause set $C = \{C_1, \ldots, C_m\}$ on $X = \{x_1, \ldots, x_n\}$, decide whether the with respect to $\langle x_1, \ldots, x_n \rangle$ lexicographically minimum $\phi(X)$ satisfying $C$, which we denote by $\phi_{min}(X)$, fulfills $\phi_{min}(x_n) = true$.

Consider the normal program $LP$ containing: (i) for each clause $a_1 \vee \cdots \vee a_l \vee \neg b_1 \vee \cdots \vee \neg b_k \in C$ $(k, l \geq 0)$, the rule

$$abs \leftarrow b_1, \cdots, b_k,\ \neg a_1,\ \cdots,\ \neg a_l$$

and, (ii) for each $1 \leq i \leq n$, the rule $x_i \vee x'_i \leftarrow$; where $abs$ and $x'_i$ $(1 \leq i \leq n)$ are fresh atoms.

Clearly, $LP$ is head-cycle-free. Moreover, there is a one-to-one correspondence between the stable models of $LP$ which do not contain $abs$ and the truth assignments to $X$ satisfying $C$. In particular, given a stable model $M$ of $LP$ not containing $abs$, the truth assignment $\phi_M$ satisfies $C$, where $\phi_M(x_i) = \mathit{true}$ iff $x_i \in M$. Vice versa, given a truth assignment $\phi$ satisfying $C$, the interpretation $I_\phi$ is a stable model of $LP$ which does not contain $abs$, where

$$I_\phi = \{x_i \in \{x_1, \ldots, x_n\} \mid x_i \in M\} \cup \{x'_i \in \{x'_1, \ldots, x'_n\} \mid x'_i \in M\}$$

Consider now the DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, where $LP$ is the HCF program above, $S = \{\leftarrow abs\}$ and $W = \langle\{\Leftarrow x_n\}, \cdots, \{\Leftarrow x_1\}\rangle$. It is easy to see that $\mathcal{P}$ has exactly one stable model $M$. Moreover, the lexicographically minimum truth assignment $\phi_{min}(X)$ satisfying $C$ fulfills $\phi_{min}(x_n) = \mathit{true}$ if and only if $x_n$ is true in $M$ (that is, iff $x_n$ is a brave consequence of $\mathcal{P}$).

Therefore, brave reasoning on HCF DATALOG$^{\vee,\neg,c}$ programs is $\Delta_2^P$-complete. $\Box$

**Theorem 28** *Given a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$, where $LP$ is an HCF program, and a literal $q$ as input, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_2^P$-complete even if strong constraints are disallowed ($S = \emptyset$) and $LP$ is stratified or even positive.*

*Proof.* Membership comes from previous theorem. To show hardness, we have to eliminate both strong constraints and negation from program $\mathcal{P}$ of the prof of theorem 27. To this end we redefine $LP$ as follows: (i) for each clause $a_1 \vee \cdots \vee a_l \vee \neg b_1 \vee \cdots \vee \neg b_k \in C$ $(k, l \geq 0)$, $LP$ contains the rule

$$abs \leftarrow b_1, \cdots, b_k,\ a'_1, \cdots,\ a'_l$$

and, (ii) for each $1 \leq i \leq n$, $LP$ contains the rule $x_i \vee x'_i \leftarrow$; where $abs$ and $x'_i$ $(1 \leq i \leq n)$ are fresh atoms. Moreover, $W$ is $\langle\{\Leftarrow x_n\}, \cdots, \{\Leftarrow x_1\}, \{\Leftarrow \neg abs\}\rangle$, and $S = \emptyset$.

The new DATALOG$^{\vee,\neg,c}$ program $\mathcal{P}$ behaves exactly like the program of Theorem 27. Therefore, the lexicographically minimum truth assignment $\phi_{min}(X)$ satisfying $C$ fulfills $\phi_{min}(x_n) = \mathit{true}$ if and only if $x_n$ is true in $M$ (that is, iff $x_n$ is a brave consequence of $\mathcal{P}$). $\Box$

**Theorem 29** *Let a ground literal $q$ and a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ given as input, where $LP$ is HCF and $W$ consists of a single component. Then, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_2^P[O(\log n)]$-complete.*

*Proof.* Membership stems by applying the same reasoning we did in Theorem 21.

To show $\Delta_2^P$-hardness we resort to abductive reasoning tasks discussed in Section 3.3. Given an *LPAP* (see Section 3.3), $\mathbf{P} = \langle Hyp, Obs, LP \rangle$, where $LP$ is a normal logic program, and a ground atom $\hbar$, deciding whether $\hbar$ belongs to some cardinality mimimal explanation of $\mathbf{P}$ is $\Delta_2^P[O(\log n)]$-hard [13]. (This decision problem is called *relevance* in the literature, and $\hbar$ is said *relevant* for $\mathbf{P}$.) Now, consider the DATALOG$^{\vee,\neg,c}$ program of Section 3.3 $pr(\mathbf{P}) = (LP \cup G, S, \langle W_0 \rangle)$, where

$$S = \{ \leftarrow \neg.o \mid o \in Obs \}$$

$$G = \{ h \vee not\_h \leftarrow \ \mid h \in Hyp \}$$

$$W_0 = \{ \Leftarrow h \mid h \in Hyp \}$$

$LP \cup G$ is clearly head-cycle-free (as $LP$ is $\vee$-free and the rules in $G$ have an empty body). Since the cardinality minimal solutions of $\mathbf{P}$ correspond one-to-one to the stable models of $pr(\mathbf{P})$, it is easy to see that $\hbar$ is relevant for $\mathbf{P}$ (under the cardinality minimal criterion) iff $pr(\mathbf{P}) \models_{brave} \hbar$. Therefore, brave reasoning on HCF DATALOG$^{\vee,\neg,c}$ programs is $\Delta_2^P$-complete. $\Box$

**Theorem 30** *Let a ground literal $q$ and a DATALOG$^{\vee,\neg,c}$ program $\mathcal{P} = (LP, S, W)$ given as input, where $LP$ is HCF and $W$ consists of a single component. Then, deciding whether $q$ is true in some stable model of $\mathcal{P}$ is $\Delta_2^P[O(\log n)]$-complete even under the following further restictions on $\mathcal{P}$:*

*1) $S = \emptyset$, and*

*2) $LP$ is stratified or even positive.*

*Proof.* Membership comes from Theorem 29. We recall that deciding whether a ground atom $\hbar$ belongs to some cardinality mimimal explanation of a given *LPAP* $\mathbf{P}$ is $\Delta_2^P[O(\log n)]$-hard even if $LP$ is a positive program (i.e., a definite Horn theory), and $Obs = \{o_1, ..., o_m\}$ is made of positive literals only [14]. Therefore, we can use a reduction from this problem (we further assume that $\mathbf{P}$ has at least one explanation; it is clear that this assumption does not decrease the complexity of the problem). To this end, consider the following DATALOG$^{\vee,\neg,c}$ program $pr(\mathbf{P}) = (LP \cup G, \emptyset, \langle W_0 \rangle)$, where

$$G = \{ h \vee not\_h \leftarrow \ \mid h \in Hyp \} \cup \{ o_i \vee o_i' \leftarrow \ \mid o_i \in Obs \} \cup \{ h \leftarrow o_i' \mid o_i \in Obs \ h \in Hyp \}$$

$$W_0 = \{ \Leftarrow h \mid h \in Hyp \}$$

It is evident that $LP \cup G$ is a positive HCF program. The rules in $\{h \leftarrow o_i' \mid o_i \in Obs\ h \in Hyp\}$ enforce that all observations are entailed in the stable models of $\mathcal{P}$ (if one observation is missed then all constraints of $W_0$ are violated). Because of the constraints in $W_0$, the stable models of $\mathcal{P}$ correspond one-to-one to the cardinality minimal explainations of $\mathbf{P}$. Thus, $\hbar$ belongs to a cardinality minimal explaination of $\mathbf{P}$ if and only if $\mathcal{P} \models_{brave} \hbar$. (We are done). $\Box$

## B.3 Complexity of Normal Programs

The computational complexity of brave reasoning for an $\vee$-free program $\mathcal{P} = (LP, S, W)$ strongly depends on the kind of negation that we allow in $LP$. Indeed, if the logic program $LP$ is stratified (or even positive), then it admits a unique stable model that can be computed in polynomial time. Therefore, in all $\vee$-free fragments of DATALOG$^{\vee,\neg,c}$ that allow only stratified negation, brave reasoning is always polynomial and its complexity is not affected at all from the presence of (strong or weak) constraints.

If unstratified negation is allowed in $LP$, and then multiple stable models are possible, the complexity does increase and is affected from the presence of weak constraints. We claim that (under brave reasoning) each $\vee$-free fragment of DATALOG$^{\vee,\neg,c}$ which allows unstratified negation has exactly the same complexity of the corresponding $\neg$-free fragment which allows HCF disjunction. In proof, we make the following remarks:

- Since stable model checking is polynomial for both ($\neg$-free) HCF disjunctive programs and ($\vee$-free) unstratified programs, it is easy to see that all upper complexity bounds that we determined for the former programs, apply to the latters as well (actually, the same proofs can be applied).

- The lower complexity bounds derived for ($\neg$-free) HCF disjunctive programs hold also for ($\vee$-free) unstratified programs, because every HCF program $LP$ can be rewritten into an unstratified $\vee$-free program having exactly the same stable models as $LP$ (by using this rewriting, we can again utilize the same proofs we used for HCF programs to derive the hardness results for normal unstratified programs).