Adding Weak Constraints to Disjunctive Datalog *

Francesco Buccafurri
DIEMA
Univ. di Reggio Calabria
89100 Reggio Calabria, Italy
bucca@si.deis.unical.it

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

Pasquale Rullo
DIEMA
Univ. di Reggio Calabria
89100 Reggio Calabria, Italy
rullo@si.deis.unical.it

Abstract

This paper presents an extension of disjunctive datalog (Datalog$^{\vee,\neg}$) by integrity constraints. In particular, besides classical integrity constraints (called *strong constraints* in this paper), the notion of *weak constraints* is introduced in the language. These are constraints that are satisfied *if possible*. The semantics of weak constraints tends to minimize the number of violated instances. As a consequence, weak constraints differ from strong constraints *only if* the latter are unsatisfiable. Weak constraints may be ordered according to their importance to express different priority levels. The formal definition of the semantics of weak constraints is given in a general way that allows to put them on top of *any* existing (model-theoretic) semantics for Datalog$^{\vee,\neg}$ programs.

A number of examples shows that the proposed language (call it Datalog$^{\vee,\neg,c}$) is well-suited to represent complex knowledge-based problems, such as, for instance, NP optimization problems.

A detailed complexity analysis of the language is also given.

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

1 Introduction

Disjunctive Datalog programs (or Datalog$^{\vee,\neg}$ programs) [9] are nowadays widely recognized as a valuable tool for knowledge representation and commonsense reasoning [1, 18, 14, 11]. An important merit of Datalog$^{\vee,\neg}$ programs over normal (that is, disjunction-free) Datalog programs is their capability to model incomplete knowledge [1, 18].

The presence of disjunction in the rules' heads makes disjunctive programming inherently non-monotonic, that is, new information can invalidate previous conclusions. Much research work has been done on the semantics of disjunctive programs and several alternative proposals have been formulated [3, 11, 19, 22, 23, 24, 25]. One which is widely accepted is the extension to the disjunctive case of the stable model semantics of Gelfond-Lifschitz. According to this semantics [11, 22], a disjunctive program may have

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

†Please address correspondence to this author.

several alternative models (possibly none), each corresponding to a possible view of the reality.

In [6] it is proved that (under stable model semantics) disjunctive programs capture the complexity class Σ_2^P (that is, they allow us to express *every* property which is decidable in non-deterministic polynomial time with an oracle in NP). In [6], Eiter et al. show that the expressiveness of disjunctive Datalog has practical relevance, as concrete real world situations can be represented by disjunctive programs, while they cannot be expressed by (disjunction-free) Datalog programs.

In this paper, we propose an extension of Datalog$^{\vee,\neg}$ by constraints. In particular, besides classical integrity constraints (that we call "strong" constraints), we introduce the notion of *weak constraints*, that is, constraints that *should possibly be satisfied*. Informal readings of a weak constraint, say, $\Leftarrow B$, where $\Leftarrow$ is the weak implication symbol and B a conjunction of literals, are "try to falsify B" or "B is preferably false", etc. Weak constraints differs from strong ones *only if* the latter are unsatisfiable. 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 problems (including, e.g., NP optimization problems) in a very natural and compact way.

The main contributions of the paper are the following:

- We add weak constraints to Datalog$^{\vee,\neg}$ and show how they can be profitably used for knowledge representation and commonsense reasoning. The semantics of weak constraints is given in a general way that allows to define them on top of any existing model-theoretic semantics for Disjunctive Datalog.

- We analyze the complexity of Datalog$^{\vee,\neg,c}$. 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 Δ_3^P-complete (resp., Δ_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 query language, which fits the needs in practice (see Section 5).

Even if strong constraints (traditionally called integrity constraints) are well-known in the logic programming community (see, e.g., [5, 17, 7]), to our knowledge this is the first paper which formally defines weak constraints and proposes their use for knowledge representation and reasoning. Related works can be considered the interesting papers of Greco and Saccà [13, 12] which analyze other extensions of Datalog to express NP optimizations problems.

Observe that we could have presented weak constraints on plain Datalog with negation (Datalog$^{\neg}$), instead that on Disjunctive Datalog. The reason why we decided to define weak constraints on top on Datalog$^{\vee,\neg}$ are the following: (a) Datalog$^{\vee,\neg}$ subsumes Datalog with negation and, thus, giving the definition in the general case we have implicitly defined weak constraints also for Datalog$^{\neg}$; (b) Datalog$^{\vee,\neg}$ provides a higher expressive power [6] and, more importantly, allow us to model several problems in a more elegant and natural way than Datalog$^{\neg}$ does (e.g., the Datalog$^{\vee,\neg}$ programs reported in Section

2 can be equivalently expressed in Datalog$^{\neg}$; however, their representation in Datalog$^{\neg}$ is less compact and less intuitive, as it requires the use of unstratified negation).

For space limitation the proofs of the complexity results are omitted. Detailed proofs of all results can be found in [4].

2 Motivating Examples

In this section we give some examples to show the suitability of Datalog$^{\vee,\neg,c}$ in representing knowledge. As will be clear in Section 3, for each problem Π we describe below, the semantics of the associated Datalog$^{\vee,\neg,c}$ program precisely captures the set of solutions to Π. A number of further examples, where the language is used to model NP optimization problems, can be found in [4].

Constraints enhance disjunctive Datalog (Datalog$^{\vee,\neg}$) with the capability of naturally expressing complex knowledge-based problems. As an example, consider the problem of scheduling 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 "incompatible"). Assuming that there are three time slots available, namely, s_1, s_2 and s_3, we express the problem in Datalog$^{\vee,\neg}$with strong constraints as follows:

$$\begin{array}{ll} r_1: & assign(X, s1) \vee assign(X, s2) \vee assign(X, s3) \leftarrow course(X) \\ s_1: & \leftarrow assign(X, S), assign(Y, S), incompatible(X, Y) \end{array}$$

Fig. 1

Here, rule r_1 says that every course is assigned with either one of the three time slots; the strong constraint s_1 (a rule with empty head) expresses that no two incompatible courses can be assigned with 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 the problem above, there could be no way to assign courses to time slots without having some overlapping between incompatible courses. That is, 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: assign courses to time slots trying to minimize the overlapping of incompatible courses. To solve this problem we resort to the notion of *weak* constraint, as shown below:

$$\begin{array}{ll} r_2: & assign(X, s1) \vee assign(X, s2) \vee assign(X, s3) \leftarrow course(X) \\ w_2: & \Leftarrow assign(X, S), assign(Y, S), incompatible(X, Y) \end{array}$$

Fig. 2

From a syntactical point of view, a weak constraint is like a strong constraint where $\leftarrow$ is replaced by $\Leftarrow$. The semantics of weak constraints tends to minimize the occurrences of violated constraints. For instance, an informal reading of the above weak constraint is: "preferably, do not assign the courses X and Y to the same slot S if they are incompatible". Note that the programs of Figure 1 and Figure 2 have the same models if all incompatible courses can be assigned to different time slots (i.e., if the problem admits an "exact" solution). We call Datalog$^{\vee,\neg,c}$ the language obtained from Datalog$^{\vee,\neg}$ by adding

strong and weak constraints.
An interesting use of weak constraint is in modeling optimization problems. Consider, for an instance, the problem of finding in a graph $G =< V, E >$, where V is the set of vertices and E is the set of edges, a clique of maximum cardinality (recall that a clique of G is a subset V' of V such that every two nodes in V' are joined by an adge in E). Using Datalog$^{\vee,\neg,c}$, this problem can be expressed as follows:

$$
\begin{array}{ll}
r_3: & v'(X) \vee \bar{v}'(X) \leftarrow node(X) \\
s_3: & \leftarrow v'(X), v'(Y), \neg edge(X,Y) \\
w_3: & \Leftarrow node(X), \neg v'(X)
\end{array}
$$

Fig. 3

Intuitively, rule r_3 partitions the set of nodes into two subsets, v' and its complement $\bar{v}'$. The strong constraint s_3 imposes that each pair of nodes in v' (the clique) is joined by an edge. Finally, the weak constraint w_3 requires that a node X is *possibly* an element of the clique v'.
As another example, consider the coloring of a planar graph $G =< V, E >$ [1]. We want to color the nodes of G with the minimum number of colors in such a way that every pairs of joined (by edges) nodes have different colors:

$$
\begin{array}{ll}
r_4: & col(X,c1) \vee col(X,c2) \vee col(X,c3) \vee col(X,c4) \leftarrow node(X) \\
s_4: & \leftarrow col(X,C), col(Y,C), edge(X,Y) \\
w_4: & \Leftarrow col(X,C1), col(Y,C2), \neg edge(X,Y), C1 \neq C2
\end{array}
$$

Fig. 4

Rule r_4 above assigns one out of four colors to every node; the strong constraint s_4 disallows two joined nodes to have the same color; the weak constraint w_4 requires two unjoined nodes to have the same color, if possible.

As weak constraints express preferences, and 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 "importance". [2] For example, consider the problem of organizing a given set of employees in two (disjoint) project groups. We wish each group to be possibly heterogeneous as far as skills are concerned. Further, it is preferable that no persons in the same group are married each other. And, finally, we would like that the components of a group already know each other. We consider the former requirement more important than the latter two.

$$
\begin{array}{l}
r_5: assign(X,p1) \vee assign(X,p2) \leftarrow employee(X) \\
w_5: \Leftarrow assign(X,P), assign(Y,P), same\_skill(X,Y) \\
w_6: \Leftarrow assign(X,P), assign(Y,P), married(X,Y) \\
w_7: \Leftarrow assign(X,P), assign(Y,P), X \neq Y, \neg know(X,Y)
\end{array}
$$

Fig. 5

where w_5 is "stronger than" both w_6 and w_7. The models of the above program are the assignments of employees to projects $p1$, $p2$ which minimize the number of violated instances of w_5 and, among these, those in which is minimum the number of violated instances of w_6 and w_7.

[1] A graph is planar if it can be embedded in the plane by identifying each vertex with a unique point and each edge with a line connecting its endpoints, so that no two lines meet except at a common endpoint. Recall that planar graphs are 4-colorable [10]

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

3 The Datalog$^{\vee,\neg,c}$ Language

A *term* is either a constant or a variable[3]. 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 set of rules (also called *disjunctive Datalog* program); 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.

We define a *Datalog$^{\vee,\neg,c}$program* (often simply "program") $\mathcal{P} = \{LP, S, W\}$, where LP is a Datalog$^{\vee,\neg}$ program, S a (possibly empty) set of strong constraints and $W = \langle W_1, ..., W_n \rangle$, is a (possibly empty) set of *components*, each consisting of a set of weak constraints, totally ordered by the binary relation $<$. [4] Informally, we say that if $w \in W_i$, $w' \in W_j$ and $W_j < W_i$ than w is "stronger than" or "more important than" w'.

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, and (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.

We next define the semantics of Datalog$^{\vee,\neg,c}$ in a general way which does not rely to a specific semantical proposal of Datalog$^{\vee,\neg}$; but, rather, can be applied to every semantics of disjunctive Datalog. [5] 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 set of candidate models for $\mathcal{P}$. Clearly, every classical semantics (i.e., set of models) for LP provides a ground semantics for $\mathcal{P}$.

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. In particular,

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

[4] For notational simplicity we will assume that $<$ follows the orderin which the componentsare listed (e.g., the first component is the smallest).

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

since weak constraints may have different priorities, we associate with each component $W_i \in W$, $1 \leq i \leq n$, a positive weight $f(W_i)$ defined as follows: $f(W_1) = 1$, $f(W_i) = f(W_{i-1}) \cdot |ground(W)| + 1$, $1 < i \leq n$, where $|X|$ denotes the cardinality of set X. Given an interpretation M for a program $\mathcal{P}$, we denote by H_M the sum $f(W_1) \cdot N_1^M + \ldots + f(W_n) \cdot N_n^M$, where N_i^M $(1 \leq i \leq n)$ is the number of the ground instances of the weak constraints in the component W_i not satisfied in M.

Definition 3.1 Given a ground semantics Γ for $\mathcal{P} = \{LP, S, W\}$, a Γ-*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_M is minimum, that is, there is no candidate model $N \in \Gamma$ verifying Point (1) such that $H_N < H_M$.

The above definition, minimizing H_M, selects the 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.

Using the notion of candidate model we have parameterized 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 [3, 11, 19, 22, 23, 24, 25]. One which is generally acknowledged is the extension of the stable model semantics to take into account disjunction [11, 22]. We next report a brief discussion on this semantics.

In [19], Minker proposed a model-theoretic semantics for positive (disjunctive) programs, whereby a positive program LP is assigned with a set 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). Observe that every positive program admits at least one minimal model.

Example 3.2 For the positive program $LP = \{a \vee b \leftarrow\}$ the (total) interpretations $\{a\}$ and $\{b\}$ are its minimal models (i.e., MM(LP) = { $\{a\}$, $\{b\}$ }).

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:

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

Now, let I be an interpretation for a program LP. I is a *stable model* for LP if $I \in$ 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 $STM(LP)$.

Example 3.3 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. □

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 3.1 provides the stable model semantics of a Datalog$^{\vee,\neg,c}$ program $\mathcal{P} = \{LP, S, W\}$ if the choosen ground semantics is $STM(LP)$, the set of the stable models of LP.

Definition 3.4 A Γ-*model* for a program $\mathcal{P} = \{LP, S, W\}$ is a *stable model for* $\mathcal{P}$ if $\Gamma = STM(LP)$. □

Example 3.5 Consider the program $\mathcal{P}_1 = \{LP_1, S_1, W_1\}$ of Fig. 1, where $LP_1 = \{r_1\}$, $S_1 = \{s_1\}$ and $W_1 = \emptyset$. According to the stable model semantics, LP_1 has as many stable models as the possibilities of assigning n courses to 3 time slots (namely, 3^n). The stable models of $\mathcal{P}_1$ are the stable models of LP_1 satisfying the strong constraint s_1, that is, those for which no two incompatible courses are assigned with the same time slot.

The program $\mathcal{P}_2 = \{LP_2, S_2, W_2\}$ of Fig. 2, where $LP_2 = \{r_2\}$, $S_2 = \emptyset$ and $W_2 = \langle\{w_2\}\rangle$, is obtained from LP_1 simply by replacing s_1 by w_2 (note that $LP_2 = LP_1$). The stable models of $\mathcal{P}_2$ are the stable models of LP_2 which minimize the number of violated instances of w_2, that is, the number of incompatible courses assigned to the same time slots. Thus, $\mathcal{P}_2$ provides a different solution from $\mathcal{P}_1$ only if 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.

Consider now program $\mathcal{P}_3 = \{LP_3, S_3, W_3\}$ of Fig. 3, where $LP_3 = \{r_3\}$, $S_3 = \emptyset$ and $W_3 = \langle\{w_3\}\rangle$. Each stable model of LP_3 is a possible 2-partitioning of the set of nodes of the given graph. The stable models of $\mathcal{P}_3$ are those of LP_3 which (1) satisfy the strong constraint s_3 (i.e., no two nodes in the clique v' are not joined) and (2) minimize the number of nodes that are not in v' (thus, maximizing the cardinality of v').

Finally, consider program $\mathcal{P}_5 = \{LP_5, S_5, W_5\}$ of Fig. 5, where $LP_5 = \{r_5\}$, $S_5 = \emptyset$ and $W_5 = \langle\{w_6, w_7\}, \{w_5\}\rangle$ (read "w_5 is stronger than both w_6 and w_7"). Each stable model of LP_5 is a possible assignment of employees to projects. The stable models of $\mathcal{P}_5$ are the stable models of LP_5 that, first of all, minimize the number of violated instance of w_5 and, in second order, minimize the overall number of violated instances of w_6 and w_7. □

4 The Complexity of Constraints: Propositional Case

In this section we analyze the complexity of brave reasoning (i.e., deciding whether a literal is true in some stable model) 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 [11, 22], which is a widely acknowledged semantics for normal and disjunctive Datalog programs.

4.1 The Polynomial Hierarchy

For NP-completeness and complexity theory, cf. [20]. The classes Σ_k^P, Π_k^P and Δ_k^P of the *Polynomial Hierarchy (PH)* (cf. [26]) are defined as follows:

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

	{}	{s}	{w}	{s, w}	{w$^<$}	{s, w$^<$}
{}	P	P	P	P	P	P
{$\neg^s$}	P	P	P	P	P	P
{$\neg$}	NP	NP	$\Delta_2^P[O(\log n)]$	$\Delta_2^P[O(\log n)]$	Δ_2^P	Δ_2^P
{$\vee^h$}	NP	NP	$\Delta_2^P[O(\log n)]$	$\Delta_2^P[O(\log n)]$	Δ_2^P	Δ_2^P
{$\vee^h, \neg^s$}	NP	NP	$\Delta_2^P[O(\log n)]$	$\Delta_2^P[O(\log n)]$	Δ_2^P	Δ_2^P
{$\vee^h, \neg$}	NP	NP	$\Delta_2^P[O(\log n)]$	$\Delta_2^P[O(\log n)]$	Δ_2^P	Δ_2^P
{$\vee$}	Σ_2^P	Σ_2^P	$\Delta_3^P[O(\log n)]$	$\Delta_3^P[O(\log n)]$	Δ_3^P	Δ_3^P
{$\vee, \neg^s$}	Σ_2^P	Σ_2^P	$\Delta_3^P[O(\log n)]$	$\Delta_3^P[O(\log n)]$	Δ_3^P	Δ_3^P
{$\vee, \neg$}	Σ_2^P	Σ_2^P	$\Delta_3^P[O(\log n)]$	$\Delta_3^P[O(\log n)]$	Δ_3^P	Δ_3^P

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

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

The classes Δ_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.

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 $Q_1 = \exists$ the 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 Φ (resp. a k-universal QBF Ψ), deciding whether $\Phi \in \mathrm{QBF}_{k,\exists}$ (resp. $\Psi \in \mathrm{QBF}_{k,\forall}$), is a classical Σ_k^P-complete (resp. Π_k^P-complete) problem.

Δ_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 $Q_1 Y_1, ..., Q_r Y_r$, deciding whether the with respect to $\langle X_1, ..., X_n \rangle$ lexicographically minimum truth assignment [6] ϕ to $X_1, \ldots, X_n$ such that $Q_1 Y_1 \cdots Q_r Y_r E_\phi \in \mathrm{QBF}_{k-2,\forall}$ (where such a ϕ is known to exist)

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

fulfills $\phi(X_n) = true$ (cf. [30]).[7] $\Delta^P_{k+1}[O(\log n)]$ 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 ([30, 8]).

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 $Q_k = \forall$ [27].

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* [29]. For space limitation the proofs of the complexity results are omitted. Detailed proofs of all results can be found in [4]. The list of all complexity results is reported in Table 1 and discussed in Section 5.

Given a program $\mathcal{P} = \{LP, S, W\}$, we denote by $maxH(\mathcal{P})$ the sum of the weights $w(c)$ of all weak constraints c occurring in W (note that $maxH(\mathcal{P})$ is polynomial time computable).

Theorem 4.1 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 Δ^P_3.

Theorem 4.2 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 Δ^P_3-complete. Hardness holds even if LP is a positive disjunctive Datalog program and $S = \emptyset$.

Theorem 4.3 Given a Datalog$^{\vee,\neg,c}$ program $\mathcal{P} = \{LP, S, W\}$, where W consists of one only component, and a literal q as input, deciding whether q is true in some stable model of $\mathcal{P}$ is $\Delta^P_3[O(\log n)]$-complete. Hardness holds even if LP is positive and $S = \emptyset$.

5 Conclusion

We have presented Datalog$^{\vee,\neg,c}$, a language which extends Datalog$^{\vee,\neg}$ by strong and weak constraints. While strong constraints are imperative, weak constraints allow to express desiderata. We have shown the suitability of the language to naturally express complex knowledge-based problems by a number of examples. A definition of semantics and a detailed complex analysis of Datalog$^{\vee,\neg,c}$ programs has been provided.

The results on the complexity of brave reasoning with stable model semantics over propositional Datalog$^{\vee,\neg,c}$ programs are compactly represented in Table 1. [8] There, each entry of a complexity class C symbolizes C-completeness. Each column in Table 1 refers to a specific form of constraint: $\{\}$ = 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

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

[8] Note that some results can be obtained by using known translations of a language into another, we report all results in the table to give a complete picture of the complexity of the fragments of Datalog$^{\vee,\neg,c}$.

particular, $\neg^s$ stands for stratified negation [21] and $\vee^h$ stands for head cycle free (hcf) disjunction [2]. [9]

It appears that, while strong constraints do not affect the complexity of the language, weak constraints "mildly" increase the computational complexity. Indeed, brave reasoning is Δ_3^P-complete for full Datalog$^{\vee,\neg,c}$ programs (while it is Σ_2^P-complete in absence of weak constraints). Interestingly, priorities among constraints affect the complexity, which decreases to $\Delta_3^P[O(\log n)]$ if priorities are disallowed. As expected, disallowing negation does not change the complexity of the language. On the contrary, the complexity of the $\vee$-free fragment of Datalog$^{\vee,\neg,c}$ lies one level down in the polynomial hierarchy, as brave reasoning is Δ_2^P-complete (resp., $\Delta_2^P[O(\log n)]$) for $\{w^<, s\}$ (resp., $\{w, s\}$) on $\vee$-free programs. Finally observe that if we limit disjunction to the head cycle free case, then we obtain exactly the same complexities than for disjunction free programs with negation, as the former programs can be translated in the latter ones. In our opinion the fragment of Datalog$^{\vee,\neg,c}$ with hcf disjunction and stratified negation ($\{\vee^h, \neg^s\}$) is very interesting: it has a very clear and easy to understand semantics (while the semantics of unstratified or non-hcf programs is intriguing), and, at the same time, allow us to express several hard problems (up to Δ_2^P-complete problems) in a natural and compact fashion (note that all sample programs reported in the paper fall in this fragment).

Ongoing work follows two main research lines. On the one hand, we are studying possible linguistic extensions and refinements of Datalog$^{\vee,\neg,c}$ (concerning constraints in particular). On the other hand, we are developing the implementation of a deductive system supporting Datalog$^{\vee,\neg,c}$. The latter implementation work is carried out at the Technical University of Vienna within *FWF Project P11580-MAT: "A Query System for Disjunctive Deductive Databases,"*.

References

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

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

[3] Brass, S. and 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.

[4] Buccafurri, F., Leone, N., Rullo, P., (1996), Strong and Weak Contraints in Disjunctive Datalog, *Technical Report ISI CNR. 96-14*.

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

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

[9] Recall that stratification forbids negation through recursion, and head cycle freeness forbids recursion in disjunction.

[7] Eiter, T. and 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.

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

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

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

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

[12] 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.

[13] Greco, S. and Saccà, D. (1997), NP Optimization Problems in Datalog, unpublished manuscript.

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

[15] 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.

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

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

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

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

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

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

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

[31] 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.