Project Number:	IST-2001-33570
Project Acronym:	INFOMIX
Project Full Name:	Infomix: Boosting the Information Integration
Deliverable Number:	D4.1
Title:	**Internal integration formalism.**
Authors:	G.Ianni, N. Leone, P. Rullo, G. Terracina
Workpackage:	WP4
Document Type:	Deliverable
Distribution:	Infomix Consortium
Status:	Final
Document file:	WP4_T1_D4.1_0102.ps
Version:	1.2
Number of pages:	31
Due Date:	June 30, 2003
Delivery Date:	July 11, 2003
Partners contributed:	UNICAL, TUWIEN, UNIROME1, RODAN
Partners owning:	UNICAL, TUWIEN, UNIROME1, RODAN
Man Months:	6

Short Description:

The report provides the declarative definition of the internal integration formalism. In particular, we evidence that it was already implicit in the proposal that the internal integration formalism is based on Disjunctive Logic Programming (DLP).
Indeed, the rewriting performed by the Query Reformulator module produces DLP programs, and we planned to employ a DLP system for the integration. However, thinking on possible applications, we realized that standard DLP is not sufficient, as real life queries often require to "aggregate data", which is not possible in standard DLP. Thus, we extended DLP by aggregate functions in order to support some form of aggregate queries in the language.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	June 30, 2003	First draft
v.1.1	July 7, 2003	Review from the project parterns
v.1.2	July 8, 2003	Final version

Contents

1 Introduction

In this report we provide the declarative definition of the Internal Integration Formalism (IIF) which will be adopted in the INFOMIX system. Since our main goal is to apply computational logic systems in a data integration framework, this formalism will constitute the glue between the data integration model, the user query language, and the computational engine. It is clear that the choice of a formalism fitting all the requirements for the project is a crucial aspect for the inter-operability of the various components of the system.

Recently, the application of computational logics to the data integration framework received great attention. Such an interest is witnessed by a growing body of proposals aiming at formalizing different repairs semantics for inconsistent databases, eventually obtained by wrapping different data sources [3, 28, 6, 11, 9]. An attractive feature of these approaches, which is also one of the main concern of the INFOMIX project, is that logic programs serve as executable logical specifications that can be processed with current implementations of stable models engines such as DLV [30] or Smodels [38].

Both the project proposal and the specifications given in the previous reports made it clear that the internal integration formalism would have been based on a suitable extension of logic programming, which is Disjunctive Logic Programming (DLP). Indeed, the specifications given in deliverable D2.2 for the run-time system architecture makes it explicit that the Query Reformulator module produces disjunctive datalog programs and that the query evaluator module exploits a disjunctive datalog executor in query evaluation and integration processes. However, thinking to real application settings in data integration, we realized that the expressiveness of standard DLP is not sufficient to express queries requiring to “aggregate data”, which often occur in applications. This consideration led us to extend standard DLP by aggregate functions in order to support some forms of aggregate queries in the user language.

The rest of the present document is structured as follows. In the next section, we point out the requirements of the Internal Integration Formalism and the limitations of DLP in expressing complex user queries that involve aggregate operators; these considerations allow us to introduce an extension of DLP which is able to model and reason on aggregate data, namely DLP$^{\mathcal{A}}$. In Section 3 we briefly recall the syntax and the semantics of the core language of DLP, and we identify some significant syntactic fragments that can be handled more efficiently. In Section 4 we formally describe DLP$^{\mathcal{A}}$; whereas, in Section 5 we overview the knowledge representation and problem-solving techniques in DLP$^{\mathcal{A}}$. Section 6 is devoted to the complexity analysis of both the DLP and the DLP$^{\mathcal{A}}$, Section 7 takes into account related works and, finally, in Section 8 we draw our conclusions.

2 Requirements for the Internal Integration Formalism

The need of representing and manipulating complex knowledge arising in Artificial Intelligence and in other emerging areas, like Knowledge Management and Information

Integration, has renewed the interest in advanced logic-based formalisms for Knowledge Representation and Reasoning (KR&R), which started in the early 1980s. Thus, the choice of the IIF requires a careful look into such growing body of proposals, in order to identify the formalism that best fits the requirements of the project. Specifically, the internal formalism should provide a compromise between two distinct needs:

- Under a knowledge representation perspective, an "ideal" IIF should be able to easily capture all the features of the Information Integration Model, that have been already specified in the deliverable D3.1. For instance, it should be able to express integrity constraints on the data to be handled as well as to specify complex mappings between the global schema and the sources.

- Under an operational, and more pragmatic, point of view it should be a language that can be easily translated in the core language of the DLV system, which constitutes the main part of the run-time architecture.

Let us start by considering the former issue. The formalism for the Information Integration Model (IIM), specified in the deliverable D3.1, defines *(i)* a model for the description of the global schema and of the constraints over such a schema, *(ii)* a model for the data sources and *(iii)* the language in which the mappings between global and source schemas can be expressed. By analyzing the IIM, we can point out the following requirements:

1. The IIF should be able to represent tuples stored in a relational database (also called *facts*). Indeed, even though INFOMIX leaves a certain degree of freedom in the data formats that can be handled at the sources, it requires the internal view of the global schema to be formalized in the relational model.

2. The IIF should be able to express first order sentences over such tuples. In fact, as stated in the deliverable D3.1, the INFOMIX system can deal with most of the common forms of integrity constraints for the relational model, i.e., functional dependencies, key dependencies, inclusion dependencies, exclusion dependencies and foreign keys. Actually, we point out that this does not mean that all these constraints can be simultaneously enforced on the global schema: D3.1 describes how these constraints can be combined (together with the the query language) in order to maintain the decidability of query answering in data integration. Nonetheless, the whole set of these constraints can be easily modelled with first order sentences.

3. The expressiveness of the IIF should be at least that of DATALOG. In fact, as for the mapping between global schema and sources the project has already specified (D3.1) some requirements. Specifically, the **GAV** mapping can be expressed by means of DATALOG queries; indeed, the choice of the language has been proved to be not crucial for the query answering decidability. Conversely, the **LAV** mapping has been restricted in a fragment of DATALOG, namely the union of conjunctive queries (UCQs), because a more liberal assumption may easily lead to undecidability problems in query answering.

Moreover, we have also to consider the impact of the above mentioned features on the intrinsic difficulty of query answering in a data integration system. Under this perspective, the IIF, which is also used to generate the executable specifications, should be able to model problems at least "difficult" as the query answering in the INFOMIX setting. The following requirement exploits many results appeared recently in the literature:

4. The IIF should be able to model at least co-NP-complete problems. In fact, recent studies (see, e.g., [3, 28, 6, 11, 9]) showed that the complexity of query answering in data integration (even though in different settings) is often complete for the class co-NP, or even for the class Π_2^P.

The second strong requirement of the IIF is that the algorithms for obtaining the executable specifications should not constitute a significant overhead in the process of query answering. Specifically:

5. The IIF should be well suited for an implementation over the DLV system. This requirement prevents complex pre-processing or post-processing of the specifications, and enables DLV to be a self contained engine for the INFOMIX applications.

The above considerations allow us to conclude that the language for the IIF should be able to express at least all the DATALOG queries and that it should be easily implementable in the core language of DLV.

Finally, the last critical requirement regards the ability of expressing user queries:

6. The IIF should be able to easily express aggregation operations on source data. Even if the query language has not been defined yet, it is reasonable to assume that users should be able to express most of the SQL2 queries. Specifically, one of the most relevant features of SQL2 to be handled is the capability to "aggregate" data.

By analyzing points 1—5, it is clear that if we restrict our attention to the current proposals only, DLP seems to be the most suited IIF in the context of the INFOMIX project. In fact, it is more expressive than DATALOG, it can be used to easily model co-NP-complete problems, and it constitutes the core language of the DLV system. Nonetheless, the last requirement (Point 6) is not satisfied by DLP, which is in fact not well suited for modelling aggregate data. This latter point is addressed in more details in the following subsection.

2.1 The need of aggregates

In spite of its high expressiveness, classical DLP presents some limitations for its applicability to real data integration settings. In data integration contexts, it is reasonable to assume that users should be able to express most of the SQL2 queries. Indeed, SQL2 is widely accepted as the standard query language in the database context. However, DLP is not comparable with SQL2 in that some queries that can be expressed in DLP cannot be expressed in SQL2 and vice versa. For instance, DLP provides the power of recursion and disjunction which cannot be simulated in SQL2, and SQL2 allows aggregate operators (such as SUM, MIN, MAX) and ordering features which cannot be expressed or easily simulated in classical DLP. In some cases, aggregate operators can be simulated in DLP, but this produces inefficient programs and unnatural encodings of the problems.

As an example, suppose that a user wants to know if the sum of the salaries of the employees working in a team exceeds a given budget (see the **Team Building** problem, in Section 5). To this end, the user should first order the employees defining a successor relation. Then she should define a *sum* predicate, in a recursive way, which computes the sum of all salaries, and compare its result with the given budget. This approach has two drawbacks: (1) It is bad from the KR perspective, as the encoding is not natural at all; (2) It is inefficient, as the (instantiation of the) program is quadratic (in the cardinality of the input set of employees).

Unfortunately, the need of expressing aggregation is quite common in data integration, since user usually want to build summary views of data residing in different databases. Thus, there is a clear need to enrich DLP with suitable constructs for the natural representation of aggregate operators and to provide means for an efficient evaluation of them.

In INFOMIX, we overcome this problem by exploiting an extension of DLP, called $\text{DLP}^{\mathcal{A}}$, allowing to handle aggregate functions, first studied in the context of deductive databases. Thus, $\text{DLP}^{\mathcal{A}}$ will constitute the Internal Integration Formalism in the INFOMIX project. ct.

3 The DLP language

Disjunctive logic programs are logic programs where disjunction is allowed in the heads of the rules and negation may occur in the bodies of the rules. Such programs are now widely recognized as a valuable tool for knowledge representation and commonsense reasoning [4, 32, 48, 18, 25, 31, 37, 5]. One of the attractions of disjunctive logic programming (DLP) is its capability of allowing the natural modeling of incomplete knowledge [4, 32]. Much research has been spent on the semantics of disjunctive logic programs, and several alternative semantics have been proposed [8, 22, 25, 36, 41, 43, 42, 45, 46] (see [1, 16, 32, 37, 35] for comprehensive surveys). The most widely accepted semantics is the *answer sets semantics* proposed by Gelfond and Lifschitz [25] as an extension of the stable model semantics of normal logic programs [24]. According to this semantics, a disjunctive logic program may have several alternative models (but possibly none), called *answer sets*, each corresponding to a possible view of the world.

Disjunctive logic programs under answer sets semantics are very expressive. It was shown in [19, 26] that, under this semantics, disjunctive logic programs capture the complexity class Σ_2^P (i.e., they allow us to express, in a precise mathematical sense, *every* property of finite structures over a function-free first-order structure that is decidable in nondeterministic polynomial time with an oracle in NP). As Eiter *et al.* [19] showed, the expressiveness of disjunctive logic programming has practical implications, since relevant practical problems can be represented by disjunctive logic programs, while they cannot be expressed by logic programs without disjunctions, given current complexity beliefs. The high expressiveness of disjunctive logic programming comes at the price of a higher computational cost in the worst case: The typical reasoning tasks, brave reasoning and cautious reasoning, on disjunctive logic programs are complete problems for the complexity class Σ_2^P and Π_2^P, respectively [19, 17], while non-disjunctive logic programs are complete for NP and co-NP, respectively (cf. [34, 13]).

In the following we briefly recall the syntax of disjunctive datalog.

3.1 Syntax and Semantics

A *disjunctive rule* (*rule*, for short) r is a formula

$$a_1 \texttt{ v } \cdots \texttt{ v } a_n \texttt{ :- } b_1, \cdots, b_k, \text{ not } b_{k+1}, \cdots, \text{ not } b_m. \tag{1}$$

where $a_1, \cdots, a_n, b_1, \cdots, b_m$ are classical literals and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \texttt{ v } \cdots \texttt{ v } a_n$ is the *head* of r, while the conjunction $b_1, ..., b_k$, not $b_{k+1}, ...,$ not b_m is the *body* of r. A rule without head literals (i.e. $n = 0$) is usually referred to as an *integrity constraint*. A rule having precisely one head literal (i.e. $n = 1$) is called a *normal rule*. If the body is empty (i.e. $k = m = 0$), it is called a fact, and we usually omit the " :- " sign.

The following notation will be useful for further discussion. If r is a rule of form (1), then $H(r) = \{a_1, \ldots, a_n\}$ is the set of the literals in the head and $B(r) = B^+(r) \cup B^-(r)$ is the set of the body literals, where $B^+(r)$ (the *positive body*) is $\{b_1, \ldots, b_k\}$ and $B^-(r)$ (*the negative body*) is $\{b_{k+1}, \ldots, b_m\}$.

A *disjunctive datalog (*DLP*) program* (alternatively, *disjunctive deductive database*) $\mathcal{P}$ is a finite set of rules. A not-free program $\mathcal{P}$ (i.e., such that $\forall r \in \mathcal{P} : B^-(r) = \emptyset$) is called *positive*,[1] and a `v`-free program $\mathcal{P}$ (i.e., such that $\forall r \in \mathcal{P} : |H(r)| \leq 1$) is called *datalog program* (or *deductive database*).

For any program $\mathcal{P}$, let $U_\mathcal{P}$ (the Herbrand Universe) be the set of all constants appearing in $\mathcal{P}$. In case no constant appears in $\mathcal{P}$, an arbitrary constant ψ is added to $U_\mathcal{P}$. For any program $\mathcal{P}$, let $B_\mathcal{P}$ be the set of all ground (classical) literals that can be constructed from the predicate symbols appearing in $\mathcal{P}$ and the constants of $U_\mathcal{P}$ (note that, for each atom p, $B_\mathcal{P}$ contains also the strongly negated literal $\neg p$).For any rule r, $Ground(r)$ denotes the set of rules obtained by applying all possible substitutions σ from the variables in r to elements of $U_\mathcal{P}$.

An interpretation I is a set of ground classical literals, i.e. $I \subseteq B_\mathcal{P}$ w.r.t. a program $\mathcal{P}$. A consistent interpretation $X \subseteq B_\mathcal{P}$ is called *closed under* $\mathcal{P}$ (where $\mathcal{P}$ is a positive disjunctive datalog program), if, for every $r \in Ground(\mathcal{P})$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation $X \subseteq B_\mathcal{P}$ is an *answer set* for a positive disjunctive datalog program $\mathcal{P}$, if it is minimal (under set inclusion) among all interpretations that are closed under $\mathcal{P}$.

The value of a ground atom l w.r.t. an interpretation I, $value_I(l)$, is *true* if $l \in I$ and $false$ otherwise. The value of a ground negated literal not l is not $value_I(l)$. The truth value of a conjunction of ground literals $C = l_1, \ldots, l_n$ is the minimum over the values of the l_i, i.e., $value_I(C) = min(\{value_I(l_i) \mid 1 \leq i \leq n\})$, while the value $value_I(D)$ of a disjunction $D = l_1 \texttt{ v } \cdots \texttt{ v } l_n$ is their maximum, i.e., $value_I(D) = max(\{value_I(l_i) \mid 1 \leq i \leq n\})$; if $n = 0$, then $value_I(C) = true$ and $value_I(D) = false$. A ground rule r is

[1] In positive programs negation as failure (not) does not occur, while strong negation ($\neg$) may be present.

satisfied by I if $value_I(H(r)) \geq value_I(B(r))$. Thus, a rule r with empty body is satisfied by I if $value_I(H(r)) = true$. In the following we also assume the existence of rules with empty head which define denials, i.e. rules which are satisfied only if the body is false ($value_I(B(r)) = false$). An interpretation M for $\mathcal{P}$ is a model of $\mathcal{P}$ if M satisfies all rules in $Ground(\mathcal{P})$.

The model-theoretic semantics for a positive program $\mathcal{P}$ assigns to $\mathcal{P}$ the set of its *minimal models* $\mathcal{MM}(\mathcal{P})$, where a model M for $\mathcal{P}$ is minimal, if no proper subset of M is a model for $\mathcal{P}$. The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation [25]. Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs.

Let $\mathcal{P}$ be a logic program and let I be an interpretation for $\mathcal{P}$, $\mathcal{P}^I$ denotes the ground positive program derived from $Ground(\mathcal{P})$ by: *(1)* removing all rules that contain a negative literal not a in the body and $a \in I$, *(2)* removing all negative literals from the remaining rules. An interpretation M is a (disjunctive) stable model of $\mathcal{P}$ if and only if $M \in \mathcal{MM}(\mathcal{P}^M)$.

For general $\mathcal{P}$, the stable model semantics assigns to $\mathcal{P}$ the set $\mathcal{SM}(\mathcal{P})$ of its *stable models*. It is well known that stable models are minimal models and that for negation-free programs minimal and stable model semantics coincide and that positive Datalog programs without constraints have a unique minimal model.

3.2 Relevant Fragments of the DLP Language

In this section, we introduce syntactic classes of DLV programs with a number of useful and interesting properties. First we need the following definition.

Definition 3.1 Functions $|| \; || : B_{\mathcal{P}} \to \{0, 1, \ldots\}$ from the ground (classical) literals of the Herbrand Literal Base $B_{\mathcal{P}}$ to finite ordinals are called *level mappings* of $\mathcal{P}$.

Level mappings give us a useful technique for describing various classes of programs.

Definition 3.2 A disjunctive datalog program $\mathcal{P}$ is called *(locally) stratified* [2], [44], if there is a level mapping $|| \; ||_s$ of $\mathcal{P}$ such that, for every rule r of $Ground(\mathcal{P})$,

1. For any $l \in B^+(r)$, and for any $l' \in H(r)$, $||l||_s \leq ||l'||_s$;
2. For any $l \in B^-(r)$, and for any $l' \in H(r)$, $||l||_s < ||l'||_s$.
3. For any $l, l' \in H(r)$, $||l||_s = ||l'||_s$.

Example 3.3 Consider the following two DLV programs.

$$\begin{array}{llll} \mathcal{P}_7: & p(a) \lor p(c) \text{ :- not } q(a). & \qquad \mathcal{P}_8: & p(a) \lor p(c) \text{ :- not } q(b). \\ & p(b) \text{ :- not } q(b). & & q(b) \text{ :- not } p(a). \end{array}$$

It is easy to see that program $\mathcal{P}_7$ is stratified, while program $\mathcal{P}_8$ is not. A suitable level mapping for $\mathcal{P}_7$ is the following:

$$||p(a)||_s = 2 \quad ||p(b)||_s = 2 \quad ||p(c)||_s = 2 \quad ||q(a)||_s = 1 \quad ||q(b)||_s = 1 \quad ||q(c)||_s = 1$$

As for $\mathcal{P}_8$, an admissible level mapping would need to satisfy $||p(a)||_s < ||q(b)||_s$ and $||q(b)||_s < ||p(a)||_s$, which is impossible. □

Another interesting class of problems consists of head-cycle free programs.

Definition 3.4 A program $\mathcal{P}$ is called *head-cycle free (HCF)* [7], if there is a level mapping $||\ ||_h$ of $\mathcal{P}$ such that, for every rule r of $\mathcal{P}$,

1. For any $l \in B^+(r)$, and for any $l' \in H(r)$, $||l||_h \leq ||l'||_h$;
2. For any pair $l, l' \in H(r)$ $||l||_h \neq ||l'||_h$.

Example 3.5 Consider the following program $\mathcal{P}_9$.

$$\begin{array}{ll} \mathcal{P}_9 : & a \vee b. \\ & a \text{ :- } b. \end{array}$$

It is easy to see that $\mathcal{P}_9$ is head-cycle free; an admissible level mapping for $\mathcal{P}_9$ is given by $||a||_h = 2$ and $||b||_h = 1$.

Consider now the program

$$\mathcal{P}_{10} = \mathcal{P}_9 \cup \{b \text{ :- } a.\}$$

Program $\mathcal{P}_{10}$ is not head-cycle free, since a and b should belong to the same level by Condition (1) of Definition 3.4, while they cannot by Condition (2) of the definition. Note, however, that $\mathcal{P}_{10}$ is stratified. □

4 The DLP$^{\mathcal{A}}$ Language

In this section, we provide a formal definition of the syntax and semantics of the DLP$^{\mathcal{A}}$ language – an extension of DLP by set-oriented functions (also called aggregate functions). In what follows, we refer to atoms, literals, rules, and programs of DLP, as *standard atoms, standard literals, standard rules*, and *standard programs,* respectively.

4.1 Syntax

A (DLP$^{\mathcal{A}}$) *set* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{Vars : Conj\}$, where *Vars* is a list of variables and *Conj* is a conjunction of standard literals. Intuitively, a symbolic set $\{X : a(X,Y), p(Y)\}$ stands for the set of X-values making $a(X,Y), p(Y)$ true, i.e., $\{X : \exists Y s.t.\ a(X,Y), p(Y) \textit{ is true}\}$. Note that also negative literals may occur in the conjunction *Conj* of a symbolic set.

A *ground set* is a set of pairs of the form $\langle \bar{t} : Conj \rangle$, where $\bar{t}$ is a list of constants and *Conj* is a ground (variable free) conjunction of standard literals. An *aggregate function* is of the form $f(S)$, where S is a set, and f is a *function name.* Possible function names are #`count`, #`min`, #`max`, #`sum`. An *aggregate atom* is $Lg \prec_1 f(S) \prec_2 Rg$, where $f(S)$ is an aggregate function, $\prec_1, \prec_2 \in \{=, <, \leq, >, \geq\}$, and Lg and Rg (called *left guard*, and *right guard*, respectively) are terms. Either "$Lg \prec_1$" or "$\prec_2 Rg$" can be omitted. An *atom* is either a standard (DLP) atom or an aggregate atom. A *literal* L is an atom A or an atom A preceded by the default negation symbol not; if A is an aggregate atom, L is an *aggregate literal.*

A *(*DLP$^{\mathcal{A}}$*) rule* r is a construct

$$a_1 \texttt{ v } \cdots \texttt{ v } a_n \texttt{ :- } b_1, \cdots, b_k, \text{ not } b_{k+1}, \cdots, \text{ not } b_m.$$

where $a_1, \cdots, a_n$ are standard atoms, $b_1, \cdots, b_m$ are atoms, and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \texttt{ v } \cdots \texttt{ v } a_n$ is the *head* of r, while the conjunction $b_1, ..., b_k, \text{ not } b_{k+1}, ..., \text{not } b_m$ is the *body* of r. A DLP$^{\mathcal{A}}$*program* is a set of DLP$^{\mathcal{A}}$ rules.

Syntactic Restrictions and Notation

For simplicity, and without loss of generality, we assume that the body of each rule contains at most one aggregate atom. A *global* variable of a rule r is a variable appearing in some standard atom of r; a *local* variable of r is a variable appearing only in an aggregate function in r.

Safety. A rule r is *safe* if the following conditions hold: *(i)* each global variable of r appears in a positive standard literal in the body of r; *(ii)* each local variable of r appearing in a symbolic set $\{\mathit{Vars} : \mathit{Conj}\}$, appears also in a positive literal in Conj; *(iii)* each guard of an aggregate atom of r is either a constant or a global variable. A program is safe if all of its rules are safe.

Example 4.1 Consider the following rules:

$$\begin{array}{l} p(X) \texttt{ :- } q(X,Y,V),\ Y < \#\texttt{max}\{Z : r(Z),\ \text{not } a(Z,V)\}. \\ p(X) \texttt{ :- } q(X,Y,V),\ Y < \#\texttt{sum}\{Z : \text{not } a(Z,S)\}. \\ p(X) \texttt{ :- } q(X,Y,V),\ T < \#\texttt{min}\{Z : r(Z),\ \text{not } a(Z,V)\}. \end{array}$$

The first rule is safe, while the second is not, since both local variables Z and S violate condition *(ii)*. The third rule is not safe either, since the guard T is not a global variable, thus violating condition *(iii)*. □

Stratification. A DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is *aggregate-stratified* if there exists a function $||\ ||$, called *level mapping*, from the set of (standard) predicates of $\mathcal{P}$ to ordinals, such that for each pair a and b of (standard) predicates of $\mathcal{P}$, and for each rule $r \in \mathcal{P}$: *(i)* if a appears in the head of r, and b appears in an aggregate atom in the body of r, then $||b|| < ||a||$, and *(ii)* if a appears in the head of r, and b occurs in a standard atom in the body of r, then $||b|| \leq ||a||$.

Example 4.2 Consider the program consisting of a set of facts for predicates a and b, plus the following two rules:

$$\begin{array}{l} q(X) \texttt{ :- } p(X),\ \#\texttt{count}\{Y : a(Y,X), b(X)\} \leq 2. \\ p(X) \texttt{ :- } q(X),\ b(X). \end{array}$$

The program is aggregate-stratified, as the level mapping $||a|| = ||b|| = 1$, $||p|| = ||q|| = 2$ satisfies the required conditions. If we add the rule $b(X) \texttt{ :- } p(X)$, then no legal level-mapping exists and the program becomes aggregate-unstratified. □

Intuitively, aggregate-stratification forbids recursion through aggregates. Stratification provides clear and well understood semantics. Moreover, we will see that stratification guarantees the existence of an answer set for positive programs without integrity constraints.

4.2 Informal Semantics

Let $\mathcal{P}$ be a DLP$^{\mathcal{A}}$ program and let I be a given interpretation for $\mathcal{P}$, i.e. a subset of the Herbrand Base $B_{\mathcal{P}}$ of this program, assigning a truth value to each possible instance of the predicates appearing in $\mathcal{P}$. In the next subsections we describe the meaning assigned from the interpretation I to the aggregate literals occurring in the body of the rules of $\mathcal{P}$.

4.2.1 Symbolic Set

Consider a rule r of $\mathcal{P}$ containing a standard aggregate function $\mathcal{F}$ defined on a symbolic set $\mathcal{S}$ of the type

$$Vars : Conj$$

The literals of the conjunction $Conj$ can have as terms both constants and variables. Among the last ones we can make a distinction between the *local* variables, i.e. the variables occurring only inside the conjunction of the symbolic set and the *global* variables, i.e. the variables appearing also inside other literals of the considered rule. $Vars$ is a subset of the local variables. For the sake of clarity, we first illustrate the meaning of the symbolic sets without global variables. The effect of global variables will be explained in the Section 4.3.2.

The instantiation of $\mathcal{S}$ consists in the generation of all the possible ground instances of the conjunction $Conj$ obtained from the association of constant values to the local variables. Consider the interpretation I defined for the program $\mathcal{P}$. Each ground conjunction obtained during the instantiation of $\mathcal{S}$ can be true or false w.r.t. I. The symbolic set $\mathcal{S}$ contains exactly the instances of $Vars$ for which there exists a ground instance of the conjunction $Conj$ which is true w.r.t. I.

Example 4.3 Let $\mathcal{S}$ be the symbolic set

$$X, Y : a(X, 3),\ \text{not } b(X),\ c(X, 2, Y)$$

and let I be the interpretation

$$I = \{a(1, 3),\ a(2, 3),\ a(4, 3),\ b(2),\ c(1, 2, 2),\ c(4, 2, 3)\}$$

The instances of the conjunction

$$a(X, 3),\ \text{not } b(X),\ c(X, 2, Y)$$

that are true w.r.t. the interpretation I are:

$$a(1, 3),\ \text{not } b(1),\ c(1, 2, 2)$$

$$a(4, 3),\ \text{not } b(4),\ c(4, 2, 3)$$

In these instances, X and Y assume the values

$$X = 1,\quad Y = 2$$

$$X = 4,\quad Y = 3$$

thus producing the symbolic set

$$\mathcal{S} = \{< X = 1, \;\; Y = 2 >, \; < X = 4, \;\; Y = 3 >\}$$

□

The value returned by the function **#count** is determined by all the variables in *Vars*; indeed it counts the number of *distinct* instances of the n-tuple produced by the variables in $Vars$ for which a ground instance of the conjunction $Conj$ exists being true w.r.t. I. As it will be clear in the following, all the other aggregate functions consider just the first variable of *Vars*.

We next consider in more detail the semantics of the aggregate functions allowed in DLP$^{\mathcal{A}}$.

4.3 Aggregate Functions

#count

Let $f(\mathcal{S})$ be of the form

$$\#\mathbf{count}\{Vars : Conj\}$$

and let I be an interpretation. Then, **#count** returns the number of ground instances of $Vars$ which satisfy the conjunction $Conj$ w.r.t. the interpretation I.

Example 4.4 Let I be the interpretation consisting of the following set of ground atoms

$$I = \{a(1,2), \; a(2,2), \; a(3,1), \; b(2), \; c(3,1)\}$$

and let us consider the aggregate function

$$\#\mathbf{count}\{X, Y : a(Y,2), \; \text{not } b(X), \; c(X,1)\}$$

There are two instantiations of the variables (Y, X)

$$< Y = 1, \; X = 3 >$$

$$< Y = 2, \; X = 3 >$$

such that the conjunction

$$a(Y,2), \; \text{not } b(X), \; c(X,1)$$

is true w.r.t. I. Thus, the value returned by the aggregate function is 2; formally

$$\#\mathbf{count}\{Y, X : a(Y,2), \; \text{not } b(X), \; c(X,1)\} = 2$$

□

#sum

Let $f(\mathcal{S})$ be of the form

$$\#\mathbf{sum}\{Vars : Conj\}$$

and let I be an interpretation. Then, the function **#sum** computes the sum of all the values of the first variable in *Vars* in the ground instances satisfying the conjunction $Conj$ w.r.t. the interpretation I.

Example 4.5 Let I be the interpretation consisting of the following set of ground atoms

$$I = \{a(1,2),\ a(2,2),\ a(3,2),\ b(3),\ c(3)\}$$

and consider the aggregate function

$$\#\mathbf{sum}\{X, Y : a(X,2),\ \text{not } b(X),\ c(Y)\}$$

There are two instantiations of (X, Y)

$$< Y = 3,\ X = 1 >$$

$$< Y = 3,\ X = 2 >$$

such that the conjunction

$$a(X,2),\ \text{not } b(X),\ c(Y)$$

is true w.r.t. I. Thus, the value of the aggregate function is the sum of the values taken by the variable X in these two instances, i.e.

$$\#\mathbf{sum}\{X, Y : a(X,2),\ \text{not } b(X),\ c(Y)\} = 1 + 2 = 3$$

□

#min

Let $f(\mathcal{S})$ be of the form

$$\#\mathbf{min}\{Vars : Conj\}$$

let I be an interpretation, and let X be the first variable of *Vars*. Then, the function **#min** returns the minimum value of the variable X among those taken in the ground instances of *Vars* making the conjunction $Conj$ true w.r.t. the interpretation I.

Example 4.6 Let I be the interpretation consisting of the following set of ground atoms

$$I = \{a(3,1),\ a(3,2),\ a(3,3),\ b(2),\ c(3,1)\}$$

and consider the aggregate function

$$\#\mathbf{min}\{X, Y : a(Y,X),\ \text{not } b(X),\ c(Y,1)\}$$

There are two instances of (X, Y)

$$< X = 1,\ Y = 3 >$$

$$< X = 3,\ Y = 3 >$$

such that the conjunction

$$a(Y, X),\ \text{not } b(X),\ c(Y, 1)$$

is true w.r.t. I. Thus, the aggregate function yields the minimum value of X for which the conjunction is true:

$$\#\textbf{min}\{X, Y : a(Y, X),\ \text{not } b(X),\ c(Y, 1)\} = min\{1, 3\} = 1$$

□

#max

Let $f(\mathcal{S})$ be of the form

$$\#\textbf{max}\{Vars : Conj\}$$

let I be an interpretation, and let X be the first variable of *Vars*. Then, the function **#max** returns the maximum value of the variable X among those taken in the ground instances of *Vars* making the conjunction $Conj$ true w.r.t. the interpretation I.

Example 4.7 Let I be the interpretation consisting of the following set of ground atoms

$$I = \{a(3, 1),\ a(3, 2),\ a(3, 3),\ b(2),\ c(1, 1)\}$$

and consider the aggregate function

$$\#\textbf{max}\{X, Y : a(3, X),\ \text{not } b(X),\ c(Y, 1)\}$$

There are two instances of (X, Y)

$$< X = 1,\ Y = 1 >$$

$$< X = 3,\ Y = 1 >$$

such that the conjunction

$$a(3, X),\ \text{not } b(X),\ c(Y, 1)$$

is true w.r.t. I. Thus, the aggregate function yields the value

$$\#\textbf{max}\{X, Y : a(3, X),\ \text{not } b(X),\ c(Y, 1)\} = max\{1, 3\} = 3$$

□

4.3.1 Aggregate Atoms and Literals

Consider the following aggregate literal

$$Lg \leq f\{\mathcal{S}\} \leq Rg$$

Given an interpretation I, the aggregate literal is true if and only if the value taken by the aggregate function $f(\mathcal{S})$ belongs to the range specified by the guard values Lg and Rg.

Example 4.8 Consider the Example 4.3 and the following aggregate literal

$$0 \leq \#\mathbf{sum}\{\mathcal{S}\} \leq 10$$

Since the function **#sum** returns the value 5 w.r.t. I, we have that $f\{\mathcal{S}\}$ is true w.r.t. the interpretation I. □

4.3.2 Sets with global variables

In this section we consider the case in which a symbolic set $\mathcal{S}$ contains also global variables. We recall that, due to the safety condition, all the global variables must appear in some positive standard atom in the body of the rule r in which $\mathcal{S}$ is defined. Thus, each instantiation of r assigns a value to these global variables. After that, the symbolic set $\mathcal{S}$ is evaluated as described above (in which we have supposed the absence of global variables).

Example 4.9 Let us consider the following rule

$$h \text{ :- } b(Y), 0 \leq \#\mathbf{count}\{X : a(X,Y)\} \leq 2.$$

Note that Y is a global variable, whereas X is a local variable. Let us consider the following interpretation

$$I = \{a(1,1),\ a(2,1),\ a(2,2),\ b(1),\ b(2)\}$$

The instantiation of the global variable Y generates

$$h \text{ :- } b(1), 0 \leq \#\mathbf{count}\{X : a(X,1)\} \leq 1.$$

$$h \text{ :- } b(2), 0 \leq \#\mathbf{count}\{X : a(X,2)\} \leq 1.$$

and, hence, the successive instantiation of the local variable X leads to the following ground instances of V

$$h \text{ :- } b(1), 0 \leq \#\mathbf{count}\{1 : a(1,1)\} \leq 1.$$

$$h \text{ :- } b(1), 0 \leq \#\mathbf{count}\{2 : a(2,1)\} \leq 1.$$

and

$$h \text{ :- } b(2), 0 \leq \#\mathbf{count}\{2 : a(2,2)\} \leq 1.$$

Note that, if $Y = 1$ we have two distinct instances of X satisfying $a(X,Y)$ and, hence, $0 \leq \#\mathbf{count}\{X : a(X,Y)\} \leq 1$ is false. Conversely, if $Y = 2$ the aggregate literal is evaluated true. □

4.4 Semantics

Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$, let $U_{\mathcal{P}}$ denote the set of constants appearing in $\mathcal{P}$, $U_{\mathcal{P}}^{\mathcal{N}} \subseteq U_{\mathcal{P}}$ the set of the natural numbers occurring in $U_{\mathcal{P}}$, and $B_{\mathcal{P}}$ the set of standard atoms obtainable from the (standard) predicates of $\mathcal{P}$ with constants in $U_{\mathcal{P}}$. Furthermore, given a set X, $\overline{2}^X$ denotes the set of all multisets over elements from X. Let us now describe the domains and the meanings of the aggregate functions we consider:
#`count`: defined over $\overline{2}^{U_{\mathcal{P}}}$, the number of elements in the set.
#`sum`: defined over $\overline{2}^{U_{\mathcal{P}}^{\mathcal{N}}}$, the sum of the numbers in the set.[2]
#`min`, #`max`: defined over $\overline{2}^{U_{\mathcal{P}}} - \{\emptyset\}$, the minimum/maximum element in the set; if the set contains strings, the lexicographic ordering is considered.[3]
If the argument of an aggregate function does not belong to its domain, the aggregate evaluates to false, denoted as $\perp$ (and our implementation issues a warning in this case).

A *substitution* is a mapping from a set of variables to the set $U_{\mathcal{P}}$ of the constants appearing in the program $\mathcal{P}$. A substitution from the set of global variables of a rule r (to $U_{\mathcal{P}}$) is a *global substitution for r*; a substitution from the set of local variables of a symbolic set S (to $U_{\mathcal{P}}$) is a *local substitution for S*. Given a symbolic set without global variables $S = \{Vars : Conj\}$, the *instantiation of S* is the following ground set of pairs $inst(S)$:
$\{\langle\gamma(Vars) : \gamma(Conj)\rangle \mid \gamma \textit{ is a local substitution for } S\}$.[4]
A *ground instance* of a rule r is obtained in two steps: (1) a global substitution σ for r is first applied over r; (2) every symbolic set S in $\sigma(r)$ is replaced by its instantiation $inst(S)$. The instantiation $Ground(\mathcal{P})$ of a program $\mathcal{P}$ is the set of all possible instances of the rules of $\mathcal{P}$.

Example 4.10 Consider the following program $\mathcal{P}_1$:

$$
\begin{array}{l}
q(1) \;\mathtt{v}\; p(2,2). \\
q(2) \;\mathtt{v}\; p(2,1). \\
t(X) \;\text{:-}\; q(X), \#\mathtt{sum}\{Y : p(X,Y)\} > 1.
\end{array}
$$

The instantiation $Ground(\mathcal{P}_1)$ is the following:

$$
\begin{array}{l}
q(1) \;\mathtt{v}\; p(2,2). \\
q(2) \;\mathtt{v}\; p(2,1). \\
t(1) \;\text{:-}\; q(1), \#\mathtt{sum}\{\langle 1 : p(1,1)\rangle, \langle 2 : p(1,2)\rangle\} > 1. \\
t(2) \;\text{:-}\; q(2), \#\mathtt{sum}\{\langle 1 : p(2,1)\rangle, \langle 2 : p(2,2)\rangle\} > 1.
\end{array}
$$

□

Interpretation and models. An *interpretation* for a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is a set of standard ground atoms $I \subseteq B_{\mathcal{P}}$. The truth valuation $I(A)$, where A is a standard ground literal or a standard ground conjunction, is defined in the usual way. Besides

[2]#`sum` applied over an empty set returns 0.
[3]The latter is not yet supported in our first implementation.
[4]Given a substitution σ and a DLP$^{\mathcal{A}}$ object *Obj* (rule, conjunction, set, etc.), with a little abuse of notation, we denote by $\sigma(Obj)$ the object obtained by replacing each variable X in *Obj* by $\sigma(X)$.

assigning truth values to the standard ground literals, an interpretation provides the meaning also to (ground) sets, aggregate functions and aggregate literals; the meaning of a set, an aggregate function, and an aggregate atom under an interpretation, is a multiset, a value, and a truth-value, respectively. Let $f(S)$ be an aggregate function. The valuation $I(S)$ of the set S w.r.t. I is the multiset composed by the first constant of the first components of the elements in S whose conjunction is true w.r.t. I. More precisely, let $S_I = \{\langle t_1, ..., t_n \rangle \mid \langle t_1, ..., t_n : Conj \rangle \in S \wedge$ *Conj is true w.r.t.* $I\}$, then $I(S)$ is the multiset $[\, t_1 \mid \langle t_1, ..., t_n \rangle \in S_I \,]$. The valuation $I(f(S))$ of an aggregate function $f(S)$ w.r.t. I is the result of the application of the function f on $I(S)$. (If the multiset $I(S)$ is not in the domain of f, $I(f(S)) = \bot$.)

An aggregate atom $A = Lg \prec_1 f(S) \prec_2 Rg$ is *true w.r.t.* I if: *(i)* $I(f(S)) \neq \bot$, and, *(ii)* the relationships $Lg \prec_1 I(f(S))$, and $I(f(S)) \prec_2 Rg$ hold whenever they are present; otherwise, A is false.

A *model* for $\mathcal{P}$ is an interpretation M for $\mathcal{P}$ such that every rule $r \in Ground(\mathcal{P})$ is true w.r.t. M. A model M for $\mathcal{P}$ is *minimal* if no model N for $\mathcal{P}$ exists such that N is a proper subset of M.

By exploiting the above notion of truth valuation for aggregate atoms, the truth valuations of aggregate literals and rules, as well as the notion of model and minimal model for DLP$^{\mathcal{A}}$ are immediate extensions of the corresponding notions in DLP (see Section 3 and [25]).

Example 4.11 Consider the aggregate atom $A = \#\texttt{sum}\{\langle 1 : p(2,1)\rangle, \langle 2 : p(2,2)\rangle\} > 1$ in Example 4.10. Let S be the ground set appearing in A. For the interpretation $I = \{q(2), p(2,2), t(2)\}$, $I(S) = [2]$, the application of #sum over [2] yields 2, and A is therefore true w.r.t. I, since $2 > 1$.
I is a minimal model for the program of Example 4.10. □

Answer Sets. First we define the answer sets of positive DLP$^{\mathcal{A}}$ programs without aggregates, then we give a reduction of disjunctive datalog programs containing negation as failure and aggregates to positive programs without aggregates and use it to define answer sets of arbitrary disjunctive datalog programs, possibly containing negation as failure and aggregates.

An interpretation $X \subseteq B_{\mathcal{P}}$ is called *closed under* $\mathcal{P}$ (where $\mathcal{P}$ is a positive disjunctive datalog program without aggregates), if, for every $r \in Ground(\mathcal{P})$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation $X \subseteq B_{\mathcal{P}}$ is an *answer set* for a positive DLP$^{\mathcal{A}}$ without aggregates $\mathcal{P}$, if it is minimal (under set inclusion) among all interpretations that are closed under $\mathcal{P}$.[5]

Example 4.12 The positive program

$$\mathcal{P}_1 = \{a \vee b \vee c.\}$$

has the answer sets $\{a\}$, $\{b\}$, and $\{c\}$. Its extension

$$\mathcal{P}_2 = \{a \vee b \vee c. \;\; ; \;\; \text{:-}\ a\}.$$

[5]Note that we consider only *consistent answer sets*, whereas in [25] also the inconsistent set of all possible literals can be a valid answer set.

has the answer sets {$\mathtt{b}$} and {$\mathtt{c}$}. Finally, the positive program

$$\mathcal{P}_3 = \{a \,\mathtt{v}\, b \,\mathtt{v}\, c. \quad ; \quad \text{:-}\, a. \quad ; \quad \mathrm{b}\ \text{:-}\ \mathrm{c}. \quad ; \quad \mathrm{c}\ \text{:-}\ \mathrm{b}.\}$$

has the single answer set $\{b, c\}$. □

The *reduct* or *Gelfond-Lifschitz transform* of a ground program $\mathcal{P}$ w.r.t. a set $X \subseteq B_{\mathcal{P}}$ is the positive ground program $\mathcal{P}^X$, obtained from $\mathcal{P}$ by

- deleting all rules $r \in \mathcal{P}$ for which a negative literal in $B(r)$ is false w.r.t. X or an aggregate literal is false w.r.t. X;
- deleting the aggregate literals and the negative literals from the remaining rules.

An answer set of a program $\mathcal{P}$ is a set $X \subseteq B_{\mathcal{P}}$ such that X is an answer set of $Ground(\mathcal{P})^X$.

Example 4.13 Given the following program with negation and stratified aggregates

$$\mathcal{P}_4 = \{d(1). \quad ; \quad a \,\mathtt{v}\, b \text{ :- } c. \quad ; \quad b \text{ :- not } a, \text{not } c, \#\mathtt{count}\{Y : d(Y)\} > 0. \quad ;$$
$$a \,\mathtt{v}\, c \text{ :- not } b, \#\mathtt{sum}\{Y : d(Y)\} > 1.\}$$

and $I = \{\mathtt{b} \quad ; \quad \mathtt{d(1)}\}$, the reduct $\mathcal{P}_4^I$ is

$$\{a \,\mathtt{v}\, b \text{ :- } c. \quad ; \quad b. \quad ; \quad \mathtt{d(1)}.\}.$$

It is easy to see that I is an answer set of $\mathcal{P}_4^I$, and for this reason it is also an answer set of $\mathcal{P}_4$.

If, on the other hand, we consider the interpretation $K = \{\mathtt{c} \quad ; \quad \mathtt{d(1)}\}$, the reduct $\mathcal{P}_4^K$ is

$$\{a \,\mathtt{v}\, b \text{ :- } c. \quad ; \quad \mathtt{d(1)}.\}$$

but K is not an answer set of $\mathcal{P}_4^K$. Indeed, for the rule

$$r : a \,\mathtt{v}\, b \text{ :- } c,$$

$B(r) \subseteq K$ holds, but $H(r) \cap K \neq \emptyset$ does not. □

Example 4.14 Given the following program with unstratified aggregates

$$\mathcal{P}_5 = \{p(1) \text{ :- } \#\mathtt{count}\{X : p(X)\} < 2. \quad ; \quad p(2) \,\mathtt{v}\, q(2).\}$$

and $I = \{\mathtt{p(1)} \quad ; \quad \mathtt{p(2)}\}$, the reduct $\mathcal{P}_5^I$ is

$$\{p(2) \,\mathtt{v}\, q(2).\}.$$

It is easy to see that I is a model but not a minimal model of $\mathcal{P}_5^I$ and, for this reason, it is not an answer set of $\mathcal{P}_5$. Now consider $J = \{\mathtt{p(1)} \quad ; \quad \mathtt{q(2)}\}$. The reduct $\mathcal{P}_5^J$ is

$$\{p(1). \quad ; \quad p(2) \,\mathtt{v}\, q(2).\}.$$

and it can be easily verified that J is an answer set of $\mathcal{P}_5^J$, so it is also an answer set of $\mathcal{P}_5$. If, on the other hand, we take $K = \{\mathtt{p(2)}\}$, the reduct $\mathcal{P}_5^K$ is equal to $\mathcal{P}_5^J$, but K is not a model of $\mathcal{P}_5^K$. Indeed, for the rule

$$r : p(1).$$

$B(r) \subseteq K$ holds, but $H(r) \cap K \neq \emptyset$ does not. It can be verified that J is the only answer sets of $\mathcal{P}_5$. □

5 Knowledge Representation in $\mathrm{DLP}^{\mathcal{A}}$

In this section, we show by two examples how aggregate functions can be used to encode relevant problems.

Team Building. A project team has to be built from a set of employees according to the following specifications:
(p_1) The team consists of a certain number of employees.
(p_2) At least a given number of different skills must be present in the team.
(p_3) The sum of the salaries of the employees working in the team must not exceed the given budget.
(p_4) The salary of each individual employee is within a specified limit.
(p_5) The number of women working in the team has to reach at least a given number.

Suppose that our employees are provided by a number of facts of the form *emp(EmpId,Sex,Skill,Salary)*; the size of the team, the minimum number of different skills, the budget, the maximum salary, and the minimum number of women are specified by the facts *nEmp(N)*, *nSkill(N)*, *budget(B)*, *maxSal(M)*, and *women(W)*. We then encode each property p_i above by an aggregate atom A_i, and enforce it by an integrity constraint containing not A_i.

$$\begin{array}{l}
in(I) \,\mathtt{v}\, out(I) \text{ :- } emp(I, Sx, Sk, Sa).\\
\text{ :- } nEmp(N), \text{not } \#\mathtt{count}\{I : in(I)\} = N.\\
\text{ :- } nSkill(M), \text{not } \#\mathtt{count}\{Sk : emp(I, Sx, Sk, Sa), in(I)\} \geq M.\\
\text{ :- } budget(B), \text{not } \#\mathtt{sum}\{Sa, I : emp(I, Sx, Sk, Sa), in(I)\} \leq B.\\
\text{ :- } maxSal(M), \text{not } \#\mathtt{max}\{Sa : emp(I, Sx, Sk, Sa), in(I)\} \leq M.\\
\text{ :- } women(W), \text{not } \#\mathtt{count}\{I : emp(I, f, Sk, Sa), in(I)\} \geq W.
\end{array}$$

Intuitively, the disjunctive rule "guesses" whether an employee is included in the team or not, while the five constraints have a one-to-one correspondence with the five requirements p_1-p_5. Thanks to the aggregates the translation of the specifications is surprisingly straightforward. The example highlights the usefulness of representing both sets and multisets in our language (a multiset can be obtained by specifying more than one variable in *Vars* of a symbolic set $\{\,Vars : Conj\}$). For instance, the encoding of p_2 requires a set, as we want to count *different* skills; two employees in the team having the same skill, should count once w.r.t. p_2. On the contrary, p_3 requires to sum the elements of a multiset; if two employees have the same salary, *both* salaries should be summed up for p_3. This is obtained by adding the variable I to *Vars*. The valuation of $\{Sa, I : emp(I, Sx, Sk, Sa), in(I)\}$ yields the set $S = \{\langle Sa, I\rangle : \textit{Sa is the salary of employee I in the team}\}$. Then, the sum function is applied on the multiset of the first components Sa of the tuples $\langle Sa, I\rangle$ in S (see Section 4.4).

Seating. We have to generate a sitting arrangement for a number of guests, with m tables and n chairs per table. Guests who like each other should sit at the same table; guests who dislike each other should not sit at the same table.

Suppose that the number of chairs per table is specified by $nChairs(X)$ and that $person(P)$ and $table(T)$ represent the guests and the available tables, respectively. Then, we can generate a seating arrangement by the following program:

% Guess whether person P sits at table T or not.
$at(P,T)$ v $not\_at(P,T)$:- $person(P), table(T)$.
% The persons sitting at a table cannot exceed the chairs.
:- $table(T), nChairs(C)$, not $\#\texttt{count}\{P : at(P,T)\} \leq C$.
% A person is seated at precisely one table; equivalent
% to :- $person(P), at(P,T), at(P,U), T <> U$.
:- $person(P)$, not $\#\texttt{count}\{T : at(P,T)\} = 1$.
% People who like each other should sit at the same table...
:- $like(P1,P2), at(P1,T)$, not $at(P2,T)$.
% ...while people who dislike each other should not.
:- $dislike(P1,P2), at(P1,T), at(P2,T)$.

6 The Complexity of Disjunctive Logic Programming

We next analyze the computational complexity of both DLP and DLP$^{\mathcal{A}}$ languages. In the sequel we first provide some preliminaries on complexity theory. Then, we define the main computational problems under consideration and derive their precise complexity.

6.1 A Reminder of the Polynomial Hierarchy

We assume that the reader has some acquaintance with the concepts of NP-completeness and complexity theory. We next provide a very short reminder of the complexity classes of the Polynomial Hierarchy which are relevant to this section. The book [39] is an excellent source for deepening the knowledge in this field.

The classes Σ_k^P, Π_k^P, and Δ_k^P of the *Polynomial Hierarchy (PH)* (cf. [29]) are defined as follows:

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

where NP^C denotes the class of decision problems that are solvable in polynomial time on a nondeterministic Turing machine with an oracle for any decision problem π in the class C. In particular, NP $= \Sigma_1^P$, co-NP $= \Pi_1^P$, and $\Delta_2^P = \text{P}^{\text{NP}}$.

The oracle replies to a query in unit time, and thus, roughly speaking, models a call to a subroutine for π that is evaluated in unit time.

Observe that for all $k \geq 1$,

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

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

6.2 Main Problems Considered

As for the classical nonmonotonic formalisms, three important decision problems, corresponding to three different reasoning tasks, arise in the context of the DLP ($\mathrm{DLP}^{\mathcal{A}}$) language:

> **Answer Set Checking.** Given a DLP ($\mathrm{DLP}^{\mathcal{A}}$) program $\mathcal{P}$, and a set M of ground literals as input, decide whether M is an answer set of $\mathcal{P}$.
>
> **Brave Reasoning**. Given a DLP ($\mathrm{DLP}^{\mathcal{A}}$) program $\mathcal{P}$, and a ground atom A, decide whether A is true in some answer set of $\mathcal{P}$ (denoted $\mathcal{P} \models_b A$).
>
> **Cautious Reasoning**. Given a DLP ($\mathrm{DLP}^{\mathcal{A}}$) program $\mathcal{P}$, and a ground atom A, decide whether A is true in all answer sets of $\mathcal{P}$ (denoted $\mathcal{P} \models_c A$).

We study the complexity of the above decision problems, which are strongly relevant to the tasks performed by the DLP ($\mathrm{DLP}^{\mathcal{A}}$) computational engine. Brave Reasoning is strictly related also to the problem of *finding* an answer set.

In the following, we analyze the computational complexity of the decision problems specified above for ground (i.e., propositional) DLP ($\mathrm{DLP}^{\mathcal{A}}$) programs; we shall address the case of non-ground programs at the end of this section.

An interesting issue, from the viewpoint of system implementation, is the impact of syntactic restrictions on the logic program $\mathcal{P}$. Starting from normal positive programs (without negation and disjunction), we consider the effect of allowing the (combined) use of the following constructs:

- stratified (nonmonotonic) negation (not_s),
- arbitrary (nonmonotonic) negation (not),
- head-cycle free disjunction ($\mathtt{v}_h$),
- arbitrary disjunction ($\mathtt{v}$).

6.3 Complexity Results for DLP

The computational complexity of the above problems for a number of fragments of the DLP language has been previously analyzed, cf. [19, 26, 10, 23, 17].

Throughout this section, we consider the ground case, i.e., we assume that programs $\mathcal{P}$ are ground, and unless stated otherwise, also atoms, literals etc. are ground. Furthermore, to simplify matters, we shall consider language fragments without strong negation and integrity constraints. However, this does not change the generality of results, since the results in presence of strong negation and integrity constraints are the same (see, e.g., [10]).

The complexity of Brave Reasoning and Cautious Reasoning from ground DLP programs are summarized in Table 1 and Table 2, respectively. In Table 3, we report the results on the complexity of Answer Set Checking.

	{}	{not$_s$}	{not}
{}	P	P	NP
{ v$_h$}	NP	NP	NP
{ v }	Σ_2^P	Σ_2^P	Σ_2^P

Table 1: The Complexity of Brave Reasoning in fragments of the DLP Language

	{}	{not$_s$}	{not}
{}	P	P	co-NP
{ v$_h$}	co-NP	co-NP	co-NP
{ v }	co-NP	Π_2^P	Π_2^P

Table 2: The Complexity of Cautious Reasoning in fragments of the DLP Language

Each row of the table specifies the form of disjunction allowed; in particular, $\{\} =$ no disjunction, $\{\,\mathtt{v}_h\} =$ head-cycle free disjunction, and $\{\,\mathtt{v}\,\} =$ unrestricted (possibly not head-cycle free) disjunction. Each column specifies the support for negation. Each entry of the table provides the complexity of the corresponding fragment of the language, in terms of a completeness result. For instance, $(\{\,\mathtt{v}_h\}, \{\mathrm{not}_s\})$ is the fragment allowing head-cycle free disjunction and stratified negation. The corresponding entry in Table 1, namely NP, expresses that brave reasoning for this fragment is NP-complete. The results reported in the tables represent completeness under polynomial time reductions, and in fact under LOGSPACE reductions. For a detailed analysis of the results reported in Tables 1, 2 and 3 the reader is referred to [19, 26, 23, 17, 10].

Looking at Table 1, we see that limiting the form of disjunction and negation reduces the respective complexity. For disjunction-free programs, brave reasoning is polynomial on stratified negation, while it becomes NP-complete if we allow unrestricted (nonmonotonic) negation. Brave reasoning is NP-complete on head-cycle free programs even if no form of negation is allowed. The complexity jumps one level higher in the Polynomial Hierarchy, up to Σ_2^P-complexity, if full disjunction is allowed. Thus, disjunction seems to be harder than negation, since the full complexity is reached already on positive programs, even without any kind of negation.

Table 2 contains results for cautious reasoning. One would expect its complexity to be symmetric to the complexity of brave reasoning, that is, whenever the complexity of a fragment is C under brave reasoning, one expects its complexity to be co-C under cautious reasoning (recall that co-P = P, co-$\Delta_2^P = \Delta_2^P$, co-$\Sigma_2^P = \Pi_2^P$, and co-$\Delta_3^P = \Delta_3^P$).

Surprisingly, there is one exception: while full disjunction raises the complexity of brave reasoning from NP to Σ_2^P, full disjunction alone is not sufficient to raise the complexity of cautious reasoning from co-NP to Π_2^P. Cautious reasoning remains in co-NP if default negation is disallowed. Intuitively, to disprove that an atom A is a cautious consequence of a program $\mathcal{P}$, it is sufficient to find a *model* M of $\mathcal{P}$ (which need not be an answer set or a minimal model) which does not contain A. Indeed, for not-free programs,

	{}	{$\text{not}_\mathbf{s}$}	{not}
{}	P	P	P
{ $\mathtt{v}_\mathbf{h}$ }	P	P	P
{ v }	co-NP	co-NP	co-NP

Table 3: The Complexity of Answer Set Checking in fragments of the DLP Language

the existence of such a model guarantees the existence of a subset of M which is an answer set of $\mathcal{P}$ (and does not contain A).

The complexity results for Answer Set Checking, reported in Table 3, help us to understand the complexity of reasoning. Whenever Answer Set Checking for programs with arbitrary disjunction is co-NP-complete, then the complexity of brave reasoning jumps up to the second level of the Polynomial Hierarchy (Σ_2^P). Indeed, brave reasoning on full DLP programs suffers from two sources of complexity:

- (s_1) the exponential number of answer set "candidates",
- (s_2) the difficulty of checking whether a candidate M is an answer set (the minimality of M can be disproved by an exponential number of subsets of M), and

Now, disjunction (unrestricted or even head-cycle free) or unrestricted negation preserve the existence of source (s_1), whereas source (s_2) exists only if full disjunction is allowed (see Table 3). The complexity goes down to the first level of PH if source (s_2) is eliminated, being in the class NP. Finally, avoiding source (s_1) the complexity falls down to P, as source (s_2) is automatically eliminated.

6.4 Complexity Results for DLP$^\mathcal{A}$

The following theorems deal with the complexity of the above mentioned reasoning tasks for ground DLP$^\mathcal{A}$ programs that respect the syntactic restrictions imposed in Section 4 (safety and aggregate-stratification). It is worth pointing out that reasoning in DLP$^\mathcal{A}$ does not produce an increase in computational complexity, which remains exactly the same as for standard DLP.

Theorem 6.1 (Answer Set Checking). *Deciding whether an interpretation M is an answer set for a ground program $\mathcal{P}$ is co-NP-complete.*

Proof. We check in NP that M is <u>not</u> an answer set of $\mathcal{P}$ as follows. Guess a subset I of M, and verify that: (1) M is not a model for $\mathcal{P}$, or (2) $I \subset M$ and I is a model of the Gelfond-Lifschitz transform of $\mathcal{P}$ w.r.t. M.

The only difference w.r.t. the corresponding tasks of (1) and (2) in standard DLP, is the computation of the truth-valuations of the aggregate atoms, which requires, in turn, to compute the valuations of aggregate functions and sets. Computing the valuation of a ground set T only requires scanning each element $\langle t_1, ..., t_n : Conj\rangle$ of T, adding t_1 to the result multiset if Conj is true w.r.t. I. This is evidently polynomial, as is the application of the aggregate operators (#`count`, #`min`, #`max`, #`sum`) on a multiset; the comparison of the guards with its result is straightforward.

Therefore, the tasks (1) and (2) are tractable as in standard DLP. Deciding whether M is not an answer set for $\mathcal{P}$, is in NP, and, consequently, deciding whether M is an answer set for $\mathcal{P}$ is in co-NP.

Hardness follows from the fact that DLP$^{\mathcal{A}}$ is a superset of DLP. □

Theorem 6.2 (Brave Reasoning). *Brave Reasoning on ground* DLP$^{\mathcal{A}}$ *programs is* Σ_2^P*-complete.*

Proof. We verify that a ground literal L is a brave consequence of a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ as follows: Guess a set $M \subseteq B_{\mathcal{P}}$ of ground literals, check that (1) M is an answer set for $\mathcal{P}$, and (2) L is true w.r.t. M. Task (2) is clearly polynomial; while (1) is in co-NP, by virtue of Theorem 6.1. The problem therefore lies in Σ_2^P.

Σ_2^P-hardness follows from the Σ_2^P-hardness of DLP [19], since DLP$^{\mathcal{A}}$ is a superset of DLP. □

The complexity of cautious reasoning follows by similar arguments as above.

Theorem 6.3 (Cautious Reasoning). *Cautious Reasoning on ground* DLP$^{\mathcal{A}}$ *programs is* Π_2^P*-complete.*

Proof. We verify that a ground literal L is *not* a cautious consequence of a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ as follows: Guess a set $M \subseteq B_{\mathcal{P}}$ of ground literals, check that (1) M is an answer set for $\mathcal{P}$, and (2) L is not true w.r.t. M. Task (2) is clearly polynomial; while (1) is in co-NP, by virtue of Theorem 6.1. Therefore, the complement of cautious reasoning is in Σ_2^P, and cautious reasoning is in Π_2^P.

Π_2^P-hardness follows from [17], since DLP$^{\mathcal{A}}$ is a superset of DLP. □

As for the case of full DLP extended with aggregate operators, by using arguments similar to the ones exploited in the above proofs, it is easy to show that the complexity results for the various fragments of DLP$^{\mathcal{A}}$ are the same as those obtained for the corresponding fragments of DLP.

We close this section by briefly addressing the complexity and expressiveness of non-ground programs. A non-ground program $\mathcal{P}$ can be reduced, by naive instantiation, to a ground instance of the problem. In the general case, where $\mathcal{P}$ is given in the input, the size of the grounding $Ground(\mathcal{P})$ is single exponential in the size of $\mathcal{P}$. Informally, the complexity of Brave Reasoning and Cautious Reasoning increases accordingly by one exponential, from P to EXPTIME, from NP to NEXPTIME, from Δ_2^P to EXPTIME$^{\mathrm{NP}}$, from Σ_2^P to NEXPTIME$^{\mathrm{NP}}$, etc. For disjunctive programs and certain fragments of the DLP language, complexity results in the non-ground case have been derived e.g. in [19, 23]. For the other fragments, the results can be derived using complexity upgrading techniques [19, 27].

7 Related Works

Aggregate functions in logic programming languages already appeared in the 80s, when their need emerged in deductive databases like LDL [12]. Several proposals for modelling aggregate functions both directly and indirectly have been presented in literature. For instance, one of the most remarkable approaches of extending logic programming with additional constructs for its successful use in practice is presented in [20, 21]. These papers define stable model semantics for logic programs with generalized quantifier atoms in the body, which can be used to easily simulate operations such as SUM and COUNT.

Despite the great interest of the logic community in aggregate function, the first implementation in Answer Set Programming, based on the Smodels system, is quite recent [47].

Comparing DLP$^{\mathcal{A}}$ to the language of Smodels, we observe a strong similarity between its cardinality constraints and function #`count`. Also weight constraints in Smodels and function #`sum` are similar in spirit. However, there are some relevant differences. For instance, weights and guards are allowed to be also variables in DLP$^{\mathcal{A}}$ aggregates, whereas they are forced to be constants in Smodels. In this way, DLP$^{\mathcal{A}}$ provides a better support to the programmer for writing <u>uniform</u> encodings. For instance, consider the DLP$^{\mathcal{A}}$ constraint

$:\!-\, budget(B), \mathrm{not}\#\mathtt{sum}\{Sa, I : emp(I, Sx, Sk, Sa), in(I)\} \leq B.$

encoding specification p_3 of **Team Building**: "the sum of the salaries of the employees working in the team cannot exceed a given budget". This constraint does not have a direct translation in Smodels language. The programmer should manually write the big set containing the weighted atoms to be summed up in Smodels. A manual writing of the instantiation can be impractical and error-prone in real-world applications, as pointed out also in [47].

As a further, important difference between DLP$^{\mathcal{A}}$ and Smodels, we point out that aggregate atoms in DLP$^{\mathcal{A}}$ can be negated, whereas cardinality and weight constraint literals in Smodels cannot.

Negated aggregates are useful for a more direct knowledge representation, and allow to express, for instance, that some value should be external to a given range. As an example, not $3 \leq \#\mathtt{count}\{X : p(X)\} \leq 7$ is true if the number of true facts for p is in $[0, 3[\cup]7, \infty[$; for expressing the same property in Smodels one needs to use 2 cardinality constraints.

Smodels, on the other hand, allows for weight constraints in the heads of rules, whereas DLP$^{\mathcal{A}}$ aggregates cannot occur in heads. However, it has been shown [33] that even if the presence of weight constraints in heads is a powerful KR feature, it causes the loss of some semantic property of nonmonotonic languages. Observe also that DLP$^{\mathcal{A}}$ aggregates like #`min` and #`max` do not have a counterpart in Smodels. Moreover, DLP$^{\mathcal{A}}$ provides a general framework where further aggregates can be easily accommodated (e.g., #`any` and #`avg`). Furthermore, note that symbolic sets of DLP$^{\mathcal{A}}$ directly represent pure (mathematical) sets, and can also represent multisets rather naturally (see the discussion on Team Building in Section 5). On the contrary, Smodels weight constraints work on multisets and additional rules are needed to encode pure sets; for instance, Condition p_2 of **Team Building** cannot be encoded directly in a constraint, but needs the definition of an extra predicate. Finally, note that DLP$^{\mathcal{A}}$ deals with sets of terms, while Smodels deals with sets of atoms. As far as the implementation is concerned, Smodels is endowed with advanced pruning operators for weight constraints, which are efficiently implemented; we are not aware, though, of techniques for the automatic recognition of duplicate sets in Smodels.

Another interesting line of research uses 4-valued logics and approximating operators to define the semantics of aggregate functions in logic-based languages [15, 14, 40]. These approaches are founded on very solid theoretical grounds, and appear very promising, as they could provide a clean formalization of a very general framework for arbitrary aggregates in logic programming and nonmonotonic reasoning. In these approaches aggregate atoms can also "produce" new values (currently, in both DLP$^{\mathcal{A}}$ and Smodels the guards of the aggregates need to be bound to some value). However, these approaches sometimes amount to a higher computational complexity [40], and there is no implementation available so far.

8 Conclusions

In this report we have defined the Internal Integration Formalism (IIF) of the INFOMIX system. In particular, we have shown that the IIF should be: *(i)* able to represent tuples stored in relational databases, *(ii)* able to express first order sentences over such tuples and, *(iii)* at least as expressive as DATALOG. From the observations above and the choices performed in the previous deliverables it appears clear that the most suited language for the specification of the IIF is Disjunctive Datalog (DLP). However, in spite of the high expressiveness of DLP we have shown that many real life queries in the context of data integration cannot be modelled with standard DLP. To overcome this problem, we have extended classical DLP to DLP$^{\mathcal{A}}$, a language capable of supporting also aggregate operators. We provided a description of the syntax and the semantics of DLP$^{\mathcal{A}}$ as well as a detailed analysis of both DLP and DLP$^{\mathcal{A}}$ complexities. Knowledge representation and problem solving in DLP$^{\mathcal{A}}$ have been addressed in an example-driven way. Finally, we have compared DLP$^{\mathcal{A}}$ with other languages already proposed in the literature for handling aggregate functions. In particular, a thorough comparison between DLP$^{\mathcal{A}}$ and Smodels has been presented.

References

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

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a Theory of Declarative Knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Inc., Washington DC, 1988.

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. of the 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000)*, pages 27–41. Springer, 2000.

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

[5] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2002.

[6] Pablo Barceló and Leopoldo Bertossi. Reapairing databases with annotated predicate logic. In *Proc. the 10th Int. Workshop on Non-Monotonic Reasoning (NMR 2002)*, pages 160–170, 2002.

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

[8] Stefan Brass and Jürgen Dix. Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation. In Leon Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming*, pages 199–213, Tokyo, June 1995. MIT Press.

[9] Loreto Bravo and Leopoldo Bertossi. Logic programming for consistently querying data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, 2003. To appear.

[10] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[11] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, 2003. To appear.

[12] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL System Prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 1990.

[13] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[14] M. Denecker, V. Marek, and M. Truszczyński. Ultimate Approximations in Monotonic Knowledge Representation Systems. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.A. Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning, KR-2002, April 22-25, 2002, Toulouse, France*, pages 177–188. Morgan Kaufmann, 2002.

[15] M. Denecker, N. Pelov, and M. Bruynooghe. Well-Founded and Stable Model Semantics for Logic Programs with Aggregates. In Philippe Codognet, editor, *Proceedings of the 17th International Conference on Logic Programming*, pages 212–226. Springer Verlag, 2001.

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

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

[18] Thomas Eiter, Wolfgang Faber, Georg Gottlob, Christoph Koch, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and F. Scarcello. The DLV System. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC*, College Park, Maryland, June 1999. Computer Science Department, University of Maryland. Workshop Notes.

[19] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[20] Thomas Eiter, Georg Gottlob, and Helmuth Veith. Modular Logic Programming and Generalized Quantifiers. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR-97)*, number 1265 in LNCS, pages 290–309. Springer, 1997.

[21] Thomas Eiter, Georg Gottlob, and Helmuth Veith. Generalized Quantifiers in Logic Programs. In Jouko Väänänen, editor, *Generalized Quantifiers and Computation, 9th European Summer School in Logic, Language, and Information, ESSLLI'97 Workshop, Aix-en-Provence, France, August 11-22, 1997, Revised Lectures*, number 1754 in Lecture Notes in Computer Science, pages 72–98. Springer, 2000.

[22] Thomas Eiter, Nicola Leone, and Domenico Saccà. On the Partial Semantics for Disjunctive Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(1–2):59–96, April 1997.

[23] Thomas Eiter, Nicola Leone, and Domenico Saccá. Expressive Power and Complexity of Partial Models for Disjunctive Deductive Databases. *Theoretical Computer Science*, 206(1–2):181–218, October 1998.

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

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

[26] Georg Gottlob. Complexity and Expressive Power of Disjunctive Logic Programming. In M. Bruynooghe, editor, *Proceedings of the International Logic Programming Symposium (ILPS '94)*, pages 23–42, Ithaca NY, 1994. MIT Press.

[27] Georg Gottlob, Nicola Leone, and Helmut Veith. Succinctness as a Source of Expression Complexity. *Annals of Pure and Applied Logic*, 97(1–3):231–260, 1999.

[28] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of the 17th Int. Conf. on Logic Programming (ICLP'01)*, volume 2237 of *Lecture Notes in Artificial Intelligence*, pages 348–364. Springer, 2001.

[29] David S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Pub., 1990.

[30] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Christoph Koch, Cristinel Mateis, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. Technical Report cs.AI/0211004, arXiv.org, November 2002. Submitted to ACM TOCL.

[31] Vladimir Lifschitz. Foundations of Logic Programming. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, Stanford, 1996.

[32] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.

[33] Victor W. Marek and Jeffrey B. Remmel. On Logic Programs with Cardinality Constraints. In Salem Benferhat and Enrico Giunchiglia, editors, *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR'2002)*, pages 219–228, Toulouse, France, April 2002.

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

[35] J. Minker. Logic and Databases: a 20 Year Retrospective. In *Proceedings International Workshop on Logic in Databases (LID '96)*, number 1154 in LNCS, pages 3–57. Springer, 1996.

[36] Jack Minker. On Indefinite Data Bases and the Closed World Assumption. In D.W. Loveland, editor, *Proceedings 6^{th} Conference on Automated Deduction (CADE '82)*, number 138 in Lecture Notes in Computer Science, pages 292–308, New York, 1982. Springer.

[37] Jack Minker. Overview of Disjunctive Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 12:1–24, 1994.

[38] Ilkka Niemelä, Patrik Simons, and Tommi Syrjänen. Smodels: A System for Answer Set Programming. In Chitta Baral and Mirosław Truszczyński, editors, *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (NMR'2000)*, Breckenridge, Colorado, USA, April 2000.

[39] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[40] Nikolay Pelov. Non-monotone Semantics for Logic Programs with Aggregates. Available at `http://www.cs.kuleuven.ac.be/~pelov/papers/nma.ps.gz`., October 2002.

[41] T. Przymusinski. Stationary Semantics for Disjunctive Logic Programs and Deductive Databases. In *Proceedings of North American Conference on Logic Programming*, pages 40–62, 1990.

[42] T. Przymusinski. Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.

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

[44] Theodor C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers, Inc., 1988.

[45] K.A. Ross. The Well-Founded Semantics for Disjunctive Logic Programs. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 385–402. Elsevier Science Publishers B. V., 1990.

[46] C. Sakama. Possible Model Semantics for Disjunctive Databases. In *Proceedings First Intl. Conf. on Deductive and Object-Oriented Databases (DOOD-89)*, pages 369–383, Kyoto, Japan, 1989. North-Holland.

[47] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, June 2002.

[48] B. Wolfinger, editor. Workshop: *Disjunctive Logic Programming and Disjunctive Databases*, Berlin, August 1994. German Society for Computer Science (GI), Springer. 13^{th} IFIP World Computer Congress, Hamburg, Germany.