The DLV Project: A Tour from Theory and Research to Applications and Market*

Nicola Leone and Wolfgang Faber

Department of Mathematics, University of Calabria, 87036 Rende (CS), Italy
{leone,faber}@mat.unical.it

Abstract. DLV is one of the most succesful and widely used ASP systems. It is based on stable model semantics, and supports a powerful language extending Disjunctive Logic Programming with many expressive constructs, including aggregates, strong and weak constraints, functions, lists, and sets. In this paper, we describe the long tour from basic research on languages and semantics, studies on algorithms and complexity, design and implementation of prototypes, up to the realization of a powerful and efficient system, which won the last ASP competition, is employed in industrial applications, and is even ready for marketing and commercial distribution. We report on the experience we got in more than twelve years of work in the DLV project, focusing on most recent developments, industrial applications, trends, and market perspectives.

1 Introduction

Disjunctive Logic Programming [1] under the stable model semantics [2,3] (DLP, ASP)[1] is a powerful formalism for Knowledge Representation and Reasoning. 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. Disjunctive logic programs under stable model semantics are very expressive: 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 [4]. The high knowledge modeling power of DLP has implied a renewed interest in this formalism in the recent years, due to the need for representing and manipulating complex knowledge, arising in Artificial Intelligence and in other emerging areas, like Knowledge Management and Information Integration.

In this paper, we overview the DLV project, which has been active for more than twelve years, and has led to the development of the DLV system – the state-of-the-art implementation of disjunctive logic programming. DLV is widely used by researchers all over the world, and it is competitive, also from the viewpoint of efficiency, with the most advanced ASP systems. Indeed, at the First Answer Set Programming System Competition [5][2], DLV won in the Disjunctive Logic Programming category. And

* Supported by M.I.U.R. within projects "Potenziamento e Applicazioni della Programmazione Logica Disgiuntiva" and "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione."

[1] ASP stands for Answer Set Programming, with answer-set being an alternative name for stable-model, which is more frequently used than the latter today.

[2] See also http://asparagus.cs.uni-potsdam.de/contest/

M. Garcia de la Banda and E. Pontelli (Eds.): ICLP 2008, LNCS 5366, pp. 53–68, 2008.

DLV finished first also in the general category MGS (Modeling, Grounding, Solving — also called royal competition, which is open to all ASP systems). Importantly, DLV is profitably employed in many real-word applications, and has stimulated quite some interest also in industry (see Section 7). The key reasons for the success of DLV can be summarized as follows:

Advanced knowledge modeling capabilities. DLV provides support for declarative problem solving in several respects:

- High expressiveness of its knowledge representation language, extending disjunctive logic programming with many expressive constructs, including aggregates, weak constraints, functions, lists, and sets. These constructs not only increase the expressiveness of the language; but they also improve its knowledge modeling power, enhancing the usability in real-world contexts.
- Full declarativeness: ordering of rules and subgoals is immaterial, the computation is sound and complete, and its termination is always guaranteed.
- A number of front-ends for dealing with specific AI applications [6,7,8,9], information extraction [10], Ontology Representation and Reasoning [11,12].

Solid Implementation. Much effort has been spent on sophisticated algorithms and techniques for improving the performance, including

- Database optimization techniques: indexing, join ordering methods [13], Magic Sets [14,15].
- Search optimization techniques: heuristics [16,17,18], backjumping techniques [19,20], pruning operators [21].

DLV is able to solve complex problems and can deal with data-intensive applications by evaluating the program in mass-memory on a language subset [22,23].

Interoperability. A number of powerful mechanisms have been implemented to allow DLV to interact with external systems:

- Interoperability with Semantic Web reasoners: DLVHEX [24].
- Interoperability with relational DBMSs: ODBC interface [25,22].
- Calling external (C++) functions from DLV programs: DLVEX [26].
- Calling DLV from Java programs: Java Wrapper [27].

In the following, we report on the long tour which has led to the DLV system implementation, focusing on most recent developments, industrial applications, trends, and market perspectives.

2 Ancestry

Probably the earliest relevant roots of DLV are to be found in the 1950ies, when McCarthy proposed the use of *logical formulas* as a basis for a knowledge representation language [28,29]. It was soon realized, however, that classical logic is not always adequate to model commonsense reasoning [30]. As an alternative, it has been

suggested to represent commonsense reasoning using logical languages with nonmonotonic consequence relations, which allow new knowledge to invalidate some of the previous conclusions. This observation has led to the development and investigation of new logical formalisms, nonmonotonic logics, on which nonmonotonic logic programming has been based.

In the 1980ies, Minker proposed Disjunctive Logic Programming (DLP) [1], which allows for disjunctions instead of just atoms in rule heads, yielding (in general) a more expressive language. Early methods for implementations have been proposed already in the book by Lobo, Minker, and Rajasekar [31]. In the early 1990ies nonmonotonic and disjunctive logic programming have been succesfully merged in the semantic proposals by Gelfond and Lifschitz [3] and Przymusinski [32], called Stable Model Semantics, and yielding what is today known as Answer Set Programming (ASP),[3] Stable Logic Programming, ASP-Prolog, or simply Disjunctive Logic Programming (DLP).

After a few early attempts on implementing DLP [33,34,35], the foundation of what would become the DLV system was laid in the seminal works [36] and [37]. These articles essentially contain an abstract description of the basic DLV algorithm.

3 Implementing the Core System

The core system of DLV works on a set of disjunctive rules, i.e., clauses of the form

$$a_1 \,\mathtt{v}\, \cdots \,\mathtt{v}\, a_n \;\mathtt{:-}\; b_1, \cdots, b_k, \mathrm{not}\; b_{k+1}, \cdots, \mathrm{not}\; b_m$$

where atoms $a_1, \ldots, a_n, b_1, \ldots, b_m$ may contain variables and each of n, k, m may be 0. If $n = 0$, then the clause is referred to as an integrity constraint, as the empty head acts like falsity. If $n = 1$ and $k = m = 0$, the rule is referred to as a fact, and for facts `:-` is usually omitted. The intuitive reading of such a rule is "If all $b_1, \ldots, b_k$ are true and none of $b_{k+1}, \ldots, b_m$ is true, then at least one atom in $a_1, \ldots, a_n$ must be true." Additionally, there is a stability criterion [2,3], which also implies minimality of truth.

Disjunctive logic programming is strictly more expressive that disjunction-free logic programming, it can represent some problems which cannot be encoded in OR-free programs, and cannot be translated even to SAT in polynomial time. We next show an example of a problem, called *strategic companies*, where disjunction is strictly needed.

Example 1. Suppose there is a collection $C = \{c_1, \ldots, c_m\}$ of companies c_i owned by a holding, a set $G = \{g_1, \ldots, g_n\}$ of goods, and for each c_i we have a set $G_i \subseteq G$ of goods produced by c_i and a set $O_i \subseteq C$ of companies controlling (owning) c_i. O_i is referred to as the *controlling set* of c_i. This control can be thought of as a majority in shares; companies not in C, which we do not model here, might have shares in companies as well. Note that, in general, a company might have more than one controlling set. Let the holding produce all goods in G, i.e. $G = \bigcup_{c_i \in C} G_i$.

A subset of the companies $C' \subseteq C$ is a *production-preserving* set if the following conditions hold: (1) The companies in C' produce all goods in G, i.e., $\bigcup_{c_i \in C'} G_i = G$.

[3] Stable Models are also called Answer Sets, and we will often use the latter name which is more frequently used today.

(2) The companies in C' are closed under the controlling relation, i.e. if $O_i \subseteq C'$ for some $i = 1, \ldots, m$ then $c_i \in C'$ must hold.

A subset-minimal set C', which is *production-preserving*, is called a *strategic set*. A company $c_i \in C$ is called *strategic*, if it belongs to some strategic set of C.

In the following, we adopt the setting from [38] where each product is produced by at most two companies (for each $g \in G$ $|\{c_i \mid g \in G_i\}| \leq 2$) and each company is jointly controlled by at most three other companies, i.e. $|O_i| \leq 3$ for $i = 1, \ldots, m$ (in this case, the problem is still Σ_2^P-hard). For a given instance of STRATCOMP, the program will contain the following facts:

- $company(c)$ for each $c \in C$,
- $prod\_by(g, c_j, c_k)$, if $\{c_i \mid g \in G_i\} = \{c_j, c_k\}$, where c_j and c_k may possibly coincide,
- $contr\_by(c_i, c_k, c_m, c_n)$, if $c_i \in C$ and $O_i = \{c_k, c_m, c_n\}$, where c_k, c_m, and c_n are not necessarily distinct.

Given this instance representation, the problem itself can be represented by the following two rules:

$$s1 : strat(Y) \vee strat(Z) \text{ :- } prod\_by(X, Y, Z).$$
$$s2 : strat(W) \text{ :- } contr\_by(W, X, Y, Z), strat(X), strat(Y), strat(Z)$$

Here $strat(X)$ means that company X is a strategic company.

DLV today can solve instances with thousands of companies in reasonable time.

The main tasks for computing a DLP program in a (by now) typical architecture are eliminating variables by instantiation (*grounding*), creating candidate answer sets (*generation*), and finally checking their stability (*checking*). It is worthwhile noting that, due to the higher expressiveness of DLP, the (stability) *checking* is a co-NP-complete task for disjunctive programs, while it is polynomially doable for OR-free programs.

In November 1996 the DLV project started in Vienna, its goal being the production of a performant system for computing answer sets of disjunctive logic programs. A working system was produced fairly quickly, and the first description of the system was presented at LPNMR 1997 [39]. The basic architecture of the system as presented in that paper stands until today more or less unchanged. The paper also introduced the grounding module, which proved to be a strong component of the system. Along with the basic model generator, it also described the model checker (a key module which is not needed for dealing with nondisjunctive programs) and various forms of dependency graphs.

The following major publication about the system was at KR 1998 [40], in which the newly created front-ends (brave and cautious reasoning, various forms of diagnostic reasoning, and a subset of the then-unpublished SQL-3 (later SQL98) language (see also Section 4), which allows for recursion in SQL. Another main focus of this work were the benchmarks. DLV was compared to two of the most competitive systems of the era, Smodels [41] (as yet without Lparse) and DeReS [42], and found to be competitive.

The computational core modules of DLV continued to be improved. A major step was the move to a more goal-oriented computation, by introducing a new truthvalue

or atom class named "must-be-true" [43] together with a suitable heuristic. These features proved to boost the system's performance on many benchmarks. In fact, work on tuning the heuristics has been continued ever since, giving rise to a number of significant improvements [16,44,45]. Other enhancements of the model generator were the comparison of various pruning operators [46,21] employed during model construction, which also yields considerable performance gains on certain kinds of problem.

Also DLV's model checker has been improved by introducing a new, lightweight technique which permits the use of a SAT solver to decide model stability [47]. It has been shown that the introduction of this technique significantly improves performance on the hardest (Σ_2^P-complete) problems that DLV can handle in a uniform way.

The grounding module is a very important part of DLV, as on the one hand it solves a difficult problem and on the other hand it should output a program that is as small as possible, as the efficiency of all subsequent computations will in general depend on this size. Thus, grounding optimizations are very important and often have a profound impact on the overall system performance, cf. [48,13,20,17]. The enhancement of grounding by "porting" optimization techniques from relational databases to DLP, has been one of the most effective improvements of DLV for real-world applications.

4 Language Extensions and Their Optimization

Early on, extensions of the basic language were a main focus of DLV. The first of these was the introduction of support for brave and cautious query answering, first described in [49]. In nonmonotonic reasoning, these are the two major modes for answering queries. In DLV, a program with a query is transformed into a program the structure and meaning of which depends on the reasoning mode. Answer sets are then computed for the transformed program: For brave reasoning, each answer set supports the truth of the query, while for cautious reasoning, an answer set is a witness for the falsity of the query.

Example 2. In order to check whether a company c is strategic in Example 1, one can write a query $strat(c)?$. Brave reasoning on this query decides whether c is strategic, while cautious reasoning decides whether c is contained in each strategic set.

For query evaluation, an adaption of the Magic Sets method to (fragments of) the DLV language has been introduced as an optimization [15,14]. The basic idea is to make the process more query oriented, and consider only a fragment of the program which is sufficient to answer the query. In addition, if constants are present in the query, this optimization attempts to minimize also the rule instantiations to those that are necessary to answer the query correctly.

The introduction of weak constraints [50,51] was the next major language extension. A weak constraint is a construct of the form

$$:\sim\ b_1, \cdots, b_k, \text{not } b_{k+1}, \cdots, \text{not } b_m.[w : l]$$

where for $m \geq k \geq 0$, $b_1, \ldots, b_m$ are atoms, while w (the *weight*) and l (the *level*) are positive integer constants or variables occuring in $b_1, \ldots, b_m$. For convenience, w

and/or l might be omitted and are set to 1 in this case. The idea is that weak constraints should preferably be satisfied, with the weight and level specifying a penalty in case a weak constraint is not satisfied. Basically, for each answer set we can associate a vector of weights, which are the sum of weights of unsatisfied weak constraints of a specific level. Optimal answer sets are then selected by first choosing those answer sets having the least weight for the highest level, among these those having the least weight for the next highest level and so on (that is, the optimum of a lexicographical ordering).

Example 3. For instance, if one wants to avoid scenarios in which company c is contained in a strategic set (and thus be bound to sold), we may add a weak constraint

$$:\sim strat(c).\ [1:1]$$

in this way, if strategic sets exist which do not contain c, then only those will be computed. However, this is a preference criterion: if there exists no one missing c, then the other answer sets will be anyway computed.

Having weak constraints actually increases the expressiveness of the language and incurred some fairly crucial modifications of the core system. For instance, the model generator potentially is activated twice in the presence of weak constraints: Once for determining the optimal value of answer sets and a second time for enumerating the optimal answer sets.

Especially with the advent of data-intensive applications, it became clear that some interface to databases is necessary, as extracting data from a database and putting it into a temporary text file is not a very practical option. After initial trials with proprietary interfaces, eventually an ODBC interface has been provided, which abstracts from the actual database used, and allows for both importing input data from and exporting answer set data to an external database.

A major language extension was the introduction of aggregates [52]. Aggregate atoms consist of an aggregation function (currently one of cardinality, sum, product, maximum, minimum), which is evaluated over a multiset of terms, which are determined by the truthvalues of standard (non-aggregate) atoms. The syntax is

$$L \prec_1 \mathbf{F}\{\mathit{Vars} : \mathit{Conj}\} \prec_2 U$$

where $\mathbf{F}$ is a function $\#\texttt{count}$, $\#\texttt{min}$, $\#\texttt{max}$, $\#\texttt{sum}$, $\#\texttt{times}$, $\prec_1, \prec_2 \in \{=, <, \leq, >, \geq\}$, and L and U, the guards, are integers or variables.

Intuitively, a symbolic set $\{X, Y : a(X,Y), \text{not } p(Y)\}$ stands for the set of pairs (X,Y) making the conjunction $a(X,Y), \text{not } p(Y)$ true, i.e., $S = \{(X,Y) \mid \exists Y \textit{ such that } a(X,Y) \wedge \text{not } p(Y) \textit{ is true}\}$. When evaluating an aggregate function over it, the projection on the first elements of the pairs is considered, which yields a multiset in general. The value yielded by the function application is compared against the guards in order to determine the truth value of the aggregate. DLV comes with full support for non-recursive aggregates, as described in [52]. To this end, specialized data structures were introduced, and the model generation algorithm was significantly enhanced in order to deal with aggregates.

In presence of recursion through aggregates, special care is needed for defining the semantics of aggregates.

Example 4. Consider $a(1) \texttt{ :- } \#\texttt{count}\{X{:}a(X)\} < 1.$
we see that in this example $a(1)$ can be true only if $a(1)$ is false. Therefore, any answer set containing $a(1)$ should not include $a(1)$, and any answer set not containing $a(1)$ should include $a(1)$, which are both infeasible conditions and therefore no answer should exist for this program.

However, looking at $a(1) \texttt{ :- } \#\texttt{count}\{X{:}a(X)\} > 0.$
intuitively, $a(1)$ can become true only if $a(1)$ is true, which would thus be a self-support for $a(1)$. One would expect that in any answer set $a(1)$ is false.

In a way, the first program behaves just like $a(1) \texttt{ :- not } a(1).$ while the second one is like $a(1) \texttt{ :- } a(1).$ Thus, "easy" approaches treating aggregate atoms like negative atoms are bound to give incorrect results on programs such as the second.

In [53,54] a semantics has been presented, which deals with these issues in a simple, but effective way. Later, in [55,56], characterizations of this semantics using an adapted version of unfounded sets has been presented, which paved the way for a reasonable implementation for recursive aggregates. Currently, a special version of DLV exists, which supports an ample class of programs with recursive aggregates under this semantics. This will eventually be integrated in the main distribution of DLV.

The latest extension of DLV language is the addition of functions, lists, and sets, along with a rich library of built-in functions for their manipulation [57]. This is a very relevant extension, which lifts up the expressive power of the language allowing to encode any computable function. Even if the integration in the main distribution of DLV is under development, this extension is already spread and succesfully used in many universities and research institutes.[4]

5 Frontends, Backends and Research-Applications

DLV has been succesfully integrated as a computational engine in systems which use it as an oracle, usually acting as frontends and/or backends to DLV. Also the implementation of brave and cautious query answering described in Section 4 can be viewed as such a frontend, but since it seamlessly integrates into the language we have described it as a language extension.

The first major frontend was the diagnosis frontend [6], which is now integrated into the DLV distribution. It supports various modes of abductive- and consistency-based diagnosis by transforming the input into a DLV program and extracting the diagnoses of the answer sets. Later, also diagnosis with penalization [9] has been studied and implemented using DLV.

The second frontend which became included in the DLV distribution supported object programs which can be linked via inheritance constructs, as described in [58]. Also this could be viewed as a language extension by considering programs not in any object as belonging to a special, isolated object. Also in this case the input is transformed into a standard DLV program and the resulting answer sets are cleaned of the intermediate symbols introduced by the translation.

[4] We refrain from providing further details, since the paper describing the extension of DLV with functions is reported in this book.

The last major frontend to be included into the DLV distribution was the support for finding plans for domains formulated in the action language $\mathcal{K}$ [59,7,60]. In this case, the interaction with DLV is somewhat more complex, and also the extraction of plans from answer sets is slightly more involved than in the frontends discussed so far.

There are several other systems which wrap around DLV; a few of these can also use other ASP systems in place of DLV.

There are actually two such systems for ASP with preferences, where the preferences are expressed between rules. The system `plp` [61] transforms these programs into a standard ASP program and extracts the preferred answer sets from the answer sets of the transformed program. A different approach has been presented in [62], which uses a metainterpretation technique. In this context, this means that the propositional atoms of the preference programs become terms in the transformed program, where the extensional database defines the program structure and an intensional fixed part characterizes the semantics.

The system `nlp` is an implementation for computing answer sets for programs with nested expressions, which relax the structural requirements for connectors occurring in rules [63]. Also here the program with nested expressions is transformed, introducing several intermediate predicates on the way, which are finally filtered from the output.

The system A-POL provides a solver for programs with partial order constructs by transforming them to standard DLV programs [64].

A major endeavor and interdisciplinary success has been the coupling between Answer Set Programming and Description Logic. System `NLP-DL` [65,66] uses DLV on its ASP side. It turned out that for certain tasks DLV can perform much better than Description Logic systems in this sort of coupling.

DLV has also been used inside a system for strong equivalence testing and associated program simplification [67]. Also in this case, it is used as a backend for deciding whether some rule is redundant or can be simplified.

Two systems have been devised which work on action descriptions in the language $\mathcal{K}$ and on plans, one for monitoring plan execution (KMonitor) [68], and another one which diagnoses plan execution failures (KDiagnose) [69]. Another system implements query answering on action descriptions (AD-Query) [70]. All of these systems use DLV for solving various computational tasks arising during their execution.

Recently, a system for Answer Set Optimization [71] has been presented, which handles programs with preferences expressed among atoms (rather than rules as for `plp` described earlier). In this case, DLV is used for producing candidate answer sets, which are then tested for optimality by other software.

Finally, we mention `spock`, a system for debugging ASP programs [72,73], which may be configured to use DLV as its computational core.

6 Spin-Off Projects

Several projects have spun off DLV over the time. A fairly early one was the DLV Java wrapper, described in [27]. Since industrial applications (cf. Section 7) are frequently developed in a Java environment, some means had to be found to interact with DLV from Java. The DLV Java wrapper project provides interfaces, which are in some way

inspired by ODBC or JDBC. They allow for creating DLV programs, passing them to DLV, invoking DLV and getting back and analyzing the answer sets produced. This software has been succesfully applied in industrial settings described in Section 7.

DLVT [74] is a project which enhances DLV by so-called templates. These templates can be viewed as abstractions for programs, which can then be used by instantiating them for a particular setting. The semantics for these constructs is defined by expanding the respective templates, and allows for modular programming in DLV.

Again experiences with industrial applications motivated the creation of DLVEX [26]. The main observation was that it is often necessary to delegate certain computational tasks in programs to functions evaluated outside of DLV's proper language. This requirement arises because ASP is not well-suited for certain tasks such as string-handling, various numeric computations and similar features. Moreover it allows for easy language extensions, the idea being to define a suitable semantics for a generic extension, the semantics for a particular extension then being automatically provided by the generic definition. It can also be seen as an easy means for providing new data types and associated operations. Several libraries have already been provided for DLVEX, including numeric operations, string handling, manipulation of biological data, and more. It is planned that these features will be merged into standard DLV in the near future.

A system which is similar in spirit is dlvhex [24], which also allows for external calls. However, while DLVEX is situated at the grounding level, in dlvhex these external predicates may be evaluated at an arbitrary stage of the computation. For instance, the truthvalue of an atom may be determined by the answer that a Description Logic reasoner provides for a query, where the state of the Description Logic reasoner itself may be determined by the truthvalue of atoms occurring in the dlvhex program. This project has received a lot of attention by the Semantic Web community.

A spin-off of DLV which seems very attractive for real-world applications, where large amount of data are to be dealt with, is DLV^{DB}. The basic idea undelying DLV^{DB} [23,22] is to create a close interaction between DLV and databases, delegating some computational tasks to the database engine. The motivation is that if some data is obtained from a database anyway, it might be more efficient to reason on it directly where it resides; this becomes particularly important if the data size does not fit main memory (which is a typical case in real world applications). Moreover, if input data is spread over different databases, DLV^{DB} provides suitable constructs to reason on them transparently. Finally, as many database engines give the possibility to attach stored function calls to queries, DLV^{DB} allows for attaching such function calls to declarative programs, allowing for solving procedural sub-tasks directly on the database.

Essentially forming a language extension, a system for supporting parametric connectives [75] in the language of DLV has been implemented, which should eventually be integrated into regular DLV. Parametric connectives allow for dynamically creating disjunctions and conjunctions during grounding. This is especially useful if one does not know in advance which or how many options there will be in a particular instance of a program. For instance, for the well-known 3-colorability problem it is known in advance that there are exactly three colors available, and one can exploit this knowledge for writing a concise program that includes a disjunction involving the three colors. When one is interested in n-colorability instead, one cannot write a similar disjunction,

as it depends on the problem instance how many colors will be available. With parametric disjunctions, this can be done as the disjunction will be dynamically created based on the extension of some predicate. The following program encodes n-colorability by means of parametric disjunction:

$$\bigvee\{col(X,C) : color(C)\} \text{ :- } vertex(X)$$
$$\text{ :- } col(X,C), col(Y,C), edge(X,Y), X \neq Y$$

A project for improving runtimes of basic DLV is to endow the model generator with a reason calculus and backjumping [19]. These techniques are quite well-known in SAT solving, and in this project those methods have been considerably adapted to suit the ASP world, and the DLV system in particular. It has been shown that these techniques are beneficial with respect to runtime, and they will eventually be included in mainline DLV.

Based on the reason calculus discussed above, another side project has been established that defines VSIDS-like heuristics for ASP, and DLV in particular [76]. This kind of heuristics tries to look back on the computation and guide choices based on previous experiences. Standard DLV does the opposite, it looks ahead by performing a tentative computational step and analyzing the output. Eventually it is planned to integrate also this kind of heuristics into DLV.

A recent effort to improve the scalability of DLV has been the parallelization of DLV's grounding module [77]. The original implementation was sequential, but conceptually the grounding procedure has potential for parallel processing. The implementation is done having a multiprocessor machine with shared memory in mind.

7 Industry-Level Applications and Commerce

Unlike many other ASP systems, DLV has a history of applications on the industrial level. An important application area, in which DLV has been succesfully applied, is Information Integration. The European Commission funded a project on Information Integration, which produced a sophisticated and efficient data integration system, called INFOMIX, which uses DLV at its computational core [78]. The powerful mechanisms for database interoperability, together with magic sets [15,14] and other database optimization techniques [13,79], which are implemented in DLV, make DLV very well-suited for handling information integration tasks. And DLV (in INFOMIX) was succesfully employed in an advanced real-life application, in which data from various legacy databases and web sources must be integrated for the information system of the University of Rome "La Sapienza".

The DLV system has been experimented also with an application for Census Data Repair [80], in which errors in census data are identified and eventually repaired. This application includes a formalization of error models and hypothetical reasoning on possible repairs. DLV has been employed at CERN, the European Laboratory for Particle Physics, for an advanced deductive database application that involves complex knowledge manipulation on large-sized databases. The Polish company Rodan Systems S.A. has exploited DLV in a tool for the detection of price manipulations and unauthorized use of confidential information, which is used by the Polish Securities and Exchange

Commission. In the area of self-healing Web Services[5] the most recent extension of DLV with function symbols is succesfully exploited for implementing the computation of minimum cardinality diagnoses [81]. Function symbols are employed to replace existential quantification, which is needed to model the existence of values in case the semantics of Web Services is unknown, e.g., because of faulty behaviors.

Thanks to the high expressivity of the language and to its solid implementation DLV has been attractive for many other similar applications. However, the most valuable applications from a commercial viewpoint are those in the area of Knowledge Management, which have been realized by the company EXEURA s.r.l., with the support of the DLVSYSTEM s.r.l. (see below).

The experience gained in these real-world settings confirmed plans to promote DLV also commercially. To this end, the key people involved in DLV founded the company DLVSYSTEM s.r.l. in September 2005. This company is located in Calabria, Italy, and its main goal is to license DLV to interested partners in industry as well as to provide consultancy and support for its use in an industrial context.

The main licensee so far has been EXEURA, a spin-off company of the University of Calabria having a branch also in Chicago, which extensively uses DLV in its Knowledge Management (KM) products. Three main industrial prototypes of Exeura, currently in production, are strongly based on DLV: OntoDLV, Olex, and HiLeX.

OntoDLV is a system for ontology specification and reasoning [82,11]. The system supports a powerful ontology representation language, called OntoDLP, extending Disjunctive Logic Programming with all the main ontology features including classes, inheritance, relations, and axioms. OntoDLP is strongly typed, and includes also complex type constructors, like lists and sets. Importantly, OntoDLV supports powerful rule-based reasoning on ontologies, by incorporating the DLV system. The semantic peculiarities of DLP, like the Closed World Assumption (CWA) and the Unique Name Assumption (UNA), allow to overcome some limits of OWL, making OntoDLV very suitable for Enterprise Ontologies. It is worth noting that OntoDLV supports a powerful interoperability mechanism with OWL, allowing the user to retrieve information from OWL ontologies, and build rule-based reasoning on top of OWL ontologies. Moreover, through the exploitation of DLV^{DB}, OntoDLV is able to deal also with data-intensive applications, by working in mass-memory when main memory is not sufficient. The system is already used in a number of real-world applications including agent-based systems, information extraction, and text classification.

HiLeX [10] supports a semantic-aware approach to information extraction from unstructured data (i.e., documents in several formats, e.g., html, txt, doc, pdf, etc). In HiLeX information extraction is "Ontology driven", and exploits a domain description expressed through an OntoDLP ontology. A pre-processing phase transforms the input document in a set of logical facts, extraction patterns are rewritten into logical rules, and the whole process of information extraction amounts to answer set computation, which is carried out by the DLV system. The HiLex system has been succesfully applied for the extraction of clinical data (stored in flat text format in Italian language) from an Electronic Medical Record (EMR), and for the extraction of data from balance sheets.

[5] `http://wsdiamond.di.unito.it`

Olex is a rule-based system for text classification [83,84]. Roughly, given an ontology of the domain, Olex assigns each input document to the classes of the ontology which are relevant for it (by recognizing and analyzing the concepts treated in the document). For instance, Olex can automatically classify ANSA news according with their contents (Sport, Economy, Politics, etc.). Olex classifiers are learned automatically in a "training phase", and expressed by DLP rules. The document classification process amounts to answer set computation, which is performed by the DLV system. Olex has been succesfully applied in a number of real world applications in various industries including health-care, tourism, and insurance.

Exeura is currently concentrating its efforts on the implementation of a data-mining suite, where DLV will be employed for reasoning on top of the results of data mining.

References

1. Minker, J.: On Indefinite Data Bases and the Closed World Assumption. In: Loveland, D.W. (ed.) CADE 1982. LNCS, vol. 138, pp. 292–308. Springer, Heidelberg (1982)
2. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP 1988, pp. 1070–1080. MIT Press, Cambridge (1988)
3. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC 9, 365–385 (1991)
4. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS 22(3), 364–418 (1997)
5. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
6. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: The Diagnosis Frontend of the dlv System. AI Communications 12(1-2), 99–111 (1999)
7. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A Logic Programming Approach to Knowledge-State Planning, II: the DLV$^{\mathcal{K}}$ System. AI 144(1-2), 157–211 (2003)
8. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Answer Set Planning under Action Costs. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS, vol. 2424, pp. 186–197. Springer, Heidelberg (2002)
9. Perri, S., Scarcello, F., Leone, N.: Abductive Logic Programs with Penalization: Semantics, Complexity and Implementation. TPLP 5(1-2), 123–159 (2005)
10. Ruffolo, M., Manna, M., Gallucci, L., Leone, N., Saccà, D.: A Logic-Based Tool for Semantic Information Extraction. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 506–510. Springer, Heidelberg (2006)
11. Ricca, F., Leone, N.: Disjunctive Logic Programming with types and objects: The DLV^{+} System. Journal of Applied Logics 5(3), 545–573 (2007)
12. Ricca, F., Leone, N., De Bonis, V., Dell'Armi, T., Galizia, S., Grasso, G.: A DLP System with Object-Oriented Features. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 432–436. Springer, Heidelberg (2005)
13. Leone, N., Perri, S., Scarcello, F.: Improving ASP Instantiators by Join-Ordering Methods. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS, vol. 2173, pp. 280–294. Springer, Heidelberg (2001)
14. Faber, W., Greco, G., Leone, N.: Magic Sets and their Application to Data Integration. JCSS 73(4), 584–609 (2007)
15. Cumbo, C., Faber, W., Greco, G.: Enhancing the magic-set method for disjunctive datalog programs. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132. Springer, Heidelberg (2004)

16. Faber, W., Leone, N., Pfeifer, G.: Experimenting with Heuristics for Answer Set Programming. In: IJCAI 2001, pp. 635–640 (2001)
17. Perri, S., Scarcello, F., Catalano, G., Leone, N.: Enhancing DLV instantiator by backjumping techniques. AMAI 51(2-4), 195–228 (2007)
18. Faber, W., Leone, N., Ricca, F.: Heuristics for Hard ASP Programs. In: IJCAI 2005, pp. 1562–1563 (2005)
19. Ricca, F., Faber, W., Leone, N.: A Backjumping Technique for Disjunctive Logic Programming. AI Communications 19(2), 155–172 (2006)
20. Leone, N., Perri, S., Scarcello, F.: BackJumping Techniques for Rules Instantiation in the DLV System. In: NMR 2004, pp. 258–266 (2004)
21. Calimeri, F., Faber, W., Leone, N., Pfeifer, G.: Pruning Operators for Disjunctive Logic Programming Systems. Fundamenta Informaticae 71(2-3), 183–214 (2006)
22. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. TPLP 8, 129–165 (2008)
23. Terracina, G., De Francesco, E., Panetta, C., Leone, N.: Enhancing a DLP system for advanced database applications. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 119–134. Springer, Heidelberg (2008)
24. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: IJCAI 2005, Edinburgh, UK, pp. 90–96 (2005)
25. Leone, N., Lio, V., Terracina, G.: DLV^{DB}: Adding Efficient Data Management Features to ASP. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS, vol. 2923, pp. 341–345. Springer, Heidelberg (2003)
26. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. AMAI 50(3-4), 333–361 (2007)
27. Ricca, F.: The DLV Java Wrapper. In: ASP 2003, Messina, Italy, pp. 305–316 (2003), `http://CEUR-WS.org/Vol-78/`
28. McCarthy, J.: Programs with Common Sense. In: Proceedings of the Teddington Conference on the Mechanization of Thought Processes, pp. 75–91. Her Majesty's Stationery Office (1959)
29. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Machine Intelligence 4, pp. 463–502. Edinburgh University Press (1969) reprinted in [85]
30. Minsky, M.: A Framework for Representing Knowledge. In: The Psychology of Computer Vision, pp. 211–277. McGraw-Hill, New York (1975)
31. Lobo, J., Minker, J., Rajasekar, A.: Foundations of Disjunctive Logic Programming. The MIT Press, Cambridge (1992)
32. Przymusinski, T.C.: Stable Semantics for Disjunctive Programs. NGC 9, 401–424 (1991)
33. Subrahmanian, V., Nau, D., Vago, C.: WFS + Branch and Bound = Stable Models. IEEE TKDE 7(3), 362–377 (1995)
34. Seipel, D., Thöne, H.: DisLog – A System for Reasoning in Disjunctive Deductive Databases. In: DAISD 1994, Universitat Politecnica de Catalunya (UPC), pp. 325–343 (1994)
35. Pfeifer, G.: Disjunctive Datalog — An Implementation by Resolution. Master's thesis, TU Wien, Wien, Österreich (1996); Supported by Eiter, T.
36. Leone, N., Rullo, P., Scarcello, F.: Declarative and Fixpoint Characterizations of Disjunctive Stable Models. In: ILPS 1995, Portland, Oregon, pp. 399–413. MIT Press, Cambridge (1995)
37. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Information and Computation 135(2), 69–112 (1997)
38. Cadoli, M., Eiter, T., Gottlob, G.: Default Logic as a Query Language. IEEE TKDE 9(3), 448–463 (1997)

39. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A Deductive System for Nonmonotonic Reasoning. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 363–374. Springer, Heidelberg (1997)
40. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: The KR System dlv: Progress Report, Comparisons and Benchmarks. In: KR 1998, pp. 406–417 (1998)
41. Niemelä, I., Simons, P.: Efficient Implementation of the Well-founded and Stable Model Semantics. In: ICLP 1996, Bonn, Germany, pp. 289–303. MIT Press, Cambridge (1996)
42. Cholewiński, P., Marek, V.W., Truszczyński, M.: Default Reasoning System DeReS. In: KR 1996, Cambridge, Massachusetts, USA, pp. 518–528 (1996)
43. Faber, W., Leone, N., Pfeifer, G.: Pushing Goal Derivation in DLP Computations. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) LPNMR 1999. LNCS, vol. 1730, pp. 177–191. Springer, Heidelberg (1999)
44. Faber, W., Leone, N., Pfeifer, G.: Optimizing the Computation of Heuristics for Answer Set Programming Systems. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS, vol. 2173, pp. 288–301. Springer, Heidelberg (2001)
45. Faber, W., Leone, N., Ricca, F.: Solving Hard Problems for the Second Level of the Polynomial Hierarchy: Heuristics and Benchmarks. Intelligenza Artificiale 2(3), 21–28 (2005)
46. Calimeri, F., Faber, W., Leone, N., Pfeifer, G.: Pruning Operators for Answer Set Programming Systems. In: NMR 2002, pp. 200–209 (2002)
47. Koch, C., Leone, N., Pfeifer, G.: Enhancing Disjunctive Logic Programming Systems by SAT Checkers. AI 15(1-2), 177–212 (2003)
48. Faber, W., Leone, N., Mateis, C., Pfeifer, G.: Using Database Optimization Techniques for Nonmonotonic Reasoning. In: DDLP 1999, pp. 135–139. Prolog Association of Japan (1999)
49. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: Progress Report on the Disjunctive Deductive Database System dlv. In: Andreasen, T., Christiansen, H., Larsen, H.L. (eds.) FQAS 1998. LNCS, vol. 1495, pp. 148–163. Springer, Heidelberg (1998)
50. Buccafurri, F., Leone, N., Rullo, P.: Enhancing Disjunctive Datalog by Constraints. IEEE TKDE 12(5), 845–860 (2000)
51. Faber, W.: Disjunctive Datalog with Strong and Weak Constraints: Representational and Computational Issues. Master's thesis, TU Wien (1998)
52. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the dlv system. TPLP (accepted for publication, 2008)
53. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
54. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. AI (accepted for publication, 2008)
55. Calimeri, F., Faber, W., Leone, N., Perri, S.: Declarative and Computational Properties of Logic Programs with Aggregates. In: IJCAI 2005, pp. 406–411 (2005)
56. Faber, W.: Unfounded Sets for Disjunctive Logic Programs with Arbitrary Aggregates. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 40–52. Springer, Heidelberg (2005)
57. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable Functions in ASP: Theory and Implementation. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 407–424. Springer, Heidelberg (2008)
58. Buccafurri, F., Faber, W., Leone, N.: Disjunctive Logic Programs with Inheritance. TPLP 2(3) (2002)
59. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. ACM TOCL 5(2), 206–263 (2004)
60. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Answer Set Planning under Action Costs. JAIR 19, 25–71 (2003)

61. Delgrande, J.P., Schaub, T., Tompits, H.: A Framework for Compiling Preferences in Logic Programs. TPLP 3(2), 129–187 (2003)
62. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing Preferred and Weakly Preferred Answer Sets by Meta-Interpretation in Answer Set Programming. In: AAAI 2001 Spring Symposium on ASP, California, USA, pp. 45–52. AAAI Press, Menlo Park (2001)
63. Pearce, D., Sarsakov, V., Schaub, T., Tompits, H., Woltran, S.: A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs: Preliminary Report. In: NMR 2002 (2002)
64. Osorio, M., Corona, E.: The A-Pol system. In: Answer Set Programming (2003)
65. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with Description Logics for the Semantic Web. In: KR 2004, Whistler, Canada, pp. 141–151 (2004); Extended Report RR-1843-03-13, Institut für Informationssysteme, TU Wien (2003)
66. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Nonmonotonic description logic programs: Implementation and experiments. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS, vol. 3452, pp. 511–527. Springer, Heidelberg (2005)
67. Eiter, T., Traxler, P., Woltran, S.: An Implementation for Recognizing Rule Replacements in Non-ground Answer-Set Programs. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 477–480. Springer, Heidelberg (2006)
68. Eiter, T., Fink, M., Senko, J.: KMonitor - A Tool for Monitoring Plan Execution in Action Theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 416–421. Springer, Heidelberg (2005)
69. Eiter, T., Erdem, E., Faber, W., Senko, J.: A Logic-Based Approach to Finding Explanations for Discrepancies in Optimistic Plan Execution. Fundamenta Informaticae 79(1-2), 25–69 (2007)
70. Eiter, T., Fink, M., Senko, J.: A Tool for Answering Queries on Action Descriptions. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 473–476. Springer, Heidelberg (2006)
71. Caroprese, L., Trubitsyna, I., Zumpano, E.: Implementing prioritized reasoning in logic programming. In: ICEIS 2007, pp. 94–100 (2007)
72. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 31–43. Springer, Heidelberg (2007)
73. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A Meta-Programming Technique for Debugging Answer-Set Programs. In: AAAI 2008, pp. 448–453. AAAI Press, Menlo Park (2008)
74. Calimeri, F., Ianni, G., Ielpa, G., Pietramala, A., Santoro, M.C.: A system with template answer set programs. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 693–697. Springer, Heidelberg (2004)
75. Perri, S., Leone, N.: Parametric connectives in disjunctive logic programming. AI Communications 17(2), 63–74 (2004)
76. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in dlv: Implementation, evaluation and comparison to qbf solvers. Journal of Algorithms in Cognition, Informatics and Logics 63(1-3), 70–89 (2008)
77. Calimeri, F., Perri, S., Ricca, F.: Experimenting with Parallelism for the Instantiation of ASP Programs. Journal of Algorithms in Cognition, Informatics and Logics 63(1-3), 34–54 (2008)
78. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: SIGMOD 2005, Baltimore, Maryland, USA, pp. 915–917. ACM Press, New York (2005)

79. Calimeri, F., Citrigno, M., Cumbo, C., Faber, W., Leone, N., Perri, S., Pfeifer, G.: New dlv features for data integration. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 698–701. Springer, Heidelberg (2004)
80. Franconi, E., Palma, A.L., Leone, N., Perri, S., Scarcello, F.: Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS, vol. 2250, pp. 561–578. Springer, Heidelberg (2001)
81. Friedrich, G., Ivanchenko, V.: Diagnosis from first principles for workflow executions. Tech. Rep., http://proserver3-iwas.uni-klu.ac.at/download_area/Technical-Reports/technical_report_2008_02.pdf
82. Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., Leone, N.: OntoDLV: an ASP-based System for Enterprise Ontologies. Journal of Logic and Computation (Forthcoming)
83. Cumbo, C., Iiritano, S., Rullo, P.: Reasoning-Based Knowledge Extraction for Text Classification. In: Proceedings of Discovery Science, 7th International Conference, Padova, Italy, pp. 380–387 (2004)
84. Curia, R., Ettorre, M., Gallucci, L., Iiritano, S., Rullo, P.: Textual Document Pre-Processing and Feature Extraction in OLEX. In: Proceedings of Data Mining 2005, Skiathos, Greece (2005)
85. McCarthy, J.: Formalization of Common Sense, papers by John McCarthy edited by V. Lifschitz, Ablex (1990)