The DLV System*

Nicola Leone[1], Gerald Pfeifer[2], Wolfgang Faber[2], Francesco Calimeri[1], Tina Dell'Armi[1], Thomas Eiter[2], Georg Gottlob[2], Giovambattista Ianni[1], Giuseppe Ielpa[1], Christoph Koch[2], Simona Perri[1], and Axel Polleres[2]

[1] Department of Mathematics, University of Calabria
87030 Rende (CS), Italy
{leone,calimeri,dellarmi,ianni,ielpa,perri}@mat.unical.it
[2] Institut für Informationssysteme, TU Wien
A-1040 Wien, Austria
{pfeifer,gottlob,koch}@dbai.tuwien.ac.at
{faber,eiter,axel}@kr.tuwien.ac.at

1 Introduction

The development of the DLV system has started as a research project financed by FWF (the Austrian Science Funds) in 1996, and has evolved into an international collaboration over the years. Currently, the University of Calabria and TU Wien participate in the project, supported by a scientific-technological collaboration between Italy and Austria. At the time of writing, the latest version of the system has been released on April 12, 2002.

The system is based on disjunctive logic programming without function symbols under the consistent answer set semantics [5],has the following important features:

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

- High expressiveness in a formally precise sense (Σ_2^P), so any such problem can be uniformly solved by a fixed program over varying input.
- Declarative problem solving following a "Guess&Check" paradigm where a solution to a problem is guessed by one part of a program and then verified through another part of the program.
- A number of front-ends for dealing with specific AI applications.

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

- deductive database optimization techniques, and
- non-monotonic reasoning optimization techniques.

Database Interfaces The DLV system provides an experimental interface to a relational database management system (Oracle) and, by means of a special

* This work was supported by FWF (Austrian Science Funds) under the projects Z29-INF and P14781, by MURST under project COFIN-2000 "From Data to Information (D2I)," and by the European Commission under project IST-2002-33570.

query tool, also to an object-oriented database management system (Objectivity), which is useful for the integration of specific problem solvers developed in DLV into more complex systems. An ODBC interface is currently being developed.

For up-to-date information on the system and a full manual please refer to the URL `http://www.dlvsystem.com`, where you can also download binaries of the current release and various examples.

2 Languages

2.1 Kernel Language

The kernel language of DLV is disjunctive datalog extended with strong negation and weak constraints under the consistent answer set semantics [5].

Let $a_1, \cdots, a_n, b_1, \cdots, b_m$ be classical literals (atoms possibly preceded by the classical negation symbol "$-$") and $n \geq 0$, $m \geq k \geq 0$. A *(disjunctive) rule* r is a formula

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

A *strong constraint* is a rule with empty head ($n = 0$). A *weak constraint* is

$$\texttt{:}\sim \ b_1, \cdots, b_k, \texttt{ not } b_{k+1}, \cdots, \texttt{ not } b_m. \ [Weight : Level]$$

where both Weight and Level are positive integers. A *(disjunctive) program* $\mathcal{P}$ is a finite set of rules and constraints.

The semantics of these programs is provided in [1] as an extension of the classical answer set semantics given in [5].

2.2 Front-ends

In addition to its kernel language, DLV provides a number of application front-ends that show the suitability of our formalism for solving various problems from the areas of Artificial Intelligence, Knowledge Representation and (Deductive) Databases. In particular, currently front-ends for model-based diagnosis [2], SQL3, logic programming with inheritance, and knowledge based planning are available.

3 System Architecture

The architecture of DLV is illustrated in Figure 1. The general flow in this picture is top-down. The principal User Interface is command-line oriented, but also a Graphical User Interface (GUI) for the core systems and most front-ends is available. Subsequently, front-end transformations might be performed. Input data can be supplied by regular files, and also by Oracle or Objectivity databases. The DLV kernel (the shaded part in the figure) then produces answer sets one at a

time, and each time an answer set is found, "Filtering" is invoked, which performs post-processing (dependent on the active front-ends) and controls continuation or abortion of the computation.

The DLV kernel consists of three major components: The "Intelligent Grounding," "Model Generator," and "Model Checker" modules share a principal data structure, the "Ground Program". It is created by the Intelligent Grounding using differential (and other advanced) database techniques together with suitable data structures, and used by the Model Generator and the Model Checker. The Ground Program is guaranteed to have exactly the same answer sets as the original program. For some syntactically restricted classes of programs (e.g. stratified programs), the Intelligent Grounding module already computes the corresponding answer sets.

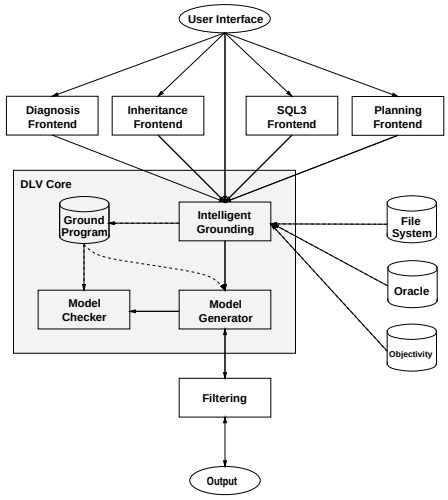

Fig. 1. The System Architecture of DLV

For harder problems, most of the computation is performed by the Model Generator and the Model Checker. Roughly, the former produces some "candidate" answer sets (models) [3, 4], the stability and minimality of which are subsequently verified by the latter.

The Model Checker (MC) verifies whether the model at hand is an answer set. This task is very hard in general, because checking the stability of a model is known to be co-NP-complete. However, MC exploits the fact that minimal model checking — the hardest part — can be efficiently performed for the relevant class of *head-cycle-free* (HCF) programs.

4 Usage

While DLV also comes with a graphical user interface (GUI), the computational engine and its front-ends are implemented as a command-line tool, controlled by

command-line options; it reads input from text files whose names are provided on the command-line, and any output is written to the standard output.

In its default mode, `DLV` simply computes all answer sets of its input; specific front-ends are invoked by means of command-line options starting with "`-F`".

For the diagnosis front-ends, hypotheses and observations have to be stored in files whose names carry the extensions `.hyp` and `.obs`, respectively. The type of diagnostic reasoning (abductive or consistency-based) and the desired minimality criterion (none, single error, subset minimality) is selected by specifying one of the options depicted in Table 1.

The SQL front-end is selected by means of the `-FS` command-line option. Input files whose names end with `.sql` are processed with an SQL parser.

Option	*Front-end/Mode*
-FD	generic abductive diagnosis
-FDsingle	single error abductive diagnosis
-FDmin	subset minimal abductive diagnosis
-FR	"Reiter's" (consistency-based) diagnosis
-FRsingle	single error "Reiter's" diagnosis
-FRmin	subset minimal "Reiter's" diagnosis
-FS	SQL3

Table 1. Command-line switches

5 Current Work

Our current activities concentrate on three issues: Language extensions, efficiency enhancements, and interoperability.

Concerning language extensions, we are working on aggregates and function symbols in the kernel system, but also on further front-ends, such as qualitative diagnosis. Regarding efficiency, we work on a generalization of magic set techniques, and we are investigating alternative heuristics. Finally, in order to support better integration of `DLV` in large systems, we work on providing an API to `DLV` and on an ODBC interface to relational databases.

References

1. F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE TKDE*, 12(5), 2000.
2. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The Diagnosis Frontend of the `dlv` System. *AI Communications*, 12(1–2):99–111, 1999.
3. W. Faber, N. Leone, and G. Pfeifer. Pushing Goal Derivation in DLP Computations. In *LPNMR'99*, pp. 177–191. Springer.
4. W. Faber, N. Leone, and G. Pfeifer. Experimenting with Heuristics for Answer Set Programming. In *IJCAI 2001*, pp. 635–640. Morgan Kaufmann.
5. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.