

# System Description: DLV with Aggregates<sup>\*</sup>

Tina Dell'Armi<sup>1</sup>, Wolfgang Faber<sup>2</sup>, Giuseppe Ielpa<sup>1</sup>,  
Nicola Leone<sup>1</sup>, Simona Perri<sup>1</sup>, and Gerald Pfeifer<sup>2</sup>

<sup>1</sup> Department of Mathematics, University of Calabria  
87030 Rende (CS), Italy

{dellarmi, ielpa, leone, perri}@unical.it

<sup>2</sup> Institut für Informationssysteme, TU Wien  
1040 Wien, Austria

faber@kr.tuwien.ac.at, pfeifer@dbai.tuwien.ac.at

## 1 Introduction

DLV is an efficient Answer Set Programming (ASP) system implementing the (consistent) answer set semantics [GL91] with various language extensions like weak constraints [BLR00], queries, and built-in predicates. A strong point of DLV is the high expressiveness of its language, which is able to express very complex problems, even problems which are hard for the complexity class  $\Delta_3^P$ . The language of DLV is therefore strictly more expressive than normal logic programming which cannot express any problem beyond NP. Another strong point of DLV is its robust and efficient implementation, which integrates algorithms and heuristics from the field of nonmonotonic reasoning with several optimization techniques from the field of deductive databases.

The expressiveness of the DLV language together with the robustness of the implementation make DLV well-suited for developing knowledge-based applications. Indeed, the DLV system is disseminated in academia, and presumably soon also in industry – the industrial exploitation of DLV in the emerging areas of Knowledge Management and Information Integration is the subject of two international projects funded by the European Commission, namely, INFOMIX (Boosting Information Integration, project IST-2002-33570) and ICONS (Intelligent Content Management System, project IST-2001-32429).

Despite this high expressiveness, encoding aggregations over a multiset of elements satisfying some conditions is often cumbersome and unintuitive, requiring ordering relations and recursive definitions using these orders. In this system description, we present an extension of DLV by aggregates `#count`, `#sum`, `#times`, `#min`, and `#max`, which greatly simplifies the encoding of those frequently occurring concepts. Moreover, in many cases also a computational benefit can be observed. The extension of the DLV language by aggregates has been proposed very recently [DFI<sup>+</sup>03a], and it has been further enhanced to also support unstratified aggregates [DFI<sup>+</sup>03b].

---

<sup>\*</sup> The work was partially supported by the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-32429 ICONS, and IST-2001-37004 WASP.

## 2 DLP<sup>A</sup>: Extending ASP by Aggregates

The language of DLV is disjunctive datalog under the consistent answer set semantics [GL91], commonly called *Answer Set Programming (ASP)*.

We assume familiarity with “classic” ASP and refer to [LPF<sup>+</sup>02] for a detailed description of the full language of DLV. In what follows, we refer to atoms, literals, etc. without aggregates as *standard atoms*, *standard literals*, and so forth.

A (DLP<sup>A</sup>) *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.<sup>3</sup> 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* among #count, #min, #max, #sum, #times.

Since symbolic or ground sets consist of tuples of constants, a projection on the first element of each tuple in the set is applied, yielding a multiset of constants on which the aggregate functions are effectively applied. Intuitively, #count evaluates to the number of elements, #min to the minimum element, and #max to the maximum element of the multiset. #sum evaluates to the sum of the numbers (0 in case of the empty set) and #times to the product of the numbers in the multiset (1 in case of the empty set). If the argument of an aggregate function does not belong to its domain, the aggregate evaluates to false and DLV issues a warning.

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. One of “ $Lg \prec_1$ ” and “ $\prec_2 Rg$ ” can be omitted; “ $0 \leq$ ” and “ $\leq +\infty$ ”, respectively, are assumed then.

As an example, consider a predicate  $person(Name, Age, Salary)$ . The aggregate atom  $19 < \#min\{A : person(N, A, S)\} \leq 30$  then evaluates to true iff the age of the youngest person is in the range [20..30].  $\#sum\{S : person(N, A, S), A > 30\} > 10000$ , on the other hand, evaluates to true if the sum of the salaries of all persons exceeds 10000; it evaluates to false else.

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

$$a_1 \vee \dots \vee a_n :- b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m.$$

A *strong constraint* is a rule with empty head ( $n = 0$ ). A *program*  $\mathcal{P}$  is a finite set of rules and constraints.

The semantics of programs without aggregates is provided in [BLR00] as an extension of the classical answer set semantics given in [GL91]. The detailed semantics of the full language including aggregates can be found in [DFI<sup>+</sup>03b].

<sup>3</sup> Intuitively, a symbolic set  $\{X : a(X, Y), p(Y)\}$  stands for the set of *X*-values making  $a(X, Y), p(Y)$  true. Note that *Conj* may contain negative literals.

### 3 Representing Problems in DLP<sup>A</sup>

DLP<sup>A</sup> is a superset of classic Answer Set Programming and thus allows to encode problems in a highly declarative fashion and solve problems of high computational complexity ( $\Delta_3^P$  for the full language supported by DLV, for details we refer to [LPF<sup>+</sup>02]).

In this section, we will focus on encodings employing the novel feature of aggregates, using the problem of Team Building, where 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  $not A_i$ .

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

Intuitively, the disjunctive rule “guesses” whether an employee is included in the team or not, while the five constraints correspond one-to-one to 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 (recall that 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  $S = \{(Sa, I) : Sa \text{ is the salary of employee } I \text{ in the team}\}$ . Then, the sum function is applied on the multiset of the first components  $Sa$  of the tuples  $\langle Sa, I \rangle$  in  $S$ .

### 4 Implementation and Usage

We have modified the implementation of DLV in order to deal with aggregates in the following way:

After parsing, each aggregate  $A$  is transformed such that both guards are present and both  $\prec_1$  and  $\prec_2$  are set to  $\leq$ . The conjunction  $Conj$  of the symbolic set of  $A$  is replaced by a single, new atom  $Aux$  and a rule  $Aux : -Conj$  is added to the program (the arguments of  $Aux$  being the distinct variables of  $Conj$ ).

The instantiator, which transforms input with variables to ground programs in a bottom-up manner, ensures that all rules which define predicates appearing in an aggregate have been processed before it considers rules containing that aggregate. Furthermore, it identifies repeated occurrences of the same aggregate function in different rules and replaces them by a single internal representation, thus reducing the size of the output.

Finally, we designed an extension of the Deterministic Consequences operator of the DLV system [FLP99] to perform both forward and backward inferences on aggregate atoms, resulting in an effective pruning of the search space during model generation. We then extended the Dowling and Gallier algorithm [DG84] to compute a fixpoint of this operator in linear time using a multi-linked data structure of pointers.

A more detailed description of these aspects as well as some benchmarking can be found in [DFI<sup>+</sup>03a].

While DLV with aggregates also offers a graphical user interface, the system per se is a command-line tool. It reads input from text files whose names are provided on the command-line, and can also obtain input from relation databases via an ODBC interface; any output is written to the standard output.

In its default mode, DLV computes all answer sets of its input. Command-line options like `-n=3` or `-filter=in` then can be used to request only a certain number of answer sets (3 in this case) or that only specific predicates (`in` in this case) are printed.

Detailed documentation, a full manual, and binaries of DLV with aggregates for various platforms are available at <http://www.dlvsystem.com>.

## References

- [BLR00] F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE TKDE*, 12(5), 2000.
- [DFI<sup>+</sup>03a] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *IJCAI 2003*, Acapulco, Mexico, August 2003. Morgan Kaufmann.
- [DFI<sup>+</sup>03b] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Semantics and Computation of Aggregate Functions in Disjunctive Logic Programming. Tech. Report INFSYS RR-1843-03-07, TU Wien, April 2003.
- [DG84] W. F. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *JLP*, 3:267–284, 1984.
- [FLP99] W. Faber, N. Leone, and G. Pfeifer. Pushing Goal Derivation in DLP Computations. *LPNMR'99*, pp. 177–191. Springer.
- [GL91] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
- [LPF<sup>+</sup>02] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. Tech. Report cs.AI/0211004, arXiv.org, November 2002. Submitted to ACM TOCL.