WASP WP3 Report: Language Extensions and Software Engineering for ASP

Ilkka Niemelä (Ed.)

WASP: European Working group on Answer Set Programming (funded by the European commission, FET "Future Emerging Technologies" initiative)
Work package 3: Language Extensions and Software Engineering for ASP
Participating nodes: HUT, UCY, UNILEIPZIG, VUB, BATH, UNIPOTSDAM, TUWIEN, URJC, KULEUVEN, UNICAL, UNIRC, UNINA

Abstract

The report gives an overview of the most important language extensions for ASP with emphasis on work done within the WASP research groups. We start from the more general purpose extensions and then move to application oriented extensions and to work on logical foundations. The report contains also a summary of available implementations of ASP language extensions and a recapitulation of interesting software engineering issues for ASP.

Contents

1 Introduction

The work on ASP systems started from implementations [145] of stable model semantics for normal logic programs. Most of the current ASP systems support a (Prolog style) logic programming language [137] based on *normal rules* of the form

$$A_0 \texttt{:-}\ A_1, \ldots, A_m, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_n.$$

However, the semantics and the basic computational services of ASP systems are different from those of Prolog. ASP programs typically have multiple (stable) models representing the different solutions to a problem to be solved and the task of an ASP solver is to search for models of a program. On the other hand, Prolog type systems search for proofs and different solutions are represented by different answer substitutions found by the proof procedure. This also leads to different approaches in problem representation. Moreover, Prolog systems are not strictly declarative but are, e.g., sensitive to the order of rules in the program and the order of subgoals in the rule bodies whereas ASP systems are based on declarative semantics. The same differences w.r.t. ASP hold for extensions of Prolog style logic programming such as abductive and constraint logic programming.

However, ASP systems support similar (deductive) database techniques as Prolog type approaches where logical variables and recursive definitions are combined. This is different from other similar frameworks for problem representation such as constraint satisfaction problems (CSPs) and integer programming which are basically propositional and do not allow effective combination of (deductive) database techniques and search.

While the work on ASP started with normal rules, fairly soon implementations extending the basic language started to emerge with disjunctive programs as the first key enhancement [127]. Below we summarize the most important extensions to ASP based on normal programs with emphasis on the work done within the WASP research groups. We start from the more general purpose extensions in Section 2, then move to application oriented extensions (Section 3) and finish with work on logical foundations (Section 4) which aims at widening the basis of ASP systems and which has led already to some interesting extensions. For each extension we provide brief motivation, comparisons, references to relevant literature and links to available software.

2 General Extensions

2.1 Disjunctions

Disjunctive logic programs are logic programs where disjunction is allowed in the heads of rules. They have first been studied by Minker [142] in the context of deductive databases, and are nowadays widely recognized as a valuable tool for knowledge representation and commonsense reasoning [7, 138, 76, 95, 131, 143, 9]. One of the attractions of disjunctive logic programming (DLP) is its capability of allowing the natural modeling of incomplete knowledge [7, 138].

DLP has always been at the core of ASP. In fact, the term "Answer Set Programming" has been coined by Gelfond and Lifschitz [95] for disjunctive logic programs with explicit negation.

DLP is the natural tool for programming following the "Guess&Check" paradigm [76]. This methodology divides problems into a guessing part, which defines the search space, and a checking part, which tests whether a solution candidate is in fact a solution. In most cases, disjunctions are a natural choice for implementing the guessing part.

As an example which matches this scheme, let us consider the well-known *3-Colorability* problem.

3COL: Given a graph $G = (V, E)$ in the input, assign each node one of three colors (say, red, green, or blue) such that adjacent nodes always have different colors.

3-Colorability is a classical NP-complete problem. Assuming that the set of nodes V and the set of edges E are specified by means of predicates *node* (which is unary) and *edge* (binary), respectively, it can be encoded by the following "Guess&Check" program:

```
col(X,r) ∨ col(X,g) ∨ col(X,b) :-  node(X).  } Guess
:-  edge(X,Y), col(X,C), col(Y,C).           } Check
```

System development for DLP systems had begun very early, and the implementations available today are mature and competitive. The `DLV` system [127] (`http://www.dlvsystem.com`) is considered to be the state of the art implementation dedicated to DLP. A competitive alternative is the `GnT` system [112, 111] (`http://www.tcs.hut.fi/Software/gnt/`), which has been built on top of `Smodels` (`http://www.tcs.hut.fi/Software/smodels/`).

Recently, disjunctive logic programs have been extended by parametric connectives (OR and AND) [125]. These connectives allow for a compact representation of disjunctions and conjunctions of a set of atoms having a given property.

2.2 Nested Programs

The syntax of disjunctive programs has been further extended to programs with nested expressions, or *nested programs* for short [129], where the bodies and heads of rules may contain arbitrary Boolean formulas.

A rule r of a nested program therefore has the form

$$H(r) \text{:-} B(r),$$

where $B(r)$ and $H(r)$ (the *body* and *head* of r resp.) are formulas in conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. A *program*, Π, is a finite set of rules. Note that nested programs properly generalise disjunctive logic programs. To each rule r we can associate a corresponding propositional formula $\hat{r} = B(r) \rightarrow H(r)$ and, accordingly, to each program Π a corresponding set of formulas $\hat{\Pi} = \{\hat{r} \mid r \in \Pi\}$. We call expressions, rules, and programs *basic* iff they do not contain the operator $\neg$. An interpretation I is a *model* of a basic program Π if it is a model of the associated set $\hat{\Pi}$ of formulas.

Given an interpretation I and an (arbitrary) program Π, the *reduct*, Π^I, of Π with respect to I is the basic program obtained from Π by replacing every occurrence of an expression $\neg\psi$ in Π which is not in the scope of any other negation by $\bot$ if ψ is true under I, and by $\top$ otherwise. I is an *answer set* of Π iff it is a minimal model (with respect to set inclusion) of the reduct Π^I.

Consider a logic program Π whose only rule is given by

$$p \text{:-} (q \wedge r) \vee (\neg q \wedge \neg s). \tag{1}$$

To check whether a candidate $I = \{p\}$ is an answer set for Π, observe that

$$\Big((q \wedge r) \vee (\neg q \wedge \neg s)\Big)^I = (q \wedge r) \vee (\top \wedge \top),$$

and, therefore the reduct of (1) is given by $p \text{:-} \top$. Consequently, the only minimal model of this rule (and thus of Π^I) is $\{p\}$, so I is an answer set of Π. In fact, there are no other answer sets of Π.

As already mentioned, nested programs properly generalise disjunctive logic programs. One difference is that the so-called *anti-chain* property does not hold for nested logic programs, while each disjunctive logic program, Π satisfies this property, viz. for all answer sets I, J of Π, $I \subseteq J$ implies $I = J$. Let $\Pi = \{p \text{:-} \neg\neg p\}$. It is easy to see that Π violates this anti-chain property since it possesses the two answer sets, $I_1 = \emptyset$ and $I_2 = \{p\}$. For I_1, $\neg p$ is true under I_1 and we have $\Pi^{I_1} = p \text{:-} \bot$, which has $\emptyset$ as its minimal model. For I_2, $\neg p$ is false under I_2 and we have $\Pi^{I_2} = p \text{:-} \top$, which has $\{p\}$ as its

minimal model. Consequently, both I_1 and I_2 are answer sets of Π, and we have $I_1 \subseteq I_2$, but $I_1 \neq I_2$.

Nested programs provide a richer syntax for modelling knowledge and declaratively encoding problems. One important property is that so-called *weight constraints*, see Section 2.3, can be expressed via nested programs, [91]. Several concepts and methods from disjunctive logic programming have been extended to nested programs. For instance, acyclic and head-cycle free programs are examined in [136] and the splitting theorem is generalised to nested programs in [87].

Nested logic programs can be encoded in terms of quantified boolean formulas (QBFs) in linear time. Based on the resulting transformations, complexity results related to logic programs with nested expressions can be derived [149]. A polynomial reduction of nested programs to disjunctive programs, where new atoms are admitted into the language, is studied in [148]. These reductions have led to various implementations of nested programs, see [161, 148, 134, 135]. For information on current systems, see `nlp` (`http://www.cs.uni-potsdam.de/~torsten/nlp/`) and `NoMoRe` (`http://www.cs.uni-potsdam.de/~linke/nomore`).

2.3 Cardinality and Weight Constraints

There are classes of constraints that frequently occur in applications but are not straightforward to encode succinctly using normal rules. Developing compact and easy-to-understand treatments of such constraints is important (i) from the software engineering point of view for enhancing the development, updating and maintenance of ASP programs and (ii) from the implementation point of view for improving efficiency.

Important classes of such conditions are cardinality and weight constraints and optimization criteria. For example, in the *product configuration* domain typical constraints include conditions of the form

- a PC should have from one to four hard disks;
- sum of the disk space in the chosen hard disks should be at least 100GB;
- the cheapest set of hard disks should be selected.

Weight constraint rules [164, 165] are an extension to normal programs which allows compact encoding of such constraints. For example, the cardinality and weight constraint above can be expressed by the constraints

```
1 { hd1, hd2, hd3, ..., hdn } 4
100 [ hd1=23, hd2=25, hd3=102, ..., hdn=44 ]
```

where the first expression is satisfied in a model if at least one and at most 4 of the `hdi` atoms are true in the model and the second expression is satisfied in a model if the sum of weights (e.g. 23 for the atom `hd1`) of the atoms true in the model is at least 100.

General weight constraint rules are expressions of the form

$$C_0 \texttt{:-}\ C_1, \ldots C_n.$$

where each C_i is a weight constraint (with normal literals and cardinality constraints seen as special cases of weight constraints). The semantics [165] is a generalization of the stable model semantics for normal programs assigning each program (a set of weight constraints rules) a class of stable models. A stable model (a set of atoms) is not necessarily subset minimal but the semantics guarantees that each atom in a stable model is justified by the rules in the program. For example, a program

```
r.
p :- q.
q :- 2 {r,p,q}.
```

has a unique stable model {`r`} and, e.g., the set of atoms {`r`, `p`, `q`} is not a stable model for the program. Hence, weight and cardinality constraints are not treated straightforwardly as ordinary aggregates but are tightly integrated into the stable model semantics.

In order to handle optimization requirements the weight constraint rule language contains minimization and maximization statements to set optimization criteria. For example, the condition on selecting the cheapest set of hard disks can be expressed as

```
minimize [ hd1=100, hd2=115, hd3=95, ..., hdn=200 ].
```

Cardinality and weight constraints have proved to be very useful in developing applications. With them typical choices and constraints can be expressed in a compact and easily maintainable way and they are in wide use as can be seen from the WP5 report on model applications and proofs-of-concepts [89].

The `Smodels` system (`http://www.tcs.hut.fi/Software/smodels/`) implements cardinality and weight constraints as well as optimization. For details on the algorithms and implementation techniques, see, e.g., [164, 165]. The `Smodels` system also supports variables, conditional literals, built-in functions and limited use of function symbols [165, 168, 169].

ASP systems such as `Smodels` can be used as stand-alone systems which take a program as input and compute stable models for it. However, in many applications an ASP system is used as a (key) component integrated with other parts of the system. For this the `Smodels` system contains a number of APIs so that it can be embedded to a larger system as a software library.

2.4 Aggregates

There are some simple properties, often arising in real-world applications, which cannot be encoded in a simple and natural manner using ASP. Especially properties that require the use of arithmetic operators (like sum, count, or maximum) on a set of elements satisfying some conditions require rather cumbersome encodings (often requiring an "external" ordering relation over terms), if one is confined to classic ASP. The size of these programs is often astonishingly large, which in many cases has adverse effects on solving time.

Similar observations have also been made in related domains, notably database systems, which led to the definition of aggregate functions. Especially in database systems this concept is by now both theoretically and practically fully integrated. When ASP systems became used in real applications, it became apparent that aggregates are needed also here. First, cardinality and weight constraints, which are special cases of aggregates, have been introduced (see previous section). However, in general one might want to use also other aggregates (like minimum, maximum, or average), and it is not clear how to generalize the framework of cardinality and weight constraints to allow for arbitrary aggregates.

To overcome this deficiency, ASP has been extended by aggregate functions. There have been several attempts for defining a suitable semantics for aggregates [119, 97, 55, 152, 153, 90]. These semantic definitions typically agree in the non-recursive case, but the picture is less clear for recursion. The `DLV` system (`http://www.dlvsystem.com/`) contains an implementation of stratified aggregates.

Consider the following example application: A project team has to be built from a set of employees according to the following specifications:

1. The team consists of a certain number of employees.
2. At least a given number of different skills must be present in the team.
3. The sum of the salaries of the employees working in the team must not exceed the given budget.
4. The salary of each individual employee is within a specified limit.

Suppose that our employees are provided by a number of facts of the form `emp(EmpId,Sex,Skill,Salary)` and then 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 stated above by an aggregate atom, and enforce it by an integrity constraint.

```
in(I) v 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)} >= M.
:- budget(B), not #sum{Sa,I:emp(I,Sx,Sk,Sa),in(I)} <= B.
:- maxSal(M), not #max{Sa:emp(I,Sx,Sk,Sa),in(I)} <= M.
```

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. Thanks to the aggregates the translation of the specifications is straightforward.

In addition, partial stable semantics for logic programs with arbitrary aggregate relations has been defined in [152] and a translation to normal logic programs which preserves the semantics in [151].

2.5 Templates

Although ASP systems have been extended in many directions, they still miss features which may be helpful towards industrial applications, like capabilities to quickly introduce new predefined constructs or to deal with compound data structures and modules. The DLP^T system is an extension of ASP with template constructs. ASP systems developers are enabled to fast prototype, making new features quickly available to the community, and later to concentrate on efficient (long lasting) implementations. Template predicates allow to define intensional predicates by means of generic, reusable subprograms, easing coding and improving readability and compactness. For instance, a template program is like

```
#template max[p(1)](1) {
    exceeded(X) :- p(X),p(Y), Y > X.
    max(X) :- p(X), not exceeded(X). }
```

The statement above defines the predicate `max`, which computes the maximum value over the domain of a generic unary predicate `p`. A template definition may be instantiated as many times as necessary, through *template atoms* (or *template invocations*), like in `:-max[weight(*)](M),M>100.` Template definitions may be unified with a template atom in many ways. The above rule contains a *plain* invocation, while in `:-max[student(Sex,$,*)](M),M>25.` there is a *compound* one.

Semantics is given through a suitable mapping to usual ASP programs. Given a DLP^T program P, the *Explode* algorithm replaces each template atom t with a standard atom, referring to a fresh intensional predicate p_t. The subprogram d_t (which may have associated more than one template atom),

defining the predicate p_t, is computed according to the template definition $D(t)$. The final output of the algorithm is a standard ASP program P'. Answer sets of the originating program P are constructed, *by definition*, from answer sets of P'.

The work takes inspiration from similar approaches such as the Hilog language [37], and the comprehensive study about generalized quantifiers of [82].

The DLP^T language has been implemented on top of the DLV system [127], creating the DLT system. The current version is available on the web [69, 108, 109].

3 Application Oriented Extensions

3.1 Abduction

In the context of logic programming, abduction has been first proposed by Eshghi and Kowalski [88], Kakas and Mancarella [113] and, during the recent years, the interest in this subject has been growing rapidly [40, 122, 117, 72, 56, 160, 20, 114, 58, 133].

Unlike most of these previous works on abduction in the logic programming framework, in [154], *abduction with penalization* from logic programs has been proposed. This form of abductive reasoning, well studied in the setting of classical logics [75], has not been previously analyzed in logic programming and turns out to encode easily and in a natural way several relevant problems, belonging to different domains.

In [154], a formal model for abduction with penalization from logic programs has been defined. Roughly, a problem of abduction with penalization (*PAP*) consists of a logic program P, a set of hypotheses, a set of observations, and a function that assigns a penalty to each hypothesis. An admissible solution is a set of hypotheses such that all observations can be derived from P assuming that these hypotheses are true. Each solution is weighted by the sum of the penalties associated with its hypotheses. The optimal solutions are those with the minimum weight, which are considered more likely to occur, and thus are preferred over other solutions with higher penalties. As an example, consider the following diagnosis problem. Let $\mathcal{N}$ be the computer network represented by the set of facts

$$F = \{\texttt{connected(a,b)}, \texttt{connected(a,f)}, \texttt{connected(b,d)}, \texttt{connected(b,c)}, \texttt{connected(c,e)}, \texttt{connected(d,e)}, \texttt{connected(f,e)}\}.$$

and suppose that we are working on machine a (and we therefore know that machine a is online) of $\mathcal{N}$ but we observe machine e is not reachable

from a, even if we are aware that e is online. We would like to know which machines could be offline. However, we do not consider all the possible explanations, rather we are interested in explanations with the minimum number of offline machines. This problem is easily represented by a *PAP* problem where the set of hypotheses is $H = \{$ `offline(X)` | X is a machine in $\mathcal{N}\}$, the set of observations is $O = \{$ `not offline(a), not offline(e), not reaches(a,e)`$\}$, for each $h \in H$, $\gamma(h) = 1$, and the logic program P consists of the set F and of the rules,

```
reaches(X,X) :- node(X),not offline(X).
reaches(X,Z) :- reaches(X,Y), connected(Y,Z), not offline(Z).
```

This *PAP* problem has the unique optimal explanation $S = \{$ `offline(f), offline(b)`$\}$, that corresponds to the unique solution of our diagnosis problem with a minimum number of offline machines. Note that, in this example the penalty function γ assigns 1 to each hypothesis. However, minimizing the number of offline machines is not necessarily the best strategy. If the reliability of the machines is very different, one should take it into account. To this end, for each machine x in $\mathcal{N}$ the penalty function γ should assign the probability of x to be offline to the hypothesis `offline(x)`.

The proposed framework for abduction with penalization over logic programs has been implemented as a front-end for the `DLV` system. The implementation is based on an algorithm that translates an abduction problem with penalties into a logic program with weak constraints [32], which is then evaluated by `DLV`. This abductive system is available in the current release of the `DLV` system (`www.dlvsystem.com`), and can be freely retrieved for experiments.

This work is evidently related to previous studies on semantic and knowledge representation aspects of abduction over logic programs [113, 130, 114, 57, 133]. Another approach is given in [1], where abductive reasoning from ground logic programs is based on the well-founded semantics with explicit negation. In other proposals, the semantics is naturally associated to a particular optimality criterion, as for [110], where the authors consider prioritized programs under the preferred answer set semantics. A similar optimization criterion is proposed for the logic programs with consistency-restoring rules (cr-rules) described in [6]. Furthermore, this work is also related to previous work on abductive logic programming systems [171, 116].

3.2 Preferences

The predominant methodology in ASP uses a generate and test method which works as follows:

1. generate answer sets which represent potential solutions,

2. specify conditions which destroy those answer sets which do not correspond to actual solutions.

For instance, in graph colouring arbitrary assignments of colours to nodes constitute potential solutions. If we add the condition that an answer set is to be disregarded if it assigns the same colour to neighbouring nodes, then the remaining answer sets will be the solutions to our original graph colouring problem.

This methodology allows the programmer to distinguish between solutions and non-solutions. However, in many realistic applications the possibility to make more fine grained distinctions is required, in particular distinctions between more and less preferred solutions. For a discussion of various types of applications where preferences play an important role (like abduction and diagnosis, revision and inconsistency handling) see [23].

For this reason, there has been a substantial amount of work on extending logic programs with preferences. The different approaches can be categorized according to the following criteria:

- representation of preference: numerical vs. qualitative
- object of preference: rules vs. atoms/formulas
- type of preference: static vs. context dependent

The major focus of recent research in WASP has been on qualitative approaches. This stems from the fact that for a variety of applications numerical information is hard to obtain (preference elicitation is rather difficult) — and often turns out to be unnecessary. We describe some main directions studied within the WASP community. Both static and context dependent approaches can be found in the category of rule preferences. Programs with ordered disjunction and optimization programs are context dependent approaches with formula preference. Ordered logic programs belong to the category of static rule preference. Comparative studies have been conducted in [162, 53]. Among others, it is shown that the approaches to rule-based preferences form a hierarchy as regards their strength in eliminating answer sets.

3.2.1 Rule Preferences

A (statically) *ordered logic program* is a pair $(\Pi, <)$, where Π is a logic program and $< \subseteq \Pi \times \Pi$ is a strict partial order. Given, $r_1, r_2 \in \Pi$, the relation $r_1 < r_2$ expresses that r_2 has *higher priority* than r_1.

For example, consider the following program $\Pi_1 = \{r_1, r_2, r_3\}$, where

$$\begin{array}{lcrcl} r_1 & = & \neg a & \leftarrow & \\ r_2 & = & b & \leftarrow & \neg a, not\ c \\ r_3 & = & c & \leftarrow & not\ b. \end{array}$$

Each r_i identifies the respective rule. This program has two regular answer sets, one given by $\{\neg a, b\}$ and the other given by $\{\neg a, c\}$. For the first answer set, rules r_1 and r_2 are applied; for the second, r_1 and r_3. However, assume that we have reason to prefer r_2 to r_3, expressed by $r_3 < r_2$. In this case we would want to obtain just the first answer set.

There have been numerous proposals for expressing preferences in extended logic programs, including [173, 92, 123, 35, 29, 163, 162, 52, 121, 120, 53, 146, 147]. The general approach has been to employ meta-formalisms for characterizing "preferred answer sets". For instance, a common approach is to generate all answer sets for a program and then, in one fashion or other, select the most preferred set(s). Consequently, non-preferred as well as preferred answer sets are first generated, and the preferred sets next isolated by a filtering step. Such approaches generally have a higher complexity than the underlying logic programming semantics.

Unlike this, the preference approaches in [52, 162, 25], referred to as D-, W-, and B-strategy, do not increase the complexity. There, preferred answer sets are (standard) answer sets, where for the rules which determine the answersetship, the "question of applicability" does not depend on lower ranked rules. More precisely, in the D-strategy, it is not allowed to apply a rule r whenever the derivation of positive prerequisites (so-called groundedness) of r is done by lower ranked rules. Furthermore, rules can never be defeated by lower ranked ones. The W-strategy weakens the D-strategy in that way that groundedness or defeat by lower ranked rules is allowed whenever the head of this rule can be derived in another way. In contrast to that, the B-strategy pays only attention to the defeat of rules. In [162], a comparative study of the D-, W-, and B-strategy is given, where all three approaches are described as uniform fixpoint characterizations wherein the different ways of deciding the applicability of rules is represented. Furthermore, a hierarchy between these three strategies is shown. That is, every preferred answer set concerning D-strategy, is also preferred in the W-strategy and every answer set preferred in the W-strategy is also preferred in the B-strategy. This hierarchy is mainly induced by the decreasing interaction between groundedness and preferences. While the D-strategy requires full compatibility between groundedness and preferences, this interaction is weakened in the W-strategy, before it is fully abandoned in the B-strategy.

In the above given example Π_1 with the preference $r_3 < r_2$, all strategies obtain $\{\neg a, b\}$ as preferred answer set. Answer set $\{\neg a, c\}$ is in no strategy preferred, since rule r_2 is defeated by lower ranked rule r_3 and there is no way to derive the head of rule r_2 in another way.

[52] provides for these strategies a polynomial transformation from ordered logic programs into extended logic programs wherein the preferences are respected, in that the answer sets obtained in the transformed theory correspond with the preferred answer sets of the original theory. This compilation process has been implemented in the `plp` system [155]. Since the result of the compilation is an extended logic program, one can make use of existing answer set solvers, such as `dlv` [86] and `smodels` [166].

In contrast to the compilation, [120] provides an operational characterization for the integration of preference information into an answer set solver. An ordered logic program is represented by a rule dependency graph which includes the preference information. Preferred answer sets are characterized by non-standard graph colorings on this graph. For the D-strategy, the `GCplp` system [94, 120] has been developed, where preferred answer sets are computed by gradually turning an uncolored rule dependency graph into a totally colored one. The interaction between groundedness and preference information affects the computation of deterministic and non-deterministic consequences.

Alternatively, preference information can also be handled by a meta-interpretation [70, 73] within answer set programming.

Instead of static preferences, one can also consider dynamic preferences, where the preference information becomes dynamically active. An application of dynamic preferences is, for example, legal reasoning. In (Gordon, 93) [99] the following legal reasoning problem was formulated:

> "A person wants to find out if her security interest in a certain ship is perfected. She currently has possession of the ship. According to the *Uniform Commercial Code* (UCC, §9-305) a security interest in goods may be perfected by taking possession of the collateral. However, there is a federal law called the *Ship Mortgage Act* (SMA) according to which a security interest in a ship may only be perfected by filing a financing statement. Such a statement has not been filed. Now the question is whether the UCC or the SMA takes precedence in this case. There are two known legal principles for resolving conflicts of this kind. The principle of *Lex Posterior* gives precedence to newer laws. In our case the UCC is newer than the SMA. On the other hand, the principle of *Lex Superior* gives precedence to laws supported by

the higher authority. In our case the SMA has higher authority since it is federal law."

This can be modeled as the following ordered logic program with dynamic preferences, where the *name* predicate is used to refer to the rules:

$$
\begin{array}{l}
perfected \leftarrow name(ucc), possession, not\ \neg perfected \\
\neg perfected \leftarrow name(sma), ship, \neg finstatement, not\ perfected \\
possession \leftarrow \\
ship \leftarrow \\
\neg finstatement \leftarrow \\
(Y < X) \leftarrow name(lex\_posterior(X,Y)), newer(X,Y), not\ \neg(Y < X) \\
(X < Y) \leftarrow name(lex\_superior(X,Y)), state\_law(X), federal\_law(Y), \\
\qquad not\ \neg(X < Y) \\
newer(ucc, sma) \leftarrow \\
federal\_law(sma) \leftarrow \\
state\_law(ucc) \leftarrow
\end{array}
$$

Adding the rule

$$(lex\_posterior(X,Y) < lex\_superior(X,Y)) \leftarrow$$

gives Lex Superior higher priority than Lex Posterior. Hence, we get the preferred answer set

$$
\begin{array}{c}
\{possession, ship, \neg finstatement, newer(ucc, sma), \\
state\_law(ucc), federal\_law(sma), \neg perfected\}.
\end{array}
$$

3.2.2 Ordered Disjunctions

Ordered disjunction is a form of disjunction where the order of disjuncts expresses preference. Logic programs with ordered disjunction [22, 26], LPODs for short, use ordered disjunction ($\times$) in the head of rules to express preferences among literals in the head: the rule

$$r = A_1 \times \ldots \times A_n \leftarrow body$$

says: if *body* is satisfied then some A_i must be in the answer set, most preferably A_1, if this is impossible then A_2, etc. Answer sets are defined through split programs containing exactly one option for each of the original LPOD rules, where, for $k \leq n$, option r^k of the rule above is

$$A_k \leftarrow body, \texttt{not}\ A_1, \ldots, \texttt{not}\ A_{k-1}.$$

An answer set S can satisfy rules like r to different degrees, where smaller degrees are better: if *body* is satisfied in S, then the satisfaction degree of r is the smallest index i such that $A_i \in S$. Otherwise, the rule is irrelevant. Since there is no reason to blame an answer set for not applying an inapplicable rule we also define the degree to be 1 in this case. The degree of r in S is denoted $deg_S(r)$.

Based on the satisfaction degrees of single rules a global preference ordering on answer sets is defined. This can be done through a number of different combination strategies. Let $S^k(P) = \{r \in P \mid deg_S(r) = k\}$. For instance, we can use one of the following conditions to define that S_1 is strictly preferred to S_2:

1. there is a rule satisfied better in S_1 than in S_2, and no rule is satisfied better in S_2 than in S_1 (Pareto);
2. at the smallest degree j such that $S_1^j(P) \neq S_2^j(P)$ we have $S_1^j(P) \supset S_2^j(P)$ (inclusion);
3. at the smallest degree j such that $|S_1^j(P)| \neq |S_2^j(P)|$ we have $|S_1^j(P)| > |S_2^j(P)|$ (cardinality);
4. the sum of the satisfaction degrees of all rules is smaller in S_1 than in S_2 (penalty sum).

Note that S_1 is inclusion preferred to S_2 whenever it is Pareto preferred to S_2, and cardinality preferred to S_2 whenever it is inclusion preferred to S_2. The penalty sum strategy can, for instance, be used to model *dlv*'s weak constraints, provided all weights are integers. In each case, we obtain a partial order on answer sets, and we are interested in optimal answer sets, that is answer sets which are not dominated by other answer sets with respect to the given ordering.

As a simple example consider the LPOD representing the dinner preferences of a simple agent (colors denote types of wine):

1. $red \times white \leftarrow meat$
2. $white \times red \leftarrow fish$
3. $meat \times fish$

Answer sets consist of 2 atoms combining a dish ($fish$ or $meat$) with a wine (red or $white$). The single maximally preferred answer set is, independent of the chosen combination strategy, $\{meat, red\}$. All rules are satisfied to degree 1. If we add the information that red wine is not available ($\neg red$), we get 2 the answer sets $\{meat, white\}$ and $\{fish, white\}$ which both are optimal (both satisfy one of the rules to degree 2 only).

LPODs can be implemented using a generate and improve strategy: we first generate an arbitrary answer set M and then use it as input for a tester program, generated from M and the original LPOD. The tester program has only answer sets strictly better than M. We can thus iterate until the tester program becomes inconsistent and know that the last answer set found is optimal. For an implementation, see `psmodels` (`http://www.tcs.hut.fi/Software/smodels/priority/`)

3.2.3 Optimization Programs

In the LPOD approach the construction of answer sets is amalgamated with the expression of preferences. Optimization programs [27], on the other hand, strictly separate these two aspects. This allows for greater modularity, flexibility and generality in the expression of preferences — in certain cases at the cost of somewhat less compact representations.

An optimization program is a pair (P_{gen}, P_{pref}). Here, P_{gen} is an arbitrary logic program (normal, extended, disjunctive, ...) used to generate answer sets. All we require is that it produces sets of literals as its answer sets. P_{pref} is a preference program. Preference programs consist of preference rules of the form

$$C_1 > \ldots > C_n \leftarrow \mathit{body}$$

where the C_i are boolean combinations of literals built from $\wedge, \vee, \neg$ and `not` . Here, a *boolean combination* over a set of atoms A is a formula built of atoms in A by means of disjunction, conjunction, strong ($\neg$) and default (`not`) negation, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation only in front of literals. For example, $a \wedge (b \vee \texttt{not}\ \neg c)$ is a boolean combination, whereas `not` $(a \vee b)$ is not. The restriction simplifies the treatment of boolean combinations. Satisfaction of a boolean combination in a given answer set can be defined in a straightforward manner.

As in the case of LPODs, answer sets can satisfy preference rules to different degrees. If the *body* of a preference rule is satisfied in an answer set S and some boolean combination C_i in its head is also satisfied, then the satisfaction degree of the rule in S is the index of the leftmost satisfied boolean combination. Otherwise the rule is irrelevant. Since S is not to be blamed for irrelevant rules, irrelevance is considered as good a satisfaction degree as 1.

Again, the combination strategies described in the context of LPODs — and many others — can be used to generate the global preference order on answer sets out of the satisfaction degrees of individual rules. It is also possible to introduce meta preferences among the preference rules themselves

by grouping them into subsets with different ranks. Preference rules with highest priority are considered first. Intuitively, rules with lower priority are only used to distinguish between answer sets which are of the same quality according to the more preferred rules.

Since none of the different combination strategies is the most adequate one for all applications, and since it may even be useful to apply different combination methods for different aspects of a single problem, a preference specification language has been proposed in [24]. The idea is to replace preference programs by expressions of the new language. The language is based on preference rules similar to the ones discussed above, but additionally allows the user to specify in a flexible manner how the satisfaction degrees of the rules are to be combined to a global preference order.

It turns out that this approach can be implemented in a similar fashion as LPODs by compiling the current answer set M, the generating program P_{gen} and the preference expression (respectively preference program) to a tester program which produces only answer sets of P_{gen} strictly better than M.

3.2.4 Ordered Choice Logic Programs

Once the fundamental issues of how to represent and store information have been addressed, one of the most important requirements for any automated reasoning system is to be able to characterise and make decisions. To make a decision a series of meta questions must first be answered; what are the alternatives, how do they relate to each other, how many are there to choose from and what should we do if alternatives are equally preferred or unrelated? Any logic which seeks to characterise decisions must address these questions.

Ordered Choice Logic Programming (OCLP) [45, 44, 19, 41, 49] is a conceptual extension of answer set programming to characterise decisions.

Choice rules [42, 43] are a formal way of saying which alternatives are available when a decision has to be made. The ordering between components gives a clear, expressive and extensive system for deciding which is the 'best' alternative, while the two decision semantics, skeptical and credulous, say what to do when two options are equally preferred. OCLP has been developed over several years and OCT [19], a tool for computing the answer sets of an OCLP program by providing a front-end to `Smodels` [145], is available from `http://www.cs.bath.ac.uk/~mdv/oct/` licenced under the GPL.

Both types of negation (classical negation and negation as-failure) can easily be simulated inside OCLP. Also available is Extended OCLP providing the necessary connectives and definitions that allow direct use of negation.

The algorithm implemented by OCT is a polynomial translation of OCLP to traditional answer set programs. Furthermore it can be shown that a bi-

directional transformation exists between OCLP and generalised extended logic programs.

OCLP is well-suited for situations where knowledge about decisions and the preference between their alternatives has to be modelled. Although the notion of defeated (sometimes called overruling) is known in many preference/order-based logic programming languages (e.g. [93, 124, 28, 36, 34, 2, 81], they mostly restrict to comparing literals and their complements, making the "decisions" static and predefined. In OCLP the decisions are dynamic and can comprise multiple alternatives. Two systems, dynamic logic programming [2] and update sequences [81], can easily be mapped to the OCLP.

So far OCLP has been used to model and extend classical game theory [47, 48] and to model the reasoning capabilities of individual agents in a multi-agent environment [46, 49].

3.3 Actions and Change

Reasoning about actions and change is an important field in knowledge representation and reasoning. In this context the use of logic-based approaches, in particular, for planning dates back many decades.

The usage of non-monotonic logic programs in this context, especially for planning, has been explored in several preliminary works, including [68, 167]. Later, the use of ASP for planning has been advocated by Lifschitz, who coined the term Answer Set Planning [132].

Action Language $\mathcal{K}$

Inspired by action languages such as $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, in [74, 79] the action language $\mathcal{K}$ has been introduced, which is based on answer set semantics. It follows like these languages a transition-oriented approach, in which transitions between states (which are described by the values of fluents, i.e., predicates that may change over time) are governed by axioms of a language which, loosely speaking, state when certain fluents are causally explained to be true resp. false in the new state given the value of other fluents in the new state, in the previous state, and the actions that have been executed (in parallel). Different from similar languages, however, $\mathcal{K}$ provides default negation "not" for usage in axioms, and allows to consider states in which the values of fluents also might be unknown. This allows, in particular, for representing secure planning problems (that is, conformant planning problems), in which a form of a plan is searched that works under all circumstances, regardless of the states and possible nondeterministic action effects, sometimes in a concise way.

An example which illustrates the flavor of $\mathcal{K}$ is the following action description of the Bomb-in-the-Toilet (BT) problem. We have been alarmed that there is a bomb (exactly one) in a lavatory. There are p suspicious packages which could contain the bomb. There is a toilet bowl, and it is possible to dunk a package into it. If the dunked package contained the bomb, then the bomb is disarmed and a safe state is reached. The obvious goal is to reach a safe state.

The following action description in $\mathcal{K}$ serves this purpose.

```
fluents :   armed(P) requires package(P).
            unsafe.
actions :   dunk(P) requires package(P).
always :    inertial  − armed(P).
            caused  − armed(P) after dunk(P).
            caused unsafe if not  − armed(P).
            executable dunk(P).
```

Here, `armed(P)` is a fluent which intuitively states that package P is armed, and `unsafe` is a fluent which says that the currents situation is unsafe. The action `dunk` can be executed at any time (this is expressed by the last line), to the effect that the respective package is disarmed. A situation is unsafe, if there is a package of which we don't know for certain that it is disarmed, i.e., it is possibly armed. Being disarmed is declared inertial (i.e., a bomb can't get armed if it wasn't). The knowledge about the packages is represented by facts `package(1).`, `package(2).`, ... `package(`n`).`

From the knowledge perspective, nothing definite is known about `armed(P)` (and about $-$`armed(P)`) for a particular package `P`, so the initial situation can be represented by one state in which neither `armed(P)` nor $-$`armed(P)` holds. Every plan for the goal `not unsafe` will lead to the desired state of safety. For example, a feasible plan would be to take the actions `dunk(1)`,...,`dunk(`n`)` one after the other. However, $\mathcal{K}$ features also parallel action execution; thus, dunking all packages in parallel would be another plan. If undesired, parallel action execution can be easily disabled using suitable constraints.

For precise definitions and more examples, see [79], where also a complexity study is carried out. For an extension to accommodate optimal planning using action costs, see [77]. An extended discussion, addressing knowledge representation issues and a comparison to related languages is given in [80].

A solid prototype of the system has been implemented, which is described and compared against other systems in [78]. A download and online examples are available at `http://www.dbai.tuwien.ac.at/proj/dlv/K/`.

Action Language $\mathcal{E}$

The work described in [67] extends further the link between action languages and ASP, by presenting a translation of action language $\mathcal{E}$ [115] into logic programs under the answer sets semantics. The main purpose of language $\mathcal{E}$ is to address, within a unifying framework, the three major problems of frame, ramification and qualification with emphasis on the problems of modularity and elaboration tolerance of the domain descriptions. The following examples illustrates the main features of language $\mathcal{E}$.

$Break$ `initiates` $Broken$
$TurnOnKey$ `initiates` $Running$ `when` $\{Battery\}$
$\neg Running$ `whenever` $\{Broken\}$
$Break$ `happens-at` 1
$TurnOnKey$ `happens-at` 2
$Battery$ `holds-at` 0
$\neg Running$ `holds-at` 0

The first proposition says that the effect of action $Break$ is that the car is $Broken$, whereas the second proposition states that the execution of action $TurnOnKey$ has the effect that $Running$ becomes true if at the time of the execution of $TurnOnKey$ the fluent $Battery$ is true. The third proposition is a *ramification* statement that expresses the fact that any action that would initiate $Broken$ it would also terminate $Running$. It also expresses the general domain constraint that at any time $Broken$ and $Running$ can not hold together. The next four propositions provide a specific narrative in which two actions are known to have been executed at time 1 and 2. The last two propositions are called *observations*, stating that $Battery$ was true and that the car was not running at time 0.

The semantics of language $\mathcal{E}$ sanctions essentially one model for the above theory. In this model the car continues to remain not running at time 3 onwards, despite the execution of action $TurnOnKey$. This is because the fact that the car is broken at time 3 (as a result of the earlier $Break$ action) prevents the initiation of $Running$ by this action as otherwise after time 3 we would end up with a state that violates the constraint that we can not have at the same time $Broken$ and $Running$. Assume now that in addition to the above we are also given the observation $Running$ `holds-at` 3. If the actions of the above theory are *strict*, the above theory is inconsistent. If however the actions are *default*, then the semantics of language $\mathcal{E}$ sanction one model where the action $Brake$ fails to produce its effect $Broken$.

In [67] we investigated how increasing the expressiveness of a language $\mathcal{E}$

so that it can capture the problems of frame, ramification and qualification, affects its computational complexity, and how a solution to these problems can be implemented within Answer Set Programming. Our current work [66] focuses on identifying subclasses of action theories in which reasoning is easier than in the general case. Our main interest is in theories that can be computed efficiently by current state-of-the-art ASP solvers.

We are currently developing a prototype system that translates language $\mathcal{E}$ theories into `Smodels` programs. We plan to use this system for experimentation and comparison with the existing $\mathcal{E}$-RES system (`http://www2.cs.ucy.ac.cy/~pslogic/`) for $\mathcal{E}$ and other similar systems.

3.4 Description Logics

Two interesting approaches to combining description logics and ASP techniques, open answer set programming and dl-programs, developed in the WASP research groups are discussed in this section.

3.4.1 Open Answer Set Programming

In traditional ASP one has the *domain-closure assumption* [96], i.e. the relevant domain elements are all supposed to be specified in the program. In many natural problems, however, this assumption may lead to wrong conclusions. For example, a DLP P consisting of the rules $pass(X) \leftarrow study(X)$; $fail(X) \leftarrow$ `not` $pass(X)$; $study(X) \vee$ `not` $study(X) \leftarrow$, and $pass(john) \leftarrow$, has the (normal) answer sets $\{pass(john)\}$ and $\{pass(john), study(john)\}$. Since none of them contains a *fail*-atom, one might, wrongfully, conclude that one can never fail. Listing more students might solve the problem in this case, however, in general, this puts a serious burden on the knowledge engineer, having to handle "all" influential constants.

Using (possibly infinite) open domains, i.e. allowing the universe to be a superset of the constants in the program, makes sure that one rightfully concludes that students may fail if they do not study. For example,

$$(\{john, bill\}, \{pass(john), fail(bill)\})$$

is an *open answer set* where the first component is the extended universe and the second one is an answer set that resulted from grounding P with this universe.

In [104, 103, 102], one can find the formal semantics of such *open answer set programming*. Reasoning with open domains is undecidable for arbitrary DLPs. To resolve this, [104, 103, 102] resort to an expressive subclass of DLPs, i.e., Conceptual Logic Programs (CLPs). Reasoning with CLPs is

shown to be decidable in [104, 103] by a reduction to automata theory. The type of CLPs in [102] allow for a reduction to normal, closed domain, finite answer set programming.

The CLPs in [104] allow to simulate reasoning in the Description Logic [5] $\mathcal{SHIF}$, [103] loosens the syntax of CLPs and is able to simulate reasoning in the more expressive DL $\mathcal{SHIQ}$. Finally, the CLPs in [102] allow for constants in programs such that the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ can be simulated. Since ASP is a logic programming paradigm, and thus suited for reasoning with rules, those simulations provide for integrated reasoning with both ontological and rule knowledge.

Hybrid approaches to reasoning with DLs and rules can be found in [71, 159] and [128], the former introduces ([159] extends) AL-log, i.e. a language consisting of two subsystems, a DL part and a (disjunctive) Datalog part, where the interaction between both happens through constraints within Datalog clauses. In [128], knowledge bases contain a set of Horn rules and a DL terminology where concepts and roles may appear as predicates in Horn rules.

Another approach is to reduce DL knowledge bases to logic programs, e.g., [107] uses an intermediate translation to first order clauses and then performs resolution-based decision procedures before effectively translating to a disjunctive Datalog program. In [144] non-recursive $\mathcal{ALC}$ is translated in disjunctive databases. [3] presents a translation from the DL $\mathcal{ALCQI}$ to answer set programs, using function symbols to accommodate for an infinite Herbrand universe. [100] simulates reasoning in DLs through simple Datalog programs. This necessitates heavily restricting the usual DL constructors, e.g., negation or number restrictions cannot be expressed.

In [10], the simulation of a DL with acyclic axioms in *open logic programming* is shown. An open logic program is a program with possibly undefined predicates, and a FOL-theory; the semantics is the completion semantics, which is only complete for a very restrictive set of programs.

There is currently no implementation for reasoning with CLPs available.

3.4.2 DL-Programs

Description logic programs, or *dl-programs* for short, are presented in [84, 85] as a novel method to couple description logics with nonmonotonic logic programs. Roughly speaking, a dl-program $KB = (L, P)$ consists of a knowledge base L in a description logic and a finite set P of generalized logic program rules, called *dl-rules*. These are similar to usual rules in logic programs with negation as failure, but they may also contain *queries to* L in their bodies, which are given by special atoms (on which possibly default negation may

apply). For example, a rule

$$cand(X, P) \leftarrow paperArea(P, A),\ DL[Referee](X),\ DL[expert](X, A)$$

may express that X is a candidate reviewer for a paper P, if the paper is in area A, and X is known to be a referee and an expert for area A. Here, the latter two are queries to the description logic knowledge base L, which has a concept *Referee* and role *expert* in its signature. For the evaluation, the precise definition of *Referee* and *expert* within L is fully transparent, and only the logical contents at the level of inference counts. Thus, dl-programs fully support encapsulation and privacy of L — this is needed if parts of L should not be accessible (for example, if L contains an ontology about risk assessment in credit assignment), and only extensional reasoning services are available.

Another important feature of dl-rules is that queries to L also allow for specifying an input from P, and thus for a *flow of information from P to L*, besides the flow of information from L to P, given by any query to L. Hence, description logic programs allow for building rules on top of ontologies, but also (to some extent) building ontologies on top of rules. This is achieved by dynamic update operators through which the extensional part of L can be modified for subjunctive querying. For example, the rule

$$paperArea(P, A) \leftarrow DL[keyword \uplus kw; inArea](P, A)$$

intuitively says that paper P is in area A, if P is in A according to the description logic knowledge base L, where the extensional part of the *keyword* role in L (which is known to influence *inArea*) is augmented by the facts of a binary predicate *kw* from the program. In this way, additional knowledge (gained in the program) can be supplied to L before querying. Using this mechanism, also more involved relationships between concepts and/or roles in L can be defined and exploited.

The description logic knowledge bases in dl-programs are specified in the well-known description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which underly OWL Lite and OWL DL [105, 106], respectively.

Two basic types of semantics have been defined for dl-programs: in [84], a generalization of the answer-set semantics for ordinary logic programs is given, and in [85], a generalization of the well-founded semantics [170, 8]. In fact, two versions of the answer-set semantics for dl-programs are introduced, namely the *weak answer-set semantics* and the *strong answer-set semantics*. Both semantics coincide with usual answer sets in the case of ordinary normal programs. Every strong answer set is also a weak answer set, but not vice versa. The two notions differ in the way they deal with *nonmonotonic*

dl-queries. While the answer set semantics resolves conflicts by virtue of permitting multiple intended models as alternative scenarios, the well-founded semantics remains agnostic in the presence of conflicting information, assigning the truth value *false* to a maximal set of atoms that cannot become true during the evaluation of a given program.

An efficient implementation of the answer-set and the well-founded semantics for dl-programs has been realized in a working prototype exploiting the two state-of-the-art solvers DLV [126] and RACER [101]. A major issue in this respect is an efficient interfacing between the two reasoning systems at hand, for which special methods have been devised in [83]. Depending on the type of stratification of the dl-program, a number of different evaluation strategies can be employed, ranging from a refined guess-and-check algorithm to iterative procedures with alternating calls of the external solvers.

3.5 Social and Sensor-Dependent Logic Programming

In a multi-agent environment it is possible to represent agents' requirements or desires as logic programs. Moreover, in order to pursue his goals, one agent must often be capable of fine-tuning his behavior on that of the other agents. Thus, in [30] we have focused on the interactions among logic-program agents by introducing the *SOcial Logic Programming* (SOLP).

SOLP is designed for the description of agent mental states and enables agent *social ability*, i.e. the ability to take into account (the models of) the other agents in the community when reasoning. The language extends COLP [31], the main extension consisting of constraints over the *behavior* (i.e. what it can be inferred) of either a given number of or a specific other agent in the system. Moreover, the language SOLP allows for social constraints to be nested, and it is provided with a suitable fixpoint-like semantics. We have shown that the *Social Semantics* of SOLP extends the Joint Fixpoint Semantics of COLP. Importantly, we have shown that the language SOLP can be translated into the well-known DLP^A, which is basically disjunctive logic programming with aggregate functions, by a polynomial source-to-source transformation. The adoption of the stable model semantics provides a sound formal characterization to the language.

A very large and widely accepted literature witnesses that logic programming is a suitable framework for representing the agents' subjective perception w.r.t. the dynamic environment in which they act. Thus, we assume that the observe-think-act activity of an agent involves a set of sensors (responsible to scan the external environment) coupled to a *reasoning* layer (where the events that may occur in the external environment are represented and causally related to agent actions). Now, it is possible to encode

into an extended logic program both the sensors (by using specific literals) and the reasoning layer (by means of literals for events and actions and rules possibly using sensor literals).

Unfortunately, such a solution is founded on a simple (yet unrealistic) assumption: The agent always relies on its perception, which is sensitive to failure. In this respect, we proposed in [33] an extension of *Answer Set Programming*, called *sensor-dependent* (SD) logic programming, for taking into account possible perception failure of sensors. It is shown by examples that the language is very suitable for the above purpose. Moreover, a suitable semantics has been defined in an *answer-set* fashion and a polynomial algorithm has been given, allowing any SD program to be translated into an equivalent extended logic program.

4 Logical Foundations

4.1 Default Logic

Default logic was conceived in [157] as a logical formalism for default reasoning. Since then, it has turned into the best known and most widely studied approach to nonmonotonic reasoning. In particular, it can be seen as the direct ancestor of answer set programming. To see this, observe that a logic programming rule is simply a syntactically restricted default rule that allows for highly efficient implementations.

The very generality of default logic means that it lacks several important properties, including *existence of extensions* [157] and *cumulativity* [140]. In addition, differing intuitions concerning the role of default rules have led to differing opinions concerning other properties, including *semi-monotonicity* [157] and *commitment to assumptions* [156]. As a result, a number of modifications to the definition of a default extension have been proposed, resulting in a number of variants of default logic. Most notably these variants include *constrained default logic* [54], *cumulative default logic* [21], *justified default logic* [139], and *rational default logic* [141]. (To be sure, there are other variants of default logic. The variants covered here are arguably the best-known and studied [4].) In each of these variants, the definition of an extension is modified, and a system with properties differing from the original is obtained.

In [51] we have shown how variants of default logic can be expressed in Reiter's original approach. Similarly, we have shown that rational default logic and default logic may be encoded, one into the other. For the most part the provided transformations have good properties, being (with exceptions) faithful, polynomial, modular, and monotonic. This work then complements

previous work in nonmonotonic reasoning which has shown links between (seeming) disparate approaches. Here we show links between (seemingly) disparate variants of default logic. As well, the translations clearly illustrate the relationships between alternative approaches to default logic.

As in answer set programming, default logics offer two kind of reasoning principles: a default conclusion that appears in some extension is called a *credulous* (or *brave*) default conclusion, while one that appears in every extension is called a *skeptical* conclusion. Intuitively it might seem that skeptical inference is the more useful notion. However, this is not necessarily the case. In diagnosis from first principles [158] for example, in one encoding there is a 1-1 correspondence between diagnoses and extensions of the (encoding) default theory. Hence one may want to carry out further reasoning to determine which diagnosis to pursue. More generally there may be reasons to prefer some extensions over others, or to somehow synthesize the information found in several extensions. In [50], we have described an approach for encoding default extensions within a single extension. Using constants and functions for naming, we can refer to default rules, sets of defaults, and instances of a rule in a set. Via these names we can, first, determine whether a set of defaults is its own set of generating defaults and, second, consider the application of sets of defaults ordered by set containment. The translated theory requires a modest increase in space: except for unique names axioms, only a constant-factor increase is needed.

4.2 Finitary and Open Programs

Finitary programs [12, 15] support function symbols and recursion, without affecting decidability. This extension is mainly motivated by the need of representing and reasoning about recursive data structures (such as XML documents) in a natural and uniform way, without resorting to external preprocessors.

Current prototypes [11] comprise a *finitary program recognizer* and a *relevant program constructor*. They can be downloaded from `http://people.na.infn.it/~bonatti/ricerca/software/index.html`. Work on integrating finitary programs into DLV is in progress.

In *open programs* [13, 14] the definition of some predicates is not complete (or exhaustive), and can be integrated by more rules. Open programs have been introduced for program module analysis, as well as reasoning with open domains. A combination of finitary programs and open programs supports abduction with unbounded, possibly open domains [16, 17].

4.3 ID-Logic

This line of work aims at developing an alternative ASP logic called ID-logic. This logic is an extension of classical logic with inductive definitions. The motivation for this work is to build a logic with solid epistemological foundations in mathematics, suitable as a rich and natural knowledge representation language and for which more efficient problem solvers can be built. There are three components of this project: semantics, knowledge representation and problem solving. Below we discuss each of these components.

The epistemological foundation of ID-logic is the notion of inductive definition as found in mathematics. Inductive definitions that appear in mathematical texts show a very strong correspondence with logic programs, both on syntactical and semantical level. To formalise this idea, we developed a theory of generalised nonmonotone inductive definitions and demonstrated its relationship with answer set programming and non-monotone reasoning. Our theory, called approximation theory, is a fixpoint theory. The standard fixpoint theory is limited to monotone induction and was developed 70 years ago by Alfred Tarski. We have contributed here by extending this theory to the case of general non-monotone induction [62, 61]. This result has been obtained in collaboration with Prof. Truszczyński and Prof. Marek from University of Kentucky. We have also showed that our fixpoint theory uniformly describes the semantics of the three main non-monotonic reasoning formalisms, i.e. logic programming, default logic and autoepistemic logic [63, 61]. Moreover, we have been able to unify default logic and autoepistemic logic; their relation was an open problem for two decades.

At the knowledge representation level, ID-logic is defined as an extension of classical logic with non-monotone inductive definitions. An inductive definition in an ID-logic theory is represented as a set of rules defining a subset of defined predicates in terms of other, open predicates. As an example, the following theory containing two definitions for the predicate $Person$ and one axiom expressing that men and women are disjunct categories:

$$\left\{ \begin{array}{l} \forall x(Person(x) \Leftarrow Man(x)), \\ \forall x(Person(x) \Leftarrow Woman(x)) \end{array} \right\}$$

$$\left\{ \begin{array}{l} \forall x(Person(x) \Leftarrow Child(x)), \\ \forall x(Person(x) \Leftarrow Adult(x)) \end{array} \right\}$$

$$\neg \exists x Man(x) \wedge Woman(x)$$

The syntax and semantics of ID-logic have been defined in [59, 65], partly in collaboration with Prof. Ternovska from Simon Fraser University. In [64]

we have demonstrated the role of ID-logic for knowledge representation by showing how situation calculus, a well-known AI-formalisms for temporal reasoning, contains hidden forms of inductive definitions and has an elegant formalisation in ID-logic. The methodology of ID-logic and its relationship to answer set programming have been studied in [64, 98, 60]. We have also studied the computational complexity of inference in ID-logic [61, 151]. We have also started with the study of extensions of definitions with aggregates in the context of the PhD work of Dr. Pelov [152, 151, 150].

At the computational level the work ID-logic solvers is still in a early stage but some results have been obtained. One important result that we have already obtained is the development of the A-system [118, 172]. This system implements abductive inference in the context of a sublogic of ID-logic. At this moment, this is one of the best abductive systems available. It has been developed in the context of the PhD-work of Dr. Van Nuffelen at the K.U.Leuven, in collaboration with Prof. Kakas of the University of Cyprus (WASP partner). The system integrates CLP-techniques, open functions and aggregates. It is available at `http://www.cs.kuleuven.ac.be/~bertv/Asystem/`.

We have also started with the development of a model generator for ID-logic. A first version exploits a translation from ID-logic to answer set programming presented in [98] and uses `Smodels` to generate models of the input ID-logic theory. It is available at `http://www.cs.kuleuven.ac.be/~maartenm`. We have started a new project to build a model generator tuned for ID-logic.

5 Summary of Implementations

In this section we summarize available implementations of language extensions described in the report.

Systems implementing disjunctive logic programs (Section 2.1)

- `DLV` (`http://www.dlvsystem.com`)
- `GnT` (`http://www.tcs.hut.fi/Software/gnt/`)
- `Cmodels` version 3 (`http://www.cs.utexas.edu/users/tag/cmodels.html`)

Systems implementing nested programs (Section 2.2)

- nlp (http://www.cs.uni-potsdam.de/~torsten/nlp/)
- NoMoRe (http://www.cs.uni-potsdam.de/~linke/nomore)

Systems implementing cardinality and weight constraints (Section 2.3)

- Smodels (http://www.tcs.hut.fi/Software/smodels/)

Systems implementing aggregates (Section 2.4)

- DLV (http://www.dlvsystem.com)
- Smodels-ag (http://www.cs.nmsu.edu/~ielkaban/smodels-ag.html)

Systems implementing templates (Section 2.5)

- DLP^T (http://dlpt.gibbi.com) build on top of DLV (http://www.dlvsystem.com)

Application oriented extensions

- Abduction with penalization (Section 3.1)
  implemented as a frontend of DLV (http://www.dlvsystem.com)
- Rule preferences (Section 3.2.1)
  plp (http://www.cs.uni-potsdam.de/~torsten/plp)
  GCplp (http://www.cs.uni-potsdam.de/~konczak/system/GCplp)
- Ordered disjunctions (Section 3.2.2)
  psmodels (http://www.tcs.hut.fi/Software/smodels/priority/)
- Ordered choice logic programs (Section 3.2.4)
  OCT (http://www.cs.bath.ac.uk/~mdv/oct/)
- Action language $\mathcal{K}$ (Section 3.3)
  implemented as a frontend of DLV (http://www.dbai.tuwien.ac.at/proj/dlv/K/)
- Action Language $\mathcal{E}$ (Section 3.3)
  $\mathcal{E}$-RES system (http://www2.cs.ucy.ac.cy/~pslogic/)
- DL-Programs (Section 3.4.2)
  NLP-DL (http://www.kr.tuwien.ac.at/staff/roman/semweblp/)

Finitary programs (Section 4.2)
Finitary program recognizer and instantiator (`http://people.na.infn.it/~bonatti/ricerca/software/index.html`)

ID-Logic (Section 4.3)
`A-system` `http://www.cs.kuleuven.ac.be/~bertv/Asystem/`.

6 Software Engineering Issues

This section recaps interesting software engineering issues for ASP.

- Generate and test programming methodology

  The predominant programming methodology for ASP is the generate and test technique briefly explained in Sections 2.1 and 3.2. This methodology has also motivated many of the language extensions. For example, disjunctions and cardinality and weight constraints have been found very useful for programming the generation of solution candidates and, e.g., aggregates and preferences for testing solution candidates.

- Core engine approach to language extensions

  Many of the language extensions have been implemented using a technique where the idea is to exploit an efficient implementation of a basic ASP language and implement new language features by compiling them to this basic language. For example, `DLV` (`http://www.dlvsystem.com`) and `Smodels` (`http://www.tcs.hut.fi/Software/smodels/`) have been extensively used as such core engines for implementing new language extensions ranging from preferences to action languages.

- Combining ASP and other approaches

  To complement the core engine approach there is new work on combining ASP engines with other approaches such as satisfiability (SAT) checkers and constraint programming systems. One trend is to use SAT checkers to implement answer set computation as, e.g., in `ASSAT` (`http://assat.cs.ust.hk/`) and `Cmodels` (`http://www.cs.utexas.edu/users/tag/cmodels.html`). Another interesting line of development is to integrate ASP solvers and constraint programming systems. The `Smodels-ag` system (`http://www.cs.nmsu.edu/~ielkaban/smodels-ag.html`) is an example of this.

- Programming Tools

  The need for better programming tools such as debuggers and tracers has been widely recognized. Now there is interesting recent work in this area, for example, in the IDEAS system [18]. Another interesting active area is equivalence testing of logic programs. Equivalence testing tools could provide a useful aid for step-wise development of an ASP program where the correctness of modifications or refinements would be checked against previous versions of the program. Interesting tools for equivalence testing are already emerging such as `SELP` [38], `setest` (`http://www.kr.tuwien.ac.at/students/prak_setest/`) and `lpeq` (`http://www.tcs.hut.fi/Software/lpeq/`).

References

[1] J. Alferes, L. M. Pereira, and T. Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Journal of the Theory and Practice of Logic Programming*, Forthcoming.

[2] José Júlio Alferes, Leite J. A., Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In Cohn et al. [39], pages 98–111.

[3] G. Alsaç and C. Baral. Reasoning in Description Logics using Declarative Logic Programming, 2002. `http://www.public.asu.edu/~guray/dlreasoning.pdf`.

[4] Grigoris Antoniou. A tutorial on default logics. *ACM Computing Surveys*, 31(4):337–359, 1999.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

[6] M. Balduccini and M. Gelfond. Logic programs with consistency-restoring rules. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, pages 9–18, 2003.

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

[8] C. Baral and V. S. Subrahmanian. Dualities between alternative semantics for logic programming and nonmonotonic reasoning. *J. Automated Reasoning*, 10(3):399–420, 1993.

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

[10] K. Van Belleghem, M. Denecker, and D. De Schreye. A Strong Correspondence between Description Logics and Open Logic Programming. In *Proc. of ICLP'97*, pages 346–360, 1997.

[11] P. A. Bonatti. Prototypes for reasoning with infinite stable models and function symbols. In *Proceedings of Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001*, number 2173 in LNCS, pages 416–419. Springer, 2001. [Link].

[12] P. A. Bonatti. Reasoning with infinite stable models. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 603–610. Morgan Kaufmann, 2001. [Link].

[13] P. A. Bonatti. Reasoning with open logic programs. In *Proceedings of Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001*, number 2173 in LNCS, pages 147–159. Springer, 2001. [Link].

[14] P. A. Bonatti. Abduction, asp and open logic programs. In *Proceedings of NMR'02*, 2002. [Link].

[15] P. A. Bonatti. Reasoning with infinite stable models II: Disjunctive programs. In *Proceedings of the 18th International Conference on Logic Programming, ICLP 2002*, number 2401 in LNCS, pages 333–346. Springer, 2002. [Link].

[16] P. A. Bonatti. Finitary open logic programs. In *Proceedings of Answer Set Programming: Advances in Theory and Implementation, NMR'02*, number 78 in CEUR Workshop Proceedings, pages 84–97, 2003. [Link].

[17] P.A. Bonatti. Abduction over unbounded domains via asp. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004*, pages 288–292. IOS Press, 2004.

[18] M. Brain and M. De Vos. Debugging logic programs under the answer set semantics. In *Answer Set Programming: Advances in Theory and Implementation (ASP05)*, pages 141–152, Bath, UK, 2005.

[19] Martin Brain and Marina De Vos. Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice. In *ASP03: Answer Set Programming: Advances in Theory and Implementation*. Ceur-WS, September 2003. online CEUR-WS.org/Vol-78/asp03-final-brain.ps; [ps.gz].

[20] R. Brena. Abduction, that ubiquitous form of reasoning. *Expert Systems with Applications*, 14(1–2):83–90, January 1998.

[21] G. Brewka. Cumulative default logic: In defense of nonmonotonic inference rules. *Artificial Intelligence*, 50(2):183–205, 1991.

[22] G. Brewka. Logic programming with ordered disjunction. In *Proc. AAAI-02*, pages 100–105. AAAI Press, 2002.

[23] G. Brewka. Answer sets: From constraint programming towards qualitative optimization. In *Proc. LPNMR-04*, pages 34–46. Springer LNCS 2923, 2004.

[24] G. Brewka. Complex preferences for answer set optimization. In *Proc. KR AAAI-04*, pages 213–223. Morgan Kaufmann, 2004.

[25] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.

[26] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, special issue on preferences in AI, 2004.

[27] G. Brewka, I. Niemelä, and M. Truszczyński. Answer set optimization. In *Proc. International Joint Conference on Artificial Intelligence(IJCAI 2003)*. Morgan Kaufman, 2003. [PS].

[28] Gerhard Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Articficial Intelligence Research*, 4:19–36, 1996.

[29] Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, April 1999.

[30] F. Buccafurri and G. Caminiti. A Social Semantics for Multi-Agent Systems. In *in Proc. of 8th LPNMR'05 (to appear)*. Springer, 2005.

[31] F. Buccafurri and G. Gottlob. *Multiagent Compromises, Joint Fixpoints, and Stable Models*, volume 2407 of *LNCS and LNAI*. Springer, 2002.

[32] F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[33] Francesco Buccafurri, Gianluca Caminiti, and Domenico Rosaci. Perception-dependent reasoning and answer sets. In Marco Cadoli, Marco Gavanelli, and Toni Mancini, editors, *Atti della Giornata di Lavoro: Analisi sperimentale e benchmark di algoritmi per l'Intelligenza Artificiale*, Dipartimento di Ingegneria, Universita' di Ferrara, Italy, June 10 2005. [PDF].

[34] Francesco Buccafurri, Wolfgang Faber, and Nicola Leone. Disjunctive Logic Programs with Inheritance. In Danny De Schreye, editor, *International Conference on Logic Programming (ICLP)*, pages 79–93, Las Cruces, New Mexico, USA, 1999. The MIT Press.

[35] Francesco Buccafurri, Wolfgang Faber, and Nicola Leone. Disjunctive logic programs with inheritance. *Journal of the Theory and Practice of Logic Programming*, 2(3), 2002.

[36] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Disjunctive ordered logic: Semantics and expressiveness. In Cohn et al. [39], pages 418–431.

[37] Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[38] Y. Chen, F. Lin, and L. Li. Selp—a system for studying strong equivalence between logic programs. In *Answer Set Programming: Advances in Theory and Implementation (ASP05)*, pages 141–152, Bath, UK, 2005.

[39] Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, Trento, June 1998. Morgan Kaufmann.

[40] L. Console, D. Theseider Dupré, and P. Torasso. On the Relationship Between Abduction and Deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[41] Marina De Vos. Implementing Ordered Choice Logic Programming using Answer Set Solvers. In *Third International Symposium on Foundations of Information and Knowledge Systems (FoIKS'04)*, volume 2942, pages 59–77, Vienna, Austria, February 2004. Springer Verlag.

[42] Marina De Vos and Dirk Vermeir. Choice Logic Programs and Nash Equilibria in Strategic Games. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 266–276, Madrid, Spain, 1999. Springer Verslag.

[43] Marina De Vos and Dirk Vermeir. On the Role of Negation in Choice Logic Programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 236–246, El Paso, Texas, USA, 1999. Springer Verslag.

[44] Marina De Vos and Dirk Vermeir. Dynamically Ordered Probabilistic Choice Logic Programming. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science Conference (FST TCS 2000)*, number 1974 in Lecture Notes in Computer Science, pages 227–239, New Delhi, India, December 2000. Springer-Verlag. [ps.gz].

[45] Marina De Vos and Dirk Vermeir. A Logic for Modelling Decision Making with Dynamic Preferences. In *Proceedings of the Logic in Artificial Intelligence (Jelia2000) workshop*, number 1999 in Lecture Notes in Artificial Intelligence, pages 391–406, Malaga, Spain, 2000. Springer Verslag.

[46] Marina De Vos and Dirk Vermeir. Logic Programming Agents and Game Theory. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 27–33, Stanford (Palo Alto), California, US, March 2001. American Association for Artificial Intelligence Press. [ps.gz].

[47] Marina De Vos and Dirk Vermeir. Dynamic Decision Making in Logic Programming and Game Theory. In *AI2002: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 36–47. Springer, December 2002. [ps.gz].

[48] Marina De Vos and Dirk Vermeir. Logic Programming Agents Playing Games. In *Research and Development in Intelligent Systems XIX (ES2002)*, BCS Conference Series, pages 323–336. Springer, December 2002. [ps.gz].

[49] Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[50] J. Delgrande and T. Schaub. Reasoning credulously and skeptically within a single extension. *Journal of Applied Non-Classical Logics*, 12(2):259–285, 2002. [PDF].

[51] J. Delgrande and T. Schaub. On the relation between Reiter's default logic and its (major) variants. In Th. Nielsen and N. Zhang, editors, *Proceedings of the Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2711 of *Lecture Notes in Artificial Intelligence*, pages 452–463. Springer-Verlag, 2003. [PDF].

[52] J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, March 2003. [PDF].

[53] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. Towards a classification of preference handling approaches in nonmonotonic reasoning. In U. Junker, editor, *Proceedings of the Workshop on Preferences in Artificial Intelligence and Constraint Programming: Symbolic Approaches*, pages 16–24. AAAI Press, 2002. [PDF].

[54] J.P. Delgrande, T. Schaub, and K. Jackson. Alternative approaches to default logic. *Artificial Intelligence*, 70(1-2):167–237, October 1995.

[55] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Aggregate Functions in DLV. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation*, pages 274–288, Messina, Italy, September 2003. Online at `http://CEUR-WS.org/Vol78/`.

[56] M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. *Journal of Logic and Computation*, 5(5):553–577, 1995.

[57] M. Denecker and D. De Schreye. SLDNFA: An Abductive Procedure for Abductive Logic Programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[58] M. Denecker and A. C. Kakas. Abduction in Logic Programming. In *Computational Logic: Logic Programming and Beyond 2002*, number 2407 in LNCS, pages 402–436. Springer, 2002.

[59] Marc Denecker. Extending classical logic with inductive definitions. In J. Lloyd et al., editor, *First International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 703–717, London, July 2000. Springer.

[60] Marc Denecker. Whatś in a model? Epistemological analysis of Logic Programming. In *Proceedings of Ninth International Conference on Principles of Knowledge Representation and Reasoning, Delta Whistler Resort, Canada*, pages 106–113, 2004. URL = http://www.cs.kuleuven.ac.be/cgibin-dtai/publ_info.pl?id=41086.

[61] Marc Denecker, Victir Marek, and Mirek Truszczyński. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, 2004. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41124.

[62] Marc Denecker, Victor Marek, and Mirosław Truszczyński. Ultimate approximations of operators in commonsense reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eigth International Conference (KR2002)*, pages 177–188, Toulouse, France, April 2002. Morgan Kaufman.

[63] Marc Denecker, Victor Marek, and Miroslaw Truszczyński. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence*, 143(1):79–122, 2003.

[64] Marc Denecker and Eugenia Ternovska. Inductive Situation Calculus. In *Proceedings of Ninth International Conference on Principles of Knowledge Representation and Reasoning, Delta Whistler Resort, Canada*, pages 545–553, 2004. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41085.

[65] Marc Denecker and Eugenia Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In Vladimir Lifschitz

and Ilkka Niemelä, editors, *7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2004. [Link].

[66] Y. Dimopoulos, L. Michael, and A. Kakas. Answer set programming algorithms and complexity for reasoning about actions and change. *Submitted*, 2004.

[67] Yannis Dimopoulos, Antonis Kakas, and Loizos Michael. Reasoning about actions and change in answer set programming. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, number 2923 in Lecture Notes in Computer Science, pages 61–73. Springer-Verlag, 2004.

[68] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding Planning Problems in Nonmonotonic Logic Programs. In *Proceedings of the European Conference on Planning 1997 (ECP-97)*, pages 169–181. Springer Verlag, 1997.

[69] The DLP^T web site. `http://dlpt.gibbi.com`.

[70] `meta-interpreter`. `http://www.dbai.tuwien.ac.at/proj/dlv/preferred/`, 2002.

[71] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.

[72] P. M. Dung. Negation as Hypotheses: An Abductive Foundation for Logic Programming. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91)*, pages 3–17. MIT Press, 1991.

[73] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *Journal of the Theory and Practice of Logic Programming*, 3(4-5):463–498, 2003.

[74] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings First International Conference on Computational Logic (CL-2000), Knowledge Representation and Non-monotonic Reasoning Stream*, number 1861 in LNCS/LNAI, pages 807–821. Springer Verlag, July 2000.

[75] 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.

[76] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-Solving Using the DLV System. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[77] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Answer Set Planning under Action Costs. *Journal of Artificial Intelligence Research*, 19:25–71, 2003.

[78] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, II: The $\texttt{dlv}^{\mathcal{K}}$ system. *Artificial Intelligence*, 144(1-2):157–211, 2003.

[79] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. *ACM Transactions on Computational Logic*, 5(2):206–263, April 2004.

[80] Thomas Eiter, Wolfgang Faber, Gerald Pfeifer, and Axel Polleres. Declarative planning and knowledge representation in an action language. In Ioannis Vlahavas and Dimitris Vrakas, editors, *Intelligent Techniques for Planning*. Idea Group, Inc., 2005. To appear.

[81] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On Properties of update Sequences Based on Causal Rejection. *Theory and Practice of Logic Programming*, 2(6), November 2002.

[82] 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 Nonmonotonic Reasoning (LPNMR-97)*, number 1265 in LNCS, pages 290–309. Springer, 1997.

[83] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Nonmonotonic Description Logic Programs: Implementation and Experiments. In F. Baader and A. Voronkov, editors, *Proceedings 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2005)*, LNCS. Springer, 2005. To appear.

[84] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web". In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*, pages 141–151. Morgan Kaufmann, 2004.

[85] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded Semantics for Description Logic Programs in the Semantic Web. In G. Antoniou and H. Boley, editors, *Proceedings RuleML 2004 Workshop, ISWC Conference, Hiroshima, Japan, November 2004*, number 3323 in LNCS, pages 81–97. Springer, 2004.

[86] Eiter, T., Leone, N., Pfeifer G., Mateis C., and Scarcello, F. The kr system `dlv`: Progress report, comparisons and benchmarks. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann Publishers, 1998.

[87] S. Erdogem and V. Lifschitz. Definitions in answer set programming. In V. Lifschitz and I. Niemel, editors, *Proceedings LPNMR 04*, volume 2923 of *Lecture Notes in Computer Science*, pages 114–126. Springer-Verlag, 2004.

[88] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255. MIT Press, 1989.

[89] T. Eiter et al. WP5 report: Model applications and proofs-of-concepts. [html], 2004.

[90] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In José Júlio Alferes and João Leite, editors, *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004)*, number 3229 in Lecture Notes in AI (LNAI), pages 200–212. Springer Verlag, September 2004.

[91] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 2004. Accepted for publication.

[92] D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217, Cambridge, Mass, 1991.

[93] D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217, Cambridge, Mass, 1991. Morgan Kaufmann.

[94] `GCplp`. `http://www.cs.uni-potsdam.de/~konczak/system/GCplp`, 2003.

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

[96] M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993.

[97] Michael Gelfond. Representing Knowledge in A-Prolog. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic. Logic Programming and Beyond*, number 2408 in LNCS, pages 413–451. Springer, 2002.

[98] D. Gilis, M. Mariën, and M. Denecker. On the relation between id-logic and answer set programming. In J. J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence (proceedings of Jelia'04)*, volume 3229 of *LNAI*, pages 108 – 120. Springer, 2004.

[99] T. Gordon. *The pleading game: An Artificial Intelligence Model of Procedural Justice*. Dissertation, Technische Hochschule Darmstadt, 1993.

[100] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003. `http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/p117-grosof.pdf`.

[101] V. Haarslev and R. Möller. RACER system description. In *Proceedings IJCAR-2001*, volume 2083 of *LNCS*, pages 701–705, 2001.

[102] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Semantic Web Reasoning with Conceptual Logic Programs. In *Proceedings of the Rules and Rule Markup Languages for the Semantic Web Workshop*, number 3323 in LNCS. Springer, 2004. To appear.

[103] S. Heymans and D. Vermeir. Integrating Description Logics and Answer Set Programming. In Francóis Bry, Nicola Henze, and Jan Maluszynski, editors, *International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, number 2901 in LNCS, pages 146–159, Mumbai, India, December 2003. Springer. `http://tinf2.vub.ac.be/~sheymans/publications/ppswr2003.ps`.

[104] S. Heymans and D. Vermeir. Integrating Ontology Languages and Answer set Programming. In *Fourteenth International Workshop on Database and Expert Systems Applications*, pages 584–588, Prague, Czech Republic, September 2003. IEEE Computer Society. `http://tinf2.vub.ac.be/~sheymans/publications/webs2003.ps`.

[105] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proceedings ISWC-2003*, volume 2870 of *LNCS*, pages 17–29, 2003.

[106] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[107] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. FZI-Report 1-8-11/03, Forschungszentrum Informatik (FZI), 2003.

[108] G. Ianni, F. Calimeri, G. Ielpa, A. Pietramala, and M. C. Santoro. Enhancing answer set programming with templates. In *10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, pages 233–239, June 2004.

[109] Giovambattista Ianni, Giuseppe Ielpa, Francesco Calimeri, Adriana Pietramala, and Maria Carmela Santoro. A system with template answer set programs. In *Logics in Artificial Intelligence: 9th European Conference, JELIA 2004, LNCS 3229*, pages 693–697, September 2004.

[110] K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 44–49, 1999.

[111] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 2005. Accepted for publication. Preliminary version available at CoRR: cs.AI/0303009.

[112] T. Janhunen, I. Niemelä, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. In A.G. Cohn, F. Guinchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 411–419, Breckenridge, Colorado, USA, April 2000. Morgan Kaufman Publishers.

[113] A. C. Kakas and P. Mancarella. Generalized Stable Models: a Semantics for Abduction. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI '90)*, pages 385–391, 1990.

[114] A. C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44(1-3):129–177, 2000.

[115] A. C. Kakas and R. Miller. Reasoning about actions, narratives and ramifications. *Electronic Transactions on Artificial Intelligence*, 1(4), 1997.

[116] A. C. Kakas, B. Van Nuffelen, and M. Denecker. A-System: Problem Solving through Abduction. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 591–596, Seattle, WA, USA, 2001.

[117] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[118] Anthonis C. Kakas, Bert Van Nuffelen, and Marc Denecker. A-system : Problem solving through abduction. In B. Nebel, editor, *Proceedings of IJCAI'01 - Seventeenth International Joint Conference on Artificial Intelligence*, volume 1, pages 591–596. Morgan Kaufmann Publishers, Inc., 2001. [Link].

[119] David B. Kemp and Peter J. Stuckey. Semantics of Logic Programs with Aggregates. In Vijay A. Saraswat and Kazunori Ueda, editors, *Proceedings of the International Symposium on Logic Programming (ISLP'91)*, pages 387–401. MIT Press, 1991.

[120] K. Konczak, T. Schaub, and T. Linke. Graphs and colorings for answer set programming with preferences. *Fundamenta Informaticae*, 57(2-4):393–421, 2003. [PDF].

[121] K. Konczak, T. Schaub, and T. Linke. Graphs and colorings for answer set programming with preferences: Preliminary report. In M. De Vos and A. Provetti, editors, *Proceedings of the Second International Workshop on Answer Set Programming (ASP'03)*, volume 78, pages 43–56. CEUR Workshop Proceedings, 2003. [Link], [PDF].

[122] K. Konolige. Abduction versus closure in causal theories. *Artificial Intelligence*, 53(2-3):255–272, 1992.

[123] Els Laenens and Dirk Vermeir. Assumption-free semantics for ordered logic programs: On the relationship between well-founded and stable partial models. *Journal of Logic and Computation*, 2(2):133–172, 1992.

[124] Els Laenens and Dirk Vermeir. A Universal Fixpoint Semantics for Ordered Logic. *Computers and Artificial Intelligence*, 19(3), 2000.

[125] N. Leone and S. Perri. Parametric connectives in disjunctive logic programming. In *Answer Set Programming: Advances in Theory and Implementation (ASP03)*, pages 124–135, Messina, Italy, 2003.

[126] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, C. Koch, C. Mateis, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. Technical Report INFSYS RR-1843-02-14, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, October 2002.

[127] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2005. To appear. Available via `http://www.arxiv.org/ps/cs.AI/0211004`.

[128] A. Y. Levy and M. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *European Conference on Artificial Intelligence*, pages 323–327, 1996.

[129] V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[130] V. Lifschitz and H. Turner. From disjunctive programs to abduction. In *Non-Monotonic Extensions of Logic Programming*, pages 23–42, 1994.

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

[132] Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.

[133] F. Lin and J. You. Abduction in logic programming: A new definition and an abductive procedure based on rewriting. *Artificial Intelligence*, 140(1–2):175–205, September 2002.

[134] T. Linke. Using nested logic programs for answer set programming. In M. De Vos and A. Provetti, editors, *Proceedings of the Second International Workshop on Answer Set Programming (ASP'03)*, volume 78, pages 181–194. CEUR Workshop Proceedings, 2003. [Link], [PDF].

[135] T. Linke, A. Bösel, and C. Anger. `noMoRe` a graph-based system for non-monotonic reasoning with logic programs under answer set semantics, 2000–2003. [PDF].

[136] T. Linke, H. Tompits, and S. Woltran. On acyclic and head-cycle free nested logic programs. In B. Demoen and V. Lifschitz, editors, *Proceedings of 19th International Conference on Logic Programming (ICLP04)*, volume 3132 of *Lecture Notes in Computer Science*, pages 225–239. Springer-Verlag, 2004.

[137] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.

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

[139] W. Łukaszewicz. Considerations on default logic: An alternative approach. *Computational Intelligence*, 4(1):1–16, Jan. 1988.

[140] D. Makinson. General theory of cumulative inference. In M. Reinfrank, editor, *Proc. of the Second International Workshop on Non-Monotonic Reasoning*, volume 346 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer Verlag, 1989.

[141] A. Mikitiuk and M. Truszczyński. Constrained and rational default logics. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1509–1515, Montréal, 1995. Morgan Kaufmann Publishers.

[142] 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.

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

[144] B. Motik, R. Volz, and A. Maedche. Optimizing Query Answering in Description Logics using disjunctive deductive databases. In *Proc. of KRDB'03*, pages 39–50, 2003.

[145] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[146] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. In *Logic in Artificial Intelligence, JELIA 2002*, pages 432–443, Cosenza, Italy, September 2002. `http://tinfpc2.vub.ac.be/papers/jelia2002.ps.gz`.

[147] D. Van Nieuwenborgh and D. Vermeir. Order and Negation as Failure. In *Proceedings of the 19'th International Conference on Logic Programming (ICLP 2003)*, Lecture Notes in Computer Science, page to appear. Springer Verlag, 2003.

[148] D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs. In P. Stuckey, editor, *Proceedings of the International Conference on Logic Programming*, volume 2401, pages 405–420. Springer-Verlag, 2002. [PDF].

[149] D. Pearce, H. Tompits, and S. Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In Pavel Brazdil and Alípio Jorge, editors, *Proceedings of 10th Portuguese Conference on Artificial Intelligence (EPIA 01)*, volume 2258 of *Lecture Notes in Computer Science*, pages 306–320. Springer-Verlag, 2001.

[150] Nikolay Pelov. *Semantics of logic programs with aggregates.* Phd, Department of Computer Science, K.U.Leuven, Leuven, Belgium, apr 2004. 152 + x pages.

[151] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Translation of aggregate programs to normal logic programs. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming: Advances in Theory and Implementation*, volume 78 of *CEUR Workshop Proceedings*, pages 29–42, 2003. [Link].

[152] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Partial stable semantics for logic programs with aggregates. In Vladimir Lifschitz and Ilkka Niemelä, editors, *7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2004. [Link].

[153] Nikolay Pelov and Mirosław Truszczyński. Semantics of disjunctive programs with monotone aggregates - an operator-based approach. In *Proceedings of the 10th International Workshop on Non-monotonic Reasoning (NMR 2004), Whistler, BC, Canada*, pages 327–334, 2004.

[154] S. Perri, F. Scarcello, and N.Leone. Abductive logic programs with penalization: Semantics, complexity and implementation. *Journal of the Theory and Practice of Logic Programming*, Forthcoming. [PDF].

[155] `plp`. `http://www.cs.uni-potsdam.de/~torsten/plp`, 2002.

[156] D.L. Poole. What the lottery paradox tells us about default reasoning (extended abstract). In *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning*, Toronto, Ont., 1989.

[157] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[158] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[159] R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. of DL'99*, pages 160–164, 1999.

[160] C. Sakama and K. Inoue. Abductive logic programming and disjunctive logic programming: their relationship and transferability. *Journal of Logic Programming*, 44(1-3):75–100, 2000.

[161] V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. nlp: A compiler for nested logic programming. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 361 – 364. Springer-Verlag Heidelberg, 2003. [PDF].

[162] T. Schaub and K. Wang. A semantic framework for preference handling in answer set programming. *Theory and Practice of Logic Programming*, 3(4-5):569–607, 2003. [PDF].

[163] Torsten Schaub and Kewen Wang. A comparative study of logic programs with preference. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*, pages 597–602, Seatle, Washington, USA, August 2001. [PDF].

[164] P. Simons. Extending and implementing the stable model semantics. Doctoral Dissertation. Research report A58, Helsinki University of Technology, Helsinki, Finland, April 2000.

[165] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[166] `smodels`. `http://www.tcs.hut.fi/Software/smodels/`, 2002.

[167] V.S. Subrahmanian and Carlo Zaniolo. Relating Stable Models and AI Planning Domains. In Leon Sterling, editor, *Proceedings of the 12^{th} International Conference on Logic Programming*, pages 233–247, Tokyo, Japan, June 1995. MIT Press.

[168] T. Syrjänen. Omega-restricted programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 267–279, Vienna, Austria, September 2001. Springer-Verlag.

[169] T. Syrjänen. Cardinality constraint programs. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence, JELIA'04*, pages 187–199, Lisbon, Portugal, September 2004. Springer-Verlag.

[170] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[171] B. Van Nuffelen and A. C. Kakas. A-System: Declarative Programming with Abduction. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNCS*, pages 393–396, Vienna, Austria, 2001. Springer.

[172] Bert Van Nuffelen. *Abductive constraint logic programming: implementation and applications*. Phd, Department of Computer Science, K.U.Leuven, Leuven, Belgium, jun 2004. 315+xxii pages.

[173] D. Vermeir, E. Laenens, and D. Sacca. Extending logic programming. In *Proceedings of the SIGMOD Conference*, pages 184–193. ACM, 1990.