Tutorial

Answer Set Programming for the Semantic Web

Thomas Eiter, Roman Schindlauer (TU Wien)
Giovambattista Ianni (TU Wien, Univ. della Calabria)
Axel Polleres (Univ. Rey Juan Carlos, Madrid)

Supported by IST REWERSE, FWF Project P17212-N04, CICyT Project TIC-2003-9001-C02.

Answer Set Programming for the Semantic Web

Thomas Eiter[1], Giovambattista Ianni[1 2], Axel Polleres[3], and Roman Schindlauer[1]

[1] TU Wien, Austria
{eiter,roman}@kr.tuwien.ac.at
[2] Universitá della Calabria, Italy
ianni@mat.unical.it
[3] Universidad Rey Juan Carlos, Madrid, Spain
axel.polleres@urjc.es

Abstract. The purpose of this tutorial is to get the audience familiar with the Answer Set Programming (ASP) Paradigm in the perspective of its fruitful usage for Semantic Web applications. ASP is a declarative programming paradigm with its roots in Knowledge Representation and Logic Programming. Systems and languages based on ASP are ready for tackling many of the challenges the Semantic Web offers, and in particular, are good candidates for solving a variety of issues which have been delegated to the Rule/Logic Layers in the Semantic Web vision. ASP systems are scalable, allow to mix monotonic with nonmonotonic reasoning, permit to combine rules with ontologies, and can interface external reasoners. Moreover, ASP is especially tailored at solving configuration and matchmaking problems involving reasoning with preferences by featuring easy to use, fully declarative soft & hard constraint specification languages. We introduce the attendees to the ASP basics and its principal extensions tailored at Semantic Web applications. We discuss the current impact of Answer Set Programming in the Semantic Web Area and possible future directions. Applications and exercises are presented. The attendees will practice through an online interface using one of the state-of-the-art ASP solvers and some of its extensions.

Introduction

Answer Set Programming is nowadays a general term for a powerful Knowledge Representation (KR) and declarative programming paradigm which includes many nonmonotonic logic language features, as well as support for reasoning with constraints and preferences.

Roughly speaking, Answer Set Programming is a variant of Datalog with negation as failure, under the stable model semantics, where multiple *answer sets* (stable models) may be ascribed to a program [36]. This inherent nondeterminism has been extended by introducing formalisms that allow to filter out models by means of constraints or to select among different models by means of *soft* constraints or similar extensions [9, 62, 56, 34].

After some pioneering work on stable model computation [5, 69], research in this field produced several, mature, implemented systems featuring clear semantics and efficient program evaluation. We cite here *DisLog* [67], SLG [15], DisLoP [4], *rel_sat* [57], DeReS [16, 17], DC [22], QUIP [23], *psgrnd* and *aspps* [21], `NoMoRe` [3], ASSAT [54], Cmodels-3 [53], `GnT`/ Smodels [47], and DLV[50].

ASP has been recently used as a reliable specification tool in a number of promising applications. For instance several tasks in information integration, knowledge management, security management, configuration, which require complex reasoning capabilities, have been successfully tackled using ASP. Particularly, these are explored in several projects funded by the European Commission (see e.g. the projects WASP, INFOMIX, and ICONS [49, 46]).

The basic language flavor has been extended with strong negation, weak constraints [9], aggregates as known from database query languages[32], cardinality and weight constraints [62]. The fruitful combination of these features allowed ASP to become an important knowledge representation formalism for declaratively solving AI problems in areas including planning [24], diagnosis and information integration [49], and more. ASP development is the main subject of WASP (Working Group on Answer Set Semantics, IST-2001-37004). For a comprehensive report on recent ASP applications we refer to [72]. A showcase is available online at [73].

We can summarize the main features of ASP as follows:

– *Fully declarative.* ASP is fully declarative. The order of rules and atoms in a logic program is not important, and in general, no knowledge of the operational semantics a specific solver adopts is required.

– *Decidable*. ASP programs are, in their basic flavor, naturally decidable. No special restrictions are needed in order to keep this important property.
– *Monotonic and nonmonotonic*. ASP supports strong negation as well as negation as failure. By means of the latter default reasoning and nonmonotonic inheritance are enabled.
– *Nondeterministic*. It is possible to define concepts 'ranging' over a space of choices without any particular restriction. Extension of the basic semantics with preferences, soft and hard constraint, enable the compact specification of search and optimization problems.
– *Scalable*. Despite the computational expressiveness of ASP, state-of-the-art solvers currently reached the maturity for dealing with large datasets.

In the Semantic Web perspective, a great effort has been made in order to prove benefits of ASP in the Rules layer of the Semantic Web language stack and its interactions with the Ontology layer. Here, the semantic integration of ASP and Description Logics (which underlies OWL DL) deserves particular interest.

Organization

The tutorial will be divided in seven units:

1. **ASP Basics**. In this unit the attendees will become familiar with the basic notions and intuitive semantics of ASP. We will lean examples and exercises on small examples such as for instance the OWL Wine ontology in order to get users familiar to ASP.
   In this unit you will learn how to use ASP as a declarative KR formalism and query language involving non-determinism, disjunctive rules, strong and default negation and (hard) constraints.
2. **ASP Extensions**. In this unit several ASP advanced extensions will be tutored.
   Particularly, you will learn about extensions by aggregates known from database query languages; weak constraints, and how these can be used to encode preferences and priorities; rule templates which add a powerful macro-language to ASP.
3. **Current ASP State-of-the-Art**. In this section, we will present an ASP application showcase, demonstrating for instance an information integration scenario solved using the ASP methodology, (see for instance http://www.kr.tuwien.ac.at/projects/WASP/showcase.html), as well as pointers to relevant past and current literature, research directions, and available systems.
4. **Contribution of ASP to the Semantic Web field**. In this unit we will analyze how ASP fits in the semantic Web picture.
   You will learn about overlaps and differences between ASP and Semantic Web Knowledge Representation languages such as RDF and OWL. We will survey related works in this area and give you an idea of how ASP can fruitfully extend the Semantic Web stack towards the Rules Layer. Two concrete approaches will be introduced in the following two units.
5. **ASP Semantic Web Extensions Part I**. In this unit, we will present an extension of ASP which extend ASP engines with existing Description logics engines through so-called DL-atoms. We will show how this extension can be used to facilitate a clean interaction of Semantic Web languages with rules still keeping a strict semantic separation of both worlds.
6. **ASP Semantic Web Extensions Part II (45min)**. In this unit the notion of DL-atom is generalized to the one of arbitrary external predicates. Combined with higher order reasoning, this extension can be plugged into existing ASP engines facilitating new fruitful applications, especially in the Semantic Web context. We will show the variety of available plugins (such as RDF and OWL interfaces). As it turns out, the general plugin approach introduced in this unit caters for arbitrary plugins which enable integration of ASP with a variety of reasoners, services, allow to introduce reification, etc.
7. **Hands-On Session (45min)**. Each of the previous units was accompanied with small practical examples which you could follow over a Web-Interface to the state-of-the-art ASP engine DLV and the solvers DLT and DLVHEX. As a final part of the tutorial, the participants will practice and combine their experiences from the different units in several exercises.

Unit 1 – ASP Basics

T. Eiter

KBS Group, Institute of Information Systems, TU Vienna

European Semantic Web Conference 2006

Sudoku

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Task

Fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9

Unit Outline

Social Dinner Example

- A SW Summer School is planning to organize its social dinner.
- In order to make the attendees happy with this event and to make them familiar with ontologies, the organizers decide to ask them to declare their preferences about wines, in terms of a class description reusing the (in)famous Wine Ontology
- The organizers realize that only one kind of wine would not achieve the goal of fulfilling all the attendees' preferences.
- Thus, they aim at automatically finding the cheapest selection of bottles such that any attendee can have her preferred wine at the dinner.

The organizers quickly realize that several building blocks are needed to accomplish this task.

Wanted!

A general-purpose approach for modeling and solving these and many other problems

Issues:

- Diverse domains
- Spatial and temporal reasoning
- Constraints
- Incomplete information
- Preferences and priority
- Frame problem

Proposal:

Answer Set Programming (ASP) paradigm!

Logic Programming – Prolog revisited

Logic as a Programming Language (?)

Kowalski (1979):

ALGORITHM = LOGIC + CONTROL

- Knowledge for problem solving (LOGIC)
- "Processing" of the knowledge (CONTROL)

Roots of ASP – Knowledge Representation (KR)

How to model

- An agent's belief sets
- Commonsense reasoning
- Defeasible inferences
- Preferences and priority
- The Frame Problem

Approach

- use a logic-based formalism
- Inherent feature: nonmonotonicity

Many logical formalisms for knowledge representation have been developed.

Prolog

Prolog = "Programming in Logic"

- Basic data structures: terms
- Programs: rules and facts
- Computing: Queries (goals)
  - Proofs provide answers
  - SLD-resolution
  - unification - basic mechanism to manipulate data structures
- Extensive use of recursion

Simple Social Dinner Example

From simple.dlv:

- Wine bottles (brands) "a", ..., "e"
- plain ontology natively represented within the logic program.
- preference by facts

```
% A suite of wine bottles and their kinds
wineBottle("a").   isA("a","whiteWine").   isA("a","sweetWine").
wineBottle("b").   isA("b","whiteWine").   isA("b","dryWine").
wineBottle("c").   isA("c","whiteWine").   isA("c","dryWine").
wineBottle("d").   isA("d","redWine").   isA("d","dryWine").
wineBottle("e").   isA("e","redWine").   isA("e","sweetWine").

% Persons and their preferences
person("axel").   preferredWine("axel","whiteWine").
person("gibbi").   preferredWine("gibbi","redWine").
person("roman") . preferredWine("roman","dryWine").

% Available bottles a person likes
compliantBottle(X,Z) :- preferredWine(X,Y), isA(Z,Y).
```


Example: Recursion

```
append([],X,X) .
append([X|Y],Z,[X|T]) :- append(Y,Z,T) .

reverse([],[]).
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
```

- both relations defined recursively
- terms represent complex objects: lists, sets, ...

Problem:

Reverse the list `[a,b,c]`

Ask query: `?- reverse([a,b,c],X).`

- A proof of the query yields a substitution: `X=[c,b,a]`
- The substitution constitutes an answer

Prolog /2

The key: Techniques to search for proofs

- Understanding of the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

```
reverse([X|Y],Z) :- append(U,[X],Z), reverse(Y,U) .
    vs
reverse([X|Y],Z) :- reverse(Y,U), append(U,[X],Z) .
```

Query: `?- reverse([a|X],[b,c,d,b])`

Is this truly declarative programming?

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates declarative descriptions (definitions)
- Needed for programmers convenience

Clauses of the form:

$$p(\vec{X}) \text{:-} q_1(\vec{X_1}), \ldots, q_k(\vec{X_k}), \mathit{not}\ \ r_1(\vec{Y_1}), \ldots, \mathit{not}\ \ r_l(\vec{Y_l})$$

Things get more complex!

Negation in Prolog

- "*not* $(\cdot)$" means "Negation as Failure (to prove)"
- **Different from negation in classical logic!**

Example

```
compliantBottle("axel","a"),
bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
bottleSkipped(X) :- fail.  % dummy declaration
```

Query:

```
?- bottleChosen(X).
   X = "a"
```


Semantics of Logic Programs with Negation

Great Logic Programming Schism

Single Intended Model Approach:

- Select a single model of all classical models
- Agreement for so-called "stratified programs": " Perfect model"

Multiple Preferred Model Approach:

- Select a subset of all classical models
- Different selection principles for non-stratified programs

Programs with Negation /2

Modified rule:

```
compliantBottle("axel","a").
 bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
bottleSkipped(X) :- not bottleChosen(X), compliantBottle(Y,X).
```

Result ????

Problem: not a single minimal model!

Two alternatives:

- M_1= { `compliantBottle("axel","a"), bottleChosen("a")` },
- M_2 = { `compliantBottle("axel","a"), bottleSkipped("a")` }.

Which one to choose?

Stratified Negation

Intuition: For evaluating the body of a rule containing *not* $r(\vec{t})$, the value of the "negative" predicates $r(\vec{t})$ should be known.

1. Evaluate first $r(\vec{t})$
2. if $r(\vec{t})$ is false, then *not* $r(\vec{t})$ is true,
3. if $r(\vec{t})$ is true, then *not* $r(\vec{t})$ is false and rule is not applicable.

Example:

```
compliantBottle("axel","a"),
bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
```

Computed model M = { `compliantBottle("axel","a"), bottleChosen("a")` }.

Note: this introduces *procedurality* (violates declarativity)!

Program Layers

- Evaluate predicates bottom up in layers
- Methods works if there is no cyclic negation (layered negation)

Example:

```
compliantBottle("axel","a"). wineBottle("a"). expensive("a").
 bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
bottleSkipped(X) :- expensive(X), wineBottle(X).
```

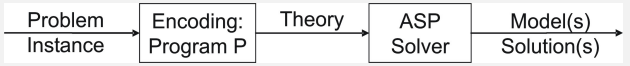
Unique model resulting by layered evaluation ("perfect model"):

M = { compliantBottle("axel","a"), wineBottle("a"), expensive("a"), bottleSkipped("a")}

Answer Set Programming Paradigm

General idea:

Reduce solving a problem instance I to computing models



1. *Encode* I as a (non-monotonic) logic program P, such that solutions of I are represented by models of P
2. *Compute* some model M of P, using an ASP solver
3. *Extract* a solution for I from M.

Variant: Compute multiple models (for multiple / all solutions)

Multiple preferred models

Unstratified Negation:

```
compliantBottle("axel","a").
 bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
bottleSkipped(X) :- not bottleChosen(X), compliantBottle(Y,X).
```

- Assign to a program (theory) not one but **several** intended models!
  - For instance, all answer sets
- How to interpret these semantics?
  - skeptical reasoning — resolution-based approaches complex
  - *preferred models represent different solutions to a problem*

Applications of ASP

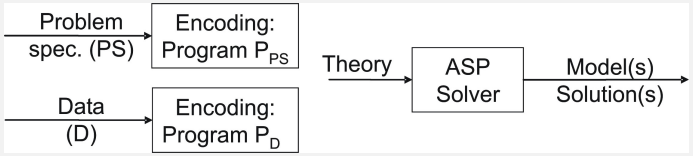
ASP facilitates *declarative problem solving*

Problems in different domains (some with substantial amount of data), see
http://www.kr.tuwien.ac.at/projects/WASP/report.html

- information integration
- constraint satisfaction
- planning, routing
- semantic web
- diagnosis
- security analysis
- configuration
- computer-aided verification
- ...

ASP Showcase: http://www.kr.tuwien.ac.at/projects/WASP/showcase.html

ASP in Practice



Uniform encoding:

Separate problem specification, *PS* and input data *D* (usually, facts)

- Compact, easily maintainable representation
- Integration of KR, DB, and search techniques
- Handling dynamic, knowledge intensive applications: data, defaults, exceptions, closures, ...

Example: Sudoku

Problem specification *PS*

$tab(i,j,n)$: cell (i,j), $i,j \in \{0,...,8\}$ has digit n

From sudoku.dlv:

```
% Assign a value to each field
tab(X,Y,1) v tab(X,Y,2) v tab(X,Y,3) v
tab(X,Y,4) v tab(X,Y,5) v tab(X,Y,6) v
tab(X,Y,7) v tab(X,Y,8) v tab(X,Y,9) :-
   #int(X),  0 <= X,  X <= 8, #int(Y),  0 <= Y,  Y <= 8.

% Check rows and columns
 :-  tab(X,Y1,Z), tab(X,Y2,Z), Y1<>Y2.
 :-  tab(X1,Y,Z), tab(X2,Y,Z), X1<>X2.

% Check subtable
 :-  tab(X1,Y1,Z), tab(X2,Y2,Z), Y1 <> Y2,
div(X1,3,W1), div(X2,3,W1), div(Y1,3,W2), div(Y2,3,W2).
 :-  tab(X1,Y1,Z), tab(X2,Y2,Z), X1 <> X2,
   div(X1,3,W1), div(X2,3,W1), div(Y1,3,W2), div(Y2,3,W2).

%Auxiliary: X divided by Y is Z
  div(X,Y,Z) :- XminusDelta = Y*Z,  X = XminusDelta + Delta, Delta < Y.
```


Sudoku (cont'd)

Data *D*:

```
% Table positions X=0..8, Y=0..8
tab(0,1,6). tab(0,3,1). tab(0,5,4). tab(0,7,5).
tab(1,2,8). tab(1,3,3). tab(1,5,5). tab(1,6,6).
...
```

Solution

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

ASP - Desiderata

Expressive Power

Capable of representing a range of problems, hard problems
Disjunctive ASP: NEXP^{NP}-complete problems !

Ease of Modeling

- Intuitive semantics
- Concise encodings: Availability of predicates and variables
  Note: SAT solvers do *not* support predicates and variables
- Modular programming: global models can be composed from local models of components

Performance

Fast solvers available

Social Dinner Example II

Extend the Simple Social Dinner Example (simple.dlv) to simpleGuess.dlv:

```
(1) doesNotLike(X,Z) :- person(X), wineBottle(Z),
                        not compliantBottle(X,Z).
% These rules generate multiple answer sets:
(2) bottleSkipped(X)  :-  not bottleChosen(X),
                          compliantBottle(Y,X).
(3) bottleChosen(X) :-  not bottleSkipped(X),
                        compliantBottle(Y,X).
% Ensure that each person gets a bottle.
(4) hasBottleChosen(X) :- bottleChosen(Z), compliantBottle(X,Z).
(5) :- person(X), not hasBottleChosen(X).
```

- Rule (1) defines doesNotLike using negation as failure on compliantBottle
- Rules (2) and (3) enforce that either `bottleChosen(X)` or `bottleSkipped(X)` is included in an answer set (but not both), if it contains `compliantBottle(Y,X)`.
- Rules (4) and (5) check that for each person some bottle must be chosen.

Example

```
compliantBottle("axel","a").  wineBottle("a").
bottleSkipped("a") :- not bottleChosen("a"),
                      compliantBottle("axel","a").
bottleChosen("a") :- not bottleSkipped("a"),
                     compliantBottle("axel","a").
hasBottleChosen("axel") :- bottleChosen("a"),
                           compliantBottle("axel","a").
```

- $HB(P) =$ { `wineBottle("a")`, `wineBottle("axel")`, `bottleSkipped("a")`, `bottleSkipped("axel")`, `bottleChosen("a")` `bottleChosen("axel")`, `compliantBottle("axel","a")`, `compliantBottle("axel","axel")`, ... `compliantBottle("a","axel")` }

Answer Set Semantics

- Variable-free programs first!
- Rules

$$a \text{ :- } b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n$$

  where all a, b_i, c_j are atoms
- a *normal logic program* P is a (finite) set of such rules
- $HB(P)$ is the set of all atoms with predicates and constants from P.

Answer Sets /2

Let

- P be a normal logic program
- $M \subseteq HB(P)$ be a set of atoms

Gelfond-Lifschitz (GL) Reduct P^M

The reduct P^M is obtained as follows:

1. remove from P each rule

$$a \text{ :- } b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n$$

   where some c_i is in M
2. remove all literals of form *not* p from all remaining rules

Answer Sets /3

- The reduct P^M is a Horn program
- It has the least model $lm(P^M)$

Definition

$M \subseteq HB(P)$ is an answer set of P if and only if $M = lm(P^M)$

Intuition:

- M makes an **assumption** about what is true and what is false
- P^M derives positive facts under the assumption of *not* $(\cdot)$ as by M
- If the result is M, then the assumption of M is "stable"

Examples

```
compliantBottle("axel","a"). wineBottle("a").
hasBottleChosen("axel") :- bottleChosen("a"),
                           compliantBottle("axel","a").
```

- P has no *not* (i.e., is Horn)
- thus, $P^M = P$ for every M
- the single answer set of P is $M = lm(P) =$ `{ wineBottle("a"), compliantBottle("axel","a") }`.

Computation of $lm(P)$

The least model of a *not*-free program can be computed by fixpoint iteration.

Algorithm Compute_LM(P)

Input: Horn program P;
Output: $lm(P)$

```
new_M := ∅;
repeat
  M := new_M;
  new_M := {a | a:-b_1,...,b_m ∈ P, {b_1,...,b_m} ⊆ M}
until new_M == M
return M
```


Examples II

```
(1) compliantBottle("axel","a"). wineBottle("a").
(2) bottleSkipped("a") :- not bottleChosen("a"),
                          compliantBottle("axel","a").
(3) bottleChosen("a") :- not bottleSkipped("a"),
                          compliantBottle("axel","a").
(4) hasBottleChosen("axel") :-  bottleChosen("a"),
                          compliantBottle("axel","a").
```

Take $M =$ `{ wineBottle("a"), compliantBottle("axel","a"), bottleSkipped("a") }`

- Rule (2) "survives" the reduction (cancel `not bottleChosen("a")`)
- Rule (3) is dropped

$lm(P^M) = M$, and thus M is an answer set

Examples III

```
(1) compliantBottle("axel","a"). wineBottle("a").
(2) bottleSkipped("a") :- not bottleChosen("a"),
                          compliantBottle("axel","a").
(3) bottleChosen("a") :- not bottleSkipped("a"),
                         compliantBottle("axel","a").
(4) hasBottleChosen("axel") :-  bottleChosen("a"),
                                compliantBottle("axel","a").
```

Take M = { wineBottle("a"), compliantBottle("axel","a"), bottleChosen("a"), hasBottleChosen("axel") }

- Rule (2) is dropped
- Rule (3) "survives" the reduction (cancel not bottleSkipped("a"))

$lm(P^M) = M$, and therefore M is another answer set

Programs with Variables

- Like in Prolog, consider Herbrand models only!
- Adopt in ASP: no function symbols ("Datalog")
- Each clause is a shorthand for all its ground substitutions, i.e., replacements of variables with constants

E.g., `b(X) :- not s(X), c(Y,X).`
is with constants "axel","a" short for:

```
b("a") :- not s("a"), c("a","a").
b("a") :- not s("a"), c("axel","a").
b("axel") :- not s("axel"), c("axel","axel").
b("axel") :- not s("axel"), c("axel","a").
```


Examples IV

```
(1) compliantBottle("axel","a"). wineBottle("a").
(2) bottleSkipped("a") :- not bottleChosen("a"),
                          compliantBottle("axel","a").
(3) bottleChosen("a") :- not bottleSkipped("a"),
                         compliantBottle("axel","a").
(4) hasBottleChosen("axel") :-  bottleChosen("a"),
                                compliantBottle("axel","a").
```

Take M = { wineBottle("a"), compliantBottle("axel","a"), bottleChosen("a"), bottleSkipped("axel"), hasBottleChosen("axel"), }

- Rules (2) and (3) are dropped

$lm(P^M)$ = { wineBottle("a"), compliantBottle("axel","a")} $\neq M$
Thus, M is not an answer set

Programs with Variables /2

- The *Herbrand base of P*, $HB(P)$, consists of all ground (variable-free) atoms with predicates and constant symbols from P
- The grounding of a rule r, *Ground*(r), consists of all rules obtained from r if each variable in r is replaced by some ground term (over P, unless specified otherwise)
- The grounding of program P, is *Ground*$(P) = \bigcup_{r \in P}$ *Ground*(r)

Definition

$M \subseteq HB(P)$ is an answer set of P if and only if M is an answer set of *Ground*(P)

Inconsistent Programs

Program

```
p :- not p.
```

- This program has NO answer sets
- Let P be a program and p be a new atom
- Adding

  ```
  p :- not p.
  ```

  to P "kills" all answer sets of P

Social Dinner Example II

Task

Add a constraint to simpleGuess.dlv in order to filter answer sets in which for some person no bottle is chosen

```
(1) doesNotLike(X,Z) :-  person(X), wineBottle(Z),
                                 not compliantBottle(X,Z).
% This rule generates multiple answer sets:
(2) bottleSkipped(X)  :-  not bottleChosen(X),
    compliantBottle(Y,X).
(3) bottleChosen(X) :- not bottleSkipped(X),
    compliantBottle(Y,X).
% Ensure that each person gets a bottle.
(4) hasBottleChosen(X) :- bottleChosen(Z),
                                  compliantBottle(X,Z).
(5) :- person(X), not hasBottleChosen(X).
```

Solution at simpleConstraint.dlv

Constraints

- Adding

  $$\texttt{p :- } q_1, \ldots, q_m \texttt{ , not } r_1, \ldots, \texttt{not } r_n, \texttt{ not p.}$$

  to P "kills" all answer sets of P that:
  - contain $q_1, \ldots, q_m$, and
  - do not contain $r_1, \ldots, r_n$
- Abbreviation:

  $$\texttt{:- } q_1, \ldots, q_m \texttt{ , not } r_1, \ldots, \texttt{not } r_n.$$

This is called a **"constraint"**

Main Reasoning Tasks

Consistency

Decide whether a given program P has an answer set.

Cautious (resp. Brave) Reasoning

Given a program P and ground literals $l_1, \ldots, l_n$, decide whether $l_1, \ldots l_n$ simultaneously hold in every (resp., some) answer set of P

Query Answering

Given a program P and non-ground literals $l_1, \ldots, l_n$ on variables $X_1, \ldots, X_k$, list all assignments of values ν to $X_1, \ldots, X_k$ such that $l_1\nu, \ldots, l_n\nu$ is cautiously resp. bravely true.

Answer Set Computation

Compute some / all answer sets of a given program P.

Simple Social Dinner Example – Reasoning

- For our simple Social Dinner Example (simple.dlv), we have a single answer set
- Therefore, cautious and brave reasoning coincides.
- *compliantBottle("axel","a")* is both a cautious and a brave consequence of the program.
- For the query *person(X)*, we obtain the answers *"axel", "gibbi", "roman"*.

ASP vs Prolog

Under answer set semantics,

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;

"Pure" declarative programming, different from Prolog

- no (unrestricted) function symbols in ASP solvers available (finitary programs; other work in progress)

Social Dinner Example II – Reasoning

For simpleGuess.dlv:

- The program has 20 answer sets.
- They correspond to the possibilities for all bottles being chosen or skipped.
- The cautious query *bottleChosen("a")* fails.
- The brave query *bottleChosen("a")* succeeds.
- For the nonground query *bottleChosen(X)*, we obtain under cautious reasoning an empty answer.

Disjunctive ASP

- The use of disjunction in rule heads is natural

```
man(X) v woman(X) :- person(X)
```

- ASP has thus been extended with disjunction

$$a_1 \vee a_2 \vee \cdots \vee a_k \text{ :- } b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n$$

- The interpretation of disjunction is "minimal" (in LP spirit)
- Disjunctive rules thus permit to encode choices

Social Dinner Example II – Disjunctive Version

Task

Replace the choice rules in simpleConstraint.dlv

```
bottleSkipped(X) :- not bottleChosen(X), compliantBottle(Y,X).
bottleChosen(X)  :- not bottleSkipped(X), compliantBottle(Y,X).
```

with an equivalent disjunctive rule

```
bottleSkipped(X) ∨ bottleChosen(X) :-compliantBottle(Y,X).
```

Solution at simpleDisj.dlv. This form is more natural and intuitive!

- Very often, disjunction corresponds to such cyclic negation
- However, disjunction is more expressive in general, and can not be efficiently eliminated

Example

```
(1) compliantBottle("axel","a"). wineBottle("a").
(2) bottleSkipped("a") v bottleChosen("a") :-
    compliantBottle("axel","a").
(3) hasBottleChosen("axel") :- bottleChosen("a"),
                               compliantBottle("axel","a").
```

This program contains no *not*, so $P^M = P$ for every M
Its answer sets are its minimal models:

- M_1 = { wineBottle("a"), compliantBottle("axel","a"), bottleSkipped("a") }
- M_2 = { wineBottle("a"), compliantBottle("axel","a"), bottleChosen("a"), hasBottleChosen("axel") }

This is the same as in the non-disjunctive version!

Answer Sets of Disjunctive Programs

Define answer sets similar as for normal logic programs

Gelfond-Lifschitz Reduct P^M

Extend P^M to disjunctive programs:

1. remove each rule in $Ground(P)$ with some literal *not* a in the body such that $a \in M$
2. remove all literals *not* a from all remaining rules in $Ground(P)$

However, $lm(P^M)$ does not necessarily exist (multiple minimal models!)

Definition

$M \subseteq HB(P)$ is an answer set of P if and only if M is a minimal (wrt. $\subseteq$) model of P^M

Properties of Answer Sets

Minimality:

Each answer set M of P is a minimal Herbrand model (wrt $\subseteq$).

Generalization of Stratified Semantics:

If negation in P is layered ("P is stratified"), then P has a unique answer set, which coincides with the perfect model.

NP-Completeness:

Deciding whether a normal propositional program P has an answer set is NP-complete in general.
$\Rightarrow$ Answer Set Semantics is an expressive formalism;
Higher expressiveness through further language constructs (disjunction, weak/weight constraints)

Answer Set Solvers

NP-completeness:

Efficient computation of answer sets is not easy!
Need to handle

1. complex data
2. search

Approach:

- Logic programming and deductive database techniques (for 1.)
- SAT/Constraint Programming techniques for 2.

Different sophisticated algorithms have been developed (like for SAT solving)
There exist many ASP solvers (function-free programs only)

Architecture of ASP Solvers

Typically, a two level architecture

1. Grounding Step

Given a program P with variables, generate a (subset) of its grounding which has the same models
DLV's grounder; lparse (Smodels), XASP, aspps
Special techniques used:

- "Safe rules" (DLV)
- domain-restriction (Smodels)

Answer Set Solvers on the Web

DLV	`http://www.dbai.tuwien.ac.at/proj/dlv/`
SModels	`http://www.tcs.hut.fi/Software/smodels/`
GnT	`http://www.tcs.hut.fi/Software/gnt/`
Cmodels	`http://www.cs.utexas.edu/users/tag/cmodels/`
ASSAT	`http://assat.cs.ust.hk/`
NoMore	`http://www.cs.uni-potsdam.de/~linke/nomore/`
XASP	distributed with XSB v2.6
	`http://xsb.sourceforge.net`
aspps	`http://www.cs.engr.uky.edu/ai/aspps/`
ccalc	`http://www.cs.utexas.edu/users/tag/cc/`

- Some provide a number of extensions to the language described here.
- Rudimentary extension to include function symbols exist ($\Rightarrow$ finitary programs, Bonatti)
- Answer Set Solver Implementation: see Niemelä's ICLP tutorial [61]

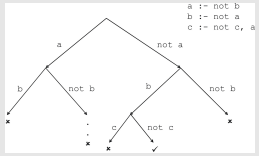
Architecture of ASP Solvers /2

2. Model search

This is applied for ground programs.

Techniques:

- Translations to SAT (e.g. Cmodels, ASSAT)
- Special-purpose search procedures (Smodels, dlv, NoMore, aspps)



- Backtracking procedures for assigning truth value to atoms
- Similar to DPPL algorithm for SAT Solving
- Important: Heuristics (which atom/rule to consider next)

Unit 2 – ASP Extensions

G. Ianni

Dipartimento di Matematica - Università della Calabria

European Semantic Web Conference 2006

Logic Programming Extensions

- Besides disjunction and strong negation, many extensions of normal logic programs have been proposed
- Some of these extensions are motivated by applications
- Some of these extensions are syntactic sugar, other strictly add expressiveness
- Comprehensive survey of extensions:
  See `http://www.tcs.hut.fi/Research/Logic/wasp/wp3/`
- Here, we consider some DLV specific extensions.

Unit Outline

Weak Constraints

- Allow the formalization of optimization problems in an easy and natural way.
- Constraints vs. weak constraints:
  - Constraints "kill" unwanted models;
  - Weak constraints express desiderata which should be satisfied, if possible.
- The answer sets of a program P with a set W of weak constraints are those answer sets of P which minimize the number of violated constraints.
- Such answer sets are called *optimal or best models of* (P, W).
- Other solvers feature similar constructs.

Syntax and Semantics

- Syntax:

  `:~` $b_1, \cdots, b_k$, `not` $b_{k+1}, \cdots,$ `not` b_m. `[Weight:Level]`
- In the presence of weights, best models minimize the sum of the weights of violated constraints.
- Semantics: minimizes the violation of constraints with highest priority level first; then with the lower priority levels in descending order.
- `Level` part is syntactic sugar, can be compiled into weights.

Weak Constraints: Examples

```
a v b.
c :- b.
:~ a.
:~ b.
:~ c.
```

Best model: a Cost ([Weight:Level]): <[1:1]> Answer set $\{b, c\}$ is discarded because it violates two weak constraints!

Weak Constraints: Examples /2

```
a v b.
:~ a. [1:]   :~ a. [1:]    :~ b. [2:]
```

Best model: b Cost ([Weight:Level]): <[2:1]>
Best model: a Cost ([Weight:Level]): <[2:1]>

```
a v b1 v b2.
:~ a. [:1]   :~ b1. [:2]   :~ b2. [:2]
```

Best model: a Cost ([Weight:Level]): <[1:1],[0:2]>

A bigger example - Employee Assignment

- Goal: Divide employees in two project groups p_1 and p_2[1].
  1. Skills of group members should be different.
  2. Persons in the same group should not be married to each other.
  3. Members of a group should possibly know each other.
- Requirement 1) is more important than 2) and 3), which are equally important
- Layers express the relative importance of the requirements.

```
assign(X,p1) v assign(X,p2) :-  employee(X).
:~ assign(X,P), assign(Y,P), same_skill(X,Y).    [:2]
:~ assign(X,P), assign(Y,P), married(X,Y).       [:1]
:~ assign(X,P), assign(Y,P), X!=Y, not know(X,Y).[:1]
```

[1] Example assignment.dlv

Guess-Check-Optimize Methodology

- Extend the "Guess & Check" Methodology
- Use weak constraints to filter out best (optimal) solutions

"Guess-Check-Optimize": Divide P into three main parts:

Guessing Part

$G \subseteq P$: $Answer\_Sets(G \cup F_I)$ represent "solution candidates" for instance I.

Checking Part (optional)

$C \subseteq P$: $Answer\_Sets(G \cup C \cup F_I)$ represent the admissible solutions for I.

Optimization Part (optional)

The optimization part $O \subseteq P$ consists of weak constraints, and implicitly defines an objective function $f : Answer\_Sets(G \cup C \cup F_I) \rightarrow \mathbb{N}$
Those answer sets minimizing f are selected.

Weak Constraints with Weights

- A single weak constraints in some layer n is more important than *all* weak constraints in lower layers ($n-1$, $n-2$,...) *together*!
- Weak constraints are weighted to make finer distinctions among elements of the same priority:
  `:~ G1.[3.5:1]  :~ G2.[4.6:1]`
- The weights of violated weak constraints are summed up for each layer.
- Example: *High School Time Tabling Problem*
  Structural Requirements > Pedagogical Requirements > Personal Wishes

Social Dinner III

Task

Now that we have defined `bottleChosen` *as the solution predicate, is there a way to select only the smallest sets of wines? Try to expand wineCover4.dlv*

```
:~ bottleChosen(X). [1:1]
```

Solution available as wineCover5.dlv

Traveling Salesperson

Given: Weighted directed graph $G = (V, E, C)$ and a node $a \in V$ of this graph.
Task: Find a minimum-cost cycle (closed path) in G starting at a and going through each node in V exactly once[2].

- G stored by facts over predicates `node(X)` and `arc(X,Y)`.
- Starting node a is specified by the predicate `start` (unary).

Guess:

```
inPath(X,Y,C) v outPath(X,Y,C) :- start(X), arc(X,Y,C).
inPath(X,Y,C) v outPath(X,Y,C) :- reached(X), arc(X,Y,C).
reached(X):- inPath(Y,X,C).
```

Check:

```
:- inPath(X,Y,_), inPath(X,Y1,_), Y <> Y1.
:- inPath(X,Y,_), inPath(X1,Y,_), X <> X1.
:- node(X), not reached(X).
```

Optimize:

```
:~ inPath(X,Y,C). [C:1]
```

[2] Example tsp.dlv

Social Dinner IV

Task

Let each wine bootle has a price encoded by `price(bottle,value)`. *Modify* `wineCover5b.dlv` *and try to choose the best cost selection of bottles.*

```
:~ bottleChosen(X),prize(X,N). [N:1]
```

Solution available at `wineCover5c.dlv`

Aggregate Atoms – Syntax

- **Symbolic Set:** Expression

  $$\{Vars : Conj\}$$

  of a list *Vars* of variables and a list *Conj* of literals (safety required) (e.g. `{ X : f(A,X,C), b(C,G) }`).
- **Aggregate Function:** Expression

  $$f\,\{Vars : Conj\}$$

  where
  - $f \in \{\#count, \#min, \#max, \#sum, \#times\}$, and
  - $\{Vars : Conj\}$ is a symbolic set (e.g. `#max{ X : f(A,X,C), b(C,G) }`)

Aggregates

- Compute aggregate functions over a set of values, similar as in SQL (count, min, max, sum)
- A few examples:

```
:- actiontime(T), #count{ move(B,L,T) } >= 4.
small :- #max{ X : f(A,X,C), b(C,G) } < 3.
ok_price :- 30 <= #sum{ Price :
                        bought(Good),
                        price(Good,Price) } < 50.
```

- other solvers (e.g. Smodels) offer similar constructs (cardinality atoms, weight constraints).

Aggregate Atoms – Syntax /2

- **Aggregate Atom:** Expression

  $$\begin{aligned} Agg\_Atom ::= \;& val \circledcirc f\,\{Vars : Conj\} \\ \mid\; & f\,\{Vars : Conj\} \circledcirc val \\ \mid\; & val_l \circledcirc_l f\,\{Vars : Conj\} \circledcirc_r val_u \end{aligned}$$

  where
  - val, val_l, val_u are constants or variables,
  - $\circledcirc \in \{<, >, \leq, \geq, =\}$,
  - $\circledcirc_l, \circledcirc_r \in \{<, \leq\}$, and
  - $f\,\{Vars : Conj\}$ is an aggregate function (e.g. `#max{ X : f(A,X,C), b(C,G) } < 3`)

Aggregate Atoms – Semantics

- Informally:
  Suppose I is an interpretation.
  - Evaluate symbolic set $\{Vars : Conj\}$ with respect to I: Collect all instances of $Vars$ for which $Conj$ is true in I (Result: $SemSet$).
  - Apply f on $SemSet$ (Result: $v = f(SemSet)$).
  - Evaluate comparison $val\ \theta\ v$ resp. $val_l\ \theta_l\ v \wedge v\ \theta_r\ val_u$ with (instantiated) value val resp. values val_l, val_u.
- Appealing formal definition of semantics is a bit tricky
- Widely acknowledged proposal: Faber et al. [32].

Social Dinner V

Task

Modify wineCover5c.dlv so that the weak constraint

```
:~ bottleChosen(X),prize(X,N). [N:1]
```

can be changed in

```
:~ totalcost(N). [N:1]

totalcost(N) :- #int(N),
     #sum{ Y : bottleChosen(X),prize(X,Y) } = N.
```

Solution at wineCover6.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[3].

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[3] Example seating.dlv

Frame logic: the idea

The molecular syntax typical of F-logic is quite useful for manipulating triple stores and complex join patterns:

Datalog Syntax

```
wineBottle("Brachetto").  isA("Brachetto","RedWine"),
isA("Brachetto","SweetWine").  prize("Brachetto",10).
```

F-Logic Syntax

```
"Brachetto" :  wineBottle[isA->>{"RedWine","SweetWine"},
prize->10].
```

Frame syntax: the idea

The molecular syntax typical of F-logic is quite useful for manipulating triple stores and complex join patterns:

Datalog Syntax

```
mainEntity(M) :- "foaf:PersonalProfileDocument"(X),
                 "foaf:primaryTopic"(X,M).
```

F-Logic Syntax

```
M : mainEntity :-
        X:"foaf:PersonalProfileDocument"["foaf:primaryTopic"->M].
```


Frame Spaces

A Frame Space directive tells how frames are mapped to regular atoms

```
@triple.
A[brother->B] :- A[father->Y],
                 B[father->Y].
```

Maps to:

```
brother(A,B,triple) :-
        father(A,Y,triple),
        father(B,Y,triple).
```

```
@.
A[brother->B] :- A[father->Y],
                 B[father->Y].
```

Maps to:

```
brother(A,B) :- father(A,Y),
                father(B,Y).
```


Informal Syntax and Semantics

F-Logic molecule

```
subject : type[predicate1->object, ...,
        predicate2->>{ object1, ..., objectn },
        ...]
```

It is a syntactic shortcut to

Datalog conjunction of facts

```
type(subject),
predicate1(subject,object), ... , predicate2(subject,object1),
                          ... , predicate2(subject, objectn)
```

- Objects can be nested frames (only atomic frames in rules' heads)
- **Subjects** and **Objects** unify with terms of the language. Under higher order extensions (see Unit 5), also **Predicates** and **Types** do.
- F-Logic semantic features (inheritance, etc.) are not currently implemented, this is only syntactic sugar.

Social Dinner VII

Task

Take wineCover7a.dlt. It is partially in frame syntax. Put the following rule in frame logic syntax:

```
compliantBottle(X,Z) :- preferredWine(X,Y), isA(Z,Y).
```

Solution at wineCover7b.dlt

The idea of templates

Imagine you want to encode all the possible permutations of a given predicate p (assume $maxint = |X : p(X)|$)

First, I guess worlds of permutations

```
permutation(X,N) v -permutation(X,N) :- p(X),#int(N).
```

Then, I cut worlds I don't like

```
:- permutation(X,A),permutation(Z,A), Z <> X.
:- permutation(X,A),permutation(X,B), A <> B.
```

Also, each element must be in the partition

```
covered(X) :- permutation(X,A).
:- p(X), not covered(X).
```


Syntax and Semantics

Template definition:

```
#template closure{e(2)}(2)
{
   closure(X,Y) :- e(X,Y).
   closure(X,Y) :- e(X,Z),
                   closure(Z,Y).
}
```

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

- `e(2), p(1)` = formal parameter list
- `..}(2), ..}(1)` = output predicate arities
- `closure, max` = output predicate names
- `exceeded` = local predicate name

The idea of templates - 2

- Thus, this "small" program encodes a search space of permutations
- But it can be reused and put in a library (let *maxint* big enough here)

```
#template permutation{p(1)}(2)
{
    permutation(X,N) v -permutation(X,N)
      :- p(X),#int(N),
         #count{ Y : p(Y) } = N1,
         N <= N1, N > 0.
    :- permutation(X,A),permutation(Z,A), Z <> X.
    :- permutation(X,A),permutation(X,B), A <> B.
    covered(X) :- permutation(X,A).
    :- p(X), not covered(X).
}
```


Syntax and Semantics - 2

Template atoms:

```
clo(X,Y) :- closure{ edge(*,*) }(X,Y).
inPath(X,N) :- permutation{ clo(*,$) }(X,N).
maxAgePerSex(S,A) :- max{ person($,S,*) }(A).
```

- `edge(*,*), clo(*,$), person($,S,*)` = actual parameters
- `closure{ edge(*,*) }(X,Y)` = a template atom
- `*` = input terms
- `$` = projection terms
- `S` = group-by (quantification) term
- `(X,Y), (X,N), S..A` = output terms

The Hamiltonian Path problem

HP: find a path between nodes of a graph s.t. I cross each node exactly once. (permutation.dlt)

If I want to encode the HP problem with templates, I can do this way:

```
path(X,N) :- permutation{node(*)}(X,N).
:- path(X,M), path(Y,N), not edge(X,Y), M = N+1.
```

Also, I can use permutation taking input predicates other than unary:

```
path(X,N) :- permutation{edge(*,$)}(X,N).
```

- `*` = parameter
- `$` = projection

Social Dinner VIII

Task

Try to expand wineCover7.dlt: define a template `subset` for specifying the search space of minimum cardinality subsets of wines.

```
#template subset{ p(1) }(1)
{
    subset(X) v nonsubset(X) :- p(X).
    :~ subset(X). [1:1]
}
bottleChosen(X) :- subset{compliantBottle($,*)}(X).
```

Solution at `wineCover8.dlt`

References

1. **Weak Constraints**: [11]
2. **Aggregates**: [32]
3. **Templates**: [13]
4. **Frame Logic**: [48]
5. **Other extensions**:
   `http://www.tcs.hut.fi/Research/Logic/wasp/wp3/`

Unit 3 – ASP: State of the Art and Applications

G. Ianni

Dipartimento di Matematica - Università della Calabria

European Semantic Web Conference 2006

A very active field

Major Scientific Events have ASP as hot topic

- Intl. Workshop on ASP ('01, '03 and '05)
- LPNMR, NMR, JELIA
- Special Issue on Answer Set Programming (ASP) in AMAI
- Working group on Answer Set Programming (WASP, 15+ nodes)

Mature Solvers

- DLV [35], Smodels [68]
- ASSAT, Cmodels, dcs, DeRes, DisLog, DisLop, NoMoRe, aspps, SLG

Unit Outline

ASP Points of strength

Totally declarative

Order of rules and atoms do not matter. You can ignore how the solver operates

Decidable

Prototypes started from Datalog without function symbols. Extensions keep decidability.

Monotonic and nonmonotonic

Negation as failure, as well as classic ("with strong semantics") negation

Nondeterministic

You can specify a set of possible worlds ("guesses") you want. Dealing with uncertainty is thus enabled.

ASP Points of strength - 2

Versatile

Weak and Soft constraints, useful special constructs with well-defined formal semantics

Scalable

Can compete with top-down solvers now

Interoperable

- External built-ins, External predicates
- DLV Java API and ODBC Interface
- RuleML schema for program exchange

Note that the price of each achievement in terms of research work is high in the context of ASP (full declarativity is a big design constraint), but it pays off

Applications

Hot Areas (non-complete list)

- **Configuration/composition**,
- **Information integration**,
- Security analysis,
- Agent systems,
- Semantic Web (see Units 4–6),
- Planning.

ASP is very well tailored at modelling problems that fits the G-C-O approach and need fast prototyping. In an increasing number of cases, ASP technologies can be kept in release versions of softwares they are embedded in.

Current state-of-the-art

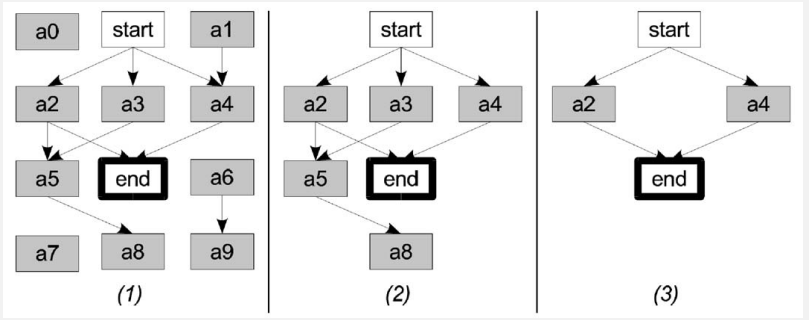
Semantics

- Introduction of Function Symbols [71, 14, 7]
- Introduction of various forms of *Generalized Quantifiers* (e.g. Aggregates [32, 55, 63, 26])
- Study of equivalence [25], and debuggers [30, 8]

Scalability

- Intelligent grounders, magic sets [51, 18]
- New heuristics for model generation [33, 31]
- Parallel execution [38]
- Intelligent reductions to SAT [37]

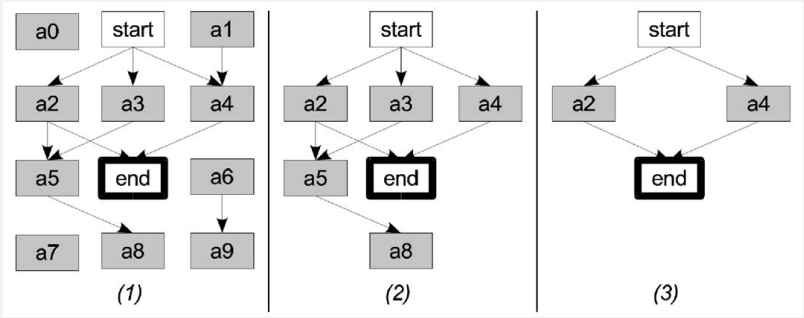
A Web Service Composition problem



Legenda

- Frame = Web Service
- Boxed White Frame = Final Goal
- $a \rightarrow b$ = Output of a fulfills input preconditions for b

A Web Service Composition problem - 2



Some assumption

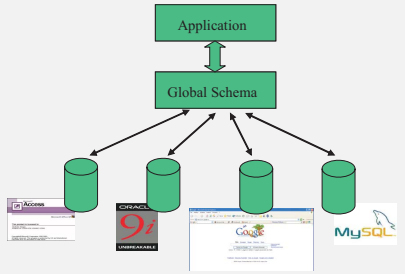
- Arrows are statically given.
- But they can come from any chosen semantic entailment.
- Also conjunctive conditions are possible (not shown).

Data Integration Systems

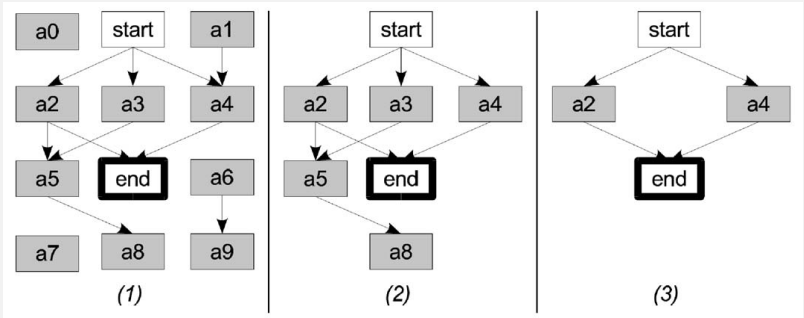
- Offer uniform access to a set of heterogeneous sources
- The representation provided to the user is called global schema
- The user is freed from the knowledge about data location and format

When the user issues a query over the global schema, the system:

- determines which sources to query and how
- issues suitable queries to the sources
- assembles the results and provides the answer



A Web Service Composition problem - 3



ASP role

- To design whatever strategy for execution plan generation.
- One can use Guess, Check, and Optimize methodology.
- [64] won the EEE-Web'05 WS contest.

The INFOMIX architecture

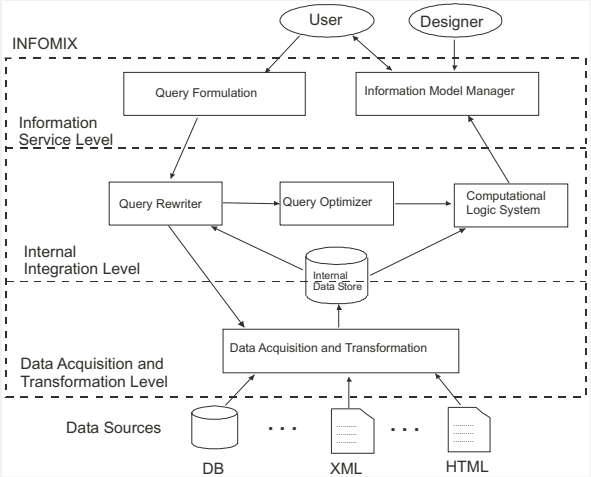
Three Layers:

Extraction:
Data Acquisition and Transformation

Processing:
Internal Integration Level

Frontend:
Information Service Level

The INFOMIX architecture - 2



Design Time - 2

The designer can specify also constraints on the global schema, e.g. `KeyConstraint(g,1)`

Data Sources

Ps1:

X	Y
1	2
2	3

Ps2:

X	Y
1	3
4	5

⇒ **GAV Mapping**

```
g(X,Y) :- ps1(X,Y).
g(X,Y) :- ps2(X,Y).
```

⇒ **Global Schema**

g:

X	Y
1	3
1	2
2	3
4	5

Conflict!!

Design Time

A designer specifies sources and mappings from sources to the global schema

Data Sources

Ps1:

X	Y
1	2
2	3

Ps2:

X	Y
1	3
4	5

⇒ **GAV Mapping**

```
g(X,Y) :- ps1(X,Y).
g(X,Y) :- ps2(X,Y).
```

⇒ **Global Schema**

g:

X	Y
1	3
1	2
2	3
4	5

GAV = Global as view

Run Time

When a query is submitted, this has to be unfolded to the sources and a merging program has to be processed. But query answering under constraints is a *NP*-hard problem also in the simpler settings.

Two possible repairs

g:

X	Y
1	3
1	2
2	3
4	5

⇒ **Definite answers (cautious reasoning)**

g:

X	Y
2	3
4	5

Idea: The query answering and conflict repair strategy can be programmed with ASP

How ASP comes into play

Bad News:

- Sources contain a huge amount of data
- Evaluating a co-NP hard problem is unfeasible

A simple program

```
p(X) v q(X) :- a(X).
```

Database and Query

Database:
`D = { a(1),a(2), ...,a(k) }`
Query: `p(1)`?

A brute force approach would consider k rules and $n = 2^k$ minimal models

Ground program

```
p(1) v q(1) :- a(1).
...
p(k) v q(k) :- a(k).
```

$\Rightarrow$

Stable Models

$M_1 : \{p(1), p(2), \ldots, p(k-1), p(k)\} \cup D$
$M_2 : \{p(1), p(2), \ldots, p(k-1), q(k)\} \cup D$
$\ldots$
$M_n : \{q(1), q(2), \ldots, q(k-1), q(k)\} \cup D$

Magic Sets approach has been extended to the ASP

Good News:

- Not all the data is necessary for answering user queries
- Magic sets can focus on relevant data
- Problems theoretically untractable become feasible

A simple program

```
p(X) v q(X) :- a(X).
```

Database and Query

Database:
`D = { a(1),a(2), ...,a(k) }`
Query: `p(1)`?

An intelligent approach could consider only one ground rule and 2 models

Ground program

```
p(1) v q(1) :- a(1).
```

$\Rightarrow$

Stable Models

$M_1 : \{p(1)\} \cup D$
$M_2 : \{q(1)\} \cup D$

Magic Sets

Given
Program P,
Query Q

$\Rightarrow$ **Magic Sets**
Program rewriting

$\Rightarrow$ **Rewritten Program**
$MS(Q, P)$

Magic sets:

- focus on the subset of P which is relevant for Q
- push down the query constants, to eliminate rule instances which cannot contribute to the derivation of Q
- simulate the top-down evaluation of Q

Applicability:

- Positive Programs (in the literature)
- Disjunctive programs (INFOMIX achievement)
- Programs with un-stratified negation (INFOMIX achievement)

The lesson of INFOMIX

- The INFOMIX prototype is the most expressive current system for consistent query answering under incompleteness
- Expresses the full range of queries (not only fragments), with different sorts of constraints (KDs, IDs, EDs)
- Rich Data Acquisition and Transformation Layer
- Fruitful use of computational logic (proof of concept)
- Experimental results are encouraging, scalability feasible
- Further efforts for optimizing data access (cf. constant pushing)
- Tighter coupling between CL system and relational engine

Live Demo

Now, let's play with the Live Demo

http://www.mat.unical.it/infomix

Unit4 – Contribution of ASP to the Semantic Web

A. Polleres

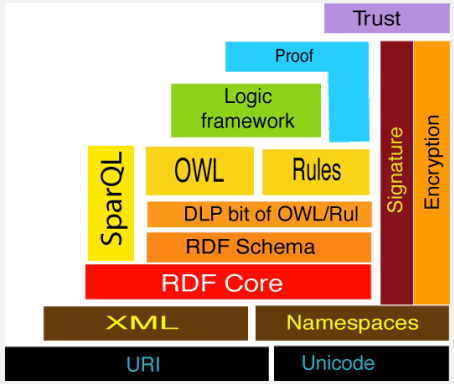
Universidad Rey Juan Carlos, Madrid

European Semantic Web Conference 2006

Introduction

In this unit, we give an overview of efforts and possibilities to deploy ASP related techniques in a Semantic Web context.

Question: Where does ASP fit in the "Layer Cake"?



Tim BL's famous, layer cake, latest version [6]

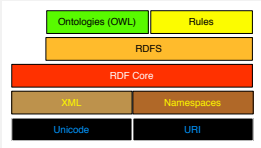
Unit Outline

Goals of this Unit:

- Learn about overlaps and differences between ASP and SW Knowledge Representation Languages.
- Get introduced to related works in this area.
- Get an idea of how ASP can fruitfully extend these languages.

The three questions treated in this Unit:

1. ASP and RDF/RDFS:
   1. What of RDF/S can be expressed directly in ASP?
   2. What is different? Blank nodes, XML Literals, etc.
   3. Related approaches: TRIPLE [19], Flora-2 [74, 75],
   4. RDF predicates in DLV (cf. Units 5 and 6)
2. ASP and OWL:
   1. What of OWL can be expressed directly in ASP?
   2. What is different? Existentials, number restrictions, equality reasoning, etc.
   3. Use ASP techniques for OWL reasoning: Baral[1], Swift[70], Hustadt, Motik, Sattler[45]
3. ASP and the Rules Layer:
   1. General undecidability [52, 44, 58]
   2. "Intersection" [39] or "two towers" [43]?
   3. The "safe interaction" approaches [52, 59, 65, 66, 42]
   4. The "safe interface" approach (cf. Units 5 and 6) [27, 19]



What of RDF/S can be expressed directly in ASP?(1/2)

The RDF data model RDF describes a labeled graph of resources (nodes) linked to other resources or literals by predicates.

- usually represented in form of triples $\langle Subject, Predicate, Object \rangle$ e.g.

```
http://www.polleres.net/index.html dc:creator http://www.polleres.net/foaf.rdf#me.
http://www.polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

```
<rdf:Description rdf:about="http://www.polleres.net/index.html">
  <dc:creator>
    <rdf:Description rdf:about="http://www.polleres.net/foaf.rdf#me">
      <foaf:Name>Axel Polleres</foaf:Name>
    </rdf:Description>
  </dc:creator>
</rdf:Description>
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://www.polleres.net/index.html","dc:creator","http://www.polleres.net/foaf.rdf#me").
triple("http://www.polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```


What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

  But: Be aware of UNA in ASP!

  Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://www.polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```

  When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
knowDifferentPeople(X,Y) :- triple(X,"foaf:knows",A),
                            triple(Y,"foaf:knows",B), A != B.
```

  Will return

```
(http://www.polleres.net/foaf.rdf#me, http://www.gibbi.com/foaf.rdf#me)
```

  as a valid pair.

  Why? '!=' in ASP means "not =" (Negation as failure of proof!)

RDFS semantics can (to a large extent) be captured by ASP style rules:

```
triple(P,rdf:type,rdf:Property) :- triple(S,P,O).
triple(S,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(O,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(S,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:domain,C).
triple(O,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:range,C).
triple(C,rdfs:subClassOf,rdfs:Resource) :- triple(C,rdf:type,rdfs:Class).
triple(C1,rdfs:subClassOf,C3) :- triple(C1,rdfs:subClassOf,C2),
                                 triple(C2,rdfs:subClassOf,C3).
triple(S,rdf:type,C2)         :- triple(S,rdf:type,C1),
                                 triple(C1,rdfs:subClassOf,C2).
triple(C,rdf:type,rdfs:Class) :- triple(S,rdf:type,C).
triple(C,rdfs:subClassOf,C)   :- triple(C,rdf:type,rdfs:Class).
triple(P1,rdfs:subPropertyOf,P3) :- triple(P1,rdfs:subPropertyOf,P2),
                                    triple(P2,rdfs:subPropertyOf,P3).
triple(S,P2,O)                :- triple(S,P1,O),
                                 triple(P1,rdfs:subPropertyOf,P2).
triple(P,rdfs:subPropertyOf,P)   :- triple(P,rdf:type,rdf:Property).
```

plus the respective axiomatic triples in RDF/RDFS, cf. Sections 3.1 and 4.1 of `http://www.w3.org/TR/rdf-mt/`.

- RDFS has infinitely many axiomatic Triples, e.g.

```
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
```

  Strictly, speaking, that means that we would always need to deal with an *infinite* Herbrand Universe, when dealing with RDF.

- Note the difference: `rdfs:domain` and `rdfs:range` restrictions boild down to RULES not to CONSTRAINTS. i.e.

```
triple(S,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:domain,C).
```

  is NOT the same as:

```
:- triple(S,P,O), triple(P,rdfs:domain,C) not triple(S,rdf:type,C).
```

  However, often people rather intend to model constraints when using RDFS, see [10]

Training Example

Learn how to import RDF data into DLV:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

```
#namespace(foaf,"http://xmlns.com/foaf/0.1/")

knownByMe(X) :- &rdf["http://polleres.net/foaf.rdf"]
                ("http://polleres.net/foaf.rdf#me","foaf:knows",X).
```

Naive solution available as knowing2.dlh

ASP and OWL

- OWL offers more expressivity than RDF/S!
- What of OWL can be expressed directly in ASP?
- What is different? Existentials, number restrictions, equality reasoning, etc.
- Approaches for using ASP-style techniques for OWL reasoning Alsac and Baral [1], Swift [70], Hustadt,Motik,Sattler [45], Heymans et al. [42]

Training Example

Learn how to import RDF data into DLV:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from ttp://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

```
#namespace(foaf,"http://xmlns.com/foaf/0.1/")

triple(X,Y,Z) :- &rdf["http://polleres.net/foaf.rdf"](X,Y,Z).
knownbyMe(X) :- triple(ID,"foaf:name","Axel Polleres"),
                triple(ID,"foaf:knows",ID2),
                triple(ID2,"foaf:name",X).
```

A bit more elegant: Solution knowing3.dlh

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://www.polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://www.polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$www.polleres.net \neq www.gibbi.com$
$www.polleres.net = platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$\geq 1 Year \sqsubseteq Paper$
$\top \sqsubseteq \forall Year.D_{xsd:integer}$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$isAuthorOf \equiv hasAuthor^-$

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper1).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper1,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where &datatype is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C1(X),..., Cn(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C1(X) :- A(X). ...Cn(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C1(X) v ...v Cn(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C1(X). ...A(X) :- Cn(X).`

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$

What of OWL cannot be expressed directly in ASP?

$A \equiv \{o_1, \ldots, o_n\}$ (owl:oneOf)	Cannot be directly translated... only approximated non-modularly if equality predicates allowed in rule bodies.
$A \sqsubseteq \bot$ (owl:Nothing)	`:- A(X).` is an Approximation only, doesn't work for complex concepts!

$A \sqsubseteq \exists R.C$ (owl:someValuesFrom rhs)	Impossible, we have no existentials in rule heads
$\forall R.C \subseteq A$ (owl:allValuesFrom lhs)	One might guess: `A(X) :- not noRC(X). noRC(X) :- R(X,Y), -C(Y).` but doesn't work :-(
cardinality restrictions, owl:sameAs, owl:differentFrom	Need reasoning with equality, expensive to implement. Recall: "=" and "!=" are not classical equality but builtin syntactic equality (UNA,CWA)!

... etc.

Main differences OWL vs. ASP?

- `not` in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - `not`: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	`Paper(paper1).`
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: `Unpublished(paper1).`

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation** are necessary!

But: not enough! ASPs is strong query ansering, algorithms not tailored for e.g. subsumption checking like DL's.

ASP for OWL reasoning? (1/4)

Several approaches in the literature [1, 70, 60, 45, 41, 42, 12], some of which we will discuss here in brief.

1) Alsac and Baral[1]: Encodes the Description Logics $\mathcal{ALCQI}$ in ASP.

- Realizes that naive translation is insufficient.
- Embedding in nondisjunctive ASP, using guesses by unstratified negation to emulate classical behavior, e.g.
  `paper(X ) :- top(X ), not not -paper(X).`
  `-paper(X ) :- top(X ), not paper(X ).`
- Facts are encoded as constraints, e.g.
  `:- not Paper(paper1).` instead of simply `Paper(paper1).`.
- Similarly Inclusion axioms $\sqsubseteq$ encoded as constraints, e.g.
  $Publication \sqsubseteq Paper$ becomes `:- Publication(X), not Paper(X).`
- for complex class descriptions, new predicates symbols are instroduced, e.g.

Problems:

- Keeps UNA (but so do prominent DL Reasoners like Racer (can be switched off), and FACT)
- Tailored for entailing facts assertions, function symbols needed for the general case, to emulate infinite domain (not supported by current ASP implementations).
- Not extensible to nominals in restrictions and enumerated classes. to emulate infinite domain (not supported by current ASP implementations).

ASP for OWL reasoning? (2/4)

2) Heymans, et al.[41] use a similar encoding of the DL $\mathcal{ALCHOQ}$, but with disjunction and "Open" answer sets.

- also keeps UNA
- no function symbols needed for the general case, instead relies on the (in general undecidable) open answer set semantics, which allows infinite, "open" domains.
- Evaluation algorithms and reductions of to existing ASP engines for decidable subsets described in [40].
- support for nominals and enumerated class again limited,

ASP for OWL reasoning? (3/4)

3) KAON approach to reduce DL reasoning to disjunctive Logic Programming, originally introduced by Motik et al. [60, 45], underlies the KAON2 system.

Remarks:

- Original approach was based on a limited translation of DL into disjunctive rules, including function symbols and a new predicate symbol for any complex class expression.
- Further optimized and developed [45] in the KAON2 system:
  - Novel implementation, not based on existing ASP solvers.
  - intermediate translation to first-order logic, clausal form transformation, function symbol elimination,
  - Algorithm based on basic superposition calculus for equality resoning, to overcome UNA.
  - Disjunctive Logic Programming as "encoding" of DL with the goal of an alternative OWL DL reasoner.
  - not really ASP in the sense presented in this Tutorial, to some extent at the cost of declarativity.
  - Also probably not extensible to nominals.
  - cf. Tutorial on KAON2 © this conference!

ASP for OWL reasoning? (4/4)

Summary:

- OWL does not really "fit" into ASP as such.
- Lossless encoding all of OWL into ASP is not only difficult, but also looses much of the declarativity and legibility of both formalisms (DL and ASP) for Knowledge Representation.

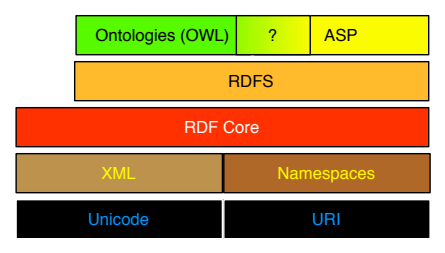
⇒ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!

- Still, an active research area from which interesting extensions of ASP itself (Open ASPs, Superposition Calculus for equality resoning etc.) arise!

ASP and the rules Layer

⇒ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!

ASP itself might be a good candidate for building a foundation of the rules layer!



But: It's not THAT easy!

ASP and the rules Layer

- Obstacles in Integrating ASP and Ontologies: Logic Programming vs. Classical Logic
  - Non-monotonicity of rules (Open world vs Closed World).
  - Equality vs. UNA.
  - Non-ground entailment.
- Strategies for combining rules and ontologies
  - Simple approaches
  - Safe interaction
  - Safe interface

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure not with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

... BUT: Non-ground entailment.

A set of Horn clauses is not the same as the corresponding logic program:

- Recall: Logic Progamming based semantics of ASP is defined in terms of minimal Herbrand models, i.e., sets of ground facts.

  ∀X potableLiquid(X) ← wine(X)
  ∀X wine(X) ← whiteWine(X)
  whiteWine("Welschriesling")

- Both the LP reading and the Horn clause reading of this yield the entailment of facts
  whiteWine("WelschRiesling"), wine("WelschRiesling"), potableLiquid("WelschRiesling").
- The Horn clauses furhter entail:

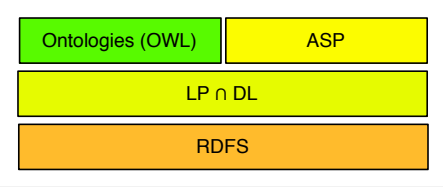
  wine("WelschRiesling") ← potableLiquid("WelschRiesling"),
  $\forall X$ .whiteWine(X) ← PotableLiquid(X).

- Logic Programs do not entail rules or other axioms, but only facts!

... Non-ground entailment, so what?

⇒ At least, we can say: Ontology reading and LP reading can interact on exchange of ground facts in the Horn **intersection** of OWL and ASP:

- i.e. the Horn fragment of $\mathcal{SHOIN}(D)$.



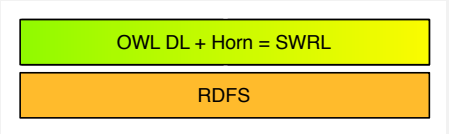
See DLP [39], and WRL [2] which extends DLP towards some features of ASP [1]

[1] not precisely true since WRL uses well-founded semantics

... Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44]takes this approach:

- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):
  shareFood(W1,W2) ← hasDrink(D,W1), hasDrink(D,W2)
  Whitewine ⊑ Wine
  "Trout grilled" ∈ Dish
  ("Trout grilled","WelschRiesling") ∈ hasDrink



- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

Well, there must be something in between, no?

Two basic approaches to retain decidability beyond DLP:

"Safe interaction" Rules interact with Ontologies in a common semantic framework, with syntactic restrictions

"Safe interface" Rules and Ontologies are kept strictly separate and only communicate via a *"safe interface"*, but do not impose syntactic restrictions on either the rules or the ontology part

Safe Interaction between LP and DL 1/2

Unrestricted recursive rules on top of DL cause the trouble of SWRL.
$\mathcal{AL}$-Log [20], extends the DL $\mathcal{AL}$ by Horn rules, with additional "safety" restriction:

> Every variable of a rule must appear in at least one of the rule atoms occurring in the body of R, where rule atoms are those predicates which do not appear in the DL Knowledge Base part, but only in rules.

This retains decidability!

Rules:

```
shareFood(W1,W2) ← hasDrink(D,W1), hasDrink(D,W2),
                   myWines(W1),myWines(W1).
cellarWine("Welchriesling").  cellarWine("Veltliner").  cellarWine("Zweigelt").
```

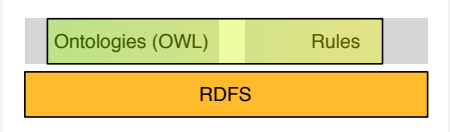
Ontology:

Whitewine $\sqsubseteq$ Wine
"Trout grilled" $\in$ Dish
⋮

Safe Interaction between LP and DL 2/2

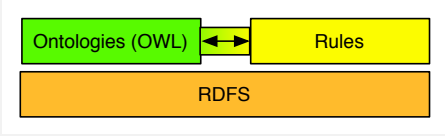
- The decidability for such so-called DL-safe rules was extended to $\mathcal{SHIQ}$ in Motik et al. [59]
- Heymans et al. [42] show decidability for query answering in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ DL-safe rules
- Rosati [65, 66] loosens the safety restriction further, by
  - allowing non-rule atoms also in rule heads, and
  - also provides an ASP style semantics for non-Horn rules.

All these approaches restrict either the DL, or rules or both:



Safe Interface between LP and DL – dl-programs

- *dl-programs* [27] Define an extension of ASP by so-called dl-atoms in rule bodies body, which allow to query a DL Reasoner, but also interchange facts in the other direction:



  - Decidability remains.
  - Full OWL DL and full ASP with all its extensions.
- Another approach in this direction: TRIPLE's [19] ability to query DL engines.

More on this in unit 5 and 6.

Unit 5 – An ASP Extension: Nonmonotonic DL-Programs

T. Eiter

KBS Group, Institute of Information Systems, TU Vienna

European Semantic Web Conference 2006

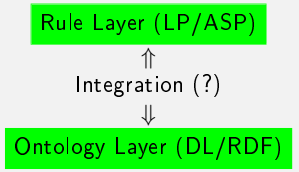
Unit Outline

Social Dinner Scenario (cont'd)

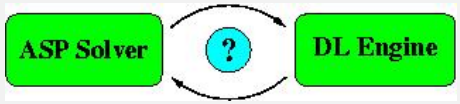
- Instead of a native, simple ontology inside the program, an external ontology should be used
- An ontology is available, formulated in OWL, which contains information about available wine bottles, as instances of a concept *Wine*.
- It has further concepts *SweetWine*, *DryWine*, *RedWine* and *WhiteWine* for different types of wine.

- How to use this ontology from the logic program ?
- How to ascribe a semantics for this usage?

Combining ASP and DL



- Add rules on top of DL, add ontologies to LP/ASP
- Safe Interface (hybrid) approach: Combine ASP and DL via interface



Nonmonotonic Description Logic Programs

- An extension of answer set programs with *queries to DL knowledge bases* (= DL-atoms)
- Formal semantics for emerging programs (*nonmonotonic DL-programs*), fostering the interfacing view

⇒ Clean technical separation of DL-engine and ASP solver.

New generalized definitions of:

- Least model of positive DL-program
- Iterative least model of stratified DL-program
- Answer sets of general DL-program

DL Atoms

Approach to enable a call to a DL-engine in ASP:

- Pose a query, Q, to a DL knowledge base, L
- Allow to modify the extensional part (ABox) of KB
- Query evaluates to true, iff Q is provable in modified L.

Examples: wine ontology

- $DL[Wine]("ChiantiClassico")$
- $DL[Wine](X)$
- $DL[DryWine \uplus my\_dry; Wine](W)$
  add all assertions $DryWine(c)$, such that $my\_dry(c)$ holds, to the ABox (extensional part) of L.

Nonmonotonic DL-Programs /2

Important: *bidirectional flow of information*

⇒ The logic program also may provide *input to DL knowledge base*

Prototype implementation, examples

`http://www.kr.tuwien.ac.at/staff/roman/semweblp/`

DL-Atoms /2

A *DL-atom* has the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t}), \qquad m \geq 0,$$

where

- each S_i is either a concept or a role
- $op_i \in \{\uplus, \uplus\}$,
- p_i is a unary resp. binary predicate (*input predicate*),
- $Q(\mathbf{t})$ is a *DL-query*.

Intuitively:

$op_i = \uplus$ increases S_i by p_i.
$op_i = \uplus$ increases $\neg S_i$ by p_i.

DL-Queries

A *DL-query* $Q(\mathbf{t})$ is one of

(a) a concept inclusion axiom $C \sqsubseteq D$, or its negation $\neg(C \sqsubseteq D)$,

(b) $C(t)$ or $\neg C(t)$, for a concept C and term t, or

(c) $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, for a role R and terms t_1, t_2.

Remarks:

- Further queries are conceivable (e.g., conjunctive queries)
- The queries above are standard queries.

DL-Programs

A *DL-rule* r is of form

$$a \leftarrow b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m, \qquad m \geq k \geq 0,$$

where

- a is a classical first-order literal
- $b_1, \ldots, b_m$ are classical first-order literals or DL-atoms (no function symbols).

Definition

A *nonmonotonic description logic (dl) program* $KB = (L, P)$ consists of

- a knowledge base L in a description logic ($\bigcup$ *Box),
- a finite set of DL-rules P.

Social Dinner IX

Task

Modify wineCover09a.dlp by fetching the wines now from the ontology.

For instance:

```
wineBottle(X) :- DL["Wine"](X).
```

Fetches all the known instances of *Wine*.

Think at how the "isA" predicate could be redefined in terms of DL atoms

```
isA(X,"SweetWine") :- DL[SweetWine](X).
isA(X,"DessertWine") :- DL[DessertWine](X).
isA(X,"ItalianWine") :- DL[ItalianWine](X).
```

Solution at wineCover9b.dlp

Social Dinner X

- Suppose now that we learn that there is a bottle, *"SelaksIceWine"*, which is a white wine and not dry.
- We may add this information to the logic program by facts[1]:

```
white("SelaksIceWine"). not_dry("SelaksIceWine").
```

- In our program, we may pass this information to the ontology by adding in the DL atoms the modification

WhiteWine⊎white,DryWine⊎not_dry.

E.g., "DL[Wine](X)" is changed to

```
DL[WhiteWine  += white, DryWine -= not_dry;Wine](X).
```

[1] See wineCover09c.dlp

Example: Review Assignment

It is given an ontology about scientific publications

- Concept *Author* stores authors
- Concept *Senior* (senior author)
- Concept *Club100* (authors with more than 100 paper)
- ...

- Goal: Assign submitted papers to reviewers
- Note: Precise definitions are not so important (encapsulation)

Review Assignment /2

Facts:

```
paper(subm1). author(subm1,"jdbr"). author(subm1,"htom").
paper(subm2). author(subm2,"ggot"). author(subm2,"gian").
author(subm2,"rsch"). author(subm2,"apol").
```

The program committee:

```
pc("vlif"). pc("mgel"). pc("dfen"). pc("fley"). pc("smil").
pc("mkif"). pc("ptra"). pc("ggot"). pc("ihor").
```

All PC members are in the "Club100" with more than 100 papers:

```
c1("club100",X) :- pc(X).
```

Consider all senior researchers as candidate reviewers adding the club100 information to the OWL knowledge base:

```
cand(X,P) :- paper(P), DL["club100" += c1;"senior"](X).
```


Review Assignment /3

Guess a reviewer assignment:

```
assign(X,P) :- not -assign(X,P), cand(X,P).
-assign(X,P) :- not assign(X,P), cand(X,P).
```

Check that each paper is assigned to at most one person:

```
:- assign(X,P), assign(X1,P), X1 != X.
```

A reviewer can't review a paper by him/herself:

```
:- assign(A,P), author(P,A).
```

Check whether all papers are correctly assigned (by projection)

```
a(P) :- assign(X,P).
error(P) :- paper(P), not a(P).
```


Semantics of $KB = (L, P)$

- HB_P^Φ: Set of all ground (classical) literals with predicate symbol in P and constants from finite relational alphabet Φ.
- Constants: those in P and (all) individuals in the ABox of L.
- Herbrand interpretation: consistent subset $I \subseteq HB_P^\Phi$
  - $I \models_L \ell$ for classical ground literal ℓ, iff $\ell \in I$;
  - $I \models_L DL[S_1 op_1\, p_1 \ldots, S_m op_m\, p_m; Q](\mathbf{c})$ if and only if

$$L \cup A_1(I) \cup \cdots \cup A_m(I) \models Q(\mathbf{c}),$$

    where
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \cup$.

- The models of $KB = (L, P)$ are the joint models of all rules in P (defined as usual)

Examples

- Suppose $L \models \mathit{Wine}(\text{“TaylorPort”})$, and I contains $\mathit{wineBottle}(\text{“TaylorPort”})$

  Then $I \models_L \mathit{DL}[\text{“Wine”}](\text{“TaylorPort”})$ and

  $$I \models_L \mathit{wineBottle}(\text{“TaylorPort”}) \text{ :- } \mathit{DL}[\text{“Wine”}](\text{“TaylorPort”})$$

- Suppose $I = \{\ \mathit{white}(\text{“siw”}),\ \ \mathit{not\_dry}(\text{“siw”})\ \}$.

  Then
  $I \models_L \mathit{DL}[\text{“WhiteWine”} \uplus \mathit{white}, \text{“DryWine”} \uplus \mathit{not\_dry}; \text{“Wine”}](\text{“siw”})$

  Note that if “siw” does not occur in L, then
  $I \not\models_L \mathit{DL}[\text{“WhiteWine”} \uplus \mathit{white}, \text{“DryWine”} \uplus \mathit{not\_dry}; \text{“Wine”}](\text{“siw”})$

Answer Sets

Answer Sets of positive $KB = (L, P)$ (no *not* in P):

- $KB = (L, P)$ has the least model $\mathit{lm}(KB)$ (if satisfiable)
- The single answer set of KB is $\mathit{lm}(KB)$

Answer Sets of general $KB = (L, P)$:

- Use a reduct KB^I akin to the Gelfond-Lifschitz (GL) reduct:

  $$KB^I = (L, P^I)$$

  where P^I is the GL-reduct of P wrt. I (treat DL-atoms like regular atoms)
- I is an answer set of KB iff $I = \mathit{lm}(KB^I)$.

Examples /2

- Suppose $L \not\models \mathit{DL}[\text{“Wine”}](\text{“Milk”})$. Then for every I,

  $$I \models_L \mathit{compliant}(\mathit{joe}, \text{“Milk”}) \text{ :- } \mathit{DL}[\text{“Wine”}](\text{“Milk”})$$
  $$I \models_L \mathit{not}\ \ \mathit{DL}[\text{“Wine”}](\text{“Milk”}).$$

- Note that $I \models_L \mathit{not}\ \ \mathit{DL}[\text{“Wine”}](\text{“Milk”})$ is different from $I \models_L \mathit{DL}[\neg\text{“Wine”}](\text{“Milk”})$.

- Inconsistency of L is revealed with unsatisfiable DL-queries:

  $$\mathit{inconsistent} \text{ :- } \mathit{DL}[\text{“Wine”} \sqsubseteq \neg\text{“Wine”}]$$

  Shorthand: $\mathit{DL}[\bot]$

- Consistency can be checked by

  $$\mathit{consistent} \text{ :- } \mathit{not}\ \ \mathit{DL}[\text{“Wine”} \sqsubseteq \neg\text{“Wine”}]$$

Some Semantical Properties

- *Existence:* Positive DL-programs without “$\neg$” and constraints always have an answer set
- *Uniqueness:* Layered use of “*not*” (stratified DL-program) $\Rightarrow$ single answer set
- *Conservative extension:* For DL-program $KB = (P, L)$ without DL-atoms, the answer sets are the answer sets of P.
- *Minimality:* answer sets of KB are models, and moreover minimal models.
- *Fixpoint Semantics:* Positive and stratified DL-programs with monotone DL-atoms possess fixpoint characterizations of the answer set.

Some Reasoning Applications

- DL-atoms allow to query description knowledge base repeatedly
- We might use DL-programs as rule-based "glue" for inferences on a DL-base.
- In this way, inferences can be combined
- Here, we show some applications where non-monotonic and minimization features of DL-programs can be exploited

Query Answering under CWA

Example: $L = \{\ SparklingWine(\text{"}VeuveCliquot\text{"}),\ (Sparklingwine \sqcap \neg WhiteWine)(\text{"}Lambrusco\text{"})\ \}.$

Query: $WhiteWine(\text{"}VeuveCliquot\text{"})$ (Y/N)?

Add CWA-literals to L:

$$\begin{aligned} \overline{sp}(X) &\leftarrow not\ DL[SparklingWine](X) \\ \overline{ww}(X) &\leftarrow not\ DL[WhiteWine](X) \\ ww(X) &\leftarrow DL[SparklingWine \uplus \overline{sp},\ WhiteWine \uplus \overline{ww};\ WhiteWine](X) \end{aligned}$$

Ask whether $KB \models ww(\text{"}VeuveCliquot\text{"})$ or $KB \models \neg ww(\text{"}VeuveCliquot\text{"})$

Closed World Assumption (CWA)

Reiter's Closed World Assumption (CWA)

For ground atom $p(c)$, infer $\neg p(c)$ if $KB \not\models p(c)$

- Express CWA for concepts $C_1, \ldots, C_k$ wrt. individuals in L:

$$\begin{aligned} \neg c_1(X) &\leftarrow not\ DL[C_1](X) \\ &\cdots \\ \neg c_k(X) &\leftarrow not\ DL[C_k](X) \end{aligned}$$

- CWA for roles R: easy extension

Extended CWA

- CWA can be inconsistent (disjunctive knowledge)
- **Example:**
  Knowledge base

$$L \ = \ \{\ Artist(\text{"Jody"}), Artist \equiv Painter \sqcup Singer\ \}$$

  - CWA for $Painter, Singer$ adds

$$\neg Painter(\text{"Jody"}), \neg Singer(\text{"Jody"}).$$

  - This implies $\neg Artist(\text{"Jody"})$

Minimal Models

- ECWA singles out "minimal" models of L wrt *Painter* and *Singer* (UNA in L on ABox):

$$\begin{aligned}
\overline{p}(X) &\leftarrow \mathit{not}\ p(X)\\
\overline{s}(X) &\leftarrow \mathit{not}\ s(X)\\
p(X) &\leftarrow DL[\mathit{Painter}\uplus\overline{p}, \mathit{Singer}\uplus\overline{s}; \mathit{Painter}](X)\\
s(X) &\leftarrow DL[\mathit{Painter}\uplus\overline{p}, \mathit{Singer}\uplus\overline{s}; \mathit{Singer}](X)\\
f &\leftarrow \mathit{not}\ f,\ DL[\bot] \quad /^*\ \text{kill model if } L \text{ is inconsistent } ^*/
\end{aligned}$$

  Answer sets:

$$\begin{aligned}
M_1 &= \{p(\text{“}\mathit{Jody}\text{”}), \overline{s}(\text{“}\mathit{Jody}\text{”})\},\\
M_2 &= \{s(\text{“}\mathit{Jody}\text{”}), \overline{p}(\text{“}\mathit{Jody}\text{”})\}
\end{aligned}$$

- Extendible to keep concepts "fixed"
  $\rightsquigarrow$ ECWA($\phi; P; Q; Z$)

OWL-DL with DL-Safe Rules

Motik et al. (2004)

- Allow rules

$$(*)\quad h(\vec{x}) \leftarrow b_1(\vec{x}_1), \ldots, b_k(\vec{x}_k), o(x_1), \ldots, o(x_m) \qquad m \geq 0,$$

  where
  - $h, b_1, \ldots, b_k$ are concept or role names,
  - $x_1 \ldots, x_m$ the variables in them, and
  - $o(x)$ holds for individuals x in the A-Box of L (*safety*).

- Rules have classical semantics (=clauses)

Default Reasoning

Add simple default rules a la Poole (1988) on top of ontologies

Example: wine ontology

$$\begin{aligned}
L = \{\ & \mathit{SparklingWine}(\text{“}\mathit{VeuveCliquot}\text{”}),\\
& (\mathit{SparklingWine} \sqcap \neg \mathit{WhiteWine})(\text{“}\mathit{Lambrusco}\text{”})\ \},
\end{aligned}$$

Use default rule: Sparkling wines are white by default

$$\begin{aligned}
r1: &\quad \texttt{white}(W) \leftarrow DL[\mathit{SparklingWine}](W), \mathit{not}\ \neg\texttt{white}(W)\\
r2: &\quad \neg\texttt{white}(W) \leftarrow DL[\mathit{WhiteWine} \uplus \texttt{white}; \neg\mathit{WhiteWine}](W)\\
r3: &\quad f \leftarrow \mathit{not}\ \ f,\ DL[\bot] \quad /^*\ \text{kill model if } L \text{ is inconsistent } ^*/
\end{aligned}$$

- In answer set semantics, $r2$ effects maximal application of $r1$.
- Answer Set: $M = \{\texttt{white}(\text{“}\mathit{VeuveCliquot}\text{”}), \texttt{-white}(\text{“}\mathit{Lambrusco}\text{”})\}$

DL-Safe Rules: Emulation with DL-programs

Method:

- Use For concept C / role R predicate p_C / p_R
- Guessing clauses for all concepts C / roles R:

$$\begin{aligned}
p_C(x) &\leftarrow \mathit{not}\ \overline{p_C}(x)\\
\overline{p_C}(x) &\leftarrow \mathit{not}\ p_C(x)\\
p_R(x,y) &\leftarrow \mathit{not}\ \overline{p_R}(x,y)\\
\overline{p_R}(x,y) &\leftarrow \mathit{not}\ p_R(x,y)
\end{aligned}$$

- Consistency test:

$$\leftarrow DL[\eta; \bot]$$

  where η consists of $C \uplus p_C, C \uplus \overline{p_R}$ / $R \uplus p_R, R \uplus \overline{p_R}$, for all concepts and roles.

DL-Safe Rules: Emulation with DL-programs /2

- Constraints for rules of form (*):

$$\leftarrow \overline{p_h}(\vec{x}), p_{b_1}(\vec{x}_1), \ldots, p_{b_k}(\vec{x}_k)$$

- Query:

$$query(\vec{t}) \leftarrow DL[\eta; Q](\vec{t})$$

- Apply cautious reasoning:

$$KB \models query(\mathbf{c})$$

Remark. (In)equality of individuals in the ABox may be explicitly addressed by guess and check (make assertions in the ABox).

Computational Complexity

Deciding strong answer set existence for DL-programs (completeness results)

$KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
positive	EXP	NEXP
stratified	EXP	$\mathrm{P}^{\mathrm{NEXP}}$
general	NEXP	$\mathrm{NP}^{\mathrm{NEXP}}$

Recall: Satisfiability problem in

- $\mathcal{SHIF}(\mathbf{D})$ / $\mathcal{SHOIN}(\mathbf{D})$ is EXP-/NEXP-complete (unary numbers).
- ASP is EXP-complete for positive/stratified programs P, and NEXP-complete for arbitrary P
- **Key observation**: The number of ground DL-atoms is polynomial
- $\mathrm{NP}^{\mathrm{NEXP}} = \mathrm{P}^{\mathrm{NEXP}}$ is less powerful than disjunctive ASP ($\equiv \mathrm{NEXP}^{\mathrm{NP}}$)
- Similar results for query answering

Further Aspects of DL-programs

- Stratified DL-programs: intuitively, composed of hierarchic layers of positive DL-programs linked via default negation.
  This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets.
- Non-monotonic DL-atoms: Operator $\curlywedge$

$$DL[WhiteWine \curlywedge my\_WhiteWine](X)$$

  Constrain *WhiteWine* to *my_WhiteWine*
- *Weak answer-set semantics* (Here: Strong answer sets)
  Treat also positive DL-atoms like *not*-literals in the reduct
- *Well-founded semantics*
  Generalization of the traditional well-founded semantics for normal logic programs.

NLP-DL Prototype

- Fully operational prototype: NLP-DL
  `http://www.kr.tuwien.ac.at/staff/roman/semweblp/`.
- Accepts ontologies formulated in OWL-DL (as processed by RACER) and a set of DL-rules, where $\leftarrow$, $\uplus$, and $\cup$, are written as ":-", "+=", and "-=", respectively.
- Model computation: compute
  - the answer sets
  - the well-founded model

  Preliminary computation of the well-founded model may be exploited for optimization.
- Reasoning: both *brave* and *cautious reasoning*; well-founded inferences

Unit 6 – Another ASP Extension: HEX-Programs

R. Schindlauer

Knowledge-bases Systems Group - TU Wien

European Semantic Web Conference 2006

Unit Outline

Motivation

- dl-programs: interfacing external source of knowledge
- Limited flexibility:
  - only one external KB possible
  - only one formalism allowed for KB (OWL)
- Spinning this idea further:
  - Access arbitrary external sources (solvers, services, different knowledge bases, etc.)
  - Standardized interface
  - Entire program: still ASP semantics
- Result: HEX-**programs**!

Motivation /2

Desirable Features for Rules in the Semantic Web:

- Software Interoperability
  - Importing external knowledge
  - Easily extendable reasoning framework
- Higher-Order Reasoning: rules that talk about predicates
  - Stating generic rules (e.g., general CWA rule)
  - Defining ontology semantics in a program

Our extension: *Higher-order logic programs with EXternal atoms* (HEX-**programs**) [26]

Syntax

Def. A HEX-**program** is a finite set P of rules:

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \mathit{not}\, \beta_{n+1}, \ldots, \mathit{not}\, \beta_m,$$

$m, k \geq 0$, where $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms.

Higher-Order Atoms are expressions of the form

$$(t_0, t_1, \ldots, t_n) \quad \text{resp.} \quad t_0(t_1, \ldots, t_n),$$

where $t_0, \ldots, t_n$ are (function-free) terms.

Examples: $(x, \mathit{rdf{:}type}, c)$, $\mathit{node}(X)$, $D(a, b)$.

Examples:

$\&\mathit{reach}[\mathit{edge}, a](X)$ ... reachable nodes from a in edge.

$\Rightarrow$ "Return 1 if $\langle a, X\rangle$ is in the extension of edge in I."

$\&\mathit{rdf}[\mathit{uri}](X, Y, Z)$ ... RDF-triples found under uri.

$\Rightarrow$ "Return 1 if $\langle X, Y, Z\rangle$ is an RDF-triple in uri."

(As already mentioned in Unit 4)

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&\mathit{sum}[p](X) \quad \Rightarrow \quad I : \{p(2), p(3), q(4)\} \quad \Rightarrow$ output list: 5
input list: p

Semantics

Define semantics of P in terms of its grounding $\mathit{grnd}(P)$.

$\Rightarrow$ *Herbrand base* HB_P of P: set of all groundings of atoms and external atoms in P (relative to set of constants $\mathcal{C}$).

- $I \subseteq HB_P$ models ground atom a, if $a \in I$
- $I \subseteq HB_P$ models ground $\&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ iff

  $$f_{\&g}(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1,$$

  where $f_{\&g}$ is an $(n+m+1)$-ary Boolean function telling whether $(x_1, \ldots, x_m)$ is in the output for input $I, y_1 \ldots, y_n$.
- $I \subseteq HB_P$ models P iff it models $\mathit{grnd}(P)$
- $I \subseteq HB_P$ is an answer set of P iff I is a minimal model of fP^I, where fP^I is the FLP-reduct of P.

Semantics top-down

If I is an answer set of a program P, then:

1. I is a minimal model of fP^I.
2. I is a model of each ground rule r in fP^I.
3. I is a model of r, iff either I is a model of the head or not a model of the body of r.
4. I is a model of the head h of r iff at least one of the atoms in h occurs in I.
5. I is a model of the body of r, iff it is a model of all positive literals in b and not a model of all negative literals in b.
6. I is a model of an atom a, iff $a \in I$.
7. I is a model of an external atom $\&g$, iff $f_{\&g}(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$,.

Applications /2

- Defining ontology semantics:

$$\begin{aligned} D(X) &\leftarrow subClassOf(D, C), C(X). \\ &\leftarrow maxCardinality(C, R, N), C(X), \\ &\quad \&count[R, X](M), M > N. \end{aligned}$$

- Closed World reasoning

$$\begin{aligned} cwa(married, single) &\leftarrow . \\ C'(X) &\leftarrow not\ \&DL[C](X), \\ &\quad concept(C), cwa(C, C'), \end{aligned}$$

Applications

- Importing external theories, stored in RDF:

$$\begin{aligned} triple(X, Y, Z) &\leftarrow \&rdf[<uri1>](X, Y, Z); \\ triple(X, Y, Z) &\leftarrow \&rdf[<uri2>](X, Y, Z); \\ proposition(P) &\leftarrow triple(P, rdfs{:}type, rdf{:}Statement). \end{aligned}$$

$\Rightarrow$ Avoid inconsistencies when merging ontologies O_1, O_2.

- Translating and manipulating reified assertions:

$$\begin{aligned} (X, Y, Z) &\leftarrow pick(P), triple(P, rdf{:}subject, X), \\ &\quad triple(P, rdf{:}predicate, Y), \\ &\quad triple(P, rdf{:}object, Z); \\ C(X) &\leftarrow (X, rdf{:}type, C). \end{aligned}$$

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety Each variable in a rule must occur in a positive body literal.

Expansion safety The input list of an external atom must be bounded:

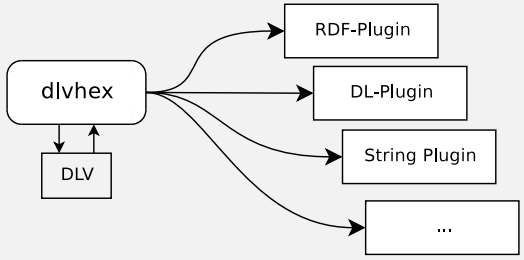
$$\begin{aligned} triple(S, P, O) &\leftarrow \&rdf[U](S, P, O), uri(U), known(U). \\ uri(X) &\leftarrow triple(\_, \text{"}seeAlso\text{"}, X). \end{aligned}$$

Safe.

dlvhex

We implemented a reasoner for HEX-programs, called **dlvhex** [29].
⇒ Command line application, that interfaces DLV and plugins for external atoms used in a program.

Design principle:



String Plugin

Purpose: String operations on names.

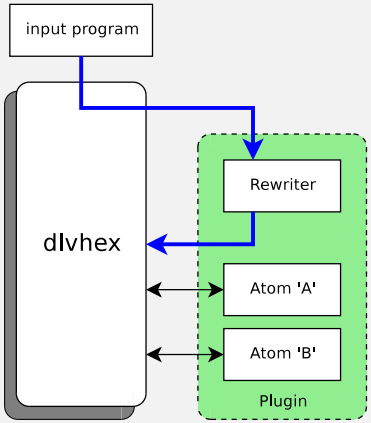
Available atoms:

`&concat`	Concatenates two strings.
`&cmp`	Compares two strings lexicographically or two numbers arithmetically.
`&strstr`	Tests two strings for substring inclusion.
`&split`	Splits a string along a specified delimiter.
`&sha1sum`	Computes SHA1 checksum from a string.

Plugin Mechanism

- A plugin is a shared library, dynamically linked at runtime
- A plugin can provide several Atoms
- A plugin can rewrite the program

Plugin development toolkit available!



String Plugin Atoms

`&concat[A,B](C)` builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

`&cmp[A,B]` is true if `A < B`.

Example:
```
before(X,Y) :- &cmp[X,Y], date(X), date(Y).
date("2006-06-18").  date("2006-06-20").
```

`&strstr[A,B]` is true if A occurs in B.

Example:
```
isURL(X) :- &strstr["http://",X].
```

String Plugin Atoms /2

`&split[A,B,C](D)` splits `A` by delimiter `B` and returns item `C`.

Example:
```
month(X) :- &split[D,"-",1](X), date(D).
date("2006-06-18").
```

`&sha1sum[A](B)` returns `B` as SHA1 checksum of `A`.

Example:
```
ownerID(X) :- &sha1sum[X](Y), mailbox(X).
```


RDF Plugin

Purpose: Access RDF knwoledge bases.

Available atom:

`&rdf` Retrieve RDF-triples from a URI.

`&rdf[U](S,P,O)` returns all triples <S,P,O> from `U`.

Example:
```
tr(X,Y,Z) :- &rdf[U](X,Y,Z), uri(U).
uri("http://www.example.org/foaf.rdf").
```


RDF Plugin Example

Program `delicious_a.dlh` retrieves triples from a del.icio.us URI.

Del.icio.us is a social bookmarking service: Users store their bookmarks and tag them with keywords. It has an RDF/RSS-interface: adding a keyword to the URL `http://del.icio.us/rss/tag/` returns all bookmarks with this tag.

- The namespace directive abbreviates long strings, simple syntactic sugar.
- The single URL given returns all bookmarks with the tag `eswc`.

Exercise

Task (1)

1. *Introduce a new predicate* `keyword` *and find a way to append its extension to the string* `"http://del.icio.us/rss/tag/"` *in order to build the URI in a more flexible way.*
2. *To get the actual bookmarks corresponding to a keyword, extract from the triples all resources that have* `"rdf:type"` *as property and* `"rss:item"` *as value.*

```
tag("eswc").
url(X) :- &concat["http://del.icio.us/rss/tag/",W](X), tag(W).
link(X) :- "rdf:type"(X,"rss:item").
```

Solution available as delicious_b.dlh

DL Plugin

Purpose: Query Description Logics knowledge bases.

Available atoms:

`&dlC`	Queries a DL concept.
`&dlR`	Queries a DL role.
`&dlDR`	Queries a DL datatype role.
`&dlConsistent`	Tests a DL KB for consistency.

These atoms descent from the corresponding DL atoms of our dl-programs and also allow for extending the DL-KB.

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)` Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to $\neg$`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to $\neg$`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```


DL Plugin Atoms /2

`&dlR[U,a,b,c,d,Q](X,Y)` Returns all pairs of `Q` in KB `U`.

`Q` has to be an ObjectProperty!

Example:

```
uncle(X,Y) :- &dlR[U,x,x,x,x,"brotherOf"](X,Z),
              &dlR[U,x,x,x,x,"parentOf"](Z,Y), url(U).
```

`&dlDR[U,a,b,c,d,Q](X,Y)` Returns all pairs of `Q` in KB `U`.

`Q` has to be a DatatypeProperty!

Example:

```
name(X,Y) :- &dlDR[U,a,b,c,d,"name"](X,Y),
             member(X), url(U).
```


Exercise

Wine example: importing wine preferences from (RDF) foaf-descriptions!

RDF-graph: `X <"foaf:name"> Name`
`X <"foafplus:winePreference"> Wine`

Task (2)

Modify wineCover10a.dlht by creating a predicate `preferredWine` *that associates names to wines.*

```
preferredWine(N,W) :- "foaf:name"(X,N),
                      "foafplus:winePreference"(X,W).
```

Solution at wineCover10b.dlht

Policy Specification

Recent project using dlvhex: **Policy Specification**

P. A. Bonatti, D. Olmedilla, and J. Peer.:
Advanced Policy Queries. For: European Commission, IST 2004-506779 (REWERSE), I2-D4, 2005.

Principle:

- Grant access to resources based on disclosed credentials.
- Various combinations of credentials might lead to the same goal.
- Credentials have specific disclosure sensitivities.
- **Optimization Problem:** Find least sensitive combination of credentials that grant access!

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  ⇒ Use aggregates
- Complicated: based on respective credentials, mutual effects
  ⇒ Use external atom!

`&policy[sens](S)` returns overall sensitivity of pred. `sens`.

`sens` is binary, associating a sensitivity value to a credential.

Example:
```
sens(ca,2).  sens(cb,3).
overall(S) :- &policy[sens](S).
```

Function inside the `&policy`-atom easily adaptable.

Policy Function Implementation: Sum

Simple version: Sum of all credential sensitivities—looking inside the plugin:

```
double
PolicySensFunction::eval(const std::vector<double>& values)
{
    double ret(0);

    for (vector<double>::const_iterator di = values.begin();
         di != values.end();
         ++di)
    {
        ret += *di;
    }

    return ret;
}
```


Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials.

Task (3)

1. *remove models without granted access (strong constraint):* *For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

```
1 :- not allow(download,R), availableFor(R,_).
2 sensitivity(S) :- &policy[sens](S).
3 :~ sensitivity(S). [S:1]
```

Solution at policy_b.dlh

Unit 7 – Hands-On Session

European Semantic Web Conference 2006

Unit 7: Hands-On Session

Each of the previous units was accompanied with small practical examples which you could follow over the Web-Interface to DLV.

Now: Try yourself!

Practice and combine your experiences from the different units in several exercises. Your tutor has the details!

Discover how to manipulate your online calendar ICAL/RDF data from an ASP application.

Specify an appointment matching strategy using declarative programming.

Grab a Tutor and get started!

The tutors will provide sets of new examples to be solved and can reexplain exercises from the different units. Don't hesitate to ask questions and let us know your opinions!

Information on the Presenters

Thomas Eiter

Affiliation: Knowledge Based Systems Group, Institute of Information Systems - Vienna University of Technology.

Home Page: `http://www.kr.tuwien.ac.at/staff/eiter/`

Short Bio: Prof. Dr. Thomas Eiter is a full professor (since 1998) in the Faculty of Informatics at Vienna University of Technology (TU Wien), Austria and head of the Institute of Information Systems. Before (1996-1998), he was an associate professor of Computer Science at the University of Giessen, Germany. Dr. Eiter's current research interests include knowledge representation and reasoning, logic programming, database foundations, knowledge-based agents, complexity in AI, and logic in computer science. He has more than 150 publications in these areas, many of which appeared in top journals and conferences. He has been involved in a number of national and international research projects, including the EU Networks of Excellence Compulog, CologNet, and REWERSE, and the EU Working Group WASP. He is a PC co-chair of RuleML 2006, and co-chaired in the past KI 2001, LPNMR 2001, FOIKS 2002, and ICDT 2005. He is on the advisory boards of the Journal of Artificial Intelligence Research (JAIR) and the Journal on Theory and Practice of Logic Programming (TPLP), and a former associate editor of the IEEE Transactions on Knowledge and Data Engineering (TKDE).

Giovambattista Ianni

Affiliation: Department of Mathematics at University of Calabria, Italy.

Home page: `http://www.gibbi.com`

Short Bio: Dr. G. Ianni has received his Ph.D. in Computer Science Engineering in 2002 at University of Calabria. Since 2002 he is Assistant Professor at the Department of Mathematics of University of Calabria, where he collaborates inside the DLV developers team, one of the major state-of-the-art ASP systems, and is responsible of the design and implementation of some of its language extensions. Dr. Ianni has been Visiting Researcher at the Institute for Information Systems of the TU-Wien from August 2004 to August 2005. He still collaborates with the Institute in the context of the project 'Answer Programming for the Semantic Web', where extensions of ASP for coping with Semantic Web are investigated. Dr. Ianni published more than 30 articles in journals and conferences. He has been organizing co-chair of the JELIA 2002 and LPNMR 2005 conferences, as well as editor of the JELIA 2002 proceedings. He served and serves in several program committees such as RuleML 2006, ECAI 2006, WI/IAT 2005, NMR 2004, AGP 2003. He is currently project manager of the EU INFOMIX project and has been involved in a number of national and international research projects, including the EU Working Group WASP. Dr. Ianni is interested in Extensions of ASP, Logic Programming and nonmonotonic reasoning, Ontology Languages, Logics in Artificial Intelligence, Rules and Semantic Web.

Axel Polleres

Affiliation: Universidad Rey Juan Carlos, Austria. **Home page**: `http://www.polleres.net`.
Short Bio: Dr. Axel Polleres studied Computer Science at the Vienna University of Technology where he finished his M.Sc. in 2001. From 2001 until 2003 he worked as a research assistant at the Institute of Information Systems at the same university. He received his Ph.D. on the topic of planning under uncertainty using Answer Set Programming in October 2003. During this period he developed the planning front-end for the DLV system. He was working as a postdoctoral researcher and leader of the RSWS cluster at DERI Innsbruck from 2003 which he still collaborates with. Dr. Polleres published more than 20 articles in journals, books and as refereed Conference and Workshop contributions. In Spring 2006 he started a research fellowship at Universidad Rey Juan Carlos, Spain, where he works in the areas of Semantic Web Services, Ontologies, and ASP. Main research topics are Logic Programming, Ontology Languages, Ontological Reasoning, Semantic Web, Semantic Web Services Description. Ongoing research projects and working groups he is participating in include WSMO, WSML, and the W3C Rule Interchange Format (RIF) working group. Dr. Polleres represented DERI Innsbruck in the Advisory Committee of the W3C between January 2005 and January 2006, where he was responsible for two W3C member submissions and acted as a local organizer of the W3C workshop on "Frameworks for Semantic Web Services" in June 2005 in Innsbruck. He will be the main organizer of the international workshop on "Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS2006)" at the Federated Logic Conference (FLOC2006) this August.

Roman Schindlauer

Affiliation: Knowledge Based Systems Group, Institute of Information Systems - Vienna University of Technology.
Home Page: `http://www.kr.tuwien.ac.at/staff/roman/`.
Short Bio: Roman Schindlauer is a PhD student at the Institute of Information Systems at Vienna University of Technology (TU Wien), Austria. He has been working in the area of Semantic Web and Answer Set Programming since 2002. He is the main responsible for the development of the DLV-HEX and the NLP-DL system and participates in the Austrian national project 'Answer Programming for the Semantic Web' as well as in the European network of excellence REWERSE.

References

1. G. Alsaç and C. Baral. Reasoning in description logics using declarative logic programming. Technical Report ASU 2001-02, Arizona State University, 2002.
2. J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web rule language (WRL), Sept. 2005. W3C Member Submission, http://www.w3.org/Submission/WRL/.
3. Christian Anger, Kathrin Konczak, and Thomas Linke. `NoMoRe`: A System for Non-Monotonic Reasoning. In *LPNMR'01*, pp. 406–410. 2001.
4. Chandrabose Aravindan, J. Dix, and I. Niemelä. DisLoP: A Research Project on Disjunctive Logic Programming. *AI Communications*, 10(3/4):151–165, 1997.
5. C. Bell, A. Nerode, R.T. Ng, and V.S. Subrahmanian. Mixed Integer Programming Methods for Computing Nonmonotonic Deductive Databases. *JACM*, 41:1178–1215, 1994.
6. T. Berners-Lee. Web for real people, April 2005. Keynote Speech at the 14th World Wide Web Conference (WWW2005), slides available at http://www.w3.org/2005/Talks/0511-keynote-tbl/.
7. P. A. Bonatti. Reasoning with Infinite Stable Models. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*, pages 603–610, Seattle, WA, USA, Aug. 2001. Morgan Kaufmann Publishers.
8. M. Brain and M. D. Vos. Debugging logic programs under the answer set semantics. In *Answer Set Programming*, 2005.
9. F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE TKDE*, 12(5):845–860, 2000.
10. J. D. Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. In *Proceedings of the 14th World Wide Web Conference (WWW2005)*, Chiba, Japan, May 2005.
11. F. Buccafurri, N. Leone, and P. Rullo. Strong and Weak Constraints in Disjunctive Datalog. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in LNAI, pages 2–17, Dagstuhl, Germany, July 1997. Springer.
12. P. Burek and R. Grabos. Dually structured concepts in the semantic web: Answer set programming approach. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, LNCS 3532*, pages 377–391, 2005.
13. F. Calimeri, G. Ianni, G. Ielpa, A. Pietramala, and M. C. Santoro. A system with template answer set programs. In *JELIA*, pages 693–697, 2004.
14. F. Calimeri and G. Ianni. External sources of computation for Answer Set Solvers. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings*, volume 3662 of *LNCS*, pages 105–118. Springer, Sept. 2005.
15. Weidong Chen and David Scott Warren. Computation of Stable Models and Its Integration with Logical Query Processing. *IEEE TKDE*, 8(5):742–757, 1996.
16. Paweł Cholewiński, V. Wiktor Marek, and M. Truszczyński. Default Reasoning System DeReS. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*, pp. 518–528, Cambridge, Massachusetts, USA, 1996.
17. Paweł Cholewiński, V.W. Marek, Artur Mikitiuk, and M. Truszczyński. Computing with Default Logic. *Artificial Intelligence*, 112(2–3):105–147, 1999.
18. C. Cumbo, W. Faber, and G. Greco. Enhancing the magic-set method for disjunctive datalog programs. In *Proceedings of the the 20th International Conference on Logic Programming – ICLP'04*, volume 3132 of *LNCS*, pages 371–385, 2004.
19. S. Decker, M. Sintek, and W. Nejdl. The model-theoretic semantics of TRIPLE, Nov. 2002.
20. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems (JIIS)*, 10(3):227–252, 1998.
21. D. East and M. Truszczyński. Propositional Satisfiability in Answer-set Programming. In *Proceedings of Joint German/Austrian Conference on Artificial Intelligence, KI'2001*, pp. 138–153. LNAI 2174, 2001.
22. D. East and M. Truszczyński. dcs: An Implementation of DATALOG with Constraints. In *NMR'2000*, 2000.
23. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In *AAAI'00*, pp. 417–422. AAAI Press / MIT Press, 2000.
24. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A Logic Programming Approach to Knowledge-State Planning, II: the DLV$^{\mathcal{K}}$ System. *Artificial Intelligence*, 144(1–2):157–211, 2003.
25. T. Eiter, M. Fink, H. Tompits, and S. Woltran. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *AAAI*, pages 695–700, 2005.

26. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *International Joint Conference on Artificial Intelligence (IJCAI) 2005*, pages 90–96, Edinburgh, UK, Aug. 2005.
27. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada*, pages 141–151, 2004. Extended Report RR-1843-03-13, Institut für Informationssysteme, TU Wien, 2003.
28. Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective Integration of Declarative Rules with external Evaluations for Semantic Web Reasoning. In *European Semantic Web Conference 2006, Proceedings*. To appear.
29. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Towards efficient evaluation of HEX programs. In *Proceedings 11th International Workshop on Nonmonotonic Reasoning (NMR-2006), Answer Set Programming Track*, June 2006. To appear.
30. O. El-Khatib, E. Pontelli, and T. C. Son. Justification and debugging of answer set programs in asp. In *AADEBUG*, pages 49–58, 2005.
31. W. Faber, N. Leone, and G. Pfeifer. A Comparison of Heuristics for Answer Set Programming. In *Proceedings of the 5th Dutch-German Workshop on Nonmonotonic Reasoning Techniques and their Applications (DGNMR 2001)*, pages 64–75, Apr. 2001.
32. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In J. J. Alferes and J. Leite, editors, *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004)*, number 3229 in LNAI, pages 200–212. Springer, Sept. 2004.
33. W. Faber, N. Leone, and F. Ricca. Heuristics for Hard ASP Programs. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1562–1563, Aug. 2005.
34. P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *TPLP*, 5(1–2):45–74, 2005.
35. W. Faber and G. Pfeifer. `DLV` homepage, since 1996. http://www.dlvsystem.com/.
36. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *NGC*, 9:365–385, 1991.
37. E. Giunchiglia and M. Maratea. On the relation between answer set and sat procedures (or, between cmodels and smodels). In *ICLP*, pages 37–51, 2005.
38. J. Gressmann, T. Janhunen, R. E. Mercer, T. Schaub, S. Thiele, and R. Tichy. Platypus: A platform for distributed answer set solving. In *LPNMR*, pages 227–239, 2005.
39. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary*, pages 48–57, 2003.
40. S. Heymans. *Decidable Open Answer Set Programming*. PhD thesis, Theoretical Computer Science Lab (TINF), Department of Computer Science, Vrije Universiteit Brussel, Feb. 2006.
41. S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Semantic web reasoning with conceptual logic programs. In *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop (RuleML 2004)*, pages 113–127, Hiroshima, Japan, Nov. 2004.
42. S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005. LNCS 3532*, pages 392–407, 2005.
43. I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers? In F. Fages and S. Soliman, editors, *Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, number 3703 in LNCS, pages 37–41. SV, 2005.
44. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml, May 2004. W3C Member Submission. http://www.w3.org/Submission/SWRL/.
45. U. Hustadt, B. Motik, and U. Sattler. Reducing shiq-description logic to disjunctive datalog programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada*, pages 152–162, 2004.
46. ICONS homepage, since 2001. http://www.icons.rodan.pl/.
47. T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J.-H. You. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM TOCL*, 2005. To appear.
48. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
49. N. Leone, G. Gottlob, R. Rosati, T. Eiter, W. Faber, Michael Fink, Gianluigi Greco, G. Ianni, Edyta Kałka, Domenico Lembo, Maurizio Lenzerini, Vincenzino Lio, Bartosz Nowicki, Marco Ruzzi, Witold Staniszkis, and Giorgio Terracina. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pp. 915–917, Baltimore, Maryland, USA, 2005. ACM Press.
50. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL*, 2005. To appear.

51. N. Leone, S. Perri, and F. Scarcello. Backjumping techniques for rules instantiation in the dlv system. In *NMR*, pages 258–266, 2004.
52. A. Y. Levy and M.-C. Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165 – 209, 1998.
53. Yuliya Lierler. Disjunctive Answer Set Programming via Satisfiability. In *Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, 2005, Proceedings*, LNCS 3662
54. F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.
55. V. Marek, I. Niemelä, and M. Truszczyński. Logic programs with monotone cardinality atoms. In *Proceedings LPNMR-2004*, volume 2923 of *LNCS*, pages 154–166, 2004.
56. V.W. Marek and J.B. Remmel. On Logic Programs with Cardinality Constraints. In *NMR'2002*, pp. 219–228, 2002.
57. N. McCain and H. Turner. Satisfiability Planning with Causal Theories. In *KR'98*, pp. 212–223. 1998.
58. J. Mei, S. Liu, A. Yue, and Z. Lin. An extension to OWL with general rules. In *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop (RuleML 2004)*, pages 155–169, Hiroshima, Japan, Nov. 2004.
59. B. Motik, U. Sattler, and R. Studer. Query answering for owl-dl with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.
60. B. Motik and R. Volz. Optimizing query answering in description logics using disjunctive deductive databases. In F. Bry, C. Lutz, U. Sattler, and M. Schoop, editors, *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003)*, volume 79 of *CEUR Workshop Proceedings*. CEUR-WS.org, Sept. 2003.
61. I. Niemel. The implementation of answer set solvers, 2004. Tutorial at ICLP 2004. Available at http://www.tcs.hut.fi/ init/papers/niemela-iclp04-tutorial.ps.gz/.
62. I. Niemelä, P. Simons, and T. Soininen. Stable Model Semantics of Weight Constraint Rules. In *LPNMR'99*, pp. 107–116.
63. N. Pelov. Non-monotone Semantics for Logic Programs with Aggregates. Available at http://www.cs.kuleuven.ac.be/ pelov/papers/nma.ps.gz., Oct. 2002.
64. A. Rainer. Web Service Composition under Answer Set Programming. In *KI-Workshop "Planen, Scheduling und Konfigurieren, Entwerfenl" (PuK)*, 2005.
65. R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1):61–73, 2005.
66. R. Rosati. $\mathcal{DL}$+*log*: Tight integration of description logics and disjunctive datalog. In *KR2006*, 2006. To appear.
67. D. Seipel and Helmut Thöne. DisLog – A System for Reasoning in Disjunctive Deductive Databases. In *Proceedings International Workshop on the Deductive Approach to Information Systems and Databases (DAISD'94)*, pp. 325–343. Universitat Politecnica de Catalunya (UPC), 1994.
68. P. Simons. Smodels Homepage, since 1996. http://www.tcs.hut.fi/Software/smodels/.
69. V.S. Subrahmanian, D. Nau, and C. Vago. WFS + Branch and Bound = Stable Models. *IEEE TKDE*, 7(3):362–377, 1995.
70. T. Swift. Deduction in Ontologies via ASP. In V. Lifschitz and I. Niemelä, editors, *Proc. of the Seventh Int.l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*, LNCS, pages 275–288, Fort Lauderdale, Florida, USA, Jan. 2004. Springer.
71. T. Syrjänen. Omega-restricted logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, Vienna, Austria, September 2001. Springer.
72. WASP report (IST-2001-37004). Model applications and proofs-of-concept. http://www.kr.tuwien.ac.at/projects/WASP/report.html.
73. WASP showcase. http://www.kr.tuwien.ac.at/projects/WASP/showcase.html.
74. G. Yang and M. Kifer. On the semantics of anonymous identity and reification. In *CoopIS/DOA/ODBASE*, pages 1047–1066, 2002.
75. G. Yang, M. Kifer, and C. Zhao. "flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web.". In R. Meersman, Z. Tari, and D. C. Schmidt, editors, *CoopIS/DOA/ODBASE*, pages 671–688, 2003.