Declarative Knowledge Processing

Lecture 9: Answer Set Programming

Magdalena Ortiz

Credits: Thank you to Thomas Eiter, who is the author of a good amount of today’s slides

Knowledge Base Systems Group
Institute of Information Systems

ortiz@kr.tuwien.ac.at

1 December 2010
Today’s lecture

We have discussed in detail **Description Logics**

- very powerful formalisms for Knowledge Representation and Reasoning
- the most prominent language for describing conceptualizations of application domains by means of ontologies

However, DLs have certain limitations that make them insufficient for certain applications

---

Today we talk about some of the things that we can not do in DLs
The things we can not do
Today’s lecture (cont’d)

But we will not only talk about things we can not do

We will instead discuss another formalism

Answer Set Programming (ASP)

- It overcomes many of the practical limitations of DLs, and
- it also plays a very prominent role in Knowledge Representation and Reasoning
Recommended reading

Outline

1. Motivation
1.1 Roots of ASP
1.2 Negation in Logic Programs
1.3 Stratified Negation

2. Answer Set Programming
2.1 ASP Basics
2.2 Disjunctive ASP
2.3 Answer Set Solvers

3. Beyond the core ASP: Extensions and frontends

4. Summary
Some limitations of DLs

- DLs are in general fragments of classical FOL
- This guarantees many positive properties of DLs
- But it also implies that DLs inherit the limitations of classical logic

A major limitation of classical FOL and DLs in the context of KR&R is their monotonicity
Monotonicity

Monotonic Logics

A logic (or more specifically, and inference relation) is **monotonic** if for every pair of theories $T, T'$ and formula $\varphi$,

$$T \vdash \varphi \quad \text{implies} \quad T \cup T' \vdash \varphi$$

that is, if something follows from a given set of knowledge, then it will still follow no matter what new knowledge we add.

- This is, in general, a natural property of inference relations
- but in real life, we often draw non-monotonic inferences.
Non-monotonic reasoning

The most classical example: Tweety

- We know that Tweety is a bird
- hence it makes sense to assume that Tweety flies!
- Later we find out that Tweety is a penguin
- We can simply withdraw our previous conclusion and infer that Tweety does not fly
Non-monotonic reasoning (cont’d)

We constantly use many forms of default reasoning that are not monotonic:

- My car is currently parked where I left it
- When I open the tap I will get running water
- On my way home I will be able to take the subway, and it comes every few minutes
- ...

Capturing this kind of reasoning through formal logics was a challenge that kept the AI community busy for quite a while
Non-monotonic logics

Many logic-based formalisms where proposed to accommodate non-monotonic common sense reasoning

Among the most famous are:

- Default logic
- Circumscription
- Auto-epistemic logic, non-monotonic modal logics

ASP is a computation-oriented formalism that captures many of the well-established languages for non-monotonic reasoning
The frame Problem

Another major limitation of Classical Logic arises when we reason about actions and change

- In FOL/DLs we can define, for example
  - Different kind of actions that an agent may carry out in some domain, e.g. load a gun, aim a target, shoot . . .
  - Infinitely many time instances over which actions can take place
  - Effects of carrying out some action, e.g. if the agent loads the gun at instant $t$, then the gun is loaded in the next instant $t + 1$
  - . . .
However, we also want our model to imply that if at instant $t$ the gun is loaded, and the agent aims it at a target, then at instant $t+1$

- the gun is still loaded
- the agent $A$ that has the gun is still the same one
- agent $A'$ still knows what the position of the door is, etc.
- ... but we do not want to explicitly say what happens to every fluent of the domain after every action!

In AI this is called the **frame problem**

There is no natural way to solve this problem in FOL/DLs

We will see that ASP provides a very elegant solution to this problem
Social Dinner Example

- Imagine the organizers of a conference plan a fancy dinner for the participants.
- 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

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

Proposal:

Answer Set Programming (ASP) paradigm!
Roots of ASP – Knowledge Representation (KR)

How to model

- An agent’s belief sets
- Commonsense reasoning
- Defeasible inferences
- Preferences and priority

The ASP Approach

- use a logic (programming)-based formalism
- Inherent feature: non-monotonicity

We will introduce ASP via examples, often using the syntax of DLV programs
Simple Social Dinner Example

A simple program simple.dlv:

- Wine bottles (brands) "a", ..., "e"
- preference by facts
- a simple (DL) ontology could be directly represented within the logic program (or even better, the program could interact with a DL ontology)

% 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).
Negation in Logic Programs

Why negation?

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

Clauses of the form:

\[ p(\vec{X}) \rightarrow q_1(\vec{X}_1), \ldots, q_k(\vec{X}_k), \text{not } r_1(\vec{Y}_1), \ldots, \text{not } r_l(\vec{Y}_l) \]

Things get more complex!
Programs with Negation

- “\textit{not} (\cdot)” means “Negation as Failure (to prove)” (aka default negation)
- Different from negation in classical logic!

Example

\begin{verbatim}
compliantBottle("axel","a"),
bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
\end{verbatim}

Query:

\begin{verbatim}
?- bottleChosen(X).
X = "a"
\end{verbatim}
Default Negation and the Frame Problem

When reasoning about actions and change, this form of negation allow us to state compact *inertia rules* for the fluents in our domain:

Example

\[
\text{gunLoaded}(T+1) :- \text{gunLoaded}(T), \neg \text{gunUnloaded}(T+1).
\]

\[
\text{doorOpen}(T+1) :- \text{doorOpen}(T), \neg \text{doorNotOpen}(T+1).
\]

This solves the frame problem!

**Note:** Here we assume that the remaining rules in the program ensure that \text{gunUnloaded} and \text{doorNotOpen} respectively behave like the negations of \text{gunLoaded} and \text{doorOpen}, and hence both predicates can not be true at the same time. This is easy to ensure in ASP.
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 single minimal model!

Two alternatives:

- $M_1 = \{ \text{compliantBottle("axel","a"), bottle Chosen("a")} \}$,
- $M_2 = \{ \text{compliantBottle("axel","a"), bottleSkipped("a")} \}$.

Which one to choose?
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
Stratified Negation

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

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

**Example:**

\[
\text{compliantBottle("axel","a"),} \\
\text{bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).}
\]

**Computed model**

\( M = \{ \text{compliantBottle("axel","a"), bottle Chosen("a") } \} \).

**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:

$L_0$: compliantBottle("axel","a"). wineBottle("a"). expensive("a").

$L_1$: bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
$L_0$: bottleSkipped(X) :- expensive(X), wineBottle(X).

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

\[ M = \{ \text{compliantBottle("axel","a"), wineBottle("a"), expensive("a"), bottleSkipped("a")}\} \]
Multiple preferred models

Unstratified Negation makes layering ambiguous:

L0: compliantBottle("axel","a").
L?: bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
L?: bottleSkipped(X) :- not bottleChosen(X), compliantBottle(Y,X).

Assign to a program (theory) not one but several intended models! For instance: Answer sets!

How to interpret these semantics? Answer set programming caters for the following views:

1. skeptical reasoning: Only take entailed answers, i.e. true in all models
2. brave reasoning: each model represents a different solution to the problem
3. additionally: one can define to consider only a subset of preferred models

(Alternative: well-founded inference takes a more “agnostic” view: One model, leaving ambiguous literals unknown.)
Answer Set Programming Paradigm

**General idea: Models are Solutions!**
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)
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/research/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: Disjunctive Logic programs with constraints: This is more than we saw so far!
- Integration of KR, DB, and search techniques
- Handling dynamic, knowledge intensive applications: data, defaults, exceptions, closures, ...
Example: Sudoku

Task

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

Problem specification $PS$

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

From $\text{sudoku.dlv}$:

% Assign a value to each field
\text{tab}(X,Y,1) \lor \text{tab}(X,Y,2) \lor \text{tab}(X,Y,3) \lor \\
\text{tab}(X,Y,4) \lor \text{tab}(X,Y,5) \lor \text{tab}(X,Y,6) \lor \\
\text{tab}(X,Y,7) \lor \text{tab}(X,Y,8) \lor \text{tab}(X,Y,9) :- \\
    \#\text{int}(X), \ 0 \leq X, \ X \leq 8, \ \#\text{int}(Y), \ 0 \leq Y, \ Y \leq 8.$

% Check rows and columns
:- \text{tab}(X,Y1,Z), \text{tab}(X,Y2,Z), Y1 \neq Y2.
:- \text{tab}(X1,Y,Z), \text{tab}(X2,Y,Z), X1 \neq X2.

% Check subtable
:- \text{tab}(X1,Y1,Z), \text{tab}(X2,Y2,Z), Y1 \neq Y2, \\
\text{div}(X1,3,W1), \text{div}(X2,3,W1), \text{div}(Y1,3,W2), \text{div}(Y2,3,W2). \\
:- \text{tab}(X1,Y1,Z), \text{tab}(X2,Y2,Z), X1 \neq X2, \\
\text{div}(X1,3,W1), \text{div}(X2,3,W1), \text{div}(Y1,3,W2), \text{div}(Y2,3,W2).

%Auxiliary: $X$ divided by $Y$ is $Z$
\text{div}(X,Y,Z) :- X\text{minusDelta} = Y*Z, \ X = X\text{minusDelta} + \text{Delta}, \ \text{Delta} < Y.$
Sudoku (cont’d)

**Data $D$:**

% Table positions X=0..8, Y=0..8  
\[
\begin{align*}
  \text{tab}(0,1,6). & \quad \text{tab}(0,3,1). \quad \text{tab}(0,5,4). \quad \text{tab}(0,7,5). \\
  \text{tab}(1,2,8). & \quad \text{tab}(1,3,3). \quad \text{tab}(1,5,5). \quad \text{tab}(1,6,6).
\end{align*}
\]

Solution:

\[
\begin{array}{cccc|cccc}
9 & 6 & 3 & 1 & 7 & 4 & 2 & 5 & 8 \\
1 & 7 & 8 & 3 & 2 & 5 & 6 & 4 & 9 \\
2 & 5 & 4 & 6 & 8 & 9 & 7 & 3 & 1 \\
8 & 2 & 1 & 4 & 3 & 7 & 5 & 9 & 6 \\
4 & 9 & 6 & 8 & 5 & 2 & 3 & 1 & 7 \\
7 & 3 & 5 & 9 & 6 & 1 & 8 & 2 & 4 \\
5 & 8 & 9 & 7 & 1 & 3 & 4 & 6 & 2 \\
3 & 1 & 7 & 2 & 4 & 6 & 9 & 8 & 5 \\
6 & 4 & 2 & 5 & 9 & 8 & 1 & 7 & 3 \\
\end{array}
\]
ASP - Desiderata

Expressive Power

Capable of representing a range of problems, hard problems
Disjunctive ASP: $\text{NEXP}^{\text{NP}}$-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:

%% These rules generate multiple answer sets:
(1) bottleSkipped(X) :- not bottleChosen(X), compliantBottle(Y,X).
(2) bottleChosen(X) :- not bottleSkipped(X), compliantBottle(Y,X).
(3) hasBottleChosen(X) :- bottleChosen(Z), compliantBottle(X,Z).

- Rules (1) and (2) enforce that either bottleChosen(X) or bottleSkipped(X) is included in an answer set (but not both), if it contains compliantBottle(Y,X).
- Rule (3) computes which persons have a bottle.
Social Dinner Example II

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

% Alternatively we could use disjunction:

(4) bottleSkipped(X) v bottleChosen(X) :- compliantBottle(Y,X).

(3) hasBottleChosen(X) :- bottleChosen(Z), compliantBottle(X,Z).

- Rules (1) and (2) enforce that either bottleChosen(X) or bottleSkipped(X) is included in an answer set (but not both), if it contains compliantBottle(Y,X).
- Rule (3) computes which persons have a bottle
- Rule (4) (disjunction!) can be used for replacing (1)-(2), more on that later!
Answer Set Semantics

- Variable-free, non-disjunctive programs first!
- Rules

\[ a :\neg b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n \]

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 \(P\).
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").

\[
HB(P) = \{ \text{wineBottle("a"), wineBottle("axel"), bottleSkipped("a"), bottleSkipped("axel"), bottleChosen("a")}
\]

bottleChosen("axel"), compliantBottle("axel","a"),
compliantBottle("axel","axel"), ... compliantBottle("a","axel") \}
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

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

   where some $c_i$ is in $M$

2. remove all literals of form $\text{not } p$ 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”
Computation of $lm(P)$

The least model of a \textit{not}-free program can be computed by fixpoint iteration.

\textbf{Algorithm Compute}_LM(P)

\begin{itemize}
  \item \textbf{Input:} Horn program $P$;
  \item \textbf{Output:} $lm(P)$
\end{itemize}

\texttt{new}\_\texttt{M} := \emptyset;
\textbf{repeat}
\begin{itemize}
  \item $M := \texttt{new}\_\texttt{M}$;
  \item \texttt{new}\_\texttt{M} := \{ $a$ | $a$ : $b_1, \ldots, b_m \in P$, \{ $b_1, \ldots, b_m$ \} $\subseteq M$ \}
\end{itemize}
\textbf{until} \texttt{new}\_\texttt{M} == $M$
\textbf{return} $M$
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

\[ M = lm(P) = \{ \text{wineBottle("a"), compliantBottle("axel","a")} \}. \]
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 = \{ \text{wineBottle("a")}, \text{compliantBottle("axel","a")}, $
\text{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 $M = \{ \text{wineBottle("a"), compliantBottle("axel","a"), bottleChosen("a"), hasBottleChosen("axel")} \}$

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

$lm(P^M) = M$, and therefore $M$ is another answer set
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 = \{ \text{wineBottle("a"), compliantBottle("axel","a"),}
\text{bottleChosen("a"), bottleSkipped("axel"), hasBottleChosen("axel")}, \}

- Rules (2) and (3) are dropped

$lm(P^M) = \{ \text{wineBottle("a"), compliantBottle("axel","a")} \} \neq M$

Thus, $M$ is not an 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., \( b(X) :- \text{not} \ s(X), \ c(Y,X). \)

is with constants "axel","a" short for:

\[
\begin{align*}
\text{b("a"):} \ &\text{not s("a"): c("a","a").} \\
\text{b("a"):} \ &\text{not s("a"): c("axel","a").} \\
\text{b("axel"):} \ &\text{not s("axel"): c("axel","axel").} \\
\text{b("axel"):} \ &\text{not s("axel"): c("axel","a").}
\end{align*}
\]
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 $Ground(P)$.

In the (function free) ASP setting, stable models are always finite. In this sense, ASP does not have the expressiveness of some DLs.
Inconsistent Programs

Program

\[ p : \neg p. \]

- This program has NO answer sets
- Let \( P \) be a program and \( p \) be a new atom
- Adding
  \[ p : \neg p. \]
  to \( P \) “kills” all answer sets of \( P \)
Constraints

- Adding

\[
p :- q_1, ..., q_m, \text{ not } r_1, ..., \text{ not } r_n, \text{ not } p.
\]

This adds a constraint to \( P \) “kills” all answer sets of \( P \) that:

- contain \( q_1, ..., q_m \), and
- do not contain \( r_1, ..., r_n \)

- Abbreviation:

\[
:- q_1, ..., q_m, \text{ not } r_1, ..., \text{ not } r_n.
\]

This is called a “constraint” (cf. integrity constraints in databases)
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. We obtain `simpleConstraint.dlv`

% This rule generates multiple answer sets:
(1) `bottleSkipped(X)` :- not `bottleChosen(X)`, compliantBottle(Y,X).
(2) `bottleChosen(X)` :- not `bottleSkipped(X)`, compliantBottle(Y,X).

% Ensure that each person gets a bottle.
(3) hasBottleChosen(X) :- `bottleChosen(Z)`, compliantBottle(X,Z).
(4) :- `person(X)`, not hasBottleChosen(X).
# 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 $\nu$ to $X_1, \ldots, X_k$ such that $l_1 \nu, \ldots, l_n \nu$ is cautiously resp. bravely true.

- seamless integration of query language and rule language
- expressivity beyond traditional query languages, e.g. SQL

## 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".
Social Dinner Example II – Reasoning

For `simpleConstraint.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.
## 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
- extensions such as **FDNC** programs, **BD** programs, finitary programs, finitely recursive programs, other work in progress.
Disjunctive ASP

- The use of disjunction in rule heads is natural

\[ \text{man}(X) \lor \text{woman}(X) :- \text{person}(X) \]

- ASP has thus been extended with disjunction

\[ a_1 \lor a_2 \lor \cdots \lor a_k :- b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \]

- 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 *to obtain* simpleDisj.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).

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
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 $\text{Ground}(P)$ with some literal $\text{not} \ a$ in the body such that $a \in M$

2. remove all literals $\text{not} \ a$ from all remaining rules in $\text{Ground}(P)$

However, $\text{lm}(P^M)$ 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 $P^M$
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:

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

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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLV</td>
<td><a href="http://www.dbai.tuwien.ac.at/proj/dlv/">http://www.dbai.tuwien.ac.at/proj/dlv/</a></td>
</tr>
<tr>
<td>SModels</td>
<td><a href="http://www.tcs.hut.fi/Software/smodels/">http://www.tcs.hut.fi/Software/smodels/</a></td>
</tr>
<tr>
<td>GnT</td>
<td><a href="http://www.tcs.hut.fi/Software/gnt/">http://www.tcs.hut.fi/Software/gnt/</a></td>
</tr>
<tr>
<td>Cmodels</td>
<td><a href="http://www.cs.utexas.edu/users/tag/cmodels/">http://www.cs.utexas.edu/users/tag/cmodels/</a></td>
</tr>
<tr>
<td>ASSAT</td>
<td><a href="http://assat.cs.ust.hk/">http://assat.cs.ust.hk/</a></td>
</tr>
<tr>
<td>NoMore</td>
<td><a href="http://www.cs.uni-potsdam.de/~linke/nomore/">http://www.cs.uni-potsdam.de/~linke/nomore/</a></td>
</tr>
<tr>
<td>Platypus</td>
<td><a href="http://www.cs.uni-potsdam.de/platypus/">http://www.cs.uni-potsdam.de/platypus/</a></td>
</tr>
<tr>
<td>clasp</td>
<td><a href="http://www.cs.uni-potsdam.de/clasp/">http://www.cs.uni-potsdam.de/clasp/</a></td>
</tr>
<tr>
<td>XASP</td>
<td>distributed with XSB v2.6</td>
</tr>
<tr>
<td></td>
<td><a href="http://xsb.sourceforge.net">http://xsb.sourceforge.net</a></td>
</tr>
<tr>
<td>aspps</td>
<td><a href="http://www.cs.engr.uky.edu/ai/aspps/">http://www.cs.engr.uky.edu/ai/aspps/</a></td>
</tr>
<tr>
<td>ccalc</td>
<td><a href="http://www.cs.utexas.edu/users/tag/cc/">http://www.cs.utexas.edu/users/tag/cc/</a></td>
</tr>
</tbody>
</table>

- Some provide a number of extensions to the language described here.
- Rudimentary extension to include function symbols exist (⇒ finitary programs, Bonatti)
- Answer Set Solver Implementation: see Niemelä’s ICLP’04 tutorial.
# Architecture of ASP Solvers

Typically, a two level architecture

<table>
<thead>
<tr>
<th>1. Grounding Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given a program $P$ with variables, generate its grounding (or a subset of its grounding which has the same models)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Model search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find a model for a given ground program.</td>
</tr>
<tr>
<td>Some solvers use special-purpose search procedures, others use variations of techniques known from SAT and theorem provers, or even export the task to sat solvers (e.g. Cmodels, ASSAT)</td>
</tr>
</tbody>
</table>
Extensions of ASP

- Besides disjunction, 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/

- For example, the extensions supported by DLV include:
  
  - weak constraints
  - aggregates

  Other solvers feature similar constructs.
Weak Constraints in DLV

- Allow the formalization of optimization problems in an easy and natural way.
- 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 $(P, W)$.*
- In our *social dinner* example, we could use weak constraints to select the least possible number of wines, or the least possible number of expensive wines.
Aggregates in DLV

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

- In our example, we can use this to calculate and minimize the price of the chosen wines
  For example,

  \[
  \text{totalcost}(N) :- \ #\text{int}(N), \\
  \ #\sum\{ Y : \text{bottleChosen}(X), \text{prize}(X,Y) \} = N.
  \]

  estimates the cost of the chosen wines, and

  \[
  \text{ok\_price} :- \ #\sum\{ Y : \text{bottleChosen}(X), \text{prize}(X,Y) \} < 500.
  \]

  says that the price is OK if the sum of the prices of the chosen wines is below 50
DLV Front Ends

DLV offers a range of front-ends for specific applications, like

- diagnosis
- planning
- inheritance reasoning
- SQL3
Summary

- ASP is a very powerful formalism for KR&R

- ASP and DLs can be seen as complementary formalisms
  - It overcomes some limitations of DLs
  - But it also has certain limitations w.r.t. DLs

- Combinations that provide the strengths of both are an active (and challenging) area of research

Anybody looking for nice research topics?