Answer-Set Programming

Basics, Combinations with Ontologies, and Debugging

Jörg Pührer

Knowledge Based Systems Group, Institute for Information Systems
Vienna University of Technology

Partially supported by the European Commission under
FP7 Project ONTORULE, Marie Curie Actions Project NET2,
and the Austrian Science Fund (FWF) P21698

Context

We deal with *answer-set programming* (*ASP*)

- Programming paradigm

Context

We deal with *answer-set programming* (*ASP*)

- Programming paradigm
- Term coined by Lifschitz (1999)
- Proposed by others people (Marek and Truszczyński, Niemelä) at about the same time

Context

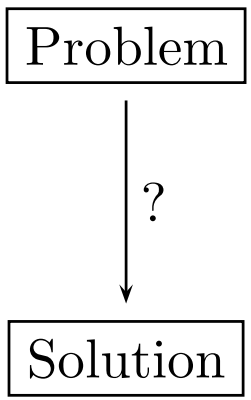
We deal with *answer-set programming* (*ASP*)

- Programming paradigm
- Term coined by Lifschitz (1999)
- Proposed by others people (Marek and Truszczyński, Niemelä) at about the same time
- Fruitful approach for *declarative* problem solving

Context

We deal with *answer-set programming* (*ASP*)

- Programming paradigm
- Term coined by Lifschitz (1999)
- Proposed by others people (Marek and Truszczyński, Niemelä) at about the same time
- Fruitful approach for *declarative* problem solving



Declarative vs. Procedural Programming

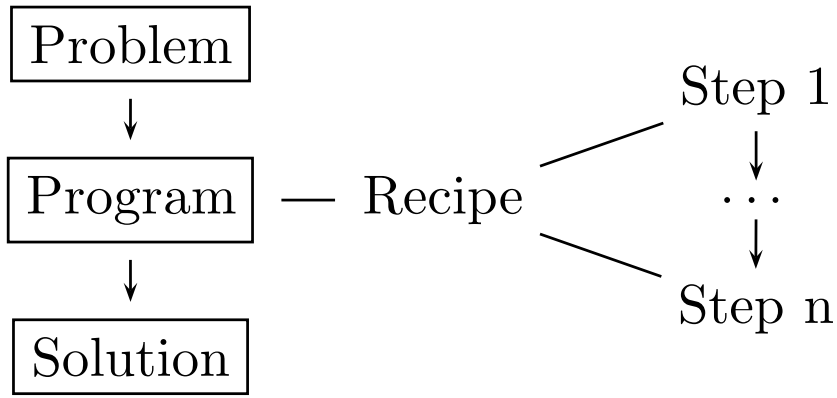
In *procedural programming*, a problem is solved by writing a program that corresponds to a *recipe*.

- General Questions:
  - What *steps* do I need to take to solve the problem?
  - *How* do I solve it?

Declarative vs. Procedural Programming

In *procedural programming*, a problem is solved by writing a program that corresponds to a *recipe*.

- General Questions:
  - What *steps* do I need to take to solve the problem?
  - *How* do I solve it?



Declarative vs. Procedural Programming (ctd.)

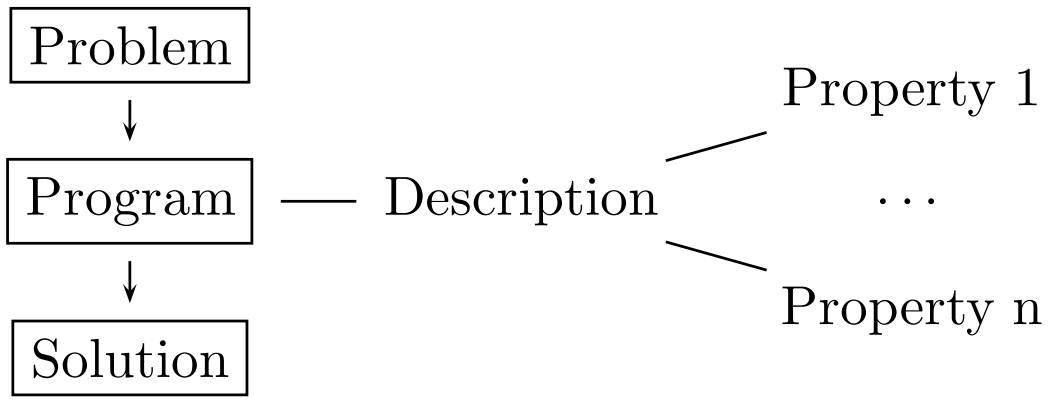
In *declarative programming*, a problem is solved by writing a program that corresponds to a *description* of the problem.

- General Question:
  - *What* is the problem?

Declarative vs. Procedural Programming (ctd.)

In *declarative programming*, a problem is solved by writing a program that corresponds to a *description* of the problem.

- General Question:
  - *What* is the problem?



- Control flow is *not* provided in the program.
- *No algorithm* is given for solving the problem.

ASP Big Picture

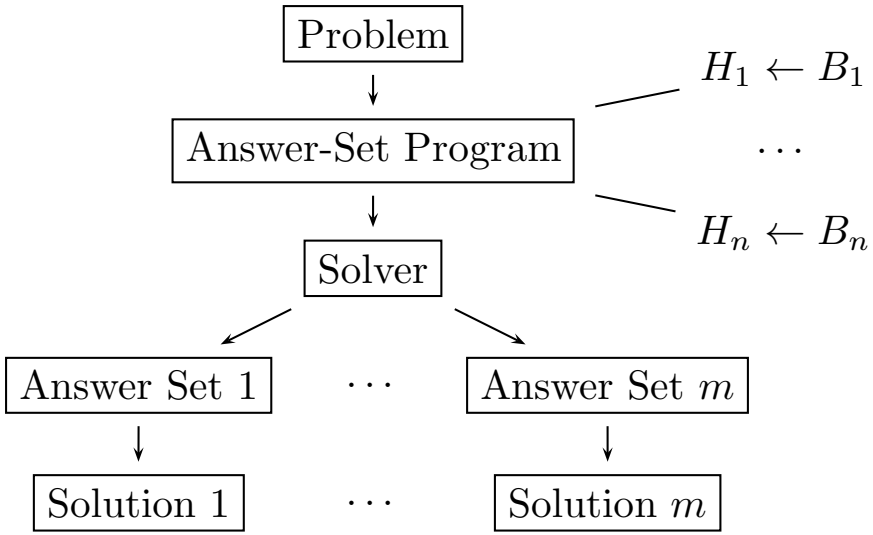
In *Answer Set Programming (ASP)*, the problem is encoded in terms of a logic theory - typically using logic programming rules.

- Answer-Set Solver computes dedicated models of the theory.
- They are called *answer sets*.

ASP Big Picture

In *Answer Set Programming (ASP)*, the problem is encoded in terms of a logic theory - typically using logic programming rules.

- Answer-Set Solver computes dedicated models of the theory.
- They are called *answer sets*.
- Correspond one-to-one to the (multiple) solutions of the problem.



Answer Sets

Multiple Answer Sets:

- Non-monotonic reasoning (use of default negation)
- Ability to express non-determinism (guessing)
- Sometimes useful: Preference relations between answer sets
- Most used semantics:
  - Stable model semantics ($\simeq$ Answer-Set semantics)
- Different modes of reasoning:
  - Model generation: compute 1, n, or all answer sets
  - Cautious reasoning: query holds in all answer sets
  - Brave reasoning: there is an answer set for which query holds

Syntax

- We deal with *disjunctive logic programs* (*DLPs*) under the answer-set semantics, which are sets of rules of form:

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$$

  - it must hold that $n \geq 1$
  - all a_i are literals over some fixed function-free first-order language $\mathscr{L}$
  - "*not*" denotes *default negation* (negation as failure)
  - a (strong) literal is either an atom or an atom preceded by *strong negation*, $-$.

Syntax (ctd.)

- We deal with rules of form:

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$$

- We distinguish the following parts of rule r of the form above:
  - the head $\mathrm{H}(r) = \{a_1, \cdots, a_l\}$,
  - the body $\mathrm{B}(r) = \{a_{l+1}, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n\}$,
  - the positive body $\mathrm{B}^+(r) = \{a_{l+1}, \ldots, a_m\}$, and
  - the negative body $\mathrm{B}^-(r) = \{a_{m+1}, \ldots, a_n\}$.

- A rule r is
  - a *fact* if $\mathrm{B}(r) = \emptyset$,
  - a *constraint* if $\mathrm{H}(r) = \emptyset$,
  - *normal* if $|\mathrm{H}(r)| \leq 1$,
  - *positive* if $\mathrm{B}^-(r) = \emptyset$, or
  - *Horn* if it is both, normal and positive.

Semantics

- The answer-set semantics for disjunctive logic programs was proposed by Gelfond and Lifschitz (1991).
- Definition based on ground programs:
  - Rules do not contain variables
- Semantics for a program P with variables based on the program's grounding $gr(P)$:
  - Replace each non-ground rule r by all ground rules that can be obtained from r by substituting variables by constants in the Herbrand Universe of the program.

Semantics (ctd.)

- An *interpretation* is a consistent set I of ground literals
  - a is true iff $a \in I$.

Semantics (ctd.)

- An *interpretation* is a consistent set I of ground literals
  - a is true iff $a \in I$.
- For a ground rule r,
  - $I \models \mathrm{B}(r)$ iff $\mathrm{B}^+(r) \subseteq I$ and $\mathrm{B}^-(r) \cap I = \emptyset$;
  - $I \models \mathrm{H}(r)$ iff $\mathrm{H}(r) \cap I \neq \emptyset$;
  - $I \models r$ iff ($I \models \mathrm{B}(r)$ implies $I \models \mathrm{H}(r)$)
- I is a model of a ground program P iff $I \models r$ for all $r \in P$.

Semantics (ctd.)

- The *reduct* of a ground program P w.r.t. I is the program

$$P^I := \{\mathrm{H}(r) \leftarrow \mathrm{B}^+(r) \mid r \in P \text{ and } I \cap \mathrm{B}^-(r) = \emptyset\}$$

Semantics (ctd.)

- The *reduct* of a ground program P w.r.t. I is the program

$$P^I := \{\mathrm{H}(r) \leftarrow \mathrm{B}^+(r) \mid r \in P \text{ and } I \cap \mathrm{B}^-(r) = \emptyset\}$$

- I is an *answer set* of a program P if it is the minimal model of $gr(P)^I$
- By $AS(P)$, we denote the set of all answer sets of P.

Semantics (ctd.)

Many alternative definitions

- By means of propositional formulas (Lin and Zhao, 2004)
- In terms of equilibrium logic and logic of here-and-there (Pearce, 1996)
- Using the FLP-reduct P^I_{FLP} instead of P^I (Faber, Leone, and Pfeifer, 2004):
  - I is an *answer set* of a program P if it is a minimal model of the rules $r \in gr(\Pi)$ such that $I \models \mathrm{B}(r)$.
  - Allows for intuitive handling of recursive aggregates.
- ...

Small example

$$
\begin{aligned}
P = \{ \quad a(X) &\leftarrow not\ e(X), d(X), \\
e(X) &\leftarrow not\ \ a(X), d(X), \\
c(X,Y) &\leftarrow a(X), a(Y), not\ -b(X), \\
d(1) &\leftarrow, \\
d(2) &\leftarrow \}.
\end{aligned}
$$

Small example

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

The grounding of P is given by

$$
\begin{aligned}
gr(P) = \{ \quad & a(1) \leftarrow not\ e(1), d(1), & & a(2) \leftarrow not\ e(2), d(2), \\
& e(1) \leftarrow not\ \ a(1), d(1), & & e(2) \leftarrow not\ \ a(2), d(2), \\
& c(1,1) \leftarrow a(1), a(1), not\ -b(1), & & c(2,2) \leftarrow a(2), a(2), not\ -b(2), \\
& c(1,2) \leftarrow a(1), a(2), not\ -b(1), & & c(2,1) \leftarrow a(2), a(1), not\ -b(2), \\
& d(1) \leftarrow, & & d(2) \leftarrow \}.
\end{aligned}
$$

Small example (ctd.)

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

Consider the interpretation $I_1 = \{d(1), d(2)\}$

$$
\begin{aligned}
gr(P)^{I_1}_{\mathrm{FLP}} = \{ \quad & a(1) \leftarrow not\ e(1), d(1), && a(2) \leftarrow not\ e(2), d(2), \\
& e(1) \leftarrow not\ \ a(1), d(1), && e(2) \leftarrow not\ \ a(2), d(2), \\
& \cancel{c(1,1) \leftarrow a(1), a(1), not\ -b(1),} && \cancel{c(2,2) \leftarrow a(2), a(2), not\ -b(2),} \\
& \cancel{c(1,2) \leftarrow a(1), a(2), not\ -b(1),} && \cancel{c(2,1) \leftarrow a(2), a(1), not\ -b(2),} \\
& d(1) \leftarrow, && d(2) \leftarrow \}.
\end{aligned}
$$

I_1 is not a model of $gr(P)^{I_1}_{\mathrm{FLP}}$.

Small example (ctd.)

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

Consider the interpretation $I_2 = \{d(1), d(2), e(1), a(2), e(2), c(2,2)\}$

$$
\begin{aligned}
gr(P)^{I_2}_{\mathrm{FLP}} = \{ \quad & \cancel{a(1) \leftarrow not\ e(1), d(1),} && \cancel{a(2) \leftarrow not\ e(2), d(2),} \\
& e(1) \leftarrow not\ \ a(1), d(1), && \cancel{e(2) \leftarrow not\ \ a(2), d(2),} \\
& \cancel{c(1,1) \leftarrow a(1), a(1), not\ -b(1),} && c(2,2) \leftarrow a(2), a(2), not\ -b(2), \\
& \cancel{c(1,2) \leftarrow a(1), a(2), not\ -b(1),} && \cancel{c(2,1) \leftarrow a(2), a(1), not\ -b(2),} \\
& d(1) \leftarrow, && d(2) \leftarrow \}.
\end{aligned}
$$

I_2 is a model of $gr(P)^{I_2}_{\mathrm{FLP}}$.

Small example (ctd.)

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

Consider the interpretation $I_2 = \{d(1), d(2), e(1), a(2), e(2), c(2,2)\}$

$$
\begin{aligned}
gr(P)^{I_2}_{\mathrm{FLP}} = \{ \quad & \cancel{a(1) \leftarrow not\ e(1), d(1),} && \cancel{a(2) \leftarrow not\ e(2), d(2),} \\
& e(1) \leftarrow not\ \ a(1), d(1), && \cancel{e(2) \leftarrow not\ \ a(2), d(2),} \\
& \cancel{c(1,1) \leftarrow a(1), a(1), not\ -b(1),} && c(2,2) \leftarrow a(2), a(2), not\ -b(2), \\
& \cancel{c(1,2) \leftarrow a(1), a(2), not\ -b(1),} && \cancel{c(2,1) \leftarrow a(2), a(1), not\ -b(2),} \\
& d(1) \leftarrow, && d(2) \leftarrow \}.
\end{aligned}
$$

I_2 is a model of $gr(P)^{I_2}_{\mathrm{FLP}}$ but not minimal.

Small example (ctd.)

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

Consider the interpretation $I_3 = \{d(1), d(2), e(1), a(2), c(2,2)\}$

$$
\begin{aligned}
gr(P)^{I_3}_{\mathrm{FLP}} = \{ \quad & \cancel{a(1) \leftarrow not\ e(1), d(1),} & & a(2) \leftarrow not\ e(2), d(2), \\
& e(1) \leftarrow not\ \ a(1), d(1), & & \cancel{e(2) \leftarrow not\ \ a(2), d(2),} \\
& \cancel{c(1,1) \leftarrow a(1), a(1), not\ -b(1),} & & c(2,2) \leftarrow a(2), a(2), not\ -b(2), \\
& \cancel{c(1,2) \leftarrow a(1), a(2), not\ -b(1),} & & \cancel{c(2,1) \leftarrow a(2), a(1), not\ -b(2),} \\
& d(1) \leftarrow, & & d(2) \leftarrow \}.
\end{aligned}
$$

Small example (ctd.)

$$
\begin{aligned}
P = \{ \quad & a(X) \leftarrow not\ e(X), d(X), \\
& e(X) \leftarrow not\ \ a(X), d(X), \\
& c(X,Y) \leftarrow a(X), a(Y), not\ -b(X), \\
& d(1) \leftarrow, \\
& d(2) \leftarrow \}.
\end{aligned}
$$

Consider the interpretation $I_3 = \{d(1), d(2), e(1), a(2), c(2,2)\}$

$$
\begin{aligned}
gr(P)^{I_3}_{\mathrm{FLP}} = \{ \quad & \cancel{a(1) \leftarrow not\ e(1), d(1),} & & a(2) \leftarrow not\ e(2), d(2), \\
& e(1) \leftarrow not\ \ a(1), d(1), & & \cancel{e(2) \leftarrow not\ \ a(2), d(2),} \\
& \cancel{c(1,1) \leftarrow a(1), a(1), not\ -b(1),} & & c(2,2) \leftarrow a(2), a(2), not\ -b(2), \\
& \cancel{c(1,2) \leftarrow a(1), a(2), not\ -b(1),} & & \cancel{c(2,1) \leftarrow a(2), a(1), not\ -b(2),} \\
& d(1) \leftarrow, & & d(2) \leftarrow \}.
\end{aligned}
$$

I_3 is a minimal model of $gr(P)^{I_3}_{\mathrm{FLP}}$, hence I_3 is an answer set of P.

Small example (ctd.)

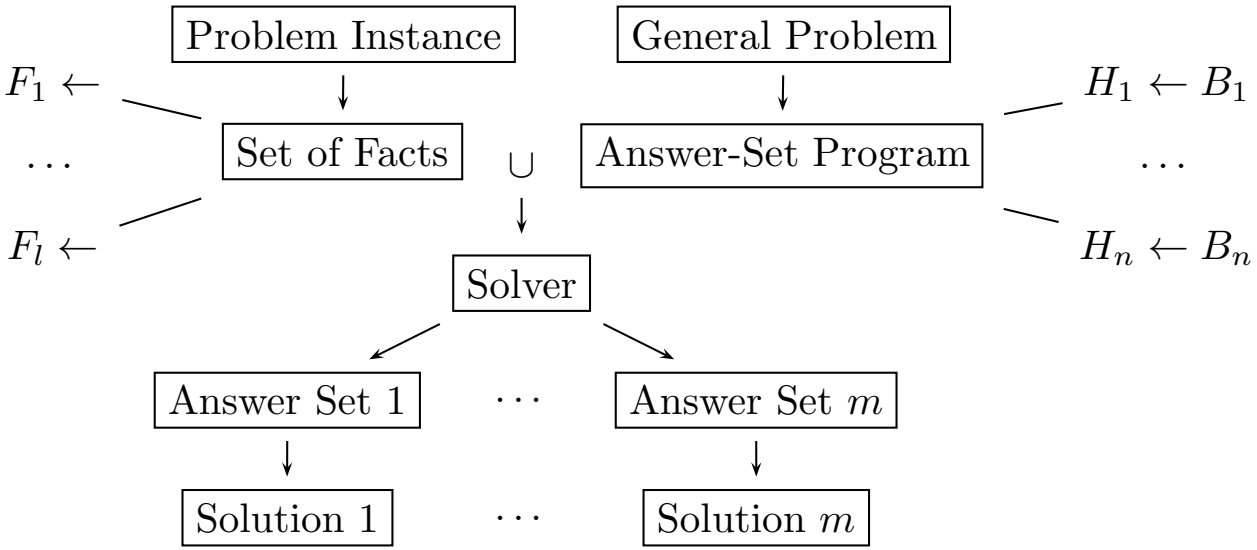
$$
\begin{aligned}
P = \{ \quad a(X) &\leftarrow not\ e(X), d(X), \\
e(X) &\leftarrow not\ \ a(X), d(X), \\
c(X,Y) &\leftarrow a(X), a(Y), not\ -b(X), \\
d(1) &\leftarrow, \\
d(2) &\leftarrow \}.
\end{aligned}
$$

P has the following answer sets:

$$\{d(1), d(2), a(1), a(2), c(1,1), c(1,2), c(2,1), c(2,2)\},$$
$$\{d(1), d(2), a(1), e(2), c(1,1)\},$$
$$\{d(1), d(2), e(1), a(2), c(2,2)\},$$
$$\{d(1), d(2), e(1), e(2)\}$$

Uniform Problem Encoding

- Typically, answer-set programs are *uniform problem encodings*.
- The program itself encodes the general problem,
- whereas individual problem instances represented by facts can be joined with the program on demand.



Graph Colouring

Uniform problem encoding:

$$P_1 = \{ another\_col(V, C) \leftarrow vertex(V), col(C), col(D), col\_of(V, D), C \neq D,$$

$$col\_of(V, C) \leftarrow vertex(V), col(C), not\ another\_col(V, C),$$

$$\leftarrow vertex(U), vertex(V), edge(U, V), col\_of(U, C), col\_of(V, C) \}.$$

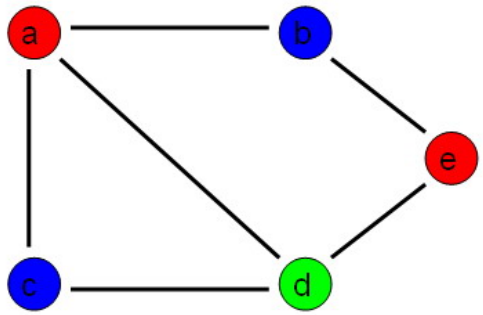
Input instance as set of facts:

$$F_1 = \{ col(red), col(green), col(blue), vertex(a), vertex(b), vertex(c), vertex(d), vertex(e), edge(a, b), edge(a, c), edge(a, d), edge(b, e), edge(c, d), edge(d, e) \}.$$

Graph Colouring (ctd.)

- The answer sets of $P_1 \cup F_1$ correspond to the solutions of the problem instance.

$$\{col\_of(a, red), col\_of(b, blue), col\_of(c, blue), col\_of(d, green), col\_of(e, red), \dots\}$$



Use

- Complexity captured by uniform problem encodings:
  - normal logic programs: NP-hard problems
  - disjunctive logic programs: Σ_2^P-hard problems
- Applications of ASP
  - configuration, computer-aided verification, software testing, health care, bio informatics, planning, music composition, database repair, *Semantic-Web reasoning*, ...
- Strengths:
  - Concise representation, elaboration tolerance, non-determinism, use of recursion
  - Efficient solvers:
    - Clasp (University of Potsdam)
    - DLV (University of Calabria, Vienna University of Technology)

ASP Language Extensions

- Many extensions have been proposed, partly motivated by applications
- Some are syntactic sugar, other strictly add expressiveness
- Incomplete list:
  - optimisation: minimize/maximize statements, weak constraints
  - aggregates, cardinality constraints
  - templates (for macros)
  - function symbols (undecidability)
  - Frame Logic syntax
  - KR frontends (diagnosis, planning,...) in DLV
  - higher-order features (variables as predicates)
  - *external atoms*

Formalisms combining Ontologies and Rules

- Hybrid formalisms - Ontologies / Logical rules.
  - loosely coupled approaches
    - DL-programs
    - F-Logic#
  - tightly coupled approaches
    - SWRL and restrictions (DLP, Carin, DL-safe rules, Desription Logic Rules)
    - R-hybrid KBs
    - F-hybrid KBs
  - Embedding approaches
    - hybrid MKNF KBs
    - Quantified Equilibrium Logic (QEL)

Ontologies and Rules in the Semantic Web

Why a "Semantic" Web?

- Undoubtedly, the World Wide Web is one of the most significant technical innovations of the last decades,
- The WWW will be (and already is) strongly impacting, and changing, our living, culture, and society.

Ontologies and Rules in the Semantic Web

Why a "Semantic" Web?

- Undoubtedly, the World Wide Web is one of the most significant technical innovations of the last decades,
- The WWW will be (and already is) strongly impacting, and changing, our living, culture, and society.
- The WWW has been designed for human users, not for machines

Ontologies and Rules in the Semantic Web

Why a "Semantic" Web?

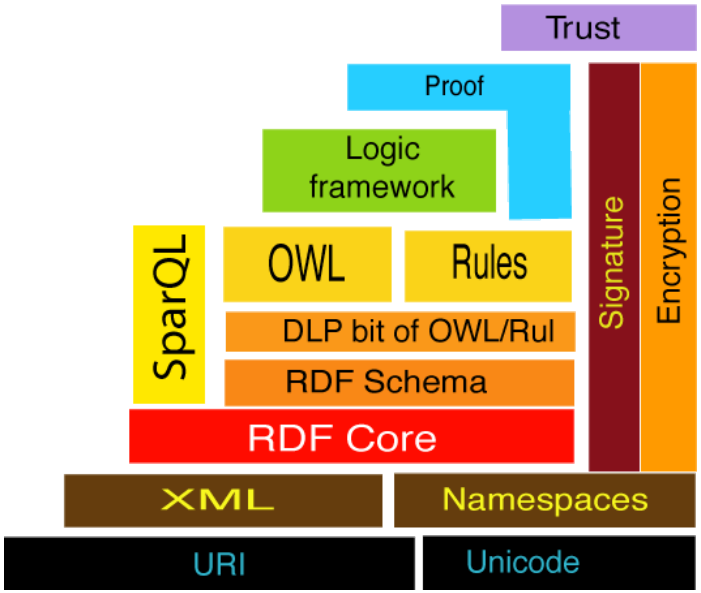
- Undoubtedly, the World Wide Web is one of the most significant technical innovations of the last decades,
- The WWW will be (and already is) strongly impacting, and changing, our living, culture, and society.
- The WWW has been designed for human users, not for machines
- To process the data and information out on the Web, semantic annotation and description is needed.

Ontologies and Rules in the Semantic Web

Why a "Semantic" Web?

- Undoubtedly, the World Wide Web is one of the most significant technical innovations of the last decades,
- The WWW will be (and already is) strongly impacting, and changing, our living, culture, and society.
- The WWW has been designed for human users, not for machines
- To process the data and information out on the Web, semantic annotation and description is needed.
- The Semantic Web is the vision of such an enriched, future generation Web
- Logic and logic-based formalisms (should/might) play an important role in this endeavor.

Building the Semantic Web (T. Berners-Lee, 04/2005)



- RDF is the data model of the Semantic Web
- RDF Schema semantically extends RDF by simple taxonomies and hierarchies
- OWL is a W3C standard, which builds on Description Logics
- Rule languages: *Rule Interchange Format (RIF)* WG of W3C

ASP vs. Ontologies

- ASP and OWL/DLs have related yet different underlying settings
- At the heart, the difference is between LP and Classical logic
- **Main Differences:**
  - Closed vs. Open World Assumption (CWA/OWA)
  - Negation as failure vs. classical negation
  - Strong negation vs. classical negation
  - Unique names, equality

LP / Classical Logic: CWA vs. OWA

- LP aims at building a single model, by closing the world

  Reiter's CWA:

  If $T \not\models A$, *then conclude* $\neg A$, for ground atom A

- FO logic / description logics keep the world open
- In the Semantic Web, this is often reasonable
- However, taking the agnostic stance of OWA may be not helpful for drawing rational conclusions under incomplete information

LP / Classical Logic: NAF vs. Classical Negation

$$P: \quad wine(X) \leftarrow whiteWine(X).$$
$$nonWhite(X) \leftarrow not\ \ whiteWine(X).$$
$$wine(myDrink).$$

$$T: \quad \forall X.(whiteWine(X) \supset wine(X)) \wedge$$
$$\forall X.(\neg whiteWine(X) \supset nonWhite(X)) \wedge$$
$$wine(myDrink).$$

LP / Classical Logic: NAF vs. Classical Negation

P : $wine(X) \leftarrow whiteWine(X).$
$nonWhite(X) \leftarrow not\ \ whiteWine(X).$
$wine(myDrink).$

T : $\forall X.(whiteWine(X) \supset wine(X)) \wedge$
$\forall X.(\neg whiteWine(X) \supset nonWhite(X)) \wedge$
$wine(myDrink).$

- Query $nonWhite(myDrink)$?

LP / Classical Logic: NAF vs. Classical Negation

P : $wine(X) \leftarrow whiteWine(X).$
$nonWhite(X) \leftarrow not\ \ whiteWine(X).$
$wine(myDrink).$

T : $\forall X.(whiteWine(X) \supset wine(X)) \wedge$
$\forall X.(\neg whiteWine(X) \supset nonWhite(X)) \wedge$
$wine(myDrink).$

- Query $nonWhite(myDrink)$?
  - Conclude $nonWhite(myDrink)$ from P.

LP / Classical Logic: NAF vs. Classical Negation

$$P: \quad wine(X) \leftarrow whiteWine(X).$$
$$nonWhite(X) \leftarrow not\ \ whiteWine(X).$$
$$wine(myDrink).$$

$$T: \quad \forall X.(whiteWine(X) \supset wine(X)) \wedge$$
$$\forall X.(\neg whiteWine(X) \supset nonWhite(X)) \wedge$$
$$wine(myDrink).$$

- Query $nonWhite(myDrink)$?
  - Conclude $nonWhite(myDrink)$ from P.
  - Do not conclude $nonWhite(myDrink)$ from T.

LP / Classical Logic: Strong vs. Classical Negation

P : $wine(X) \leftarrow whiteWine(X).$
$-wine(myDrink).$

T : $\forall X.(WhiteWine(X) \supset Wine(X)) \wedge$
$\neg Wine(myDrink).$

LP / Classical Logic: Strong vs. Classical Negation

P : $wine(X) \leftarrow whiteWine(X).$
$-wine(myDrink).$

T : $\forall X.(WhiteWine(X) \supset Wine(X)) \wedge$
$\neg Wine(myDrink).$

- Conclude $\neg WhiteWine(myDrink)$ from T;

LP / Classical Logic: Strong vs. Classical Negation

P : $wine(X) \leftarrow whiteWine(X).$
$-wine(myDrink).$

T : $\forall X.(WhiteWine(X) \supset Wine(X)) \wedge$
$\neg Wine(myDrink).$

- Conclude $\neg WhiteWine(myDrink)$ from T;
- Do not conclude $-whiteWine(myDrink)$ from P
- Note: no contraposition in LP!

$$-whiteWine(X) \leftarrow -wine(X).$$

is not equivalent to

$$wine(X) \leftarrow whiteWine(X).$$

LP / Classical Logic: Unique Names, Equality

- In LP, usally we have *Unique Names Assumption (UNA)*:
  Syntactically different ground terms are different objects.
- Thus, usually only Herbrand interpretations are considered in LP
- Ontology languages like OWL don't make UNA, and allow to link objects (owl:sameAs)
- OWL considers also non-Herbrand interpretations
- Further, related problems with existential quantifiers:

$$T: \quad \forall X \exists Y . (Wine(X) \supset hasColor(X, Y))$$

(in DL Syntax, $Wine \sqsubseteq \exists hasColor$)

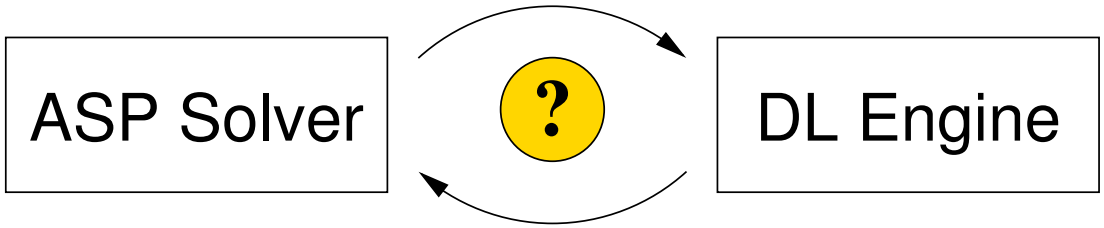
Simple skolemisation does not work in general

DL-programs

- Recently, much effort has been put into the study of combinations of *rules* and *ontologies* in the context of the Semantic Web.
- *DL-programs* have been proposed as a fruitful approach for combining logic programs (LP) under the answer-set semantics and description logic (DL) ontologies [Eiter et al. 2004,2008].
- The interface beetween LP and DL is based on *loose coupling*.

Loose Coupling

- The two components do not rely on a shared language or common logical models.
- Instead, they interact via dedicated atoms, called *DL-atoms*, in the LP.
- A DL-atom can be seen as a *query to the ontology*.
- Bi-directional flow of information:
  - before the query is evaluated, predicates from the LP can be temporarily imported to the DL.
  - this way, the DL-reasoner takes information into account that has been derived by rules in the LP.



Syntax

- A *DL-atom* $a(\mathbf{t})$ has the form

$$\mathrm{DL}[S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m; Q](\mathbf{t}), \text{ where}$$

  - $m \geq 0$,
  - each S_i is either a DL concept or a role predicate
  - $op_i \in \{\uplus, \cup, \cap\}$,
  - p_i is a unary, resp. binary, LP predicate symbol, and
  - $Q(\mathbf{t})$ is a DL-query
- We call $\gamma = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m$ the *input signature*.
- Intuitively,
  - $op_i = \uplus$ increases S_i by the extension of p_i,
  - $op_i = \cup$ increases $\neg S_i$ by the extension of p_i, and
  - $op_i = \cap$ constrains S_i to p_i.

Syntax (ctd.)

- A *DL-rule* r is a normal rule of form:

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

  - a is a strong literal, and
  - $b_1, \ldots, b_m$ are strong literals or DL-atoms.
  - "*not*" denotes *default negation* (negation as failure).
  - $\mathrm{H}(r) = a$, $\mathrm{B}^+(r) = \{b_1, \ldots, b_k\}$, $\mathrm{B}^-(r) = \{b_{k+1}, \ldots, b_m\}$, $\mathrm{B}(r) = \{b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\}$

- A *DL-program* $KB = (T, P)$ consists of a DL ontology T and a finite set P of DL-rules.
- The signatures of T and P have the same set F of (0-ary) function symbols.
- The grounding $gr(P)$ of P is the set of all ground rules that can be obtained by replacing variables of rules in P by constants occurring in F.

Semantics

- Let $KB = (T, P)$ be a DL-program.
- An *interpretation* for KB is a consistent set of strong literals from $gr(P)$.
- A ground strong literal a is *true* under interpretation I ($I \models^T a$) iff $a \in I$.

Semantics (ctd.)

- A ground DL-atom $a = DL[S_1 op_1\ p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})$ is *true* under DL T ($I \models^T a$) iff
  - $T \cup \tau_I(a) \models Q(\mathbf{c})$, where
  - the *extension* of a under I is $\tau_I(a) = \bigcup_{i=1}^{m} A_i(I)$ such that
    - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \cup$;
    - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}$, for $op_i = \cap$.
- $I \models^T \mathrm{B}^+(r)$ iff $I \models^T a$ for all $a \in \mathrm{B}^+(r)$.
- $I \models^T \mathrm{B}^-(r)$ iff $I \not\models^T a$ for all $a \in \mathrm{B}^-(r)$.
- $I \models^T \mathrm{B}(r)$ iff $I \models^T \mathrm{B}^+(r)$ and $I \models^T \mathrm{B}^-(r)$.
- $I \models^T r$ iff $I \models^T \mathrm{H}(r)$ whenever $I \models^T \mathrm{B}(r)$.
- I is a *model* of KB ($I \models KB$) iff $I \models^T r$ for all $r \in gr(P)$.

Semantics (ctd.)

- Answer sets defined analogously to answer-set programs using the FLP-reduct
  - ☞ remember: introduced for an intuitive handling of nonmonotone aggregates in ASP (Faber, Leone, Pfeifer 2004).
- The *FLP-reduct* KB^I_{FLP} of DL-program $KB = (T, P)$ relative to I is the set of rules $r \in gr(P)$ such that $I \models^T \mathrm{B}(r)$.

- Definition

An interpretation I is an *answer set* of KB if it is a minimal model of KB^I_{FLP}.

- The set of all answer sets of KB is denoted by $AS(KB)$.

Originally, two different semantics have been defined (Eiter, Lukasiewicz,Schindlauer, and Tompits 2004):

- Strong Answer-Set semantics
- Weak Answer-Set semantics

Example

- A computer store obtains hardware from several vendors.
- DL knowledge base T_{ex} contains information about the product range provided by each vendor.
- For some parts, a shop may already be contracted as supplier.
- T_{ex} T-Box

$$\geq 1\ \mathit{provides} \sqsubseteq \mathit{shop};\quad \top \sqsubseteq \forall \mathit{provides}.\mathit{part};$$

- T_{ex} A-Box

$$\mathit{provides}(s_1, \mathit{cpu});\quad \mathit{provides}(s_1, \mathit{case});\quad \mathit{provides}(s_2, \mathit{cpu});$$
$$\mathit{provides}(s_3, \mathit{harddisk});\quad \mathit{provides}(s_3, \mathit{case});$$
$$\mathit{supplies}(s_3, \mathit{case});$$

Example (ctd.)

- DL-program $KB_{ex} = (T_{ex}, P_{ex})$ not-deterministically chooses a vendor for each needed part.
- LP part P_{ex}:

$$
\begin{aligned}
needed(X) \leftarrow & \mathrm{DL}[; part](X);\\
alreadyContracted(P) \leftarrow & \mathrm{DL}[; supplies](S,P), needed(P);\\
offer(S,P) \leftarrow & \mathrm{DL}[; provides](S,P), needed(P),\\
& \quad not\ alreadyContracted(P);\\
chosen(S,P) \leftarrow & offer(S,P), not\ notChosen(S,P);\\
notChosen(S,P) \leftarrow & offer(S,P), not\ chosen(S,P);\\
supplier(S,P) \leftarrow & \mathrm{DL}[supplies \uplus chosen; supplies](S,P), needed(P);\\
anySupplied(P) \leftarrow & supplier(S,P);\\
\leftarrow & not\ needed(P), not\ anySupplied(P).
\end{aligned}
$$

Example (ctd.)

- KB_{ex} has two answer sets, I_1 and I_2
- both containing the same atoms of predicates *needed*, *offer*, *alreadyContracted*, and *anySupplied*:

$$I' = \{needed(cpu), needed(harddisk), needed(case), alreadyContracted(case), offer(s_1, cpu), offer(s_2, cpu), offer(s_3, harddisk), anySupplied(cpu), anySupplied(harddisk), anySupplied(case)\}$$

- The remaining atoms of I_1 are given by

$$I_1 \setminus I' = \{chosen(s_1, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), supplier(s_1, cpu), supplier(s_3, harddisk), supplier(s_3, case)\}$$

- The remaining atoms of I_2 are given by

$$I_2 \setminus I' = \{chosen(s_2, cpu), chosen(s_3, harddisk), notChosen(s_1, cpu), supplier(s_2, cpu), supplier(s_3, harddisk), supplier(s_3, case)\}$$

Complexity of DL-programs

Theorem

Given and a dl-program $KB = (T, P)$ with P in SHIF($\mathbf{D}$), deciding whether KB has an answer set is complete for NEXP *in the general case, and complete for* EXP *when KB is positive or stratified.*

Tractable DL-programs

- Tractable Reasoning with DL-programs (Heymans, Eiter, and Xiao 2010)
  - Datalog-rewritable Description Logics
  - DL-program is translated to a "pure" logic program
  - Approximation of answer-set semantics: well-founded semantics
  - new DL defined: LDL^+
  - OWL2 fragments:
    - OWL 2 RL: fully datalog rewritable (strict subset of LDL^+)
    - OWL 2 EL: partly d. r. (neg. and existential qu. in axiom r.h.s.)
    - OWL 2 QL: partly d. r. (neg. and existential qu. in axiom r.h.s.)

Conclusion

- Answer-set programming is a formalism for declarative problem solving
- Solvers become more and more efficient and expressive.
- Combinations of logical rules and ontologies: e.g., DL-programs
- An approach for debugging ASP programs has been presented.
- Future work:
  - Debugging of DL-programs and HEX-programs.