Datalog-Based Data Access over Ontology Knowledge Bases
Unit 1 – Rules and Ontologies

Thomas Eiter
Institut für Informationssysteme, TU Wien

ICCL Summer School 2013, August 29-30, 2013

Austrian Science Fund (FWF) grants P20841, P24090
Unit Outline

1. Introduction

2. Description Logic Ontologies

3. LP/ASP Introduction

4. OWL vs Rules

5. Hybrid Knowledge Bases

6. Conclusion
**Issue:** Combining rules and ontologies (logic framework)

- Rules and ontology formalisms like RDF/s, OWL resp. Description Logics have related yet different underlying settings

- Combination is nontrivial (at the heart, the difference is between LP and classical logic)
OWL Ontologies

- Knowledge about concepts, individuals, their properties and relationships
- W3C standard (2004): *Web Ontology Language (OWL)*
- Three increasingly expressive sublanguages
  - **OWL Lite**: Concept hierarchies, simple constraint features.  
    \[ (\equiv \text{SHIF(D)}) \]
  - **OWL DL**: Basically, DAML+OIL.  
    \[ (\equiv \text{SHOIN(D)}) \]
  - **OWL Full**: Allow e.g. to treat classes as individuals.

- **OWL2** (2009): tractable profiles OWL2 EL, OWL2 QL, OWL2 RL
- OWL syntax is based on RDF
Description Logics (DLs)

Description Logics offer more expressivity than RDF/S!

- The vocabulary of basic DLs comprises:
  - Concepts (e.g., Wine, WhiteWine)
  - Roles (e.g., hasMaker, madeFromGrape)
  - Individuals (e.g., SelaksIceWine, TaylorPort)

- Statements relate individuals and their properties using
  - logical connectives (\(\cap, \cup, \neg, \subseteq\), etc), and
  - quantifiers (\(\exists, \forall, \leq k, \geq k\), etc)

- A DL knowledge base \(L = (\mathcal{T}, \mathcal{A})\) (ontology) usually comprises
  - a TBox \(\mathcal{T}\) (terminology, conceptualization), and
  - an ABox \(\mathcal{A}\) (assertions, extensional knowledge)

- DLs are tailored for decidable reasoning (key task: satisfiability)
Example: Wine Ontology

Available at http://www.w3.org/TR/owl-guide/wine.rdf
Example: Wine Ontology, cont’d

Some axioms from the TBox

\[ \text{Wine} \sqsubseteq \text{PotableLiquid} \sqcap =1\text{hasMaker} \sqcap \forall \text{hasMaker} . \text{Winery}; \]
\[ \exists \text{hasColor}^- . \text{Wine} \sqsubseteq \{"White", "Rose", "Red"\}; \]
\[ \text{WhiteWine} \equiv \text{Wine} \sqcap \forall \text{hasColor} . \{"White"\}. \]

- A wine is a potable liquid, having exactly one maker, who is a member of the class “Winery”.
- Wines have colors “White”, “Rose”, or “Red”.
- A \text{WhiteWine} is a wine with exclusive color “White”.

The ABox contains, e.g.,

\[ \text{WhiteWine("StGenevieveTexasWhite")}, \text{hasMaker("TaylorPort", "Taylor")} \]
Formal OWL / DL Semantics

- The semantics of core DLs is given by a mapping to first-order logic
- In many DLs, basic reasoning tasks can be reduced to core DLs

In essence, DLs are FO logic in disguise

<table>
<thead>
<tr>
<th>OWL property axioms as RDF Triples</th>
<th>DL syntax</th>
<th>FOL short representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨P rdfs:domain C⟩</td>
<td>∀P ⊑ ⊤</td>
<td>∀x, y.P(x, y) ⊑ C(x)</td>
</tr>
<tr>
<td>⟨P rdfs:range C⟩</td>
<td>∀P ⊑ ⊤</td>
<td>∀x, y.P(x, y) ⊑ C(y)</td>
</tr>
<tr>
<td>⟨P owl:inverseOf P₀⟩</td>
<td>P ≡ P₀</td>
<td>∀x, y.P(x, y) ≡ P₀(y, x)</td>
</tr>
<tr>
<td>⟨P rdf:type owl:SymmetricProperty⟩</td>
<td>1 ⊑ P</td>
<td>∀x, y₁, y₂.P(x₁, y₁) ∧ P(x₂, y₂) ⊑ y₁ = y₂</td>
</tr>
<tr>
<td>⟨P rdf:type owl:FunctionalProperty⟩</td>
<td>0 ⊑ P</td>
<td>∀x, y, z.P(x, y) ∧ P(y, z) ⊑ P(x, z)</td>
</tr>
<tr>
<td>⟨P rdf:type owl:TransitiveProperty⟩</td>
<td>+ ⊑ P</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OWL complex class descriptions</th>
<th>DL syntax</th>
<th>FOL short representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Thing</td>
<td>⊤</td>
<td>x = x</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>⊥</td>
<td>¬x = x</td>
</tr>
<tr>
<td>owl:intersectionOf (C₁ . . . Cₙ)</td>
<td>C₁ ⊓ . . . ⊓ Cₙ</td>
<td>∨ Cᵢ(x)</td>
</tr>
<tr>
<td>owl:unionOf (C₁ . . . Cₙ)</td>
<td>C₁ ⊔ . . . ⊔ Cₙ</td>
<td>∨ Cᵢ(x)</td>
</tr>
<tr>
<td>owl:complementOf (C)</td>
<td>¬C</td>
<td>¬C(x)</td>
</tr>
<tr>
<td>owl:oneOf (o₁ . . . oₙ)</td>
<td>{o₁ . . . oₙ}</td>
<td>∨ x = oᵢ</td>
</tr>
<tr>
<td>owl:restriction (P owl:someValuesFrom (C))</td>
<td>∃P.C</td>
<td>∃y.P(x, y) ∧ C(y)</td>
</tr>
<tr>
<td>owl:restriction (P owl:allValuesFrom (C))</td>
<td>∀P.C</td>
<td>∀y.P(x, y) ⊑ C(y)</td>
</tr>
<tr>
<td>owl:restriction (P owl:value (o))</td>
<td>∃P. {o}</td>
<td>P(x, o)</td>
</tr>
<tr>
<td>owl:restriction (P owl:minCardinality (n))</td>
<td>⊳n P</td>
<td>∃₁⩽₁y₁ ∩₁⩽₁^n P(x, y_j) ∧ ∩₁≠j y_i ≠ y_j</td>
</tr>
</tbody>
</table>
Logic Programming – Prolog revisited

1960s/70s: Logic as a Programming Language (?)

- Breakthrough in Computational Logic by Robinson’s discovery of the Resolution Principle (1965)

Kowalski (1979):

ALGORITHM = LOGIC + CONTROL

- Knowledge for problem solving (LOGIC)
- “Processing” of the knowledge (CONTROL)
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
Prolog, cont’d

**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).

**Example**

\[
\text{reverse}([X|Y], Z) :- \text{append}(U, [X], Z), \text{reverse}(Y, U). \\
\text{vs} \\
\text{reverse}([X|Y], Z) :- \text{reverse}(Y, U), \text{append}(U, [X], Z).
\]

**Query:** ?- \text{reverse}([a|X], [b,c,d,b])

**Is this truly declarative programming?**
LP Desiderata

Relieve the programmer from several concerns.

It is desirable that

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;
- termination is not subject to such order.

“Pure” declarative programming

- Prolog does not satisfy these desiderata
- Satisfied e.g. by the answer set semantics of logic programs
Positive Logic Programs

A positive logic program $P$ is a finite set of clauses (rules) of the form

$$a \leftarrow b_1, \ldots, b_m,$$

where $a, b_1, \ldots, b_m$ are atoms of a first-order language $L$.

- $a$ is the head of the rule
- $b_1, \ldots, b_m$ is the body of the rule.
- If $m = 0$, the rule is a fact (written shortly $a$)

Roughly, (1) can be seen as material implication $(\forall) b_1 \land \cdots \land b_m \supset a$.

If $L$ has no (proper) functions symbols, we have Datalog programs.

Example

$$\text{connected(cagliari)} \leftarrow \text{hub(rome)}, \text{link(rome, cagliari)}$$

$$\text{connected}(X) \leftarrow \text{hub}(Y), \text{link}(Y, X)$$
Semantics

Semantics is based on *Herbrand interpretations* (the domain is the *Herbrand universe* $HU(P)$, i.e. the set of all ground terms $t$; each $t$ is interpreted by itself).

- Herbrand interpretations are identified with subsets $I$ of the *Herbrand base* $HB(P)$ of $P$, i.e., the set of all ground atoms $p(t_1, \ldots, t_n)$ with predicate $p$ and terms $t_i$ from $HU(P)$.

- Programs $P$ are semantically equivalent to their grounding $\text{grnd}(P)$, i.e., all rules $r$ in $P$ are replaced by their ground instances over $HU(P)$.

- A (Herbrand) interpretation $I$ *satisfies (is a model)* of a rule $a \leftarrow b_1, \ldots, b_m$, if $\{b_1, \ldots, b_m\} \subseteq I$ implies $a \in I$, i.e., $a$ is true whenever $b_1, \ldots, b_m$ are true.

- $I$ *satisfies (is a model of)* $P$ if $I$ satisfies every $r$ in $\text{grnd}(P)$. 
Example (Program $P_1$)

$$p(f(X), Y, Z) \leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r).$$

$$h(0, 0).$$

$$h(a, b) \leftarrow h(a, b).$$

Which of the following (Herbrand) interpretations are models of $P_1$?

- $I_1 = \emptyset$  no
- $I_2 = \{p(t_1, t_2, t_3), h(t_1, t_2), t(t_1, t_2, t_3) \mid t_1, t_2, t_3 \text{ ground terms} \}$  yes
  
  Note: due to the function symbol $f$, there are infinitely many ground terms $t_i (0, f(0), f(f(0)), \ldots \text{etc})$

- $I_3 = \{h(0, 0), t(a, b, r), p(0, 0, b)\}$  no
Minimal Model Semantics

- A logic program has multiple models in general.
- Select one of these models as the canonical model.
- Commonly accepted: truth of an atom in model $I$ should be “founded” by clauses.

**Example**

**Given**

$$P_2 = \{ a \leftarrow b. \quad b \leftarrow c. \quad c \},$$

truth of $a$ in the model $I = \{ a, b, c \}$ is “founded.”

**Given**

$$P_3 = \{ a \leftarrow b. \quad b \leftarrow a. \quad c \},$$

truth of $a$ in the model $I = \{ a, b, c \}$ is not founded.
Minimal Model Semantics (cont’d)

**Semantics:** prefer models with true-part as small as possible.

**Minimal Model**

A model $I$ of $P$ is *minimal*, if there exists no model $J$ of $P$ such that $J \subset I$.

**Theorem**

*Every logic program $P$ has a single minimal model (called the least model), denoted $LM(P)$.*

**Example**

- For $P_2 = \{ a \leftarrow b. \ b \leftarrow c. \ c \}$, we have $LM(P_2) = \{a, b, c\}$.
- For $P_3 = \{ a \leftarrow b. \ b \leftarrow a. \ c \}$, we have $LM(P_3) = \{c\}$.
Computation

The minimal model can be computed via fixpoint iteration.

**$T_P$ Operator**

Let $T_P : 2^{HB(P)} \rightarrow 2^{HB(P)}$ be defined as

$$T_P(I) = \left\{ a \mid \text{there exists some } a \leftarrow b_1, \ldots, b_m \text{ in } \text{grnd}(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \right\}.$$

We let denote $T_P^0 = \emptyset$, $T_P^{i+1} = T_P(T_P^i)$, $i \geq 0$.

Fundamental result:

**Theorem**

*For every positive logic program $P$, the operator $T_P$ has a least fixpoint, $\text{lfp}(T_P)$, and the sequence $\langle T_P^i \rangle$, $i \geq 0$, converges to $\text{lfp}(T_P)$.***

Proof by the fixpoint theorems of Knaster-Tarski and Kleene.
Example

- For \( P_2 = \{ a \leftarrow b. \ b \leftarrow c. \ c \} \), we have

\[
T^0_{P_2} = \{\}, \ T^1_{P_2} = \{c\}, \ T^2_{P_2} = \{c, b\}, \ T^3_{P_2} = \{c, b, a\}, \ T^4_{P_2} = T^3_{P_2}
\]

Hence \( \text{lfp}(T_{P_2}) = \{c, b, a\} \)

- For \( P_3 = \{ a \leftarrow b. \ b \leftarrow a. \ c \} \), we have

\[
T^0_{P_3} = \{\}, \ T^1_{P_3} = \{c\}, \ T^2_{P_3} = T^1_{P_3}
\]

Hence \( \text{lfp}(T_{P_3}) = \{c\} \)
F-Logic Programming

Logic programs target unstructured (flat) objects

F-Logic Programs [Kifer et al., 1995]: prominent formalism to describe structured objects

Example

\[
\begin{align*}
\text{rome} & : \text{hub} \\
\text{rome}[\text{link} \rightarrow \text{cagliari}] & \\
X & : \text{connected} \leftarrow Y : \text{hub}, Y[\text{link} \rightarrow X]
\end{align*}
\]

- object-oriented:
  - \text{rome} : \text{hub} — Rome isa hub (\text{type}, \text{hub}(\text{rome}))
  - \text{rome}[\text{link} \rightarrow \text{cagliari}] — Rome has a link to Cagliari (\text{attributes}, \text{link}(\text{rome}, \text{cagliari}))

- higher-order: \text{rome} : X is like \text{X(rome)} (can be compiled away)
Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)
  
  E.g., "Men who are not husbands are singles."

Normal Logic Program

A *normal logic program* is a set of rules of the form

\[ a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \quad (n, m \geq 0) \]  

where \( a \) and all \( b_i, c_j \) are atoms in a first-order language \( L \).

\text{not} is called “negation as failure”, “default negation”, or “weak negation”

Things get more complex!
Programs with Negation

Prolog: “not \(X\)” means “Negation as Failure (to prove to \(X\))”

Different from negation in classical logic!

Example (Program \(P_4\))

\[
\text{man(dilbert).} \\
\text{single}(X) \leftarrow \text{man}(X), \text{not husband}(X). \\
\text{husband}(X) \leftarrow \text{fail}. \quad \% \text{fail = "false" in Prolog}
\]

Query:

\[? \leftarrow \text{single}(X).\]

Answer:

\[X = \text{dilbert} .\]
Example (cont’d)

Modifying the last rule of $P_4$, we get $P_5$:

\[
\begin{align*}
\text{man}(\text{dilbert}). \\
\text{single}(X) & \leftarrow \text{man}(X), \text{not } \text{husband}(X). \\
\text{husband}(X) & \leftarrow \text{man}(X), \text{not } \text{single}(X).
\end{align*}
\]

Result in Prolog ????

**Problem:** not a single intuitive model!

Two intuitive Herbrand models:

\[
\begin{align*}
M_1 & = \{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}) \}, \text{ and } \\
M_2 & = \{ \text{man}(\text{dilbert}), \text{husband}(\text{dilbert}) \}.
\end{align*}
\]

Which one to choose?
Semantics of Logic Programs With Negation

- “War of Semantics” in Logic Programming (1980/90ies):
  Meaning of programs like the Dilbert example above
- Great Schism: Single model vs. multiple model semantics
- To date:
  - **Answer Set (alias Stable Model) Semantics** by Gelfond and Lifschitz [1988, 1991].
    Alternative models: \( M_1 = \{ \text{man(dilbert), single(dilbert)} \} \),
    \( M_2 = \{ \text{man(dilbert), husband(dilbert)} \} \).
  - **Well-Founded Semantics** [van Gelder et al., 1991]
    Partial model: \( \text{man(dilbert)} \) is true,
    \( \text{single(dilbert), husband(dilbert)} \) are unknown
- Agreement for so-called “stratified programs” (acyclic negation)
  Different selection principles for non-stratified programs
Rules and OWL

- What of OWL can be expressed directly in rules?

- What is different? Existentials, number restrictions, equality reasoning, etc.
What of OWL can be expressed directly in rules?

**ABox** factual knowledge about Class membership and property values and can be translated to LP facts “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>Intuitive correspondence with LP rules/facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( paper_1 \in Paper )†</td>
<td>( Paper(paper_1) )</td>
</tr>
<tr>
<td>( (paper_1, stHeymans) \in hasAuthor )</td>
<td>( hasAuthor(paper_1, stHeymans) )</td>
</tr>
</tbody>
</table>

**RBox/TBox**: A subset of OWL can be straightforwardly translated to Rules, e.g.:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>Intuitive correspondence with LP rules/facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R \sqsubseteq S ) (SubPropertyOf)</td>
<td>( S(X, Y) \leftarrow R(X, Y) )</td>
</tr>
<tr>
<td>( R^+ \sqsubseteq R ) (Transitive Property)</td>
<td>( R(X, Z) \leftarrow R(X, Y), R(Y, Z) )</td>
</tr>
<tr>
<td>( C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A)</td>
<td>( A(X) \leftarrow C_1(X), \ldots, C_n(X) )</td>
</tr>
<tr>
<td>( A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n)</td>
<td>( C_1(X) \leftarrow A(X); \ldots; C_n(X) \leftarrow A(X) )</td>
</tr>
<tr>
<td>( \exists R.C \sqsubseteq A ) (SomeValuesFrom, lhs)</td>
<td>( A(X) \leftarrow R(X, Y), C(Y) )</td>
</tr>
<tr>
<td>( A \sqsubseteq \forall R.C ) (AllValuesFrom, rhs)</td>
<td>( C(Y) \leftarrow R(X, Y), A(X) )</td>
</tr>
<tr>
<td>( C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A ) (UnionOf lhs)</td>
<td>( A(X) \leftarrow C_1(X); \ldots; A(X) \leftarrow C_n(X) )</td>
</tr>
</tbody>
</table>

†: use this notation for assertions
What of OWL cannot be expressed directly in Rules?

- Some OWL statements can only be approximated by a naive translation:

\[ A \equiv \{o_1, \ldots, o_n\} \text{ (OneOf)} \]

\[ A \sqsubseteq \exists R.C \text{ (SomeValuesFrom rhs)} \]

\[ A(o_1), \ldots, A(o_n) \text{ does not work (what with } A(b)?) \]

Can only be approximated using function symbols (Skolem terms) \( \leadsto \) need existential rules

- Other OWL statements are even problematic to be approximated:

\[ \forall R.C \sqsubseteq A \text{ (AllValuesFrom lhs)} \]

One might guess:

\[ A(X) \leftarrow \text{not noRC}(X); \]

\[ \text{noRC}(X) \leftarrow R(X, Y), \text{not } C(Y). \]

but doesn’t work :-(

cardinality restrictions, SameAs

Need reasoning with equality, expensive to implement.

Recall: “=” and “!=” are not classical equality but builtin syntactic equality (UNA, CWA)!

\[ \ldots \text{etc.} \]
Main Differences OWL vs. Rules?

- **not** in rule paradigms is different from negation (e.g., ComplementOf) in OWL:
  - \( \neg \): Classical negation! Open world assumption! Monotonicity!
  - \( \text{not} \): Different purpose! Closed world assumption! Non-monotonicity!

\[
\begin{align*}
\text{Publication} & \sqsubseteq \text{Paper} \\
\neg \text{Publication} & \sqsubseteq \text{Unpublished} \\
\text{paper}_1 & \in \text{Paper}.
\end{align*}
\]

in DL: \( \not\equiv \text{paper}_1 \in \text{Unpublished} \)

\[
\begin{align*}
\text{Paper}(X) & \leftarrow \text{Publication}(X) \\
\text{Unpublished}(X) & \leftarrow \text{not} \ \text{Publication}(X) \\
\text{Paper}(\text{paper}_1) & \leftarrow \\
\end{align*}
\]

Does infer in LP: \( \text{Unpublished}(\text{paper}_1) \).

- Also **strong negation** in LP ("\( \neg \)"), sometimes "\( \rightarrow \)"
  is not completely the same as classical negation in DLs, e.g.

\[
\begin{align*}
\text{Publication} & \sqsubseteq \text{Paper} \\
\text{stHeymans} & \in \neg \text{Paper}.
\end{align*}
\]

in DL:
\( \not\equiv \text{stHeymans} \in \neg \text{Publication} \)

\[
\begin{align*}
\text{Paper}(X) & \leftarrow \text{Publication}(X) \\
\neg \text{Paper}(\text{stHeymans}) & \leftarrow \\
\end{align*}
\]

Does not automatically infer in LP:
\( \neg \text{Publication}(\text{stHeymans}) \).
Main Differences OWL vs. Rules?

- LPs are strong in query answering, but subsumption checking as in DLs is infeasible (undecidable even for positive function-free programs).
- OWL DL allows complex statements in the “head” (rhs of $\sqsubseteq$), while use of variables in LP rule bodies is more flexible.
- DLs are stronger in type inference, while LPs are stronger in type checking:

<table>
<thead>
<tr>
<th>LP Rule</th>
<th>OWL DL Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Person} \sqsubseteq \exists \text{hasName}.xs: \text{string}$</td>
<td>$\text{Person}(X), \neg \text{hasName}(X, Y) \rightarrow \text{Person}(\text{john})$</td>
</tr>
<tr>
<td>$\text{john} \in \text{Person}$</td>
<td>$\text{john} \in \exists \text{hasName}$</td>
</tr>
<tr>
<td>is consistent in DL and infers $\text{john} \in \exists \text{hasName}$</td>
<td>is inconsistent, since there is no known name for $\text{john}$</td>
</tr>
</tbody>
</table>
Experience from Practice

- Rules are more flexible than OWL for expressing ternary relations.
- Use of aggregate functions and built-ins common in LP (e.g., `<`).
- Minimality in rules allows to express transitive closure.
- Different usage: OWL semantics would infer values (or use *null values*) if not present, while LP semantics indicates inconsistency if not present.
- Disadvantage rules: closed world reasoning (need a representative data set – which in practical cases is usually there).
Main Differences Summary

- CWA vs OWA
- Existential quantification
- UNA
- Negation as failure, strong negation vs. classical negation
- Symmetry between “head” and “body” (DL) vs. more complex bodies (LP)
- Type inference (DL) vs. type checking (LP)
Marrying Rules and Ontologies

- Hybrid knowledge base: $\mathcal{K} = (O, P)$
  - $O$ is an ontology
    
    $$Father \equiv \text{Man} \sqcap \exists \text{hasChild.Human}$$
  
  - $P$ is the rules part (program)
    
    $$\text{rich}(X) \leftarrow \text{famous}(X), \text{not scientist}(X)$$

- Description Logic Programs [Grosof et al., 2003]
- DL-safe rules [Motik et al., 2005]
- $r$-hybrid KBs [Rosati, 2005]
- hybrid MKNF KBs [Motik and Rosati, 2010]
- Description Logic Rules [Krötzsch et al., 2008a]
- ELP [Krötzsch et al., 2008b]
- $\mathcal{DL}+\text{log}$ [Rosati, 2006]
- SWRL [Horrocks et al., 2004]
- dl-programs [E_ et al., 2008]
- ...
Semantics

Different ways to give semantics to $\mathcal{K} = (\mathcal{O}, P)$
overviews e.g. [Motik and Rosati, 2010], [de Bruijn et al., 2009]

- Tight semantic integration
- Full integration
- Strict semantic separation (loose coupling)

Nonmonotonic semantics:

- answer sets
- well-founded semantics
- ...
Tight Semantic Integration

- Integrate FOL statements and the logic program to a large extent, but keep predicates of $\Sigma_\mathcal{O}$ and $\Sigma_P$ separate.
- Build an integrated model $M$ as the “union” of a model $M_\mathcal{O}$ of the FO theory $\mathcal{O}$ and a model $M_P$ of $P$ with the same domain.
- Ensure “safe interaction” between $M_\mathcal{O}$ and $M_P$.

Examples

- **CARIN** [Levy and Rousset, 1998], **DLP (≈ OWL 2 RL)** [Grososf et al., 2003],
- **dl-safe rules** [Motik et al., 2005], **R-hybrid KBs** [Rosati, 2005]
- **DL+LOG** [Rosati, 2006]
Full Integration

- No fundamental separation between $\Sigma_O$, $\Sigma_P$ (but special axioms)

Examples

- **Hybrid MKNF knowledge bases** [Motik and Rosati, 2010; Knorr et al., 2008]
- **FO-Autoepistemic Logic** [de Bruijn et al., 2007a]
- **Quantified Equilibrium Logic** [de Bruijn et al., 2007b] (use special axioms)
Loose Coupling

- Strict semantic separation between rules / ontology

- View rule base $P$ and FO theory $\mathcal{O}$ as separate, independent components. $\Sigma_{\mathcal{O}}$ and $\Sigma_{P}$ do (a priori) not share meaning.

- They are connected through a minimal “safe interface” for exchanging knowledge (formulas, usually ground atoms).

- Well-suited for implementation on top of LP & DL reasoners.

Examples

- **nonmonotonic dl-programs** [E_ et al., 2008], [E_ et al., 2011]
- **defeasible logic+DLs** [Wang et al., 2004]
Notions of Safety

- Levy and Rousset [1998]: combinations of Horn logic and very simple DLs are undecidable
- Problems with recursion and \textit{unsafety} of rules
- Traditional in LP: A rule $r$ is \textit{safe}, if each variable in $r$ occurs in a positive literal in $r$’s body
- Variants of safety are a key tool for decidability of combinations
  - \textit{role-safety}: [Levy and Rousset, 1998]
    For every role atom $R(X, Y)$ in rule $r$, either $X$ or $Y$ occurs with a $\Sigma_P$-predicate in $r$ that does not occur in any rule head of $P$.
  - \textit{dl-safe rules}: [Motik \textit{et al.}, 2005]
    each variable occurs in some positive body literal with a $\Sigma_P$-predicate
  - \textit{weakly dl-safe rules}: [Rosati, 2006]
    the $\Sigma_P$-subrule must be safe, and each variable that occurs with a $\Sigma_O$-predicate in the head must occur in some positive body atom with a $\Sigma_P$-predicate
Notions of Safety, cont’d

Example

\[ \text{uncleOf}(X, Y) \leftarrow \text{parentOf}(Z, Y), \text{brotherOf}(X, Z). \]

is not DL-safe; its variant

\[ \text{uncleOf}(X, Y) \leftarrow \text{parentOf}(Z, Y), \text{brotherOf}(X, Z), \]
\[ \text{person}(X), \text{person}(Y), \]

where \text{person} is for facts in \(P\), is DL-safe (and weakly dl-safe).

Example

\[ \text{parent}(X) \leftarrow \text{person}(X), \text{parentOf}(X, Y). \]

is weakly dl-safe, for \text{person} as above, but not dl-safe.

The rule

\[ \text{email}(X) \leftarrow \text{person}(X), \neg \text{hasBought}(X, Y), \text{Article}(Y). \]

is weakly dl-safe, also if \text{Article}(Y) is missing.
Conclusion

- (Logic Programming) rules and ontologies behave differently
- A number of combination formalisms
- Different levels of integration
- Issues like decidability, complexity come up
- Other notions of rules (e.g. production rules) were considered
- Standardization of combinations of rules and ontologies is embryonic (OWL-RIF)
Jos de Bruijn, Thomas Eiter, Axel Florian Polleres, and Hans Tompits.
Embedding non-ground logic programs into autoepistemic logic for knowledge base
combination.
In Manuela Veloso, editor, Proceedings of the 20th International Joint Conference on Artificial

Jos de Bruijn, David Pearce, Axel Polleres, and Agustín Valverde.
Quantified equilibrium logic and hybrid rules.

Jos de Bruijn, Philippe Bonnard, Hugues Citeau, Sylvain Dehors, Stijn Heymans, Jörg Pührer,
and Thomas Eiter.
http://ontorule-project.eu/.

T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits.
Combining answer set programming with description logics for the Semantic Web.

T. Eiter, G. Ianni, T. Lukasiewicz, and R. Schindlauer.
Well-founded semantics for description logic programs in the Semantic Web.
M. Gelfond and V. Lifschitz.
The Stable Model Semantics for Logic Programming.

M. Gelfond and V. Lifschitz.
Classical Negation in Logic Programs and Disjunctive Databases.

Description logic programs: Combining logic programs with description logics.

SWRL: A semantic web rule language combining OWL and RuleML.

M. Kifer, G. Lausen, and J. Wu.
Logical foundations of object-oriented and frame-based languages.
M. Knorr, J.J. Alferes, and P. Hitzler.
A coherent well-founded model for hybrid MKNF knowledge bases.

M. Krötzsch, S. Rudolph, and P. Hitzler.
Description logic rules.

M. Krötzsch, S. Rudolph, and P. Hitzler.
ELP: Tractable rules for OWL 2.

Alon Y. Levy and Marie-Christine Rousset.
Combining horn rules and description logics in CARIN.

Boris Motik and Riccardo Rosati.
Reconciling description logics and rules.
To appear.

Boris Motik, Ulrike Sattler, and Rudi Studer.
Query answering for OWL-DL with rules.
Riccardo Rosati.
On the decidability and complexity of integrating ontologies and rules.

Riccardo Rosati.
*DL+log*: Tight Integration of Description Logics and Disjunctive Datalog.

A. van Gelder, K.A. Ross, and J.S. Schlipf.
The Well-Founded Semantics for General Logic Programs.

Kewen Wang, David Billington, Jeff Blee, and Grigoris Antoniou.
Combining description logic and defeasible logic for the semantic web.