OWL, DL and Rules

Based on slides from Grigoris Antoniou, Frank van Harmele and Vassilis Papataxiarhis
The Semantic Web is grounded in logic

But what logic?
- OWL Full = Classical first order logic (FOL)
- OWL-DL = Description logic
- N3 rules $\sim=$ logic programming (LP) rules
- SWRL $\sim=$ DL + LP
- Other choices are possible, e.g., default logic, Markov logic, ...

How do these fit together?

What are the consequences
We need both structure and rules

- **OWL’s ontologies** are based on Description Logics (and thus in FOL)
  - The Web is an open environment.
  - Reusability / interoperability.
  - An ontology is a model easy to understand.

- Many **rule systems** based on logic programming
  - For the sake of decidability, ontology languages don’t offer the expressiveness we want (e.g. constructor for composite properties?). Rules do it well.
  - Efficient reasoning support already exists.
  - Rules are well-known in practice.
A common approach

High Expressiveness

Conceptualization of the domain

Rules Layer

Ontology Layer

OWL-DL

SWRL

Trusted SW

Proof

Logic

Rules/Query

Ontology

RDF Model & Syntax

XML Query

XML Schema

XML

Namespaces

URI/IRI

Unicode
LP and classical logic overlap

FOL: (All except (6)), (2)+(3)+(4): DLs
(4): Description Logic Programs (DLP), (3): Classical Negation
(4)+(5): Horn Logic Programs, (4)+(5)+(6): LP
(6): Non-monotonic features (like NAF, etc.) (7): ^head and, ∨ body
Description Logics vs. Horn Logic

- Neither of them is a subset of the other.
- It is impossible to assert that persons who study and live in the same city are “home students” in OWL.
  - This can be done easily using rules:
    \[
    \text{studies}(X,Y), \text{lives}(X,Z), \text{loc}(Y,U), \text{loc}(Z,U) \rightarrow \text{homeStudent}(X)
    \]
- Rules cannot assert the information that a person is either a man or a woman.
  - This information is easily expressed in OWL using disjoint union.
Basic Difficulties

- Monotonic vs. Non-monotonic Features
  - Open-world vs. Closed-world assumption
  - Negation-as-failure vs. classical negation
- Non-ground entailment
- Strong negation vs. classical negation
- Equality
- Decidability

Classical Logic vs. Logic Programming
Prolog and most ‘logic’-oriented rule languages use horn clause logic
- Cf. UCLA mathematician Alfred Horn

Horn clauses are a subset of FOL where every sentence is a disjunction of literals (atoms) where at most one is positive

\[ \sim P \lor \sim Q \lor \sim R \lor S \]
\[ \sim P \lor \sim Q \lor \sim R \]
An alternate formulation

- Horn clauses can be re-written using the implication operator
  
  \[ \sim P \lor Q = P \rightarrow Q \]
  
  \[ \sim P \lor \sim Q \lor R = P \land Q \rightarrow R \]
  
  \[ \sim P \lor \sim Q = P \land Q \rightarrow \]

- What we end up with is \sim “pure prolog”
  
  - Single positive atom as the rule conclusion
  - Conjunction of positive atoms as the rule antecedents (conditions)
  - No \textbf{not} operator
  - Atoms can be predicates (e.g., mother(X,Y))
Where are the quantifiers?

- Quantifiers (for all, exists) are implicit
  - Variables in head are universally quantified
  - Variables only in body are existentially quantified

Example:
- isParent(X) ← hasChild(X,Y)
- forAll X: isParent(X) if Exits Y: hasChild(X,Y)
We can relax this a bit

Head can contain a conjunction of atoms
- \( P \land Q \leftarrow R \) is equivalent to \( P \leftarrow R \) and \( Q \leftarrow R \)

Body can have disjunctions
- \( P \leftarrow R \lor Q \) is equivalent to \( P \leftarrow R \) and \( P \leftarrow Q \)

But something are just not allowed:
- No disjunction in head
- No negation operator, i.e. NOT
Facts & rule conclusions are definite

- A fact is just a rule with the trivial true condition
- Consider these true facts:
  - P ∨ Q
  - P ➝ R
  - Q ➝ R
- What can you conclude?
- Can this be expressed in horn logic?
Facts & rule conclusions are definite

- Consider these true facts:
  - $\neg(P) \rightarrow Q$, $\neg(Q) \rightarrow P$
  - $P \rightarrow R$
  - $Q \rightarrow R$

- A horn clause reasoner (e.g., Prolog) will be unable to prove that either $P$ or $Q$ is necessarily true or false

- And can not show that $R$ must be true
Open- vs. closed-world assumption

- **Logic Programming – CWA**
  - If $\text{KB} \not\models a$, then $\text{KB} = \text{KB} \cup \neg a$

- **Classical Logic – OWA**
  - It keeps the world open.
    - KB:
      - $\text{Man} \subseteq \text{Person}, \text{Woman} \subseteq \text{Person}$
      - $\text{Bob} \in \text{Man}, \text{Mary} \in \text{Woman}$
  - Query: “find all individuals that are not women”
Non-ground entailment

- The LP-semantics is defined in terms of minimal Herbrand model, i.e. sets of ground facts.
- Because of this, Horn clause reasoners cannot derive rules, so they cannot do general subsumption reasoning.
Decidability

- The largest obstacle!
  - Tradeoff between expressiveness and decidability.

- Facing decidability issues from 2 different angles
  - In LP: Finiteness of the domain
  - In classical logic (and thus in DL): Combination of constructs

- Problem:
  Combination of “simple” DLs and Horn Logic are undecidable. (Levy & Rousset, 1998)
Still a challenging task!

A number of different approaches exists: SWRL, DLP (Grosof), dl-programs (Eiter), DL-safe rules, Conceptual Logic Programs (CLP), AL-Log, DL +log

Two main strategies:
- Tight Semantic Integration (Homogeneous Approaches)
- Strict Semantic Separation (Hybrid Approaches)
Homogeneous Approach

- Interaction with tight semantic integration.
- Both ontologies and rules are embedding in a common logical language.
- No distinction between rule predicates and ontology predicates.
- Rules may be used for defining classes and properties of the ontology.
- Example: SWRL, DLP
Hybrid Approach

- Integration with strict semantic separation between the two layers.
- Ontology is used as a conceptualization of the domain.
- Rules cannot define classes and properties of the ontology, but some application-specific relations.
- Communication via a “safe interface”.
- Example: Answer Set Programming (ASP)
The Essence of DLP

- Simplest approach for combining DLs with Horn logic: their intersection
  - the Horn-definable part of OWL, or equivalently
  - the OWL-definable part of Horn logic
Advantages of DLP

- **Modeling**: Freedom to use either OWL or rules (and associated tools and methodologies)
- **Implementation**: use either description logic reasoners or deductive rule systems
  - extra flexibility, interoperability with a variety of tools
- **Expressivity**: existing OWL ontologies frequently use very few constructs outside DLP
<table>
<thead>
<tr>
<th>RDFS and Horn Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement(a, P, b)</td>
</tr>
<tr>
<td>type(a, C)</td>
</tr>
<tr>
<td>C subClassOf D</td>
</tr>
<tr>
<td>P subPropertyOf Q</td>
</tr>
<tr>
<td>domain(P, C)</td>
</tr>
<tr>
<td>range(P, C)</td>
</tr>
</tbody>
</table>
C sameClassAs D \quad C(X) \rightarrow D(X)
D(X) \rightarrow C(X)

P samePropertyAs Q \quad P(X,Y) \rightarrow Q(X,Y)
Q(X,Y) \rightarrow P(X,Y)
transitiveProperty(P) \[ P(X, Y), P(Y, Z) \rightarrow P(X, Z) \]

inverseProperty(P, Q) \[ Q(X, Y) \rightarrow P(Y, X) \]
\[ P(X, Y) \rightarrow Q(Y, X) \]

functionalProperty(P) \[ P(X, Y), P(X, Z) \rightarrow Y = Z \]
(C₁ ∩ C₂) subClassOf D
   C₁(X), C₂(X) → D(X)

C subClassOf (D₁ ∩ D₂)
   C(X) → D₁(X)
   C(X) → D₂(X)
(C₁ \cup C₂) \text{ subClassOf } D
\begin{align*}
C₁(X) & \rightarrow D(X) \\
C₂(X) & \rightarrow D(X)
\end{align*}
C \text{ subClassOf } (D₁ \cup D₂)

Translation not possible!
C subClassOf AllValuesFrom(P,D)
C(X), P(X,Y) → D(Y)

AllValuesFrom(P,D) subClassOf C

Translation not possible!
C subClassOf SomeValuesFrom(P,D)

Translation not possible!

SomeValuesFrom(P,D) subClassOf C
D(X), P(X,Y) → C(Y)
MinCardinality cannot be translated due to existential quantification

MaxCardinality 1 may be translated if equality is allowed

Complement cannot be translated, in general
The Essence of SWRL

- Combines OWL DL (and thus OWL Lite) with function-free Horn logic.
- Thus it allows Horn-like rules to be combined with OWL DL ontologies.
Rules in SWRL

\[ B_1, \ldots, B_n \rightarrow A_1, \ldots, A_m \]

\[ A_1, \ldots, A_m, B_1, \ldots, B_n \] have one of the forms:

- \( C(x) \)
- \( P(x,y) \)
- sameAs(x,y) differentFrom(x,y)

where C is an OWL description, P is an OWL property, and x,y are variables, OWL individuals or OWL data values.
**Drawbacks of SWRL**

- **Main source of complexity:**
  - arbitrary OWL expressions, such as restrictions, can appear in the head or body of a rule.

- **Adds significant expressive power to OWL, but causes undecidability**
  - there is no inference engine that draws exactly the same conclusions as the SWRL semantics.
SWRL Sublanguages

- SWRL adds the expressivity of DLs and function-free rules.
- One challenge: identify sublanguages of SWRL with right balance between expressivity and computational viability.
- A candidate OWL DL + **DL-safe rules**
  - every variable must appear in a non-description logic atom in the rule body.
<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-PlaysWith2CentralDefenders</td>
<td>Team(?t) ∧ isOpponentTeam(?t, true) ∧ hasStriker(?t, ?x1) ∧ hasStriker(?t, ?x2) ∧ differentFrom(?x1, ?x2) ∧</td>
</tr>
<tr>
<td>Def-PlaysWith2Strikers</td>
<td>hasStriker(?t, ?x) ∧ hasStriker(?t, ?y) ∧ differentFrom(?x, ?y) → hasStriker Predicate (?t)</td>
</tr>
</tbody>
</table>
| Def-PlaysWith2Strikers-OPPONENT                | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, striker) ∧ Player(?p2) ∧ playsInPosition ...
| Def-PlaysWith3Strikers-OPPONENT                | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, striker) ∧ Player(?p2) ∧ playsInPosition ...
| Def-PlaysWithUniqueStriker-OPPONENT            | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, striker) → playsWithUniqueStriker(?t) |
| Def1-hasStriker                                | Player(?x) ∧ Team(?t) ∧ memberOfTeam(?x, ?t) ∧ playsInPosition(?x, ?p) ∧ Striker(?p) → hasStriker(?t, ?x) |
| Def1-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, good) |
| Def1-PlaysWith2CentralDefenders-ZONE-OPPONENT  | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, stoper) ∧ Player(?p2) ∧ playsInPosition ...
| Def1-PlaysWith3CentralDefenders-ZONE-OPPONENT  | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, stoper) ∧ Player(?p2) ∧ playsInPosition ...
| Def2-hasStriker                                | Player(?x) ∧ Team(?t) ∧ memberOfTeam(?x, ?t) ∧ playsInPosition(?x, ?p) ∧ SecondStriker(?p) → hasStriker(?t, ?x) |
| Def2-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, veryG) |
| Def2-PlaysWith2CentralDefenders-WithCoverPlayer-OPPONENT | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, stoper) ∧ Player(?p2) ∧ playsInPosition ...
| Def2-PlaysWith3CentralDefenders-WithCoverPlayer-OPPONENT | Team(?t) ∧ isOpponentTeam(?t, true) ∧ Player(?p1) ∧ playsInPosition(?p1, stoper) ∧ Player(?p2) ∧ playsInPosition ...
| Def3-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, medium) |
| Def4-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, good) |
| Def5-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, veryG) |
| Def6-isQuickerThan                             | Player(?x1) ∧ Player(?x2) ∧ hasAcceleration(?x1, ?a1) ∧ hasAcceleration(?x2, ?a2) ∧ hasQuality(?a1, veryG) |
See http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab for SWRLJessTab documentation.

Press the "OWL+SWRL->Jess" button to transfer SWRL rules and relevant OWL knowledge to Jess.
Press the "Run Jess" button to run the Jess rule engine.
Press the "Jess->OWL" button to transfer the inferred Jess knowledge to OWL knowledge.

IMPORTANT: With the exception of owl:SameAs, owl:DifferentFrom and owl:AllDifferent, the Jess rule engine is currently ignoring OWL restrictions. To ensure consistency, a reasoner should be run on an OWL knowledge base before SWRL rules and OWL knowledge are transferred to Jess. Also, if inferred knowledge from Jess is inserted back into an OWL knowledge base, a reasoner should again be executed to ensure that the new knowledge does not conflict with OWL restrictions in that knowledge base.


### SWRL Rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-PlaysWith2CentralDefenders</td>
<td>$\rightarrow \text{Team}(?t) \land \text{isOpponentTeam}(?t, \text{true}) \land \text{hasStriker}(?t, ?x1) \land \text{hasStriker}(?t, ?x2) \land \text{differentFrom}(?x1, ?x2) \land \text{hasStriker}(?x1, ?x3) \land \text{hasStriker}(?x2, ?x4) \land \text{differentFrom}(?x3, ?x4)$</td>
</tr>
<tr>
<td>Def-PlaysWith2Strikers</td>
<td>$\rightarrow \text{hasStriker}(?t, ?x) \land \text{hasStriker}(?t, ?y) \land \text{differentFrom}(?x, ?y) \rightarrow \text{&lt;classPredicate&gt;}(?t)$</td>
</tr>
<tr>
<td>Def1-hasStriker</td>
<td>$\rightarrow \text{Player}(?x) \land \text{Team}(?t) \land \text{memberOfTeam}(?x, ?t) \land \text{playsInPosition}(?x, ?p) \land \text{Striker(?p)} \land \text{hasStriker}(?p, ?x)$</td>
</tr>
<tr>
<td>Def2-hasStriker</td>
<td>$\rightarrow \text{Player}(?x) \land \text{Team}(?t) \land \text{memberOfTeam}(?x, ?t) \land \text{playsInPosition}(?x, ?p) \land \text{SecondStriker(?p)}$</td>
</tr>
</tbody>
</table>
Non-monotonic rules

- Non-monotonic rules exploit an “unprovable” operator
- This can be used to implement default reasoning, e.g.,
  - assume \( P(X) \) is true for some \( X \) unless you can prove that it is not
  - Assume that a bird can fly unless you know it can not
canFly(X) :- bird(X)
bird(X) :- eagle(X)
bird(X) :- penguin(X)
eagle(sam)
penguin(tux)
canFly(X) :- bird (X), \+ not(canFly(X))
bird(X) :- eagle(X)
bird(X) :- penguin(X)
not(canFly(X)) :- penguin(X)
eagle(sam)
penguin(tux)
Rule priorities

- This approach can be extended to implement systems where rules have priorities.
- This seems to be intuitive to people – used in many human systems.
  - E.g., University policy overrules Department policy.
  - The “Ten Commandments” can not be contravened.
Two Semantic Webs?
Limitations

- The rule inference support not integrated with OWL classifier.
- New assertions by rules may violate existing restrictions in ontology. New inferred knowledge from classification may in turn produce knowledge useful for rules.
Limitations

- **Existing solution:**
  Solve these possible conflicts manually.

- **Ideal solution:**
  Have a single module for both ontology classification and rule inference.

- **What if we want to combine non-monotonic features with classical logic?**
  - **Partial Solutions:**
    - Answer set programming
    - Externally (through the use of appropriate rule engines)
Summary

- Horn logic is a subset of predicate logic that allows efficient reasoning, orthogonal to description logics.
- Horn logic is the basis of monotonic rules.
- DLP and SWRL are two important ways of combining OWL with Horn rules.
  - DLP is essentially the intersection of OWL and Horn logic.
  - SWRL is a much richer language.
Nonmonotonic rules are useful in situations where the available information is incomplete.

They are rules that may be overridden by contrary evidence.

Priorities are sometimes used to resolve some conflicts between rules.

Representation XML-like languages is straightforward.