Semantic Web Rule Language

Martin O’Connor
Stanford Medical Informatics
What is SWRL?

• SWRL is an acronym for Semantic Web Rule Language.
• SWRL is intended to be the rule language of the Semantic Web.
• SWRL is based on OWL: all rules are expressed in terms of OWL concepts (classes, properties, individuals, literals...).
Example SWRL Rule: Has brother

Person(?p) ^ hasSibling(?p, ?s) ^ Man(?s) -> hasBrother(?p, ?s)
Example SWRL Rule with Named Individuals: Has brother

\[ \text{Person}(Fred) \land \text{hasSibling}(Fred, \ ?s) \land \text{Man}(\ ?s) \rightarrow \text{hasBrother}(Fred, \ ?s) \]
Example SWRL Rule with Literals: Has brother

Person(Fred) ^ hasSibling(Fred, ?s) ^ Man(?s) ^ hasAge(?s, 40)
→ has40YearOldBrother(Fred, ?s)
Example SWRL Rule with Built-ins: Has brother

\[
\text{hasBrother}(\text{?x1}, \text{?x2}) \land \text{hasAge}(\text{?x1}, \text{?age1}) \\
\land \text{hasAge}(\text{?x2}, \text{?age2}) \land \text{swrlb:greaterThan}(\text{?age2}, \text{?age1}) \\
\rightarrow \text{hasOlderBrother}(\text{?x1}, \text{?x2})
\]
Example SWRL Rule with Built-ins: Has brother

\[
\text{hasBrother}(\text{?x1}, \text{?x2}) \land \text{hasAge}(\text{?x1}, \text{?age1}) \\
\land \text{hasAge}(\text{?x2}, \text{?age2}) \land \text{swrlb:subtract}(10, \text{?age2}, \text{?age1}) \\
\rightarrow \text{hasDecadeOlderBrother}(\text{?x1}, \text{?x2})
\]
What is the SWRL Tab?

- The SWRL Tab is an extension to the Protégé-OWL Plugin that permits the creation and execution of SWRL rules.

- The editor can be used to create SWRL rules, edit existing SWRL rules, and read and write SWRL rules.

- Allows a variety of third-party rule engines to be plugged in to do inference

- Currently supports inference with the Jess rule engine
How do I activate the SWRL Tab?

- The SWRL Tab is included as part of the OWL Plugin.
- This tab should be visible for all OWL knowledge bases that import the SWRL Ontology:
- To activate the SWRL Tab in a project that does not include this ontology, go to Project/Configure and check the "SWRLTab" box.
- Option will be given to import the SWRL ontology.
<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-hasAunt</td>
<td>hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasBrother</td>
<td>hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasDaughter</td>
<td>hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasFather</td>
<td>hasParent(?x, ?y) ∧ Man(?x) → hasFather(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasMother</td>
<td>hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNephew</td>
<td>hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasNiece</td>
<td>hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasParent</td>
<td>hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSibling</td>
<td>hasChild(?x, ?y) ∧ hasChild(?z, ?y) ∧ differentFrom(?x, ?y) → hasSibling(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSister</td>
<td>hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSon</td>
<td>hasChild(?x, ?y) ∧ Man(?x) → hasSon(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasUncle</td>
<td>hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)</td>
</tr>
</tbody>
</table>
• To edit a rule in-place, simply left-click on the rule text.

• An icon panel will pop up and the rule can then be directly edited.

• To save the edited rule, press the Enter key.

• To abandon editing, press the Escape key.
• To edit in a multi-line editor, right-click on the rule and select "Edit rule in multi-line editor..." from the popup menu.

• A modal popup dialog box will appear with a text area and an icon panel.

• To save the edited rule press "Ok".

• To abandon editing, press "Cancel".
A simple Rule to capture the definition of "uncle". Note that in contrast to pure OWL, SWRL provides mechanisms to represent variables, and therefore is quite rich.

```
hasParent(?x, ?y) ∧
hasBrother(?y, ?z)
→ hasUncle(?x, ?z)
```
What are all those pretty icons?

- SWRL rules can be entered completely from the keyboard.

- The icon panel allows the user to select the appropriate OWL classes, properties, and individuals from the knowledge base.

- It also includes shortcuts and selection dialog boxes for various other entities.

- The tool tips text for each icon will explain its purpose.
Finding SWRL Rules about particular OWL Entities

- The OWL Plugin allows users to find rules relating to the currently selected OWL class, property, or individual.

- For example, users can click on the arrow symbol at the bottom left of class editor subpane to bring up a list of SWRL rules that refer to that class.

- The same mechanism applies in the properties and individuals tabs.
What checking does the SWRL Editor do?

- Only syntactically valid rules can be saved.
- The SWRL editor will only allow saving of rules relating to currently loaded OWL entities.
- Basic semantic checking, e.g., no variables can be used in a rule consequent that were not referred to in the antecedent.
- However, no elaborate sanity checking is performed, e.g., rule could contradict OWL constraints.
How are SWRL Rules Saved?

- SWRL rules are saved as OWL individuals with their associated OWL file.
- Classes that describe this ontology are contained in SWRL Ontology: http://www.daml.org/rules/proposal/swrl.orl
- These classes include:
  - `swrl:Imp` – represents a single SWRL rule
  - `swrl:Atom` – represents a single rule atom
  - `swrl:AtomList` – represent a list of atoms
- Other rule engines can use these rules, e.g., SweetRules.
- The SWRL API provides a mechanism to create and manipulate SWRL rules in an OWL knowledge base.

- This API is used by the SWRL Editor.

- However, it is accessible to all OWL Plugin developers.

- Third party software can use this API to work directly with SWRL rules.
Adding a Third Party Rule Engine

• SWRL Editor has been available as part of Protégé-OWL for a year.
• Is open source (like Protégé-OWL itself).
• Initially had no inference capabilities.
• We then integrated the Jess rule engine with Protégé-OWL to perform inference with SWRL rules.
High-level Steps to Integrate Rule Engine with Protégé-OWL

- Use SWRL API to get all rules in knowledge base.
- Use OWL API to get all relevant OWL knowledge.
- Map OWL knowledge to rule engine knowledge.
- Perform inference!
- Map created rule engine knowledge to OWL.
- Use OWL API to put new information into OWL knowledge base.
- Also: GUI real estate is usually required.
- Other issues: integrity checking.
Before mapping, extracting relevant OWL knowledge for inference is an important optimization.
Not all knowledge needs to be extracted.
Required knowledge can be determined from each rule.
For example, the rule: Man(Fred) ∧ Man(?y) ∧ hasParent(Fred, ?y) ∧ hasBrother(?y,?z) → hasUncle(Fred, ?z) requires:
   - The individual named Fred
   - All individuals of class Man and subclasses
   - Fred’s hasParent properties and subproperties.
   - All individuals with the hasBrother property and subproperties.
Protégé-OWL Provides a SWRL Bridge API

- Given an OWL knowledge base it will extract SWRL rules and relevant OWL knowledge.
- Also provides an API to assert inferred knowledge.
- Knowledge (and rules) are described in non Protégé-OWL API-specific way.
- These can then be mapped to a rule-engine specific rule and knowledge format.
- This mapping is developers’s responsibility.
We used the SWRL Bridge to Integrate Jess Rule Engine with Protégé-OWL

- Jess is a Java-based rule engine.
- Jess system consists of a rule base, fact base, and an execution engine.
- Available free to academic users, for a small fee to non-academic users
- Has been used in Protégé-based tools, e.g., SWRLJessTab, SweetJess, JessTab.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule-1</td>
<td>AClass(?x) → BClass(?x)</td>
<td></td>
</tr>
<tr>
<td>Rule-2</td>
<td>BClass(?x) ∧ hasObjectProperty2(?x, ?e) → hasObjectProperty2(?x, ?f)</td>
<td></td>
</tr>
<tr>
<td>Rule-3</td>
<td>AClass(?e) → hasDatatypeProperty1(?e, &quot;This class has a hasDatatypeProperty2 of 8.%&quot;)</td>
<td></td>
</tr>
<tr>
<td>Rule-4</td>
<td>AClass(?a) → hasDatatypeProperty2(?a, 8.8) → hasDatatypeProperty1(?a, &quot;This class has a hasDatatypeProperty2 of 8.8.&quot;)</td>
<td></td>
</tr>
<tr>
<td>Rule-5</td>
<td>AClass(?a) → CClass(?a)</td>
<td></td>
</tr>
<tr>
<td>Rule-6</td>
<td>AClass(?a) → hasObjectProperty1(?a, ?e)</td>
<td></td>
</tr>
<tr>
<td>Rule-7</td>
<td>AClass(?a) → hasDatatypeProperty1(?e, &quot;This class has a hasDatatypeProperty1&quot;)</td>
<td></td>
</tr>
<tr>
<td>Rule-8</td>
<td>AClass(?a) → hasDatatypeProperty2(?a, 8.8) → hasDatatypeProperty1(?a, &quot;This class has a hasDatatypeProperty2 of 8.8.&quot;)</td>
<td></td>
</tr>
<tr>
<td>Rule-9</td>
<td>CClass(?a) → hasDatatypeProperty1(?a, &quot;This class has a hasDatatypeProperty1&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

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.
Defining Built-Ins using the SWRL Bridge

- SWRL provides mechanisms to add user-defined predicates, e.g.,
  - hasDOB(?x, ?y) ^ temporal:before(?y, 1997)...
  - hasDOB(?x, ?y) ^ temporal:equals(?y, 2000)...
- These built-ins could be implemented by each rule engine
- However, the SWRL Bridge provides a dynamic loading mechanism for Java-defined built-ins
- Can be used by any rule engine implementation
- Documented here:
Outstanding Issues

- SWRL Bridge does not know about all OWL constraints.
  - Contradictions with rules possible!
  - Consistency must be assured by the user.
  - Hard problem to solve in general.

- Integrated reasoner and rule engine would be ideal.
SWRL Resources

- **SWRL Language**

- **SWRL Tab**

- **SWRL API**

- **SWRL Bridge**