Knowledge Representation and Reasoning

University "Politehnica" of Bucharest
Department of Computer Science

Fall 2012

Adina Magda Florea

Lecture 10

SWRL - Semantic Web Rule Language

SWRL

- SWRL = Semantic Web Rule Language
- Proposal of W3C in 2004
- http://www.w3.org/Submission/SWRL/
- SWRL is built on the same description logic foundation as OWL
  Combines OWL DL and OWL Lite with RuleML
- Rules are expressed referring to OWL concepts (classes, properties, individuals)
- SWRL – abstract syntax for "Horn-like rules"
- The rules are saved as part of the ontology
- Implemented in: Bossam, R2ML, Hoolet, Pellet, KAON2, RacerPro, SWRLTab

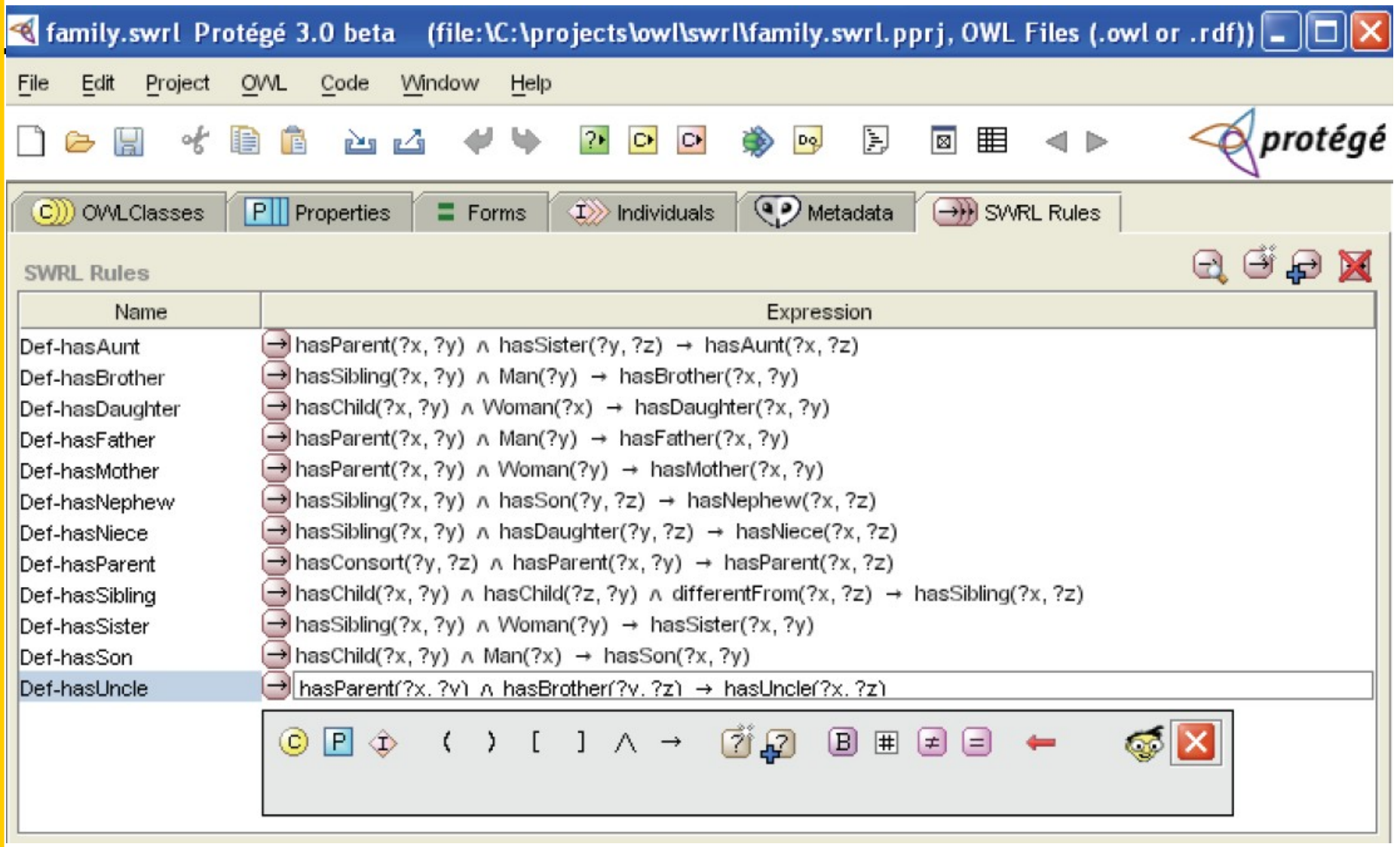
SWRL

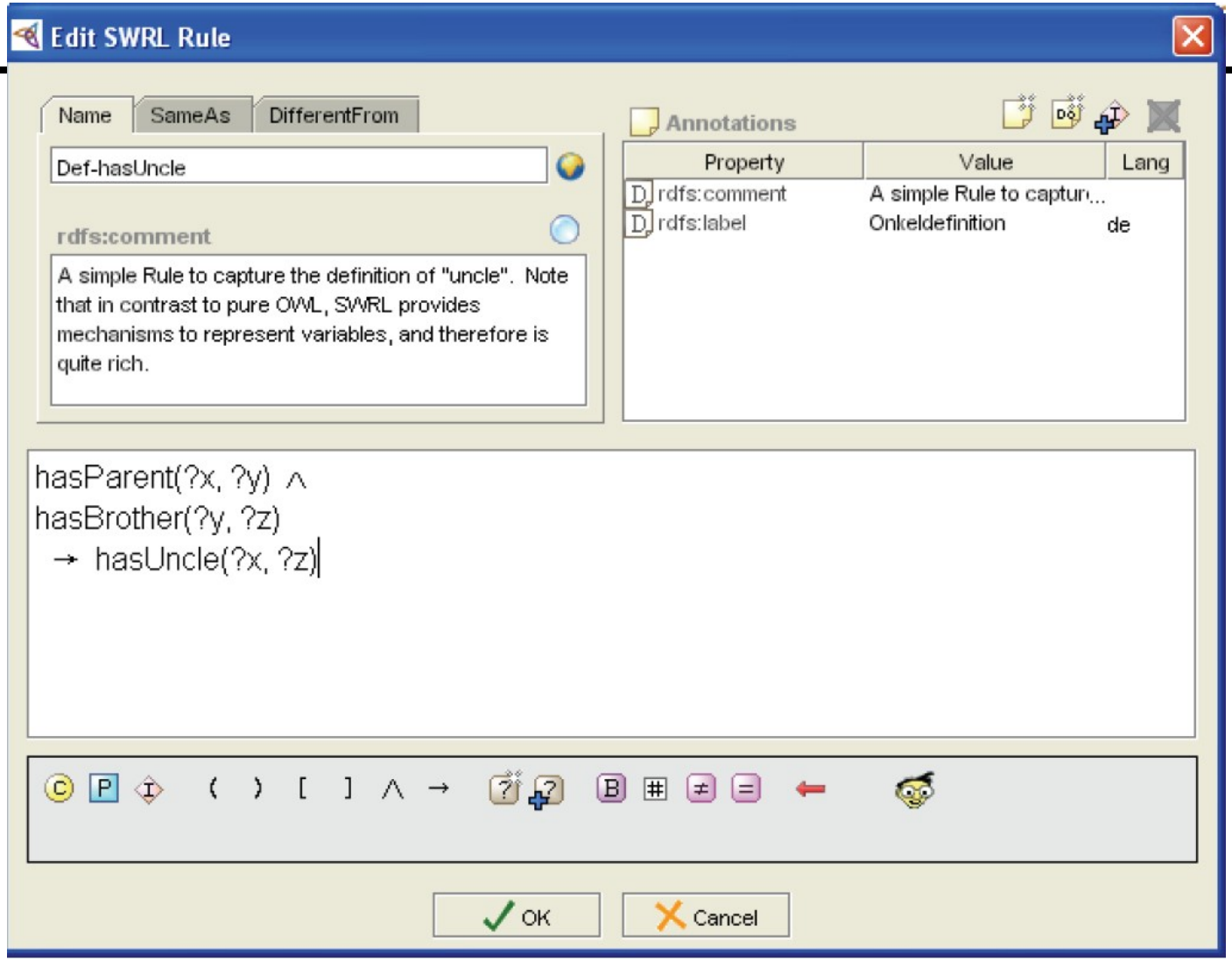
hasParent(?x, ?y) ^ hasBrother(?y, ?z) -> hasUncle(?x, ?z)

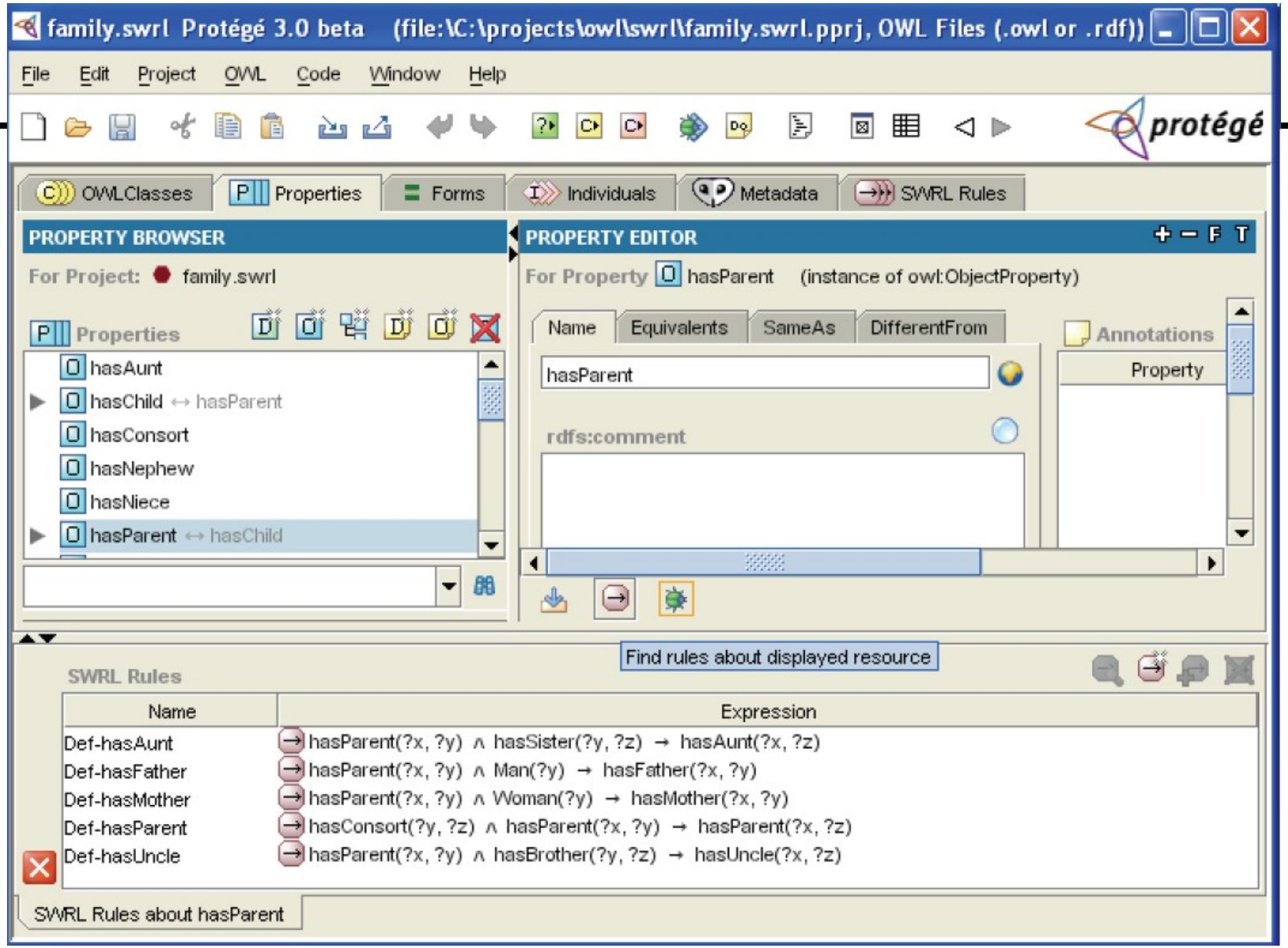
```
<ruleml:imp>
 <ruleml:_rlab ruleml:href="#example1"/>
 <ruleml:_body>
   <swrlx:individualPropertyAtom  swrlx:property="hasParent">
     <ruleml:var>x</ruleml:var>
     <ruleml:var>y</ruleml:var>
   </swrlx:individualPropertyAtom>
   <swrlx:individualPropertyAtom  swrlx:property="hasBrother">
     <ruleml:var>y</ruleml:var>
     <ruleml:var>z</ruleml:var>
   </swrlx:individualPropertyAtom>
 </ruleml:_body>
 <ruleml:_head>
   <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
     <ruleml:var>x</ruleml:var>
     <ruleml:var>z</ruleml:var>
   </swrlx:individualPropertyAtom>
 </ruleml:_head>
</ruleml:imp>
```

SWRL Editor

- SWRL Editor – extension of Protegé OWL which allows SWRL rule editing;
- Included in Protégé OWL
- Accessible as a tab
- Offers a Java API for interoperability wuth inference engine


family.swrl Protégé 3.0 beta (file:\C:\projects\owl\swrl\family.swrl.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Code Window Help
protégé
OWLClasses Properties Forms Individuals Metadata SWRL Rules
SWRL Rules
Name
Expression
Def-hasAunt hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
Def-hasBrother hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)
Def-hasDaughter hasChild(?x, ?y) ∧ Woman(?x) → hasDaughter(?x, ?y)
Def-hasFather hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
Def-hasMother hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
Def-hasNephew hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)
Def-hasNiece hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)
Def-hasParent hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
Def-hasSibling hasChild(?x, ?y) ∧ hasChild(?z, ?y) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)
Def-hasSister hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)
Def-hasSon hasChild(?x, ?y) ∧ Man(?x) → hasSon(?x, ?y)
Def-hasUncle hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)


Edit SWRL Rule
Name
SameAs
DifferentFrom
Def-hasUncle
rdfs:comment
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.
Annotations
Property
Value
Lang
rdfs:comment
A simple Rule to captur...
rdfs:label
Onkeldefinition
de
hasParent(?x, ?y) ∧
hasBrother(?y, ?z)
→ hasUncle(?x, ?z)
OK
Cancel


family.swrl Protégé 3.0 beta (file:\C:\projects\owl\swrl\family.swrl.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Code Window Help
protégé
OWLClasses Properties Forms Individuals Metadata SWRL Rules
PROPERTY BROWSER
For Project: family.swrl
Properties
hasAunt
hasChild ↔ hasParent
hasConsort
hasNephew
hasNiece
hasParent ↔ hasChild
PROPERTY EDITOR
For Property hasParent (instance of owl:ObjectProperty)
Name Equivalents SameAs DifferentFrom
Annotations
Property
hasParent
rdfs:comment
Find rules about displayed resource
SWRL Rules
Name
Expression
Def-hasAunt hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
Def-hasFather hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
Def-hasMother hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
Def-hasParent hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
Def-hasUncle hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)
SWRL Rules about hasParent

What does a SWRL Rule look like?

- A **SWRL rule** contains an antecedent part(*body)*, and a consequent (*head)*.
- Both the body and head consist of positive conjunctions of *atoms*:

Atom ^ Atom … **->** ***Atom ^ Atom ….***

ANTECEDENT CONSEQUENT

- An **atom** is an expression of the form:

p(arg1, arg2,…, argn)

- ***P*** is a predicate symbol (OWL classes, properties or data types)
- ***arg1*, *arg2*, ..., *argn*** - arguments of the expression.(OWL individuals or data values, or variables referring to them)

How many Atom types are provided by SWRL?

- SWRL provides seven types of atoms:
  - Class Atoms *owl:Class*
  - Individual Property atoms *owl:ObjectProperty*
  - Data Valued Property atoms *owl:DatatypeProperty*
  - Different Individuals atoms
  - Same Individual atoms
  - Built-in atoms
  - Data Range atoms

What is a Class atom?

- A class atom consists of an **OWL named class** or **class expression** and a single argument representing an OWL individual.

- **Person(?p)**
- **Man(Fred)**

  - Person and Man - OWL named classes
  - ?p - variable representing an OWL individual
  - Fred - name of an OWL individual.

- All individual of type Man are also or type Person:
  - **Man(?p) -> Person(?p)**
    - Of course, this statement can also be made directly in OWL.

What is an Individual Property atom?

- Consists of an **OWL object property** and two arguments representing OWL individuals.

- **hasBrother(?x, ?y)**
- **hasSibling(Fred, ?y)**
  - hasBrother, hasSibling - OWL object properties
  - ?x and ?y - variables representing OWL individuals
  - Fred -name of an OWL individual.

- person with a male sibling has a brother
  Person(?p) ^ hasSibling(?p,?s) ^ Man(?s) -> hasBrother(?p,?s)

- Person and male can be mapped to OWL class called Person with a subclass Man
- The sibling and brother relationships can be expressed using OWL object properties hasSibling and hasBrother with a domain and range of Person.

What is a Data Valued Property atom?

- A data valued property atom consists of an **OWL data property** and two arguments (OWL individual , data value)
- **hasAge(?x, ?age)**
- **hasHeight(Fred, ?h)**
- **hasAge(?x, 232)**

- All persons that own a car should be classified as drivers

  Person(?p) ^ hasCar(?p, true) -> Driver(?p)
  - This rule classifies all car-owner individuals of type Person to also be members of the class Driver.

- Named individuals can be referred directly

  Person(Fred) ^ hasCar(Fred, true) -> Driver(Fred)
  - This rule works with a known individual called Fred in an ontology, and new individual can not be created using this rule.

What is a Different and a Same Individuals atom?

- SWRL supports sameAs and differentFrom atoms to determine if individuals refer to the same underlying individual or are distinct, and can use **owl:sameAs** , **owl:allDifferents**
- **No unique name assumption**

- A different individuals atom consists of the *differentFrom* symbol and two arguments representing OWL individuals.

  differentFrom(?x, ?y)

  differentFrom(Fred, Joe)

  sameAs(?x, ?y)

  sameAs(Fred, Freddy)

- If two OWL individuals of type Author cooperate on the same publication that they are collaborators

  Publication(?a) ^ hasAuthor(?x, ?y) ^ hasAuthor(?x, ?z) ^ differentFrom(?y, ?z) -> cooperatedWith(?y, ?z)

What are Data Range atoms?

- A data range atom consists of a **datatype name** or a **set of literals** and a single argument representing a **data value**.

- **xsd:int(?x)**
- **[3, 4, 5](?x)**
- Here, ?x is a variable representing a data value.

What is a Built-In Atom?

- A built-in is a predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate.
- Core SWRL built-ins are preceded by the namespace qualifier **swrlb**.
- SWRL supports user-defined built-ins

- Person with an age of greater than 17 is an adult
- **Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) -> Adult(?p)**

- Person's telephone number starts with the international access code "+"
  Person(?p)^hasNumber(?p, ?number) ^ swrlb:startsWith(?number, "+") -> hasInternationalNumber(?p,true)

- Built-ins can take any number or combination of OWL datatype property values.
- They can not take object, class or property values (some extensions could override this)
- Argument number and types checking is the responsibility of the built-in implementor.
- If an incorrect number or type of arguments is passed, built-in should evaluate to false.
- SWRL allows new libraries of built-ins to be defined and used in rules

Can Built-Ins assign their Arguments?

- Built-ins can also assign (or bind) values to arguments.

- **swrlb:add(?x, 2, 3)** - uses the core SWRL built-in method to add two literals.
- If x is unbound when this built-in is invoked, it will be assigned the the value 5
- If x is already bound when the built-in is invoked, it will simply determine if its value is 5.

- A built-in method that successfully assigns a value to an argument should return true.
- If more than one unbound argument is present, all arguments must be bound.
- If the built-in returns false no assignments are expected.

- Calculate the area of a rectangle

  Rectangle(?r) ^ hasWidthInMeters(?r, ?w) ^ hasHeightInMeters(?r, ?h) ^ swrlb:multiply(?areaInSquareMeters, ?w, ?h) -> hasAreaInSquareMeters(?r, ?areaInSquareMeters)

Can Built-Ins assign their Arguments?

- The use of a variable in any non built-in atom automatically ensures it is bound .

- Classify a rectangle with an area of over 100 square meters as a BigRectangle:
- **Rectangle(?r) ^ hasWidthInMetres(?r, ?w) ^ hasHeightInMetres(?r, ?h) ^ swrlb:multiply(?areaInSquareMeters, ?w, ?h) ^ swrlb:greaterThan(?100, ?areaInSquareMeters) -> hasAreaInSquareMetres(?r, ?areaInSquareMeters) ^ BigRectangle(?r)**

- Binding precedence is from left to right.
- Not built-ins will perform argument binding.
  - **greaterThan**
- The designer of a built-in must decide which - if any - arguments are to be bound.

Can OWL Class Expressions be used in SWRL Rules?

- SWRL also supports the use of OWL class descriptions in rules.
- If individual is a member of a class with a hasChild property, with a minimum cardinality of one, classify him as a parent

  (hasChild >= 1)(?x) -> Parent(?x)
  - matches all individuals for which *it can be proven* that they are members of a class that has the specified cardinality restriction on a hasChild property
  - It does *not* match all individuals in an OWL ontology that have one or more values for a hasChild property.
  - this rule may actually match individuals that have no values for the hasChild property in the current ontology but for which the existence of such values can be deduced from OWL axioms
- Assert conclusions about individuals:

  Parent(?x) ->(hasChild >= 1)(?x)

  Publication(?p) ^ (hasAuthor = 1)(?p) -> SingleAuthorPublication(?p)

Does SWRL support the use of annotation values to refer to OWL entities?

- OWL entities (classes, properties, and individuals) are referred to directly in rules using their underlying OWL name (rdf:ID or rdf:About).
- We could use values of properties (e.g. rdfs:label) instead, which contain a user-friendly name.
- These annotation values can be used in rules by enclosing them in single quotes.

- **Driver(?d) ^ hasAge(?d, ?age) ^ swrlb:greaterThan(?age, 25) -> isInsurable(?d, true)**

  The class Driver has a selected annotation property value of 'a Driver'. The

  'a Driver'(?d) ^ hasAge(?d, ?age) ...

- The approach is the same when referring to both properties and individuals using annotation values.

- Annotation values, used in rules, should be made unique.

Overview

- SWRL adopt the Open World Assumption from OWL
- SWRL does not support Nonmonotonic Inference
  - does *not* change the existing value for OWL entity
- SWRL does not support negated atoms or negation as failure
- Classical negation is possible in SWRL with the use of **owl:complementOf** class descriptions in rules, or by concluding disjoinment by rules

  (not Person)(?x) -> NonHuman(?x)
- SWRL does not support disjunctions of Atoms

  A(?x) ∨ B(?y) -> C(?x)

  C(?x) -> A(?x) ∨ B(?x)

  (A ∪ B)(?x) -> C(?x) Correct – owl:unionOf

Overview

- SWRL does not support OWL Full directly
- SWRL does not support RDF or RDFS
- Stay within OWL if possible and only use SWRL when its additional expressive power is required

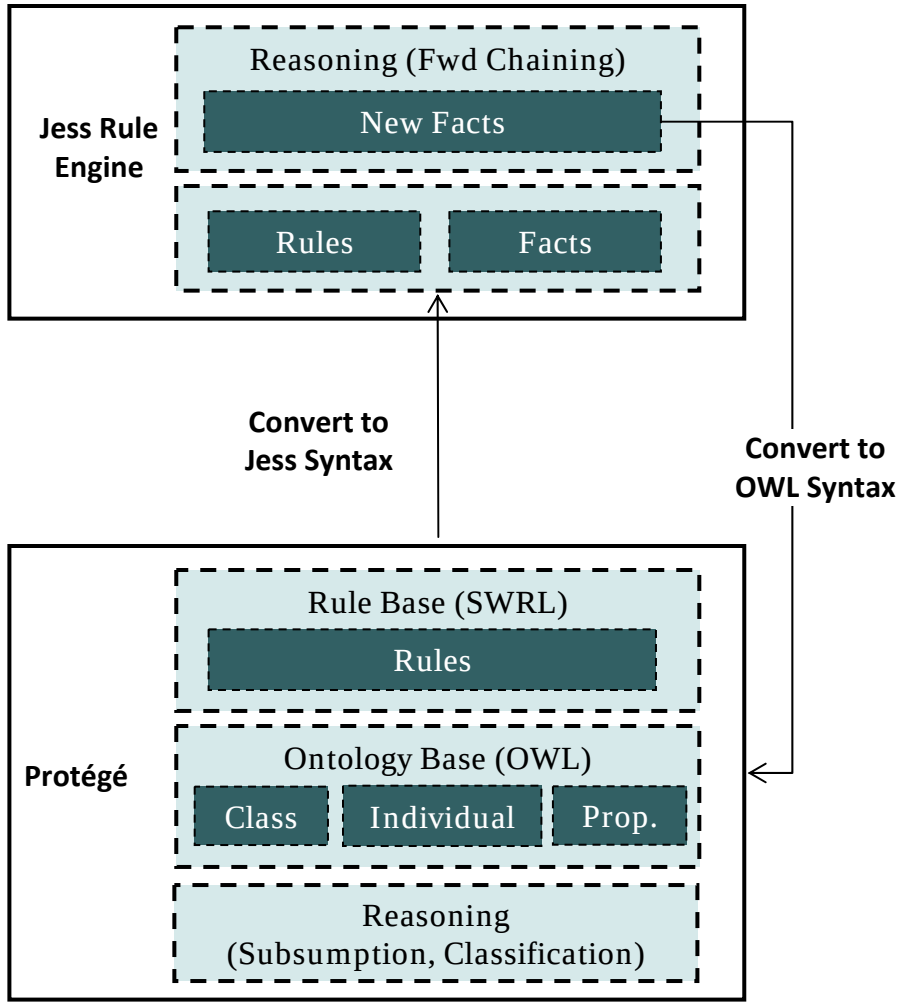
OWL/SWRL integration

- ***Human Readable Syntax***

hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒
hasUncle(?x1,?x3)

- ***XML Concrete Syntax***

```
<ruleml:imp>
<ruleml:_rlab ruleml:href="#example1"/>
<ruleml:_body>
<swrlx:individualPropertyAtom
    swrlx:property="hasParent">
    <ruleml:var>x1</ruleml:var>
    <ruleml:var>x2</ruleml:var>
</swrlx:individualPropertyAtom>
<swrlx:individualPropertyAtom
    swrlx:property="hasBrother">
    <ruleml:var>x2</ruleml:var>
    <ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom> </ruleml:_body>
<ruleml:_head> <swrlx:individualPropertyAtom
    swrlx:property="hasUncle">
    <ruleml:var>x1</ruleml:var>
    <ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom> </ruleml:_head>
</ruleml:imp>
```



Bibliography

- Slides based on Marko Stupar slides

- http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ#nid6ZK, **SWRLLanguageFAQ,** June 2, 2011
- Protégé – OWL – SWRL Tutorial, **CELab, IeSys, KAIST, Gyeongjune Hahm**
- http://en.wikipedia.org/wiki/Semantic_Web_Rule_Language