

# Semantic Web Rule Language (SWRL)

Vahid Karimi

June 2008

# Outline

- 1 Introduction
  - OWL-DL
  - RuleML
- 2 SWRL Syntax
- 3 SWRL Semantics
- 4 Rules and Safety
- 5 Implementation and Applications
- 6 Prior and Related Works

# SWRL

## Semantic Web Rule Language (SWRL) [HPSB<sup>+</sup>04]:

- A proposal to combine ontologies and rules:
  - Ontologies: OWL-DL
  - Rules: RuleML

SWRL = OWL-DL + RuleML

- OWL-DL: variable free
  - corresponding to *SHOIN*(D)
- RuleML: variables are used.

# Why Do We Need a Rule Language?

A rule language is needed for several reasons [PSG<sup>+</sup>05]:

- The existing rule sets can be reused.
- Expressivity can be added to OWL
  - Although expressivity always comes with a price, i.e.,
  - Decidability
- It is easier to read and write rules with a rule language.
  - Rules are called syntactic sugar;
  - True in some cases but not in all situations

# $\mathcal{SHOIN}$

$\mathcal{SHOIN} = \mathcal{S} + \mathcal{H} + \mathcal{O} + \mathcal{I} + \mathcal{N} = \mathcal{ALCR}_+ \mathcal{HOIN}$  [Str07]

$\mathcal{S}$  :  $\mathcal{ALC}$  with transitive roles  $\mathcal{R}_+$

$\mathcal{AL} : C, D \rightarrow \top \mid \perp \mid A \mid C \sqcap D \mid \neg A \mid \exists R. \top \mid \forall R. C$

$\mathcal{C}$  : Concept negation,  $\neg C$ . Thus,  $\mathcal{ALC} = \mathcal{AL} + \mathcal{C}$

$\mathcal{R}_+$  : transitive role, e.g.,  $\text{trans}(\text{isPartOf})$

$\mathcal{H}$  : Role inclusion axioms,  $R1 \sqsubseteq R2$ , e.g.,  $\text{isComponentOf} \sqsubseteq \text{isPartof}$

$\mathcal{O}$  : Nominals (singleton class),  $\{a\}$ , e.g.,  $\exists \text{hasChild}.\{mary\}$

$\mathcal{I}$  : Inverse role,  $R^-$ , e.g.,  $\text{isPartOf} = \text{hasPart}^-$

$\mathcal{N}$  : Number restrictions,  $(\geq n R)$  and  $(\leq n R)$ , e.g.,  
 $(\geq 3 \text{ hasChild})$  (has at least 3 children)

$\mathcal{SHOIN}$  with concrete domain:  $\mathcal{SHOIN}(\mathcal{D})$  corresponding to OWL-DL

# Rules and RuleML

RuleML, the datalog sublanguage of Horn clause [BGT05]:

- ❶ Datalog, syntactically, a subset of Prolog:
  - Function-free:  $P(f(2),5)$  not allowed
- ❷ Horn clause (a disjunction of literals with at most one positive), e.g.,
  - $\neg p \vee \neg q \vee \dots \vee \neg t \vee u$  can be written as,
  - $p \wedge q \wedge \dots \wedge t \rightarrow u$   
only conjunctions (not disjunctions) of atoms

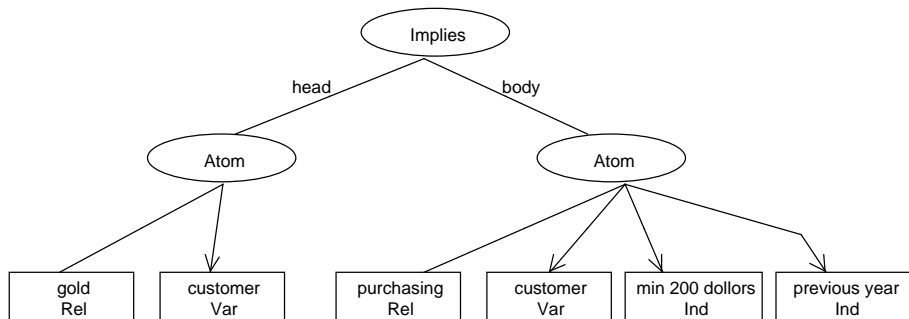
# An Example of a RuleML's Rule

A relation can be n-ary ( $n = 0, 1, 2, \dots$ ) in RuleML.

**Example:** A customer is gold if her purchasing has been minimum 200 dollars in the previous year.

**head (a unary relationship):** A customer is gold.

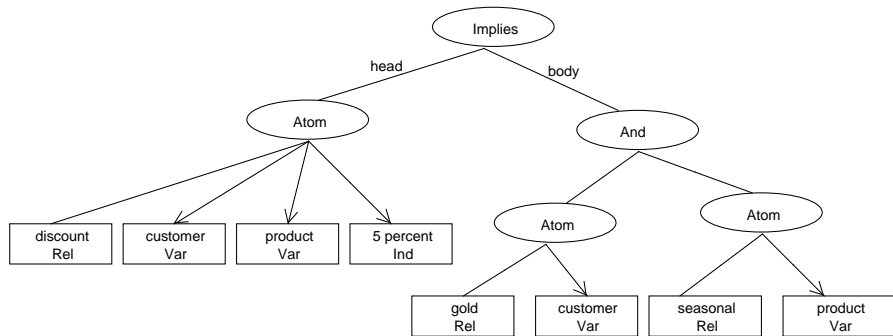
**body (a 3-ary relationship):** Her purchasing ...



# Rules with Multiple Atoms in Antecedent

Rules with and-ing multiple atoms in the body are common, e.g.:

- The discount for a customer on a product is 5% if the customer is gold and the product is seasonal.





# Rules of RuleML in SWRL

- A restricted version of RuleML
- i.e., only unary/binary relations
  - The previous slide's RuleML example is not a valid SWRL's rule.
  - n-ary relations are divided into binary relations.

Rules with and-ing multiple atoms in the body are common, e.g.:

- The discount for a customer on a product is 5% if the customer is gold and the product is seasonal.

Rules with and-ing multiple atoms in the consequent:

- Lloyd-Topor transformation:  
translate the rule into multiple rules

# SWRL Syntax

SWRL atoms are defined as follows:

$Atom \leftarrow C(i) \mid D(v) \mid R(i, j) \mid U(i, v) \mid builtIn(p, v_1, \dots, v_n) \mid i = j \mid i \neq j$

$C$  = Class

$D$  = Data type

$R$  = Object Property

$U$  = Data type Property

$i, j$  = Object variable names or  
Object individual names

$v_1, \dots, v_n$  = Data type variable names or  
Data type value names

$p$  = Built-in names

## SWRL Syntax (cont'd)

The SWRL rule syntax follows:

$a \leftarrow b_1, \dots, b_n$  where,

$a$  : head (an atom)     $b_s$ : body (all atoms)

A SWRL knowledge base ( $k$ ) is defined as follows:

$k = (\Sigma, P)$  where,

$\Sigma$  = Knowledge base of  $\mathcal{SHOIN}(D)$

$P$  = A finite set of rules

# SWRL Semantics

let  $I = (\Delta^I, \Delta^D, .^I, .^D)$  where,

$I$  = interpretation

$\Delta^I$  = Object Interpretation domain

$\Delta^D$  = Datatype Interpretation domain

$.^I$  = Object Interpretation function

$.^D$  = Datatype Interpretation function

$$\Delta^I \cap \Delta^D = \emptyset$$

such that  $V_{IX} \rightarrow P(\Delta^I)$      $V_{DX} \rightarrow P(\Delta^D)$  where,

$V_{IX}$  = object variables     $V_{DX}$  = datatype variables

$P$  = the powerset operator

# SWRL Semantics (cont'd)

The following table shows Binding  $B(I)$  for the SWRL atoms:

SWRL Atoms	Condition on Interpretation
$C(i)$	$i^I \in C^I$
$R(i, j)$	$(i^I, j^I) \in R^I$
$U(i, v)$	$(i^I, v^D) \in U^I$
$D(v)$	$v^D \in D^D$
$builtIn(p, v_1, \dots, v_n)$	$(v_1^D, \dots, v_n^D \in p^D)$
$i = j$	$i^I = j^I$
$i \neq j$	$i^I \neq j^I$

## SWRL Semantics (cont'd)

SWRL atoms in the antecedent are satisfied,

- if it is empty (trivially true)
- or every atom of it is satisfied

SWRL atom in the consequent is satisfied,

- if it is not empty
- and it is satisfied

A rule is satisfied by an interpretation of  $I$  iff

- every binding  $B(I)$  that satisfies the antecedent
- $B(I)$  satisfies the consequent

## A SWRL Example

In a table format, the SWRL terms follow:

$C(i)$	$R(i, j)$	$D(v)$	$U(i, v)$	$builtIn(p, v_1, \dots, v_n)$	$i = j$	$i \neq j$
--------	-----------	--------	-----------	-------------------------------	---------	------------

Variable are indicated by prefixing them with question marks in rules.

**Example** of a rule asserting a fast computer:

*FastComputer(?c) ← Computer(?c) ∧ hasCPU(?c, ?cpu) ∧  
hasSpeed(?cpu, ?sp) ∧ HighSpeed(?sp)*

*FastComputer(?c)*: C(i) term of table above

*hasCPU(?c, ?cpu)*: R(i, j) term of table above

# Expressing Rules without Using SWRL

It is possible to express some rules using only DL:

- i.e., rules are syntactic sugar in these cases

The previous SWRL's Rule:

*FastComputer(?c) ← Computer(?c) ∧ hasCPU(?c, ?cpu) ∧  
hasSpeed(?cpu, ?sp) ∧ HighSpeed(?sp)*

The above rule using only DL can be expressed as follows:

*Computer ⊑ ∃hasCPU.∃hasSpeed.HighSpeed ⊑ FastComputer*

The translating of rules from SWRL to DL,

- depends on the number of variables based on shared variables between the consequent and antecedent



# Translating Rules from SWRL to DL

The number of variables shared between consequent and antecedent:

- translating is possible, if 0 variable is shared, but at least one individual is shared
- translating is possible, if 1 variable is shared
- translating not possible, if 2 or more variables are shared

# Translating Process from SWRL to DL

## The Translation Procedure:

- 1 The consequent and antecedent become two conjunctive queries.
- 2 The resulting queries are translated into class expressions.
  - using the rolling-up technique
- 3 The antecedent becomes the subclass of the consequent.

# Translating Rules from SWRL to DL (Example)

*FastComputer(?c) ← Computer(?c) ∧ hasCPU(?c, ?cpu) ∧  
hasSpeed(?cpu, ?sp) ∧ HighSpeed(?sp)*

1. The consequent and antecedent become two conjunctive queries:

- 1a) *?c : FastComputer*
- 1b) *?c : Computer ∧ (?c, ?cpu) : hasCPU ∧  
(?cpu, ?sp) : hasSpeed ∧ ?sp : HighSpeed*
- Conjunctive terms a directed graph:
  - each node is a variable or a named individual
  - each edge is a relation
  - a query graph, e.g.,
  - *?c : Computer  $\xrightarrow{\text{hasCPU}}$  ?cpu  $\xrightarrow{\text{hasSpeed}}$  ?sp : HighSpeed*

## Example (cont'd)

2. The rolling-up technique applies to query graphs, e.g.,

$$?c : \textit{Computer} \xrightarrow{\textit{hasCPU}} ?cpu \xrightarrow{\textit{hasSpeed}} ?sp : \textit{HighSpeed}$$

- each outgoing edge is presented as an existential quantifier
- edges are presented as restrictions
- each outgoing edge  $(?x, ?y) : R$  transfers to
- expression of  $\exists R.Y$ 
  - $Y$  is the named class restriction on  $?y$

$\exists \textit{hasCPU} . \exists \textit{hasSpeed} . \textit{HighSpeed}$

## Example (cont'd)

The named class of target variable  $?c$  : is *Computer*

- Intersection with the named class *Computer*  $\sqcap$

Rolling-up result: *Computer*  $\sqcap \exists hasCPU. \exists hasSpeed. HighSpeed$

The named class of target  $?c$  variable for the consequent,

- $?c : FastComputer$  transforms to *FastComputer*

3. Make the antecedent the subclass of the consequent, i.e.,

*Computer*  $\sqcap \exists hasCPU. \exists hasSpeed. HighSpeed \sqsubseteq FastComputer$

## A Rule in SWRL but not in DL

Some non-trivial rules can be translated from SWRL into DL, but

- SWRL can express some rules that DL cannot

Example:

$hasUncle(?nephew, ?uncle) \leftarrow$   
 $hasParent(?nephew, ?parent) \wedge hasBrother(?parent, ?uncle)$

The above rule cannot be translated into DL:

- two different variables in the consequent
- generating a subsumption for each variable not enough
- the above rule, not syntactic sugar

## A Rule in SWRL but not in DL (cont'd)

Although cannot infer *hasUncle*(*Bob*, *Bill*) from

- *hasParent*(*Bob*, *Mary*) and *hasBrother*(*Mary*, *Bill*),
- *hasUncle* relation can be used explicitly or implicitly

**Example:** People whose uncles are all lawyers,

with explicit use of *hasUncle*:  $\forall hasUncle.Lawyer$

with implicit use of *hasUncle*:  $\forall hasParent.\forall hasBrother.Lawyer$

can even express uncle without explicit use of it:

$$\exists hasBrother^-. \exists hasParent^-. \top$$

# DL-Safe SWRL Rules

The safety condition for SWRL rules means,

- 1 to add additional expressive power and
- 2 to maintain decidability at the same time

A combined knowledge base:

Knowledge base of  $\mathcal{SHOIN}(D) = \Sigma$

A finite set of rules =  $P$

combined knowledge base =  $(\Sigma, P)$

Both OWL-DL and datalog are decidable:

- The goal is for their combinations to be decidable too.



## DL-Safe SWRL Rules (cont'd)

### A Datalog Safe Rule Definition:

- A rule is safe if every variable in the consequent also appears in the antecedent.

### DL-safety Restrictions:

- more restriction than Datalog safety
- conditions on the combination of DL and Datalog
- restrictions on variable uses in DL-atoms of Datalog rules

$$DL - Atom \leftarrow C(i) \mid D(v) \mid R(i, j) \mid U(i, v)$$

## DL-Safe SWRL Rules (cont'd)

**Strong DL-Safety (Definition):** Let  $\Sigma$  be an OWL-DL ontology and  $P$  a Datalog program. A rule  $r$  in  $P$  is **strongly DL-safe** if each variable in  $r$  occurs in a non-DL atom in the rule body. The program  $P$  is strongly DL-safe if all its rules are strongly DL-safe.

**Weak DL-Safety (Definition):** Let  $\Sigma$  be an OWL-DL ontology and  $P$  a Datalog program. A rule  $r$  in  $P$  is **weakly DL-safe** if each variable in  $r$  occurs in a non-DL atom in the rule. The program  $P$  is weakly DL-safe if all its rules are weakly DL-safe.

# Role Safety

**Definition of Role Safety:** Let  $\Sigma$  be an OWL-DL ontology and  $P$  a datalog program. A rule  $r$  in  $P$  is **role safe** if, for each DL-atom  $p$  with arity 2 in the antecedent of  $r$  **at least one variable** in  $p$  also appears in a non-DL atom  $q$  in the antecedent of  $r$  and  $q$  never appears in the consequent of any rule in the program  $P$ .

Example: *discountAvailable(?cust, ?printer) ←*

*previouslyBought(?cust, ?comp) ∧ sameBrand(?comp, ?printer) ∧  
hasPrice(?comp, ?price) ∧ Customer(?cust) ∧ Printer(?printer) ∧  
Computer(?comp) ∧ HighPrice(?price)*

# Adding Built-ins

Built-ins are special symbols, e.g.,

- $>, \geq, \leq, <$
- for concrete domain (e.g., integers, strings)
- to be used in a rule

**Datalog Example:**  $SmallMonitor(?monitor) \leftarrow$   
 $Monitor(?monitor) \wedge hasScreenSize(?monitor, ?size) \wedge ?size \leq 15$

**DL Example:**  $Monitor \sqcap \exists hasScreenSize. \leq 15 \sqsubseteq SmallMonitor$

# Reasoner Supports for SWRL

Examples of some existing reasoners supporting SWRL:

- 1 SWRLTab
  - SWRL rules in Protege-OWL (open source)
- 2 Pellet
  - An open source OWL DL reasoner in Java
- 3 RacerPro
  - A commercial product

# Rules and Policies

One application of rules is for expressing policies:

- far beyond some provided examples, e.g., hasUncle
- various areas of policies: access control, ...

Example: Allowing certain services, related to certain projects, to the individuals who are members of an organization participating in a project.

*hasPermission(?person, ?service)  $\leftarrow$  relatedTo(?service, ?project)  $\wedge$  jointProject(?project)  $\wedge$  participates(?org, ?project)  $\wedge$  memberOf(?person, ?org)*

The above rule cannot be expressed using only DL.

# Prior Work on Integrating DL and Datalog

Two main research papers using DL and datalog:

1. **AL-Log** [Donini, Lenzerini, Nardi, and Schaerf, 1998]

- Description Logic:  $\mathcal{ALC}$
- Rules: Datalog rules

Integration of DL and datalog:

- DL as the structural component; datalog as the relational component
- DL concepts: constraints on the rule bodies of datalog rules

Example:  $\text{convenient}(\text{?cust}, \text{?serv}) \leftarrow \text{livesIn}(\text{?cust}, \text{?loc}) \wedge \text{fastDelivery}(\text{?serv}, \text{?loc}) \wedge \text{Customer}(\text{?cust}) \wedge \text{SalesService}(\text{?serv})$

DL concepts: Customer and SalesService constraints on

- fastDelivery and livesIn defined in datalog.

# Prior Work on Integrating DL and Datalog (cont'd)

## 2. CARIN [Levy and Rousset, 1998]

- Description Logic:  $\mathcal{ALCN}\mathcal{R}$

Integration of DL and datalog:

- DL: the structural and relational components
- Datalog: the relational component
- DL concepts and roles: constraints on the rule bodies of datalog rules

Example:  $discountAvailable(?cust, ?printer) \leftarrow$

$previouslyBought(?cust, ?comp) \wedge sameBrand(?comp, ?printer) \wedge$   
 $hasPrice(?comp, ?price) \wedge Customer(?cust) \wedge Printer(?printer) \wedge$   
 $Computer(?comp) \wedge HighPrice(?price)$



# Summary of Decidable Combinations

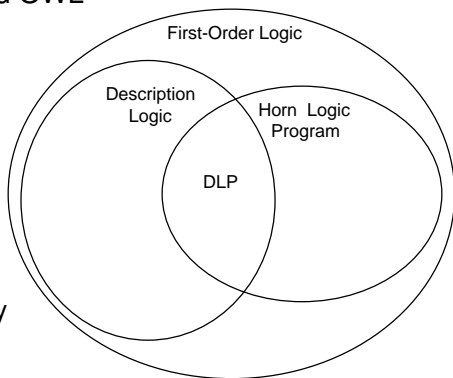
	AL-Log	CARIN	CARIN	General Case
Unary DL-Atoms in Antecedent	✓	✓	✓	Strongly DL-safe
Unary DL-Atoms in Consequent	x	x	x	Strongly DL-safe
Binary DL-Atoms in Antecedent	x	role-safe	✓	Strongly DL-safe
Binary DL-Atoms in Consequent	x	x	x	Strongly DL-safe
n-ary non-DL atoms	✓	✓	✓	✓
Most expressive OWL subset	<i>SHOIN</i> (D)	<i>SHI</i> (D)	DL-Lite <sup>-</sup>	<i>SHOIN</i> (D)

Table: Decidable Combinations

# DLP

## Description Logic Programs (DLP) [GHVD03]:

- A related suggestion for including rules to an ontology
- Intersection of Horn logic and OWL
- Rules in DL or datalog
- DLP part of DL, then
- why do we need it?
  - reuse criterion
  - pragmatic criterion
  - not applicable: expressivity



# Description Logic Programs (DLP) (cont 'd)

Comparison of SWRL and DLP, according to Parsia et al. [PSG<sup>+</sup>05]:

- SWRL: (roughly) the union of Horn logic and OWL
- DLP: Intersection of Horn logic and OWL
  - An inexpressive language

Description Logics for SWRL and DLP:

- For DLP:  $\mathcal{SHOIQ}(\mathcal{D})$ 
  - $\mathcal{Q}$ : Qualified Number restrictions, ( $\geq n$  R.C) and ( $\leq n$  R.C), e.g., ( $\geq 3$  hasChild.Adult) (has at least 3 adult children)
- For SWRL:  $\mathcal{SHOIN}(\mathcal{D})$ 
  - $\mathcal{N}$ : Number restrictions, ( $\geq n$  R) and ( $\leq n$  R), e.g., ( $\geq 3$  hasChild) (has at least 3 children)

## One Other Approach

The rule component of SWRL is restricted to datalog

- not very expressive

Another approach to overcome this drawback:

- Combining answer set programming with description logics

Answer Set Programming:

- declarative programming
- decidable

Description Logics:

- $\mathcal{SHOIN}(D)$  (OWL-DL) or
- $\mathcal{SHIF}(D)$  (OWL-lite)



# SWRL Extension

SWRL Extension towards First-Order Logic (SWRL-FOL):

- addition of function-free first-order formula over unary and binary predicates

RuleML is a family of languages.

- SWRL uses the kernel RuleML sub-language of datalog.
- FOL-RuleML uses another language of the family.
  - includes FOL operators,
    - or, not, implies, equivalent, forall, exists
  - FOL operators are used to describe rules

-  Harold Boley, Benjamin Grosf, and Said Tabet. *RuleML Tutorial*. Available at <http://www.ruleml.org/papers/tutorial-ruleml-20050513.html>, May 2005.
-  Benjamin Grosf, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *Proceedings of the Twelfth International World Wide Web Conference (WWW) ACM 2003*, Budapest, Hungary, May 2003.
-  Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Available at <http://www.w3.org/Submission/SWRL/>, May 2004.
-  Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, Edna Ruckhaus, and Daniel Hewlett. Cautiously approaching SWRL. Preprint submitted to Elsevier Science, 2005.
-  Umberto Straccia. From OWL to Description Logics. Available at <http://www.cs.uwaterloo.ca/~gweddell/cs848/OWLToDL.pdf>, 2007.