COMP3220:
Document Processing and Semantic Technologies
Rule Languages

Rolf Schwitter

Rolf.Schwitter@mq.edu.au

Today's Agenda

- What is a Rule?
- Rule Interchange Format (RIF)
- Answer Set Programming

© Macquarie University 2022

What is a Rule?

- Two different but related ways to understand rules:
  - declarative rules
  - production rules.
- A <u>declarative rule</u> makes a statement about the world but does not specify an action to be carried out:

  If condition, then consequence.
- A <u>production rule</u> is similar to an instruction in a computer program:

  If a certain condition holds, then some action is carried out.

Declarative Rules

- An example of a declarative rule is:

  If a person is human, then that person is mortal.

- Declarative rules are useful to specify domain knowledge.
- Declarative rules describe how the world is, rather than prescribing how things ought to be.
- Declarative rules can be processed by a forward or a backward chaining algorithm.
- Forward chaining is "data-driven".
- Backward chaining is "goal-driven".

© Macquarie University 2022

Production Rules

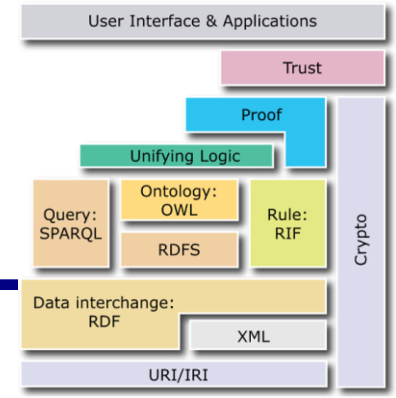
- An example of a production rule is:

  If a customer has flown more than 100,000 miles

  then upgrade him to Gold Member status.
- Production rules are often used for business applications.
- Production rules are generally executed by a forward chaining algorithm.
- The condition part of each rule is tested against the current state of the working memory.
- The rule interpreter usually uses a prioritising mechanism, if more than one rule can be triggered.

© Macquarie University 2022

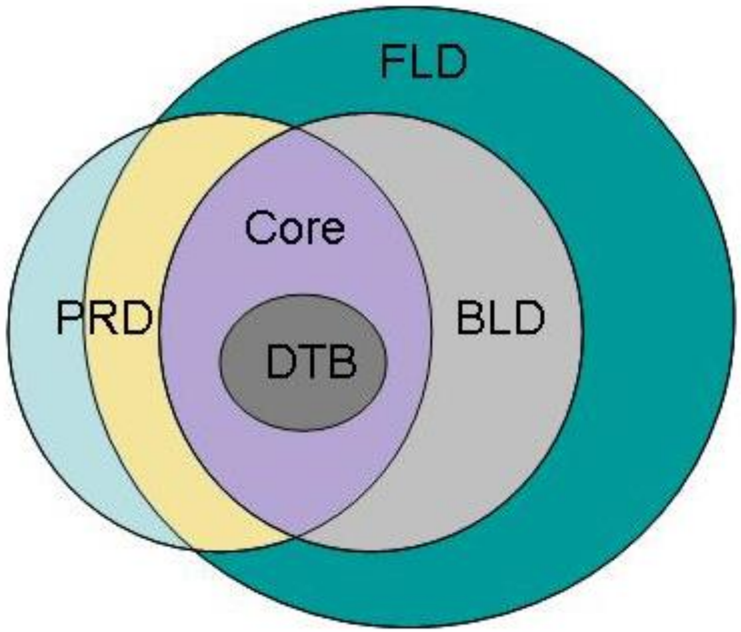
What is RIF?



- The Rule Interchange Format (RIF) facilitates rule set integration and synthesis.
- RIF is a W3C recommendation:
  `http://www.w3.org/TR/rif-overview/`
- There exist many declarative rule languages:
  - SILK, OntoBroker, SWRL, Answer Set Programming
- There exist many procedural rule languages:
  - Jess, Drools, IBM ILOG, Oracle Business Rules
- Prolog includes features of declarative and production rule languages.

© Macquarie University 2022

Family of RIF Dialects



- FLD: RIF Framework for Logic-based Dialects
- BLD: Basic Logic Dialect
- PRD: Production Rule Dialect
- Core: Core Dialect
- DTB: Datatypes and Built-ins

© Macquarie University 2022

RIF and OWL 2

- OWL 2 RL is an OWL 2 profile.
- OWL 2 RL is the intersection of RIF Core and OWL 2.
- Inferences in OWL RL can be expressed via RIF rules.
- Simple OWL 2 axioms correspond to rules, for example:

```
Class1    ⊑ Class2
Property1 ⊑ Property2
```

correspond to:

```
Forall ?X (If class1(?X) Then class2(?X))
Forall ?X ?Y (If property1(?X ?Y) Then property2(?X ?Y))
```

RIF and OWL 2 RL

- Some axioms can be transformed into rules, for example:

  Orphan $\sqsubseteq \forall$**hasParent.Dead**

  corresponds to:

```
Forall ?X ?Y
  ( If And(orphan(?X) hasParent(?X ?Y))
    Then dead(?Y) ).
```

RIF and OWL 2 RL

- Property chains provide further rule-like axioms, for example:

  hasParent ∘ hasBrother ⊑ hasUncle

  corresponds to:

```
Forall ?X ?Y ?Z
  (If (And(hasParent(?X ?Y) hasBrother(?Y ?Z))
      Then hasUncle(?X ?Z))
```

Example of an OWL 2 RL Ontology

TBox

(1) $Person \sqsubseteq Animal$	(2) $Person \equiv Human$
(3) $Man \sqsubseteq Person$	(4) $Woman \sqsubseteq Person$
(5) $Person \sqcap \exists author\_of.Manuscript \sqsubseteq Writer$	(6) $Paper \sqcup Book \sqsubseteq Manuscript$
(7) $Book \sqcap \exists topic.\{\text{“XML”}\} \sqsubseteq XMLbook$	(8) $Manuscript \sqcap \exists reviewed\_by.Person \sqsubseteq Reviewed$
(9) $OneAuthor \sqsubseteq \exists authored\_by_{\leq 1}.Person$	(10) $Manuscript \sqsubseteq \forall rating.Score$
(11) $Manuscript \sqsubseteq \forall topic.Topic$	(12) $average\_rating \sqsubseteq rating$
(13) $author\_of \equiv writes$	(14) $authored\_by \equiv author\_of^{-}$
(15) $\top \sqsubseteq \forall\ author\_of.Manuscript$	(16) $\top \sqsubseteq \forall\ author\_of^{-}.Person$
(17) $\top \sqsubseteq \forall\ reviewed\_by.Person$	(18) $\top \sqsubseteq \forall\ reviewed\_by^{-}.Manuscript$
(19) $\top \sqsubseteq \forall\ friendOf.Person$	(20) $\top \sqsubseteq \forall\ friendOf^{-}.Person$
(21) $first\_reader \equiv friendOf\ o\ author\_of$	(22) $friendOf \equiv friendOf^{-}$
(23) $Funct(average\_rating)$	

ABox

(1) Man(“Abiteboul”)	
(2) Man(“Buneman”)	(3) Man(“Suciu”)
(4) Book(“Data on the Web”)	(5) Book(“XML in Scottland”)
(6) Paper(“Growing XQuery”)	(7) Person(“Anonymous”)
(8) author_of(“Abiteboul”,“Data on the Web”)	(9) authored_by(“Data on the Web”,“Buneman”)
(10) author_of(“Suciu”,“Data on the Web”)	(11) author_of(“Buneman”,“XML in Scottland”)
(12) writes(“Simeon”,“Growing XQuery”)	(13) reviewed_by(“Data on the Web”,“Anonymous”)
(14) reviewed_by(“Growing XQuery”,“Almendros”)	(15) average_rating(“Data on the Web”,“Good”)
(16) rating(“XML in Scottland”,“Excellent”)	(17) average_rating(“Growing XQuery” ,“Good”)
(18) topic(“Data on the Web”,“XML”)	(19) topic(“Data on the Web”,“Web”)
(20) topic(“XML in Scottland”,“XML”)	(21) friendOf(Abiteboul,Buneman)
(22) $Almendros \equiv Jesus$	

© Macquarie University 2022

OWL 2 RL Ontology as a Logic Program

- Logic programming rules for OWL RL have the form:

```
triple(…) :- triple(…), triple(…).
```

- Example:

```
triple(C1, rdfs:subClassOf, C3):-triple(C1, rdfs:subClassOf, C2),triple(C2, rdfs:subClassOf, C3).
triple(X, rdf:type, C2):-triple(C1, rdfs:subClassOf, C2),triple(X, rdf:type, C1).
triple(X, rdf:type, C):-triple(P, rdfs:domain, C),triple(X, P, _).
triple(Y, rdf:type, C):-triple(P, rdfs:range, C),triple(_, P, Y).
triple(Y, P, X):-triple(P, rdf:type,  'http://www.w3.org/2002/07/owl#SymmetricProperty'),triple(X, P, Y).
triple(X,P2,Y):-triple(P1, rdfs:subPropertyOf, P2),triple(X,P1,Y).
triple(X,P2,Y):-triple(P1, owl:equivalentProperty, P2), triple(X,P1,Y).
triple(U, rdf:type, X):-triple(X, owl:someValuesFrom, Y),
                  triple(X, owl:onProperty, P),triple(U,P,V),triple(V, rdf:type, Y).
triple(U,P,Y):-triple(X, owl:hasValue,Y),triple(X, owl:onProperty,P),triple(U, rdf:type, X).
triple(C, rdfs:subClassOf, D):-triple(C, owl:intersectionOf, X),rdfs_member(D,X).
```

What is Missing?

- Rules are good for representing knowledge.
- Rule languages have powerful features that are not supported by OWL 2 RL:
  - non-monotonic rules
  - arbitrary functions with side effects
  - working with probabilities.

© Macquarie University 2022

Monotonic Reasoning

```
flies(X) :- bird(X).
bird(X)  :- eagle(X).
bird(X)  :- penguin(X).
eagle(sam).
penguin(tweety).

?- flies(Who).
Who = sam;
Who = tweety.    % Oops!
```

Non-monotonic Reasoning

```
flies(X)     :- bird(X), \+ abnormal(X).
bird(X)      :- eagle(X).
bird(X)      :- penguin(X).
abnormal(X)  :- penguin(X).
eagle(sam).
penguin(tweety).

?- flies(Who).
Who = sam.
```

What is Answer Set Programming (ASP)?

- ASP is a form of declarative programming.
- ASP has its roots in logic programming, deductive data-bases and non-monotonic reasoning.
- The basic idea of answer set programming is:
  - to represent a given problem by a set of rules,
  - to find answer sets for the program using an ASP solver,
  - to extract the solutions from the answer sets.
- By the way, ASP has been used in a decision support system for the Space Shuttle and many other real world applications.

© Macquarie University 2022

What is Answer Set Programming?

- An ASP program consists of a set of rules of the form:

  $\mathtt{L_0 \ ; \ \ldots \ ; \ L_i \ :\text{-} \ L_{i+1}, \ \ldots \ L_m, \ not \ L_{m+1}, \ \ldots, \ not \ L_n.}$
- `L` is a literal.
- A literal is either a positive atom `a` or a negative atom `-a`.
- The symbol `:-` stands for an *if*.
- The symbol `;` stands for a disjunction.
- `not` stands for negation as failure.
- `Head :- Body.` is a rule.
- A `Head.` without a body is a fact.
- `:- Body.` without a `Head` is a constraint.

© Macquarie University 2022

Example

```
% Facts

wine_bottle(brand1).
wine_bottle(brand2).
wine_bottle(brand3).
wine_bottle(brand4).
wine_bottle(brand5).
```

© Macquarie University 2022

Example

```
% Facts

type(brand1, whiteWine).
type(brand1, sweetWine).
type(brand2, whiteWine).
type(brand2, dryWine).
type(brand3, whiteWine).
type(brand3, dryWine).
type(brand4, redWine).
type(brand4, dryWine).
type(brand5, redWine).
type(brand5, sweetWine).
```

© Macquarie University 2022

Example

```
% Facts

person(john).
person(mary).
person(sue).

has_preference(john, whiteWine).
has_preference(mary, redWine).
has_preference(sue,  dryWine).
```

Example

```
% Rule

is_suitable_for(Brand, Person) :-
  has_preference(Person, Sort),
  type(Brand, Sort).

#show is_suitable_for/2.
```

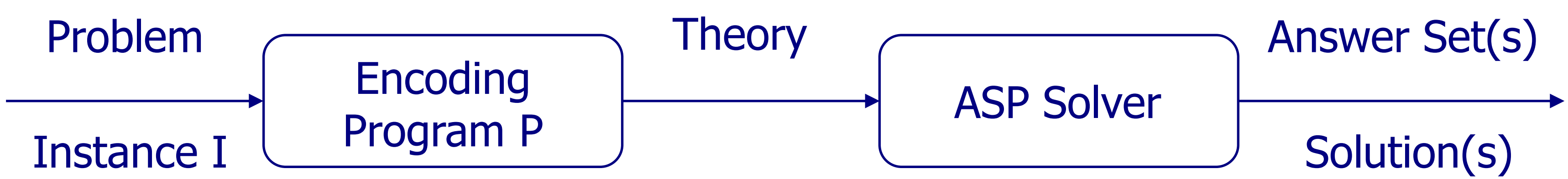
Answer Set

```
Answer: 1
  is_suitable_for(brand1, john)
  is_suitable_for(brand2, john)
  is_suitable_for(brand3, john)
  is_suitable_for(brand4, mary)
  is_suitable_for(brand5, mary)
  is_suitable_for(brand2, sue)
  is_suitable_for(brand3, sue)
  is_suitable_for(brand4, sue)
SATISFIABLE
```

ASP Solver

- General idea:

  problems are encoded as finite logic theories using rules and are solved by reducing them to answer sets which describe the solution(s) to the problem in a declarative way.



- Clingo is an ASP solver:

  `https://potassco.org/`

Generating the Closed World Assumption

```
% The following rule generates the Closed World Assumption
% (CWA) for the literal has_preference/2.
% That means we have now complete (positive and negative)
% information about has_preference/2

-has_preference(Person, Sort) :-
  person(Person),
  type(Brand, Sort),
  not has_preference(Person, Sort).

#show -has_preference/2.
```

© Macquarie University 2022

Example

```
has_preference(john, whiteWine).
has_preference(mary, redWine).
has_preference(sue,  dryWine).
```

```
Answer: 1
  -has_preference(john, sweetWine)
  -has_preference(john, dryWine)
  -has_preference(john, redWine)
  -has_preference(mary, whiteWine)
  -has_preference(mary, sweetWine)
  -has_preference(mary, dryWine)
  -has_preference(sue,  whiteWine)
  -has_preference(sue,  sweetWine)
  -has_preference(sue,  redWine)
SATISFIABLE
```

Generating Multiple Answer Sets

```
is_suitable_for(brand1, john)
is_suitable_for(brand2, john)
is_suitable_for(brand3, john)
is_suitable_for(brand4, mary)
is_suitable_for(brand5, mary)
is_suitable_for(brand2, sue)
is_suitable_for(brand3, sue)
is_suitable_for(brand4, sue)
```

```
% Rule with disjunction generates multiple answer sets.
% It actually generates 256 answer sets for our example.
% John can select in 8 possible ways.
% Mary can select in 4 possible ways.
% Sue can select in 8 possible ways.
% Therefore: 8 * 4 * 8 = 256

selects(Person, Brand) ; skips(Person, Brand) :-
  is_suitable_for(Brand, Person).

#show selects/2.
```

© Macquarie University 2022

Result

```
Answer: 1
selects(john, brand1)
selects(john, brand2)
selects(john, brand3)
selects(mary, brand4)
selects(mary, brand5)
selects(sue,  brand2)
selects(sue,  brand3)
selects(sue,  brand4)

Answer: 255
selects(sue,  brand4)
Answer: 256

SATISFIABLE
```

Excluding Answer Sets

```
% Constraint
% Exclude that the same person selects more than one brand.
:- selects(Person, Brand1),
   selects(Person, Brand2),
   Brand1 != Brand2.

% This constraint reduces the number of answer sets to 48.
% John can now select in 4 ways.
% Mary can now select in 3 ways.
% Sue can now select in 4 ways.
% Therefore: 4 * 3 * 4 = 48.
```

Result

```
Answer: 1
  selects(john, brand1)
  selects(mary, brand4)
  selects(sue,  brand2)
Answer: 2
  selects(john, brand1)
  selects(sue,  brand2)
…
Answer: 48
  selects(john, brand3)
  selects(mary, brand5)
  selects(sue,  brand4)
SATISFIABLE
```

© Macquarie University 2022

Excluding More Answer Sets

```
% Constraint
% Include only those answer sets where 3 persons select a
% brand.
:- P = #count { Person : selects(Person, Brand) },
   P != 3.

% This constraint reduces the number of answer sets
% to 18.
% John can now select in 3 ways.
% Mary can now select in 2 ways.
% Sue can now select in 3 ways.
% Therefore: 3 * 2 * 3 = 18
```

Result

```
Answer: 1
  selects(john, brand1)
  selects(mary, brand4)
  selects(sue,  brand2)
…
Answer: 18
  selects(john, brand3)
  selects(mary, brand5)
  selects(sue,  brand4)
SATISFIABLE
```

Now, we learn that ...

```
% Every person for which brand4 is suitable actually
% skips brand4.
% Every person for which brand2 is suitable actually
% skips brand2.
% John does not select brand3.

skips(Person, brand4) :-
   is_suitable_for(brand4, Person).

skips(Person, brand2) :-
   is_suitable_for(brand2, Person).

-selects(john, brand3).
```

Result

```
Answer: 1
selects(john, brand1)
selects(mary, brand5)
selects(sue,  brand3)
SATISFIABLE
```

Expressing RDF(S) in ASP

- Use a predicate `triple/3`:

```
triple("http://www.xxx.net/foaf.rdf#me",
       "foaf:name",
       "John Miller").
```

- Subclass relations:

```
triple(S, "rdf:type", C2) :-
   triple(S, "rdf:type", C1),
   triple(C1, "rdfs:subClassOf", C2).
```

Expressing Ontologies in ASP

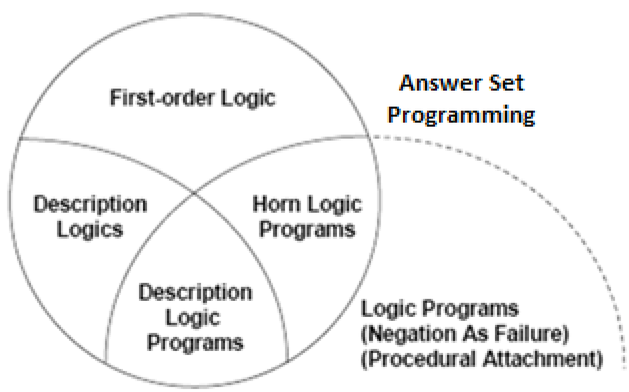
- Works <u>only</u> under limitations.
- Important differences between OWL/DLs and ASP.
- Main differences are:
  - open versus closed world assumption;
  - classical negation versus negation as failure;
  - unique name assumption in ASP;
  - existential quantifiers in DLs, e.g.:

    `∀X∃Y.(Wine(X) ⊃ hasColor(X, Y))`

© Macquarie University 2022

Description Logic Programs and ASP

- Description Logic Programs (basically OWL RL) are the intersection between Description Logics and Horn Logic programs.
- Answer Set Programming (ASP) includes Description Logic Programs and Horn Logic programs.



© Macquarie University 2022

Take-Home Messages

- RIF is a rule interchange format.
- RIF enables rule exchange across different formalisms.
- ASP uses rules to represent logic theories in a declarative way.
- An ASP solver is a tool that grounds the logic theory and searches for solutions in form of answer sets.
- Important differences between OWL/DL and ASP.
- Description logic programs (DLP) are the intersection of Description logics and Horn logic programs (HLP).
- ASP includes DLP and HLP.
- The Semantic Web needs also probabilistic rules.

© Macquarie University 2022