

*Paul Fodor*

# Logic and Inference: Rules

CSE 595 — Semantic Web

Instructor: Dr. Paul Fodor

Stony Brook University

<http://www3.cs.stonybrook.edu/~pfodor/courses/cse595.html>



# Lecture Outline

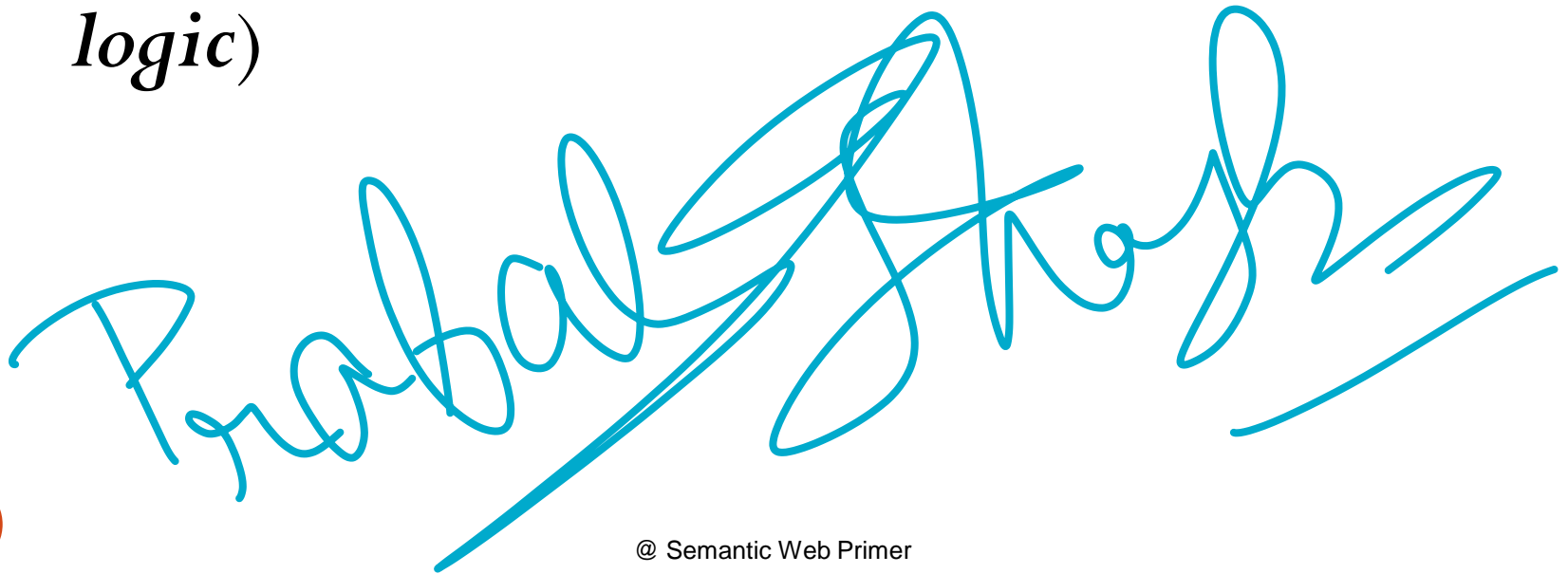
- Monotonic Rules 2 - 22
- OWL2 RL: Description Logic Meets Rules 49 - 56
- Rule Interchange Format: RIF 57 - 68
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN 77 - 80
- Nonmonotonic Rules 81 - 84
  - Example: Brokered Trade 85 - ( )
- Rule Markup Language (RuleML) 104 - 109

# Introduction

- All we did until now are forms of knowledge representation (*KR*), like knowledge about the content of web resources, knowledge about the concepts of a domain of discourse and their relationships (ontology)
- Knowledge representation had been studied long before the emergence of the World Wide Web in the area of artificial intelligence and, before that, in philosophy

# Introduction

- KR can be traced back to ancient Greece (because Aristotle is considered to be the father of logic)
- Logic is the foundation of knowledge representation, particularly in the form of *predicate logic* (also known as *first-order logic*)

A large, stylized blue signature or scribble that spans across the lower half of the slide, partially overlapping the footer area.

# Introduction

- Logic:
  - provides a high-level language in which knowledge can be expressed in a transparent way
  - has a high expressive power (maybe too high because it is intractable or undecidable in some cases)
  - has a well-understood formal semantics, which assigns an unambiguous meaning to logical statements
  - has a precise notion of *logical consequence*, which determines whether a statement follows semantically from a set of other statements (*premises*)

# Introduction

- There exist proof systems that can automatically derive statements syntactically from a set of premises.
- There exist proof systems for which semantic logical consequence coincides with syntactic derivation within the proof system.
  - Proof systems should be sound (all derived statements follow semantically from the premises) and complete (all logical consequences of the premises can be derived in the proof system).
- Predicate logic is unique in the sense that sound and complete proof systems do exist - More expressive logics (higher-order logics) do not have such proof systems.
- It is possible to trace the proof that leads to a logical consequence, so logic can provide explanations for answers.

# Introduction

- RDF and OWL2 profiles can be viewed as specializations of predicate logic:
  - One justification for the existence of such specialized languages is that they provide a syntax that fits well with the intended use (in our case, web languages based on tags).
  - Another justification is that they define reasonable subsets of logic where the computation is tractable (there is a trade-off between the expressive power and the computational complexity of certain logics: the more expressive the language, the less efficient the corresponding proof systems)

# Introduction

- Most OWL variants correspond to a *description logic*, a subset of predicate logic for which efficient proof systems exist
- Another subset of predicate logic with efficient proof systems comprises the *Horn rule systems* (also known as *Horn logic* or *definite logic programs*)

- A rule has the form:

$$\mathbf{A}_1, \dots, \mathbf{A}_n \rightarrow \mathbf{B}.$$

where  $\mathbf{A}_i$  and  $\mathbf{B}$  are atomic formulas.

- In Prolog notation:

$$\mathbf{B} \text{ :- } \mathbf{A}_1, \dots, \mathbf{A}_n.$$



# Introduction

- There are two intuitive ways of reading a Horn rule:
  - **deductive rules:** If  $\mathbf{A}_1, \dots, \mathbf{A}_n$  are known to be true, then  $\mathbf{B}$  is also true
    - There are two ways of **applying** deductive rules:
      - from the body ( $\mathbf{A}_1, \dots, \mathbf{A}_n$ ) to the conclusion ( $\mathbf{B}$ ) (*forward chaining*)
      - from the conclusion (goal) to the body (*backward reasoning*)
  - **reactive rules:** If the conditions  $\mathbf{A}_1, \dots, \mathbf{A}_n$  are true, then carry out the action  $\mathbf{B}$ .

# Introduction

- Description logics and Horn logic are **orthogonal** in the sense that neither of them is a subset of the other
- For example, it is impossible to define the class of *happy spouses* as those who are married to their best friend in description logics, but this piece of knowledge can easily be represented using rules:

**married(X, Y), bestFriend(X, Y) →  
happySpouse(X).**

- On the other hand, rules cannot (in the general case) assert:
  - (a) **negation/complement** of classes
  - (b) **disjunctive/union information** (for instance, that a person is either a man or a woman)
  - (c) **existential quantification** (for instance, that all persons have a father).

# Monotonic and nonmonotonic rules

- Predicate logic is *monotonic*: if a conclusion can be drawn, it remains valid even if new knowledge becomes available

- Even if a rule uses negation,

R1 : If birthday, then special discount.

R2 : If **not birthday**, then not special discount.

it works properly in cases where the birthday is known

# Monotonic and nonmonotonic rules

- Imagine a customer who refuses to provide his birthday because of privacy concerns, then the preceding rules cannot be applied because their premises are not known.

R1 : If birthday, then special discount.

R2 : If not birthday, then not special discount.

**R2' : If birthday is not known, then not special discount.**

- R2' is not within the expressive power of predicate logic because its conclusion may become invalid if the customer's birthday becomes known at a later stage and it happens to coincide with the purchase date.
- Adding knowledge later that invalidates some of the conclusions is called nonmonotonic because the addition of new information leads to a loss of a consequence

# Rules on the Semantic Web

- Rule technology has been around for decades, has found extensive use in practice, and has reached significant maturity
  - led to a broad variety of approaches
  - it is more difficult to standardize this area in the context of the (semantic) web
- A W3C working group has developed the Rule Interchange Format (RIF) standard
  - Whereas RDF and OWL are languages meant for directly representing knowledge, RIF was designed primarily for the exchange of rules across different applications
    - For example, an online store might wish to make its pricing, refund, and privacy policies, which are expressed using rules, accessible to intelligent agents

# Rules on the Semantic Web

- Due to the underlying aim of serving as an interchange format among different rule systems, RIF combines many of their features, and is quite complex
- Those wishing to develop rule systems for the Semantic Web have various alternatives:
  - Rules over RDF can be expressed using SPARQL constructs. SPARQL is not a rule language, as basically it carries out one application of a rule.
  - SPIN is a rule system developed on top of SPARQL
  - SWRL couples OWL DL functionalities with certain types of rules
  - Model in terms of OWL but use rule technology for implementation purposes: OWL2 RL

# Example **Monotonic Rules: Family**

- Imagine a database of facts about some family relationships which contains facts about the following **base predicates**:

**mother(X, Y)** X is the **mother** of Y ✓

**father(X, Y)** X is the **father** of Y ✓

**male(X)** X is **male** ✓

**female(X)** X is **female** ✓

*Basic Rules*

# Example Monotonic Rules: Family

- We can infer further relationships using appropriate rules:
  - a parent is either a father or a mother.

**mother**(X, Y) → **parent**(X, Y) .

**father**(X, Y) → **parent**(X, Y) .

- a brother to be a male person sharing a parent:

**male**(X) , **parent**(P, X) , **parent**(P, Y) ,  
**notSame**(X, Y) → **brother**(X, Y) .

- The predicate **notSame** denotes inequality; we assume that such facts are kept in a database

**female**(X) , **parent**(P, X) ,  
**parent**(P, Y) , **notSame**(X, Y) →  
**sister**(X, Y) .



# Example Monotonic Rules: Family

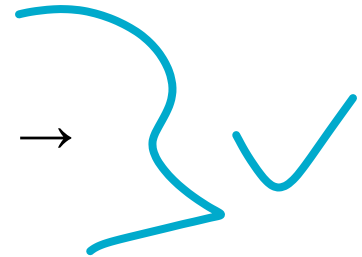
- An uncle is a brother of a parent:

**brother (X, P) , parent (P, Y) →  
uncle (X, Y) .**



- A grandmother is the mother of a parent:

**mother (X, P) , parent (P, Y) →  
grandmother (X, Y) .**



- An ancestor is either a parent or an ancestor of a parent:

**parent (X, Y) → ancestor (X, Y) .  
ancestor (X, P) , parent (P, Y) →  
ancestor (X, Y) .**

# Monotonic Rules: Syntax

- Let us consider a simple rule stating that all loyal customers with ages over 60 are entitled to a special discount:

**loyalCustomer**(X) , **age**(X) > 60 →  
**discount**(X) .

- Rules have:

- variables, which are placeholders for values: **X**
- constants, which denote fixed values: **60**
- predicates, which relate objects: **loyalCustomer**, **>**
- function symbols, which denote a value, when applied to certain arguments: **age**

- In case no function symbols are used, we discuss *function-free (Horn) logic*.

# Rules

- A rule has the form:

$$\mathbf{B}_1, \dots, \mathbf{B}_n \rightarrow \mathbf{A}$$

where  $\mathbf{A}$  and  $\mathbf{B}_i$  are atomic formulas

- $\mathbf{A}$  is the *head* of the rule
- $\mathbf{B}_i$  are the *premises* of the rule
- The set  $\{\mathbf{B}_1, \dots, \mathbf{B}_n\}$  is referred to as the *body* of the rule
- The commas in the rule body are read *conjunctively*:  
if  $\mathbf{B}_1$  and  $\mathbf{B}_2$  and ... and  $\mathbf{B}_n$  are true, then  $\mathbf{A}$  is also true
  - (or equivalently, to prove  $\mathbf{A}$  it is sufficient to prove *all* of  $\mathbf{B}_1, \dots, \mathbf{B}_n$ )

# Rules

- Variables may occur in  $\mathbf{A}, \mathbf{B}_1, \dots, \mathbf{B}_n$ .

- For example,

$\text{loyalCustomer}(X), \text{age}(X) > 60 \rightarrow$   
 $\text{discount}(X).$

is applied for **any** customer: if a customer happens to be loyal and over 60, then they gets the discount.

- The variable  $\mathbf{X}$  is implicitly universally quantified (using  $\forall \mathbf{X}$ )
- In general, all variables occurring in a rule are implicitly universally quantified.

# Rules

- A rule  $r$ :

$$\underline{B_1}, \dots, \overline{B_n} \rightarrow A$$

is interpreted as the following formula, denoted by  $pl(r)$ :

$$\forall X_1 \dots \forall X_k ( (B_1 \wedge \dots \wedge B_n) \rightarrow A )$$

or equivalently,

$$\forall X_1 \dots \forall X_k ( A \vee \neg B_1 \vee \dots \vee \neg B_n )$$

where  $X_1, \dots, X_k$  are all variables occurring in  $A$ ,

$B_1, \dots, B_n$ .

# Logic Programs

- A *fact* is an atomic formula, such as

**loyalCustomer(a345678) .**

which says that the customer with ID **a345678** is loyal

- If there are variables in a fact, then they are implicitly universally quantified.
- A *logic program*  $P$  is a finite set of facts and rules
  - Its predicate logic translation  $pl(P)$  is the set of all predicate logic interpretations of rules and facts in  $P$ .



# Logic Programs

- A *goal* or *query* **G** asked to a logic program has the form

$$\mathbf{B}_1, \dots, \mathbf{B}_n \rightarrow$$

- If  $n = 0$ , we have the empty goal  $\square$ .
- The interpretation of a goal is:

$$\forall \mathbf{X}_1 \dots \forall \mathbf{X}_k (\neg \mathbf{B}_1 \vee \dots \vee \neg \mathbf{B}_n)$$

where  $\mathbf{X}_1, \dots, \mathbf{X}_k$  are all variables occurring in  $\mathbf{B}_1, \dots, \mathbf{B}_n$

# Logic Programs

- The goal formula is equivalent to

$$\forall \mathbf{x}_1 \dots \forall \mathbf{x}_k (\mathbf{false} \vee \neg \mathbf{B}_1 \vee \dots \vee \neg \mathbf{B}_n)$$

so the missing rule head can be thought of as a contradiction **false**.

- An equivalent representation in predicate logic is:

$$\neg \exists \mathbf{x}_1 \dots \exists \mathbf{x}_k (\mathbf{B}_1 \wedge \dots \wedge \mathbf{B}_n)$$



# Logic Programs

- Suppose we know the fact

**$p(a)$**  .

and we have the goal

**$p(X)$**   $\rightarrow$

- we want to know whether there is a value for which  $p$  is true
- We expect a positive answer because of the fact  **$p(a)$**  .
- Thus  **$p(X)$**  is existentially quantified
- Why do we negate the formula?
  - The explanation is that we use a proof technique from mathematics called *proof by contradiction*
    - This technique proves that a statement  $A$  follows from a statement  $B$  by assuming that  $A$  is false and deriving a contradiction when combined with  $B$ . Then  $A$  must follow from  $B$ .

# Logic Programs

- In logic programming we prove that a goal can be answered positively by negating the goal and proving that we get a contradiction using the logic program.
- For example, given the logic program

**$p(a)$  .**

and the goal

**$\neg \exists x p(x)$**

we get a logical contradiction: the second formula says that no element has the property  **$p$** , but the first formula says that the value of  **$a$**  does have the property  **$p$** .

Thus  **$\neg \exists x p(x)$**  follows from  **$p(a)$**  .

# Monotonic Rules: Semantics

- Given a logic program  $P$  and a query  $\mathbf{B}_1, \dots, \mathbf{B}_n \rightarrow$  with the variables  $\mathbf{X}_1, \dots, \mathbf{X}_k$ , we answer positively if, and only if,

$$pl(P) \models \exists \mathbf{X}_1 \dots \exists \mathbf{X}_k (\mathbf{B}_1 \wedge \dots \wedge \mathbf{B}_n)$$

or equivalently, if

$$pl(P) \cup \{ \neg \exists \mathbf{X}_1 \dots \exists \mathbf{X}_k (\mathbf{B}_1 \wedge \dots \wedge \mathbf{B}_n) \} \text{ is}$$

**unsatisfiable**

- We give a positive answer if the predicate logic representation of the program  $P$ , together with the predicate logic interpretation of the query, is unsatisfiable (a contradiction).

# Monotonic Rules: Semantics

- Predicate Logic Semantics
  - A predicate logic model,  $A$ , consists of
    - a domain  $\text{dom}(A)$ , a nonempty set of objects about which the formulas make statements
    - an element from the domain for each constant
    - a concrete function on  $\text{dom}(A)$  for every function symbol
    - a concrete relation on  $\text{dom}(A)$  for every predicate
  - When the symbol  $=$  is used to denote equality (i.e., its interpretation is fixed), we talk of *Horn logic with equality*
  - Logical connectives  $\neg, \vee, \wedge, \rightarrow, \forall, \exists$
  - A formula  $\phi$  *follows* from a set  $M$  of formulas if  $\phi$  is true in all models  $A$  in which  $M$  is true (that is, all formulas in  $M$  are true in  $A$ ).

# Monotonic Rules: Semantics

- A formula  $\phi$  *follows* from a set  $M$  of formulas if  $\phi$  is true in all models  $A$  in which  $M$  is true (that is, all formulas in  $M$  are true in  $A$ ).
- Regardless of how we interpret the constants, predicates, and function symbols occurring in  $P$  and the query, once the predicate logic interpretation of  $P$  is true,  $\exists \mathbf{X}_1 \dots \exists \mathbf{X}_k (\mathbf{B}_1 \wedge \dots \wedge \mathbf{B}_n)$  must be true: that is, there are values for the variables  $\mathbf{X}_1, \dots, \mathbf{X}_k$  such that all atomic formulas  $\mathbf{B}_i$  become true.

# Monotonic Rules: Semantics

- Suppose  $P$  is the program

$$p(a) .$$

$$p(x) \rightarrow q(x) .$$

- Consider the query

$$q(x) \rightarrow$$

- $q(a)$  follows from  $pl(P)$
- $\exists x q(x)$  follows from  $pl(P)$
- $pl(P) \cup \{ \neg \exists x q(x) \}$  is unsatisfiable
- If we consider the query

$$q(b) \rightarrow$$

then we give a negative answer because  $q(b)$  does not follow from  $pl(P)$

# Least Herbrand Model Semantics

- Instead of considering any domain  $dom(A)$ , we can consider only the names in the program (predicate names, constants, functors)
- Then we have *Herbrand semantics*:
  - Given an alphabet  $A$ , the set of all **ground terms** constructed from the constant and function symbols of  $A$  is called the *Herbrand Universe* of  $A$  (denoted by  $U_A$ ).
  - Consider the program:

**$p(\text{zero})$  .**

**$p(s(s(X))) \leftarrow p(X)$  .**

- The Herbrand Universe of the program's alphabet is:  
 $U_A = \{\text{zero}, s(\text{zero}), s(s(\text{zero})), \dots\}$

# Least Herbrand Model Semantics

- Consider a "relations" program:

```
parent(pam, bob) .      parent(bob, ann) .  
parent(tom, bob) .      parent(bob, pat) .  
parent(tom, liz) .      parent(pat, jim) .  
grandparent(X,Y) :-  
    parent(X,Z) , parent(Z,Y) .
```

- The Herbrand Universe of the program's alphabet is:

$$U_A = \{\text{pam, bob, tom, liz, ann, pat, jim}\}$$



# Least Herbrand Model Semantics

- Given an alphabet  $A$ , the set of all **ground atomic formulas** over  $A$  is called the *Herbrand Base* of  $A$  (denoted by  $B_A$ ).

- Consider the program:

**$p(\text{zero})$  .**

**$p(s(s(X))) \leftarrow p(X)$  .**

- The Herbrand Base of the program's alphabet is:

$$B_A = \{p(\text{zero}), p(s(\text{zero})), p(s(s(\text{zero}))), \dots\}$$

# Least Herbrand Model Semantics

- Consider the "relations" program:

```
parent(pam, bob) .      parent(bob, ann) .  
parent(tom, bob) .      parent(bob, pat) .  
parent(tom, liz) .      parent(pat, jim) .  
grandparent(X,Y) :-  
    parent(X,Z) , parent(Z,Y) .
```

- The Herbrand Base of the program's alphabet is:

$B_A = \{\text{parent}(\text{pam}, \text{pam}), \text{parent}(\text{pam}, \text{bob}),$   
 $\text{parent}(\text{pam}, \text{tom}), \dots, \text{parent}(\text{bob}, \text{pam}), \dots,$   
 $\text{grandparent}(\text{pam}, \text{pam}), \dots, \text{grandparent}(\text{bob}, \text{pam}),$   
 $\dots\}.$

# Least Herbrand Model Semantics

- A *Herbrand Interpretation* of a program  $P$  is an interpretation  $I$  such that:
  - The domain of the interpretation:  $|I| = U_P$
  - For every constant  $\mathbf{c}$ :  $\mathbf{c}_I = \mathbf{c}$
  - For every function symbol  $\mathbf{f}/\mathbf{n}$ :  
 $\mathbf{f}_I(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_n)$
  - For every predicate symbol  $\mathbf{p}/\mathbf{n}$ :  $\mathbf{p}_I \subseteq (U_P)^n$  (i.e. some subset of  $\mathbf{n}$ -tuples of ground terms)
- A *Herbrand Model* of a program  $P$  is a Herbrand interpretation that is a model of  $P$ .

# Least Herbrand Model Semantics

- All Herbrand interpretations of a program give the same “*meaning*” to the constant and function symbols.
  - Different Herbrand interpretations differ only in the “*meaning*” they give to the predicate symbols.
- We often write a Herbrand model simply by listing the subset of the Herbrand base that is true in the model
  - Example: Consider our numbers program, where  $\{p(\text{zero}), p(s(s(\text{zero}))), p(s(s(s(s(\text{zero}))))), \dots\}$  represents the Herbrand model that treats  $p_I = \{\text{zero}, s(s(\text{zero})), s(s(s(s(\text{zero})))) , \dots\}$  as the meaning of **p**.

# Sufficiency of Herbrand Models

- Let  $P$  be a definite program. If  $I'$  is a model of  $P$  then  $I = \{A \in B_p \mid I' \models A\}$  is a Herbrand model of  $P$ .

Proof (by contradiction):

Let  $I$  be a Herbrand interpretation.

Assume that  $I'$  is a model of  $P$  but  $I$  is not a model.

Then there is some ground instance of a clause in  $P$ :

$$\mathbf{A_0} \quad :- \quad \mathbf{A_1}, \quad \dots, \quad \mathbf{A_n}.$$

which is not true in  $I$  i.e.,  $I \models A_1, \dots, I \models A_n$  but  $I \not\models A_0$

By definition of  $I$  then,  $I' \models A_1, \dots, I' \models A_n$  but  $I' \not\models A_0$

Thus,  $I'$  is not a model of  $P$ , which contradicts our earlier assumption.

# Definite programs only

- Let  $P$  be a definite program. If  $I'$  is a model of  $P$  then  $I = \{A \in B_p \mid I' \models A\}$  is a Herbrand model of  $P$ .
- This property holds only for definite programs!
  - Consider  $P = \{\neg p(a), \exists X.p(X)\}$ 
    - There are two Herbrand interpretations:  $I_1 = \{p(a)\}$  and  $I_2 = \{\}$ 
      - The first is not a model of  $P$  since  $I_1 \not\models \neg p(a)$ .
      - The second is not a model of  $P$  since  $I_2 \not\models \exists X.p(X)$
    - But there is a non-Herbrand model  $I$ :
      - $|I| = \mathbb{N}$ , the set of natural numbers
      - $a_I = 0$
      - $p_I = \text{"is odd"}$

# Properties of Herbrand Models

- 1) If  $M$  is a set of Herbrand Models of a definite program  $P$ , then  $\bigcap M$  is also a Herbrand Model of  $P$ .
- 2) For every definite program  $P$  there is a unique least model  $M_p$  such that:
  - $M_p$  is a Herbrand Model of  $P$  and,
  - for every Herbrand Model  $M$ ,  $M_p \subseteq M$ .
- 3) For any definite program, if every Herbrand Model of  $P$  is also a Herbrand Model of  $F$ , then  $P \models F$ .
- 4)  $M_p =$  the set of all ground logical consequences of  $P$ .

# Properties of Herbrand Models

- If  $M_1$  and  $M_2$  are Herbrand models of  $P$ , then  $M = M_1 \cap M_2$  is a model of  $P$ .
- Assume  $M$  is not a model.
- Then there is some clause  $A_0: \neg A_1, \dots, A_n$  such that  $M \models A_1, \dots, M \models A_n$  but  $M \not\models A_0$ .
- Which means  $A_0 \notin M_1$  or  $A_0 \notin M_2$ .
- But  $A_1, \dots, A_n \in M_1$  as well as  $M_2$ .
- Hence one of  $M_1$  or  $M_2$  is not a model.



# Properties of Herbrand Models

- There is a unique least Herbrand model
- Let  $M_1$  and  $M_2$  are two incomparable minimal Herbrand models, i.e.,  
 $M = M_1 \cap M_2$  is also a Herbrand model (previous theorem), and  $M \subseteq M_1$  and  $M \subseteq M_2$
- Thus  $M_1$  and  $M_2$  are not minimal.

# Least Herbrand Model

- The least Herbrand model  $M_p$  of a definite program  $P$  is the set of all ground logical consequences of the program.

$$M_p = \{A \in B_p \mid P \models A\}$$

- First,  $M_p \supseteq \{A \in B_p \mid P \models A\}$ :
  - By definition of logical consequence,  $P \models A$  means that  $A$  has to be in every model of  $P$  and hence also in the least Herbrand model.

# Least Herbrand Model

- Second,  $M_p \subseteq \{A \in B_p \mid P \models A\}$ :
  - If  $M_p \models A$  then  $A$  is in every Herbrand model of  $P$ .
  - But assume there is some model  $I' \models \neg A$ .
  - By sufficiency of Herbrand models, there is some Herbrand model  $I$  such that  $I \models \neg A$ .
  - Hence  $A$  is not in some Herbrand model, and hence is not in  $M_p$ .

# Finding the Least Herbrand Model

- Immediate consequence operator:

- Given  $I \subseteq B_p$ , construct  $I'$  such that

$$I' = \{A_0 \in B_p \mid A_0 \leftarrow A_1, \dots, A_n \text{ is a ground instance of a clause in } P \text{ and } A_1, \dots, A_n \in I\}$$

- $I'$  is said to be the *immediate consequence of*  $I$ .
- Written as  $I' = Tp(I)$ ,  $Tp$  is called the immediate consequence operator.
- Consider the sequence:

$$\emptyset, Tp(\emptyset), Tp(Tp(\emptyset)), \dots, Tp^i(\emptyset), \dots$$

- $M_p \supseteq Tp^i(\emptyset)$  for all  $i$ .
- Let  $Tp \uparrow \omega = \bigcup_{i=0, \infty} Tp^i(\emptyset)$

- Then  $M_p \subseteq Tp \uparrow \omega$

# Computing Least Herbrand Models: An Example

```
parent(pam, bob) .
parent(tom, bob) .
parent(tom, liz) .
parent(bob, ann) .
parent(bob, pat) .
parent(pat, jim) .
```

```
anc(X,Y) :-
    parent(X,Y) .
anc(X,Y) :-
    parent(X,Z) ,
    anc(Z,Y) .
```

$M_1$	$\emptyset$
$M_2 = T_P(M_1) =$	$\{\text{parent}(\text{pam}, \text{bob}),$ $\text{parent}(\text{tom}, \text{bob}),$ $\text{parent}(\text{tom}, \text{liz}),$ $\text{parent}(\text{bob}, \text{ann}),$ $\text{parent}(\text{bob}, \text{pat}),$ $\text{parent}(\text{pat}, \text{jim}) \}$
$M_3 = T_P(M_2) =$	$\{\text{anc}(\text{pam}, \text{bob}), \quad \text{anc}(\text{tom}, \text{bob}),$ $\text{anc}(\text{tom}, \text{liz}), \quad \text{anc}(\text{bob}, \text{ann}),$ $\text{anc}(\text{bob}, \text{pat}), \quad \text{anc}(\text{pat}, \text{jim}) \}$ $\cup M_2$
$M_4 = T_P(M_3) =$	$\{\text{anc}(\text{pam}, \text{ann}), \quad \text{anc}(\text{pam}, \text{pat}),$ $\text{anc}(\text{tom}, \text{ann}), \quad \text{anc}(\text{tom}, \text{pat}),$ $\text{anc}(\text{bob}, \text{jim}) \} \cup M_3$
$M_5 = T_P(M_4) =$	$\{\text{anc}(\text{pam}, \text{jim}), \quad \{\text{anc}(\text{tom}, \text{jim}) \}$ $\cup M_4$
$M_6 = T_P(M_5) =$	$M_5$

# Ground and Parameterized Witnesses

- Suppose we know the fact

**$p(a)$**  .

and we have the goal

**$p(X)$**   $\rightarrow$

- Responding **true** to parametrized queries is correct, but not satisfactory
- The appropriate answer is a substitution  $\{X/a\}$  which gives an instantiation for  **$X$** , making the answer positive.
- The constant  **$a$**  is called a *ground witness*
- Given two facts:  **$p(a)$**  . and  **$p(b)$**  . there are two ground witnesses to the same query:  **$a$**  and  **$b$** .

# Ground and Parameterized Witnesses

- Ground witnesses are not always the optimal answer

- Consider the logic program:

**add**(**X**, **0**, **X**) .

**add**(**X**, **Y**, **Z**)  $\rightarrow$  **add**(**X**, **s**(**Y**), **s**(**Z**)) .

- This program computes addition: if we read **s** as the “successor function,” which returns as value the value of its argument plus 1
    - The **add** predicate computes the sum of its first two arguments into its third argument
    - Consider the query:

**add**(**X**, **s**<sup>8</sup>(**0**), **Z**)  $\rightarrow$

- Possible ground witnesses are determined by the substitutions:  
 $\{\mathbf{X}/\mathbf{0}, \mathbf{Z}/\mathbf{s}^8(\mathbf{0})\},$   
 $\{\mathbf{X}/\mathbf{s}(\mathbf{0}), \mathbf{Z}/\mathbf{s}^9(\mathbf{0})\},$   
 $\{\mathbf{X}/\mathbf{s}(\mathbf{s}(\mathbf{0})), \mathbf{Z}/\mathbf{s}^{10}(\mathbf{0})\}, \dots$

# Ground and Parameterized Witnesses

- The parameterized witness

$$\mathbf{Z} = \mathbf{s}^8(\mathbf{X})$$

is the most general way to witness the existential query

$$\exists \mathbf{X} \exists \mathbf{Z} \text{ add}(\mathbf{X}, \mathbf{s}^8(0), \mathbf{Z})$$

since it represents the fact that  $\text{add}(\mathbf{X}, \mathbf{s}^8(0), \mathbf{Z})$  is true whenever the value of  $\mathbf{Z}$  equals the value of  $\mathbf{X}$  plus 8.

- The computation of most general witnesses is the primary aim of a proof system, called *SLD resolution*



# Lecture Outline



- Monotonic Rules
- OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)

# OWL2 RL: Description Logic Meets Rules

- OWL2 RL represents the intersection of OWL and Horn logic, that is, the part of one language that can be translated in a semantics-preserving way from OWL to rules, and vice versa.
- From the modeler's perspective, there is freedom to use either OWL or rules (and associated tools and methodologies) for modeling purposes, depending on the modeler's experience and preferences.
- From the implementation perspective, either description logic reasoners or deductive rule systems can be used: it is possible to model using one framework, such as OWL, and to use a reasoning engine from the other framework, such as rules.

OWL  $\cap$  Horn logic

# OWL2 RL: Description Logic Meets Rules

Some constructs of RDF Schema and OWL2 RL can be expressed in Horn logic, while some constructs, in general cannot be expressed

- A triple of the form **(a, P, b)** in RDF can be expressed as a fact:

**P(a, b) .**

- an instance declaration of the form **type(a, C)**, stating that **a** is an instance of class **C**, can be expressed as

**C(a) .**

- The fact that **C** is a subclass of **D** is expressed as

**C(X) → D(X) .**

- The fact that **P** is a subproperty of **Q** is expressed as

**P(X, Y) → Q(X, Y) .**

# OWL2 RL: Description Logic Meets Rules

- Domain and range restrictions can also be expressed in Horn logic: **C** is the domain of property **P**, while **D** is the range of property **P**:

$$\mathbf{P}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathbf{C}(\mathbf{X}) .$$

$$\mathbf{P}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathbf{D}(\mathbf{Y}) .$$

- **equivalentClass(C, D)** can be expressed by the pair of rules:

$$\mathbf{C}(\mathbf{X}) \rightarrow \mathbf{D}(\mathbf{X}) .$$

$$\mathbf{D}(\mathbf{X}) \rightarrow \mathbf{C}(\mathbf{X}) .$$

- **equivalentProperty(P, Q)** can be expressed by the pair of rules:

$$\mathbf{P}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathbf{Q}(\mathbf{X}, \mathbf{Y}) .$$

$$\mathbf{Q}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathbf{P}(\mathbf{X}, \mathbf{Y}) .$$

# OWL2 RL: Description Logic Meets Rules

- Transitivity of a property **P** is expressed as:

$$\mathbf{P}(\mathbf{X}, \mathbf{Y}), \mathbf{P}(\mathbf{Y}, \mathbf{Z}) \rightarrow \mathbf{P}(\mathbf{X}, \mathbf{Z}).$$

- The intersection of classes **C1** and **C2** is a subclass of **D**:

$$\mathbf{C1}(\mathbf{X}), \mathbf{C2}(\mathbf{X}) \rightarrow \mathbf{D}(\mathbf{X}).$$

- **C** is a subclass of the intersection of **D1** and **D2**:

$$\mathbf{C}(\mathbf{X}) \rightarrow \mathbf{D1}(\mathbf{X}).$$

$$\mathbf{C}(\mathbf{X}) \rightarrow \mathbf{D2}(\mathbf{X}).$$

- the union of **C1** and **C2** is a subclass of **D**:

$$\mathbf{C1}(\mathbf{X}) \rightarrow \mathbf{D}(\mathbf{X}).$$

$$\mathbf{C2}(\mathbf{X}) \rightarrow \mathbf{D}(\mathbf{X}).$$

- The opposite direction is outside the expressive power of Horn logic (see next slide).

# OWL2 RL: Description Logic Meets Rules

- **C** is a subclass of the union of **D1** and **D2** would require a **disjunction** in the head of the corresponding rule

**$C(x) \rightarrow D1(x) \vee D2(x) .$**

which is not available in Horn logic

# OWL2 RL: Description Logic Meets Rules

- OWL range restriction:

```
:C rdfs:subClassOf [  
  rdf:type owl:Restriction ;  
  owl:onProperty :P ;  
  owl:allValuesFrom :D ] .
```

can be represented as the rule:

**$C(X), P(X, Y) \rightarrow D(Y) .$**

- the opposite direction cannot be expressed in Horn logic

# OWL2 RL: Description Logic Meets Rules

```
[ rdf:type owl:Restriction ;  
  owl:onProperty :P ;  
  owl:allValuesFrom :D ]  
  rdfs:subClassOf :C.
```

can be represented as the rule:

$$P(X, \_) , (\text{Forall } Y, (P(X, Y) \rightarrow D(Y))) \\ \rightarrow C(X) .$$

which is not available in Horn logic

- Also, cardinality constraints and complement of classes cannot be expressed in Horn logic.



# Lecture Outline

- Monotonic Rules
- OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)

# Rule Interchange Format: RIF

- Rules exhibit a broad variety (e.g., action rules, first order rules, logic programming)
  - As a consequence, the aim of the *W3C Rule Interchange Format Working Group* was not to develop a new rule language that would fit all purposes, but rather to focus on the **interchange** among the various (existing or future) rule systems on the web
  - The approach taken was to develop a family of languages, from **basic** to existing state of the art, called *dialects*, that can be interchanged on the Web
  - Most of the work of the RIF Working Group was dedicated to semantic aspects
    - Of course, rule interchange takes place at the syntactic level (e.g., using XML) using mappings between the various syntactic features of a logic system and RIF, but the main objective is to interchange rules in a **semantics preserving way**.

# Rule Interchange Format: RIF

- Documents with links:
  - [RIF Overview \(Second Edition\)](#)
  - [RIF Use Cases and Requirements \(Second Edition\)](#)
  - [RIF Core Dialect \(Second Edition\)](#)
  - [RIF Basic Logic Dialect \(Second Edition\)](#)
  - [RIF Production Rule Dialect \(Second Edition\)](#)
  - [RIF Framework for Logic Dialects \(Second Edition\)](#)
  - [RIF Datatypes and Built-Ins 1.0 \(Second Edition\)](#)
  - [RIF RDF and OWL Compatibility \(Second Edition\)](#)
  - [OWL 2 RL in RIF \(Second Edition\)](#)
  - [RIF Combination with XML data \(Second Edition\)](#)
  - [RIF In RDF \(Second Edition\)](#)
  - [RIF Test Cases \(Second Edition\)](#)
  - [RIF Primer \(Second Edition\)](#)

# Rule Interchange Format: RIF

- RIF defined two kinds of dialects:
  - ***Logic-based dialects*** are meant to include rule languages that are based on some form of logic; for example, first-order logic and various logic programming approaches with different interpretations of negation (*answer-set programming*, *well-founded semantics*, etc.)
    - The concrete dialects developed so far under this branch are:
      - ***RIF Core*** corresponding to function-free Horn logic
      - ***RIF Basic Logic Dialect (BLD)*** corresponding to Horn logic with equality
  - ***Rules with actions*** are meant to include *production systems* and *reactive rules*. The concrete dialect developed so far:
    - ***Production Rule Dialect (RIF-PRD)***

# Rule Interchange Format: RIF

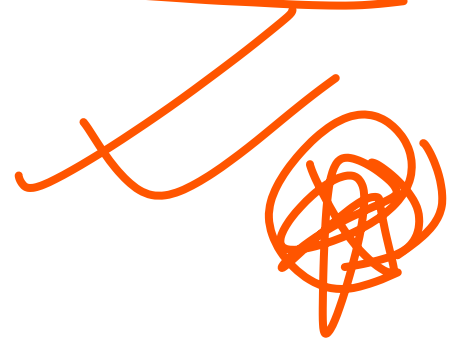
- RIF was designed to be both uniform and extensible
  - Uniformity is achieved by expecting the syntax and semantics of all RIF dialects to share basic principles
  - Extensibility refers to the possibility of future dialects being developed and added to the RIF family
- For the logic-based side, the RIF Working Group developed the *Framework for Logic Dialects* (**RIFFLD**) which allows one to specify various rule languages by instantiating the various parameters of the approach.

# RIF-BLD

- The *RIF Basic Logic Dialect* corresponds to Horn logic with equality plus:
  - data types (such as, integer, boolean, string, date),
  - “built-in” predicates (such as, numeric-greater-than, starts-with, date-less-than), and functions (such as numeric-subtract, replace, hoursfrom-time), and
  - frames (like in F-Logic) represent objects with their properties as *slots* (for example, a class professor with slots such as name, office, phone, department)  
`oid[slot1 -> value1, ..., slotn -> valuen]`

# RIF-BLD

- The syntax of RIF is straightforward, though quite verbose (of course, there is also an XML-based syntax to support interchange between rule systems)
- Variable names begin with a question mark ?
- The symbols =, #, and ## are used to express:  
equality, class membership, and subclass relationship



# RIF-BLD

- Examples:

- *A film is considered successful if it has received critical acclaim (say, a rating higher than 8 out of 10) or was financially successful (produced more than \$100 million in ticket sales).*

- *An actor is a movie star if he has starred in more than three successful movies, produced in a span of at least five years.*



# RIF-BLD

- These rules should be evaluated against the DBpedia data set:

Document (

```
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>
Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>
Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>
Prefix(imdbrel <http://example.com/imdbrelation#>
Prefix(dbpedia <http://dbpedia.org/ontology/>
Prefix(ibdbrel http://example.com/ibdbrelation#>
```

Group (


```
Forall ?Actor ?Film ?Year (
  If And( dbpedia:starring(?Film ?Actor)
    dbpedia:dateOfFilm(?Film ?Year) )
  Then dbpedia:starredInYear(?Film ?Actor ?Year)
)
```

# RIF-BLD

```
forall ?Film (  
  If Or (  
    External (pred:numeric-greater-than(  
      dbpedia:criticalRating(?Film) 8)  
    External (pred:numeric-greater-than(  
      dbpedia:boxOfficeGross(?Film) 100000000)))  
  Then dbpedia:successful(?Film)  
)
```

- **External** applies built-in predicates.
- **Group** to put together a number of rules.

# RIF-BLD

A decorative wavy line in orange and purple colors runs vertically along the left side of the slide.

```
Forall ?Actor (  
  If ( Exists ?Film1 ?Film2 ?Film3 ?Year1 ?Year2 ?Year3  
    And ( dbpedia:starredInYear(?Film1 ?Actor ?Year1)  
      dbpedia:starredInYear(?Film2 ?Actor ?Year2)  
      dbpedia:starredInYear(?Film3 ?Actor ?Year3)  
      External ( pred:numeric-greater-than(  
        External(func:numeric-subtract ?Year1 ?Year3) 5)))  
      dbpedia:successful(?Film1)  
      dbpedia:successful(?Film2)  
      dbpedia:successful(?Film3)  
      External (pred:literal-not-identical(?Film1 ?Film2))  
      External (pred:literal-not-identical(?Film1 ?Film3))  
      External (pred:literal-not-identical(?Film2 ?Film3))  
    )  
  Then dbpedia:movieStar(?Actor)  
)
```

# Compatibility with RDF and OWL

- A major feature of RIF is that it is compatible with the RDF and OWL standards
- Represent RDF triples using RIF frame formulas: a triple **(s p o)** is represented as **s [p -> o]**
- That is, one can reason with a combination of RIF, RDF, and OWL documents
- RIF facilitates the interchange of not just rules, but also RDF graphs and/or OWL axioms

# Compatibility with RDF and OWL

- The semantic definitions are such that the triple is satisfied **if and only if** the corresponding RIF frame formula is also satisfied
  - for example, if the RDF triple  
**ex:GoneWithTheWind ex:FilmYear ex:1939**  
is true, then so is the RIF fact  
**ex:GoneWithTheWind[  
    ex:FilmYear -> ex:1939] .**

# Compatibility with RDF and OWL

- Given the RIF rule (which states that the Hollywood Production Code was in place between 1930 and 1968)

```
Group (  
  Forall ?Film (  
    If And( ?Film[ex:Year -> ?Year]  
      External(pred:dateGreaterThan(?Year 1930))  
      External(pred:dateGreaterThan(1968 ?Year)))  
    Then ?Film[ex:HollywoodProductionCode -> ex:True]))
```

one can conclude

```
ex:GoneWithTheWind [  
  ex:HollywoodProductionCode -> ex:True] .
```

as well as the corresponding RDF triple

# Compatibility with RDF and OWL

- Similar techniques are used to achieve compatibility between OWL and RIF:
  - The semantics of OWL and RIF are compatible
  - One can infer conclusions from certain combinations of OWL axioms and RIF knowledge
  - OWL2 RL can be implemented in RIF

# OWL2 RL in RIF

- OWL2 RL is partially described by a set of first-order rules that can form the basis for an implementation using rule technology
  - To enable interoperability between rule systems and OWL2 RL ontologies, this axiomatization can be described using RIF (BLD, actually even in the simpler Core) rules
  - The OWL2 RL rules can be categorized in four (non-disjoint) categories: *triple pattern rules*, *inconsistency rules*, *list rules*, and *datatype rules*



# OWL2 RL in RIF

- *Triple Pattern Rules*: derive RDF triples from a conjunction of RDF triple patterns

**Group (**

**Forall ?V1 . . . ?Vn (**

**s[p->o] :-**

**And(s1[p1->o1] . . . sn[pn->on] ) )**

**)**

# OWL2 RL in RIF

- *Inconsistency Rules*: indicate inconsistencies in the original RDF graph (w.r.t. the existing OWL knowledge)
  - represented in RIF as rules with the conclusion **rif:error**, a predicate symbol within the RIF namespace that can be used to express inconsistency
    - Example: an inconsistency occurs when two predicates have been declared to be disjoint, but connect the same entities

Group (

Forall ?P1 ?P2 ?X ?Y (

**rif:error** :- And (

**?P1 [owl:propertyDisjointWith ?P2]**

**?X [?P1->?Y]**

**?X [?P2->?Y] ) ) )**

# OWL2 RL in RIF

- *List Rules:*

- A number of OWL2 RL rules involve processing OWL expressions that include RDF lists (for example

**owl:AllDifferent**)

```
Forall ?x ?y ?z1 ?z2 ?iz1 ?iz2 (  
  rif:error() :- And (  
    ?x[rdf:type -> owl:AllDifferent]  
    ?x[owl:members -> ?y]  
    External(pred:list-contains(?y ?z1))  
    ?iz1 = External(func:index-of(?y ?z1))  
    External(pred:list-contains(?y ?z2))  
    ?iz2 = External(func:index-of(?y ?z2))  
    External( pred:numeric-not-equal(?iz1 ?iz2))  
    ?z1[owl:sameAs->?z2] ) )
```

# OWL2 RL in RIF

- ***Datatype Rules***: provide type checking and value equality/inequality checking for typed literals in the supported datatypes
  - For example, generate an inconsistency if a literal is specified to be an instance of a data type but its **value is outside the value space of that data type**

```
Forall ?lt (
  rif:error() :- And (
    ?lt[rdf:type->xsd:decimal]
    External (
      pred:is-literal-not-decimal( ?lt )) ) )
```

# Lecture Outline

- Monotonic Rules
- OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)

# Semantic Web Rules Language (SWRL)

- SWRL is a proposed Semantic Web language combining OWL DL with function-free Horn logic and is written in Unary/Binary Datalog RuleML
- it allows Horn-like rules to be combined with OWL DL ontologies
- A rule in SWRL has the form:

$$\mathbf{B}_1, \dots, \mathbf{B}_n \rightarrow \mathbf{A}_1, \dots, \mathbf{A}_m$$

where the commas denote conjunction on both sides of the arrow

# Semantic Web Rules Language (SWRL)

- $\mathbf{B}_1, \dots, \mathbf{B}_n, \mathbf{A}_1, \dots, \mathbf{A}_m$  can be of the forms:
  - $\mathbf{C}(\mathbf{x})$ ,
  - $\mathbf{P}(\mathbf{x}, \mathbf{y})$ ,
  - $\mathbf{sameAs}(\mathbf{x}, \mathbf{y})$ , or
  - $\mathbf{differentFrom}(\mathbf{x}, \mathbf{y})$ ,

where

$\mathbf{C}$  is an OWL class,

$\mathbf{P}$  is an OWL property, and

$\mathbf{x}, \mathbf{y}$  are Datalog variables, OWL individuals, or OWL data values.

# Semantic Web Rules Language (SWRL)

- The main complexity of the SWRL language stems from the fact that arbitrary OWL expressions, such as restrictions, can appear in the head or body of a rule
- adds significant expressive power to OWL, but at the high price of undecidability
- a sublanguage is the extension of OWL DL with ***DL-safe*** rules, in which every variable must appear in a non-description logic atom in the rule body



# Lecture Outline

- Monotonic Rules
- ✓ • OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- ✓ • Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)

# Rules in SPARQL: SPIN

- Rules can be expressed in SPARQL using CONSTRUCT:

**grandparent(X, Z) ←  
parent(Y, Z), parent(X, Y) .**

can be expressed as:

```
CONSTRUCT {  
    ?X grandParent ?Z.  
} WHERE {  
    ?Y parent ?Z.  
    ?X parent ?Y.  
}
```

# Rules in SPARQL: SPIN

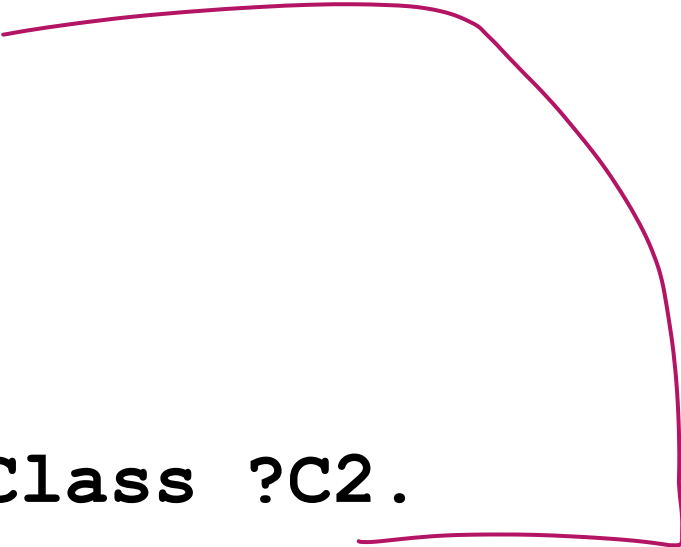
- SPIN provides abstraction mechanisms for rules using *Templates*, which encapsulate parameterized SPARQL queries; and user-defined SPIN functions as a mechanism to build higher-level rules (complex SPARQL queries) on top of simpler building blocks.

# Rules in SPARQL: SPIN

**C2 (X) ← C1 (X) , equivalentClass (C1 , C2) .**

can be represented in SPARQL as:

```
CONSTRUCT {  
  ?X a ?C2 .  
}  
WHERE {  
  ?X a ?C1 .  
  ?C1 equivalentClass ?C2 .  
}
```



and then instantiated as a **spin:rule** for the class **owl:Thing** to allow the rule to be applied to all possible instances.

# Lecture Outline

- Monotonic Rules
- OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)

# Nonmonotonic Rules: Motivation and Syntax

- In nonmonotonic rule systems, a rule may not be applied even if all premises are known because we have to consider contrary reasoning chains
  - the rules we consider from now on are called *defeasible* because they can be defeated by other rules
  - negated atomic formulas may occur in the head and the body of rules

$p(X) \Rightarrow q(X) .$

$r(X) \Rightarrow \neg q(X) .$

given also the facts

$p(a) .$

$r(a) .$

we can conclude both  $q(a)$  and  $\neg q(a)$  (impossible)

# Nonmonotonic Rules: Motivation and Syntax

- Conflicts may be resolved using **priorities** among rules
  - Suppose we knew somehow that the first rule is stronger than the second; then we could derive only  **$q(a)$** .
  - **Priorities arise naturally in practice:**
    - The source of one rule may be more **reliable** than the source of the second rule, or it may have higher authority
      - For example, federal law preempts state law
      - And in business administration, higher management has more authority than middle management
    - One rule may be preferred over another because it is more **recent**
    - One rule may be preferred over another because it is more **specific**
      - A typical example is a general rule with some exceptions; in such cases, the exceptions are stronger than the general rule
        - A classical example is that, in general, birds fly, however penguins are birds that do not fly

# Nonmonotonic Rules: Motivation and Syntax

- Extend the rule syntax to include a unique label:

**$r1: p(X) \Rightarrow q(X) .$**

**$r2: r(X) \Rightarrow \neg q(X) .$**

**$r1 > r2 .$**

given the facts

**$p(a) .$**

**$r(a) .$**

we can conclude  **$q(a) .$**

- We can require the priority relation to be acyclic: it is impossible to have cycles of the form

**$r1 > r2 > \dots > r_n > r1$**



# Nonmonotonic Rules: Motivation and Syntax

- Priorities are meant to resolve conflicts among *competing rules*: two rules are **competing** only if the head of one rule is the negation of the head of the other
- In applications it is often the case that once a predicate **p** is derived, some other predicates are excluded from holding
  - For example, an investment consultant may base his recommendations on three levels of risk that investors are willing to take: **low**, **moderate**, and **high**.
    - Only one risk level per investor is allowed to hold at any given time
  - Technically, these situations are modeled by maintaining a *conflict set* **C(L)** for each literal **L**
    - **C(L)** always contains the negation of **L** but may contain more literals

# Nonmonotonic Rules: Motivation and Syntax

- A defeasible rule has the form:

$$\mathbf{r} : \mathbf{L1}, \dots, \mathbf{Ln} \Rightarrow \mathbf{L}$$

where

$\mathbf{r}$  is the label,

$\{\mathbf{L1}, \dots, \mathbf{Ln}\}$  the body (or premises), and

$\mathbf{L}$  the head of the rule.

- $\mathbf{L}, \mathbf{L1}, \dots, \mathbf{Ln}$  are positive or negative literals (a *literal* is an atomic formula  $\mathbf{p}(\mathbf{t1}, \dots, \mathbf{tm})$  or its negation  $\neg \mathbf{p}(\mathbf{t1}, \dots, \mathbf{tm})$ ).
- No function symbols may occur in the rule
- Sometimes we denote the head of a rule as **head**( $\mathbf{r}$ ), and its body as **body**( $\mathbf{r}$ )

# Nonmonotonic Rules: Motivation and Syntax

- We use the label  $\mathbf{r}$  to refer to the whole rule
- A *defeasible logic program* is a triple  $(\mathbf{F}, \mathbf{R}, >)$  consisting of a set  $\mathbf{F}$  of facts, a finite set  $\mathbf{R}$  of defeasible rules, and an acyclic binary relation  $>$  on  $\mathbf{R}$  (precisely, a set of pairs  $\mathbf{r} > \mathbf{r}'$  where  $\mathbf{r}$  and  $\mathbf{r}'$  are labels of rules in  $\mathbf{R}$ )

# Example of Nonmonotonic Rules: Brokered Trade

- Electronic commerce application
  - Brokered trades take place via an independent third party, the broker
  - The broker matches the buyer's requirements and the sellers' capabilities and proposes a transaction in which both parties can be satisfied by the trade
- Concrete application: **apartment renting**  
(common activity that is often tedious and time-consuming)

# Example of Nonmonotonic Rules: Brokered Trade

- *Carlos is looking for an apartment of at least 45 sq m with at least two bedrooms.*
- *If it is on the third floor or higher, the house must have an elevator.*
- *Also, pet animals must be allowed.*
- *Carlos is willing to pay \$300 for a centrally located 45 sq m apartment, and \$250 for a similar apartment in the suburbs.*
- *In addition, he is willing to pay an extra \$5 per square meter for a larger apartment, and \$2 per square meter for a garden.*
- *He is unable to pay more than \$400 in total.*
- ***If given the choice, he would go for the cheapest option.***
- *His second priority is the presence of a garden; his lowest priority is additional space.*

# Formalization of Carlos's Requirements

- Predicates that describe properties of apartments:
  - **apartment**(**x**) stating that: **x** is an apartment
  - **size**(**x**, **y**) : **y** is the size of apartment **x** (in sq m)
  - **bedrooms**(**x**, **y**) : **x** has **y** bedrooms
  - **price**(**x**, **y**) : **y** is the price for **x**
  - **floor**(**x**, **y**) : **x** is on the **y**th floor
  - **garden**(**x**, **y**) : **x** has a garden of size **y**
  - **elevator**(**x**) : there is an elevator in the house of **x**
  - **pets**(**x**) : pets are allowed in **x**
  - **central**(**x**) : **x** is centrally located

# Formalization of Carlos's Requirements

- We also make use of the predicates:
  - **acceptable(x)** : apartment **x** satisfies Carlos's requirements
  - **offer(x, y)** : Carlos is willing to pay \$**y** for flat **x**
- Any apartment is a priori acceptable:

**r1 : apartment(X)  $\Rightarrow$  acceptable(X) .**

- However, apartment **Y** is unacceptable if one of Carlos's requirements is not met:

**r2 : bedrooms(X, Y), Y < 2  $\Rightarrow$   $\neg$ acceptable(X) .**

**r3 : size(X, Y), Y < 45  $\Rightarrow$   $\neg$ acceptable(X) .**

**r4 :  $\neg$ pets(X)  $\Rightarrow$   $\neg$ acceptable(X) .**

**r5 : floor(X, Y), Y > 2,  $\neg$ lif t(X)  $\Rightarrow$   $\neg$ acceptable(X) .**

**r6 : price(X, Y), Y > 400  $\Rightarrow$   $\neg$ acceptable(X) .**

# Formalization of Carlos's Requirements

- Rules **r2-r6** are exceptions to rule **r1**:

**r2** > **r1**.

**r3** > **r1**.

**r4** > **r1**.

**r5** > **r1**.

**r6** > **r1**.



# Formalization of Carlos's Requirements

- We calculate the price Carlos is willing to pay for an apartment:

**r7 : size(X, Y) , Y ≥ 45 , garden(X, Z) ,  
central(X) ⇒**

**offer(X, 300 + 2Z + 5(Y - 45)) .**

**r8 : size(X, Y) , Y ≥ 45 , garden(X, Z) ,  
¬central(X) ⇒**

**offer(X, 250 + 2Z + 5(Y - 45)) .**

- An apartment is only acceptable if the amount Carlos is willing to pay is higher than the price specified by the landlord (we assume no bargaining can take place)

**r9 : offer(X, Y) , price(X, Z) , Y < Z ⇒  
¬acceptable(X) .**

**r9 > r1 .**

# Representation of Available Apartments

- Each available apartment is given a unique name (for example **a1**), and its properties are represented as facts:

**bedrooms (a1, 1) .**

**size (a1, 50) .**

**central (a1) .**

**floor (a1, 1) .**

**¬elevator (a1) .**

**pets (a1) .**

**garden (a1, 0) .**

**price (a1, 300) .**

- In practice, the apartments on offer could be stored in a relational database, CSV file, or an RDF storage system

# Selecting an Apartment

Flat	Bedrooms	Size	Central	Floor	Elevator	Pets	Garden	Price
$a_1$	1	50	yes	1	no	yes	0	300
$a_2$	2	45	yes	0	no	yes	0	335
$a_3$	2	65	no	2	no	yes	0	350
$a_4$	2	55	no	1	yes	no	15	330
$a_5$	3	55	yes	0	no	yes	15	350
$a_6$	2	60	yes	3	no	no	0	370
$a_7$	3	65	yes	1	no	yes	12	375

- If we match Carlos's requirements and the available apartments, we see:
  - flat  $a_1$  is not acceptable because it has one bedroom only (rule r2)
  - flats  $a_4$  and  $a_6$  are unacceptable because pets are not allowed (rule r4)
  - for  $a_2$ , Carlos is willing to pay \$300, but the price is higher (rules r7, r9)
  - **flats  $a_3$ ,  $a_5$ , and  $a_7$  are acceptable (rule r1)**

# Selecting an Apartment

- Carlos's preferences are based on price, garden size, and size, in that order:

**r10** :  $\text{acceptable}(X) \Rightarrow \text{cheapest}(X)$  .

**r11** :  $\text{acceptable}(X), \text{price}(X, Z),$   
 $\text{acceptable}(Y), \text{price}(Y, W), W < Z \Rightarrow$   
 $\neg \text{cheapest}(X)$  .

**r11** > **r10** .

- Rule **r10** says that every acceptable apartment is cheapest by default.
- However, if there is an acceptable apartment cheaper than **X**, rule **r11** (which is stronger than **r10**) fires and concludes that **X** is not cheapest.

# Selecting an Apartment

- Carlos's preferences are based on price, garden size, and size, in that order:

**r12** : **cheapest**(X)  $\Rightarrow$  **largestGarden**(X) .

**r13** : **cheapest**(X) , **gardenSize**(X, Z) ,  
**cheapest**(Y) , **gardenSize**(Y, W) ,  $W > Z \Rightarrow$   
 $\neg$ **largestGarden**(X) .

**r13** > **r12** .

- Rules **r12** and **r13** select the apartments with the largest garden among the cheapest apartments.

# Selecting an Apartment

- Carlos's preferences are based on price, garden size, and size, in that order:

**r14** :  $\text{largestGarden}(X) \Rightarrow \text{rent}(X)$  .

**r15** :  $\text{largestGarden}(X), \text{size}(X, Z),$   
 $\text{largestGarden}(Y), \text{size}(Y, W), W > Z \Rightarrow$   
 $\neg \text{rent}(X)$  .

**r15** > **r14** .

- Rules **r14** and **r15** select the proposed apartments to be rented, based on apartment size.

# Selecting an Apartment

Flat	Bedrooms	Size	Central	Floor	Elevator	Pets	Garden	Price
$a_1$	1	50	yes	1	no	yes	0	300
$a_2$	2	45	yes	0	no	yes	0	335
$a_3$	2	65	no	2	no	yes	0	350
$a_4$	2	55	no	1	yes	no	15	330
$a_5$	3	55	yes	0	no	yes	15	350
$a_6$	2	60	yes	3	no	no	0	370
$a_7$	3	65	yes	1	no	yes	12	375

- Apartments  $a_3$  and  $a_5$  are cheapest
- $a_5$  has the largest garden and will be rented (after is inspected)
  - in this case the apartment size criterion does not play a role: **r14** fires only for  $a_5$ , so rule **r15** is not applicable for it

# Lecture Outline

- Monotonic Rules
- OWL2 RL: Description Logic Meets Rules
- Rule Interchange Format: RIF
- Semantic Web Rules Language (SWRL)
- Rules in SPARQL: SPIN
- Nonmonotonic Rules
  - Example: Brokered Trade
- Rule Markup Language (RuleML)



# Rule Markup Language (RuleML)

- RuleML is a long-running effort to develop markup of rules on the web
- It is actually not one language but a family of rule markup languages, corresponding to different kinds of rule languages: **derivation rules, integrity constraints, reaction rules**
- The kernel of the RuleML family is **Datalog**, which is **function-free Horn logic**
- The RuleML family provides descriptions of rule markup languages in XML

# Rule Markup Language (RuleML)

- Vocabulary of Datalog RuleML:

Rule Ingredient	RuleML
fact	Asserted Atom
rule	Asserted Implies
head	then
body	if
atom	Atom
conjunction	And
predicate	Rel
constant	Ind
variable	Var

# Rule Markup Language (RuleML)

- Example rule: “*The discount for a customer buying a product is 7.5 percent if the customer is premium and the product is luxury*”
  - In RuleML 1.0:

```
<Implies>
  <if>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>luxury</Rel>
        <Var>product</Var>
      </Atom>
    </And>
  </if>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <Var>customer</Var>
      <Var>product</Var>
      <Ind>7.5 percent</Ind>
    </Atom>
  </then>
</Implies>
```

# Rule Markup Language (RuleML)

- SWRL is an extension of RuleML and is represented in RuleML 1.0:  
**brother(X, Y), childOf(Z, Y) → uncle(X, Z) .**

```
<ruleml:Implies>
  <ruleml:then>
    <swrlx:individualPropertyAtom swrlx:property="uncle">
      <ruleml:Var>X</ruleml:Var>
      <ruleml:Var>Z</ruleml:Var>
    </swrlx:individualPropertyAtom>
  </ruleml:then>
  <ruleml:if>
    <ruleml:And>
      <swrlx:individualPropertyAtom swrlx:property="brother">
        <ruleml:Var>X</ruleml:Var>
        <ruleml:Var>Y</ruleml:Var>
      </swrlx:individualPropertyAtom>
      <swrlx:individualPropertyAtom swrlx:property="childOf">
        <ruleml:Var>Z</ruleml:Var>
        <ruleml:Var>Y</ruleml:Var>
      </swrlx:individualPropertyAtom>
    </ruleml:And>
  </ruleml:if>
</ruleml:Implies>
```

# Summary

- Rules on the (semantic) web form a very rich and heterogeneous landscape
- Horn logic is a subset of predicate logic that allows efficient reasoning
  - It forms a subset orthogonal to description logics
  - Horn logic is the basis of monotonic rules
- RIF is a new standard for rules on the web
  - Its logical dialect BLD is based on Horn logic
  - OWL2 RL, which is essentially the intersection of description logics and Horn logic, can be embedded in RIF
- SWRL is a much richer rule language, combining description logic features with restricted types of rules
- Nonmonotonic rules are useful in situations where the available information is incomplete
  - They are rules that may be overridden by contrary evidence (other rules)
  - Priorities are used to resolve some conflicts between nonmonotonic rules