Knowledge Representation

- The subjects presented so far were related to the representation of knowledge
- Knowledge Representation was studied long before the emergence of WWW in AI
- Logic is still the foundation of KR, particularly in the form of predicate logic (first-order logic)

The Importance of Logic

- High-level language for expressing knowledge
- High expressive power
- Well-understood formal semantics
- Precise notion of logical consequence
- 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
- Soundness & completeness

Predicate logic is unique in the sense that sound and complete proof systems do exist.
- Not for more expressive logics (higher-order logics)

trace the proof that leads to a logical consequence.

Logic can provide explanations for answers
- By tracing a proof

RDF/S and OWL (Lite and DL) are specializations of predicate logic
- correspond roughly to a description logic

They define reasonable subsets of logic

Trade-off between the expressive power and the computational complexity:
- The more expressive the language, the less efficient the corresponding proof systems

- A rule has the form: $A_1, \ldots, A_n \rightarrow B$
  - $A_i$ and $B$ are atomic formulas
- There are 2 ways of reading such a rule:
  - Deductive rules: If $A_1, \ldots, A_n$ are known to be true, then $B$ is also true
  - Reactive rules: If the conditions $A_1, \ldots, A_n$ are true, then carry out the action $B$

Neither of them is a subset of the other

It’s impossible to assert that people who study and live in the same city are “home students” in OWL
- This can be done easily using rules:
  \[
  \text{studies}(X,Y), \text{lives}(X,Z), \text{loc}(Y,U), \text{loc}(Z,U) \rightarrow \text{homeStudent}(X)
  \]

Rules cannot assert the information that a person is either a man or a woman
- This information is easily expressed in OWL using disjoint union
Monotonic vs. Non-monotonic Rules

- **Example:** An online vendor wants to give a special discount if it is a customer’s birthday

**Solution 1**

- R1: If birthday, then special discount
- R2: If not birthday, then not special discount

  - But what happens if a customer refuses to provide his birthday due to privacy concerns?

Monotonic vs. Non-monotonic Rules (2)

**Solution 2**

- R1: If birthday, then special discount
- R2': If birthday is not known, then not special discount

  - Solves the problem but:
    - The premise of rule R2' is not within the expressive power of predicate logic
    - We need a new kind of rule system

Monotonic vs. Non-monotonic Rules (3)

- The solution with rules R1 and R2 works in case we have complete information about the situation
- The new kind of rule system will find application in cases where the available information is incomplete
- R2' is a nonmonotonic rule

Exchange of Rules

- Exchange of rules across different applications
  - E.g., an online store advertises its pricing, refund, and privacy policies, expressed using rules
- The Semantic Web approach is to express the knowledge in a machine-accessible way using one of the Web languages we have already discussed
- We show how rules can be expressed in XML-like languages (“rule markup languages”)
Lecture Outline

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules

Family Relations

- Facts in a database about relations:
  - 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
- Inferred relation parent: A parent is either a father or a mother
  - mother(X,Y) → parent(X,Y)
  - father(X,Y) → parent(X,Y)

Inferred Relations

- male(X), parent(P,X), parent(P,Y), notSame(X,Y) → brother(X,Y)
- female(X), parent(P,X), parent(P,Y), notSame(X,Y) → sister(X,Y)
- brother(X,P), parent(P,Y) → uncle(X,Y)
- mother(X,P), parent(P,Y) → grandmother(X,Y)
- parent(X,Y) → ancestor(X,Y)
- ancestor(X,P), parent(P,Y) → ancestor(X,Y)

Lecture Outline

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules
Monotonic Rules – Syntax

\[ \text{loyalCustomer}(X), \text{age}(X) > 60 \rightarrow \text{discount}(X) \]

- We distinguish some ingredients of rules:
  - variables which are placeholders for values: \( X \)
  - constants denote fixed values: 60
  - Predicates relate objects: \text{loyalCustomer}, >
  - Function symbols which return a value for certain arguments: \text{age}

Rules

\[ B_1, \ldots, B_n \rightarrow A \]

- \( A, B_1, \ldots, B_n \) are atomic formulas
- \( A \) is the head of the rule
- \( B_1, \ldots, B_n \) are the premises (body of the rule)
- The commas in the rule body are read conjunctively
- Variables may occur in \( A, B_1, \ldots, B_n \)
  - \text{loyalCustomer}(X), \text{age}(X) > 60 \rightarrow \text{discount}(X)
  - Implicitly universally quantified

Facts and Logic Programs

- A fact is an atomic formula
- E.g. \text{loyalCustomer}(a345678)
- The variables of a fact are implicitly universally quantified.
- A logic program \( P \) is a finite set of facts and rules.
- Its predicate logic translation \( \text{pl}(P) \) is the set of all predicate logic interpretations of rules and facts in \( P \)

Goals

- A goal denotes a query \( G \) asked to a logic program
- The form: \( B_1, \ldots, B_n \rightarrow \)
- If \( n = 0 \) we have the empty goal \( \square \)
First-Order Interpretation of Goals

- ∀X₁ . . . ∀Xₖ (¬B₁ ∨ . . . ∨ ¬Bₙ)
  - Where X₁, . . . , Xₖ are all variables occurring in B₁, . . . , Bₙ
  - Same as pl(r), with the rule head omitted
- Equivalently: ¬∃X₁ . . . ∃Xₖ (B₁ ∧ . . . ∧ Bₙ)
  - Suppose we know p(a) and we have the goal p(X) →
  - We want to know if 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 Negate the Formula?

- We use a proof technique from mathematics called proof by contradiction:
  - Prove that A follows from B by assuming that A is false and deriving a contradiction, when combined with B
- 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
  - E.g., given the following logic program we get a logical contradiction

An Example

p(a)
¬∃X p(X)

- The 2nd formula says that no element has the property p
- The 1st formula says that the value of a does have the property p
- Thus ∃X p(X) follows from p(a)

Monotonic Rules – Predicate Logic Semantics

- Given a logic program P and a query B₁, . . . , Bₙ →
- with the variables X₁, . . . , Xₖ we answer positively if, and only if,
  pl(P) |= ∃X₁ . . . ∃Xₖ(B₁ ∧ . . . ∧ Bₙ) (1)
- or equivalently, if
  pl(P) ∪ {¬∃X₁ . . . ∃Xₖ (B₁ ∧ . . . ∧ Bₙ)} is unsatisfiable (2)
The Semantics of Predicate Logic

- The components of the logical language (signature) may have any meaning we like
  - A predicate logic model \( A \) assigns a certain meaning
- 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

The Semantics of Predicate Logic (2)

- The meanings of the logical connectives \( \neg, \lor, \land, \to, \forall, \exists \) are defined according to their intuitive meaning:
  - not, or, and, implies, for all, there is
- We define when a formula is true in a model \( A \), denoted as \( A \models \phi \)
- A formula \( \phi \) follows from a set \( M \) of formulas if \( \phi \) is true in all models \( A \) in which \( M \) is true

Motivation of First-Order Interpretation of Goals

\[
\begin{align*}
p(a) \\
p(X) & \to q(X) \\
q(X) & \to \\
q(a) & \text{follows from } p(a) \\
\exists X \ q(X) & \text{follows from } p(a),
\end{align*}
\]

- We must give a negative answer because \( q(b) \) does not follow from \( p(a) \)
Ground Witnesses

- So far we have focused on yes/no answers to queries
- Suppose that we have the fact \( p(a) \) and the query \( p(X) \rightarrow \)
  - The answer yes is correct but not satisfactory
- The appropriate answer is a substitution \( \{X/a\} \) which gives an instantiation for \( X \)
- The constant \( a \) is called a ground witness

Parameterized Witnesses

\[
\begin{align*}
\text{add}(X, 0, X) \\
\text{add}(X, Y, Z) & \rightarrow \text{add}(X, s(Y), s(Z)) \\
\text{add}(X, s(0), Z) & \rightarrow
\end{align*}
\]

- Possible ground witnesses:
  - \( \{X/0, Z/s(0)\}, \{X/s(0), Z/s(0)\} \ldots \)
- The parameterized witness \( Z = s^8(X) \) is the most general answer to the query:
  - \( \exists X \exists Z \text{add}(X, s^8(0), Z) \)
- The computation of most general witnesses is the primary aim of SLD resolution

Lecture Outline

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules

Motivation – Negation in Rule Head

- In nonmonotonic rule systems, a rule may not be applied even if all premises are known because we have to consider contrary reasoning chains
- Now we consider defeasible rules that can be defeated by other rules
- Negated atoms may occur in the head and the body of rules, to allow for conflicts
  - \( p(X) \rightarrow q(X) \)
  - \( r(X) \rightarrow \neg q(X) \)
Defeasible Rules

\[ p(X) \Rightarrow q(X) \]
\[ r(X) \Rightarrow \neg q(X) \]

- Given also the facts \( p(a) \) and \( r(a) \) we conclude neither \( q(a) \) nor \( \neg q(a) \)
  - This is a typical example of 2 rules blocking each other
- Conflict may be resolved using priorities among rules
- Suppose we knew somehow that the 1st rule is stronger than the 2nd
  - Then we could derive \( q(a) \)

Origin of Rule Priorities

- Higher authority
  - E.g., in law, federal law pre-empts state law
  - E.g., in business administration, higher management has more authority than middle management
- Recency
- Specificity
  - A typical example is a general rule with some exceptions
- We abstract from the specific prioritization principle
  - We assume the existence of an external priority relation on the set of rules

Rule Priorities

\[ r1: p(X) \Rightarrow q(X) \]
\[ r2: r(X) \Rightarrow \neg q(X) \]
\[ r1 > r2 \]

- Rules have a unique label
- The priority relation to be acyclic

Competing Rules

- In simple cases two rules are competing only if one head is the negation of the other
- But in many cases once a predicate \( p \) is derived, some other predicates are excluded from holding
  - E.g., an investment consultant may base his recommendations on three levels of risk investors are willing to take: low, moderate, and high
  - Only one risk level per investor is allowed to hold
Competing Rules (2)

- These situations are modelled by maintaining a conflict set \( C(L) \) for each literal \( L \)
- \( C(L) \) always contains the negation of \( L \) but may contain more literals

Defeasible Rules: Syntax

\[ r : L_1, ..., L_n \Rightarrow L \]

- \( r \) is the label
- \( \{L_1, ..., L_n\} \) the body (or premises)
- \( L \) the head of the rule
- \( L, L_1, ..., L_n \) are positive or negative literals
- A literal is an atomic formula \( p(t_1, ..., t_m) \) or its negation \( \neg p(t_1, ..., t_m) \)
- No function symbols may occur in the rule

Defeasible Logic Programs

- A defeasible logic program is a triple \( (F, R, >) \) consisting of
  - a set \( F \) of facts
  - a finite set \( R \) of defeasible rules
  - an acyclic binary relation \( > \) on \( R \)
    - A set of pairs \( r > r' \) where \( r \) and \( r' \) are labels of rules in \( R \)

Lecture Outline

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules
Brokered Trade

- 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 when both parties can be satisfied by the trade
- The application is apartment renting an activity that is common and often tedious and time-consuming

The Potential Buyer's Requirements

- At least 45 sq m with at least 2 bedrooms
- Elevator if on 3rd floor or higher
- Pets must be allowed
- Carlos is willing to pay:
  - $ 300 for a centrally located 45 sq m apartment
  - $ 250 for a similar flat in the suburbs
  - An extra $ 5 per square meter for a larger apartment
  - An extra $ 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 Used

- 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
- gardenSize(x,y), x has a garden of size y
- lift(x), there is an elevator in the house of x
- pets(x), pets are allowed in x
- central(x), x is centrally located
- acceptable(x), flat x satisfies Carlos’s requirements
- offer(x,y), Carlos is willing to pay $ y for flat x

Formalization of Carlos’s Requirements – Rules

r1: \( \Rightarrow \) acceptable(X)
r2: \( \text{bedrooms}(X,Y), Y < 2 \Rightarrow \neg\text{acceptable}(X) \)
r3: \( \text{size}(X,Y), Y < 45 \Rightarrow \neg\text{acceptable}(X) \)
r4: \( \neg\text{pets}(X) \Rightarrow \neg\text{acceptable}(X) \)
r5: \( \text{floor}(X,Y), Y > 2,\neg\text{lift}(X) \Rightarrow \neg\text{acceptable}(X) \)
r6: \( \text{price}(X,Y), Y > 400 \Rightarrow \neg\text{acceptable}(X) \)
r2 > r1, r3 > r1, r4 > r1, r5 > r1, r6 > r1
**Formalization of Carlos's Requirements – Rules**

- **r7**: \( \text{size}(X,Y), Y \geq 45, \text{garden}(X,Z), \text{central}(X) \Rightarrow \text{offer}(X, 300 + 2*Z + 5*(Y - 45)) \)
- **r8**: \( \text{size}(X,Y), Y \geq 45, \text{garden}(X,Z), \neg\text{central}(X) \Rightarrow \text{offer}(X, 250 + 2*Z + 5(Y - 45)) \)
- **r9**: \( \text{offer}(X,Y), \text{price}(X,Z), Y < Z \Rightarrow \neg\text{acceptable}(X) \)

**Representation of Available Apartments**

<table>
<thead>
<tr>
<th>Flat</th>
<th>Bedrooms</th>
<th>Size</th>
<th>Central</th>
<th>Floor</th>
<th>Lift</th>
<th>Pets</th>
<th>Garden</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>1</td>
<td>50</td>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>yes</td>
<td>0</td>
<td>300</td>
</tr>
<tr>
<td>a2</td>
<td>2</td>
<td>45</td>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>0</td>
<td>335</td>
</tr>
<tr>
<td>a3</td>
<td>2</td>
<td>65</td>
<td>no</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>0</td>
<td>350</td>
</tr>
<tr>
<td>a4</td>
<td>2</td>
<td>55</td>
<td>no</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>15</td>
<td>330</td>
</tr>
<tr>
<td>a5</td>
<td>3</td>
<td>55</td>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>15</td>
<td>350</td>
</tr>
<tr>
<td>a6</td>
<td>2</td>
<td>60</td>
<td>yes</td>
<td>3</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>370</td>
</tr>
<tr>
<td>a7</td>
<td>3</td>
<td>65</td>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>yes</td>
<td>12</td>
<td>375</td>
</tr>
</tbody>
</table>

**Determining Acceptable Apartments**

- If we match Carlos's requirements and the available apartments, we see that
- **flat a1** is not acceptable because it has one bedroom only (rule r2)
- flats **a4** and **a6** are unacceptable because pets are not allowed (rule r4)
- for **a2**, Carlos is willing to pay $300, but the price is higher (rules r7 and r9)
- flats **a3**, **a5**, and **a7** are acceptable (rule r1)
Selecting an Apartment

r10: cheapest(X) ⇒ rent(X)
r11: cheapest(X), largestGarden(X) ⇒ rent(X)
r12: cheapest(X), largestGarden(X), largest(X) ⇒ rent(X)
r12 > r10, r12 > r11, r11 > r10

- We must specify that at most one apartment can be rented, using conflict sets:
  - $C(\text{rent}(x)) = \{\neg \text{rent}(x)\} \cup \{\text{rent}(y) \mid y \neq x\}$

Lecture Outline

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules

Atomic Formulas

- $p(X, a, f(b, Y ))$

```
<atom>
  <predicate>p</predicate>
  <term><var>X</var></term>
  <term><const>a</const></term>
  <term><function>f</function>
    <term><const>b</const></term>
    <term><var>Y</var></term>
  </term>
</atom>
```

Facts

```
<fact>
  <atom>
    <predicate>p</predicate>
    <term><const>a</const></term>
  </term>
</fact>
```
Rules

<rule>
  <head>
    <atom>
      <predicate>r</predicate>
      <term><var>X</var></term>
      <term><var>Y</var></term>
    </atom>
  </head>
</rule>

Rule Markup in XML: A DTD

<!ELEMENT program ((rule|fact)*)>
<!ELEMENT fact (atom)>
<!ELEMENT rule (head,body)>
<!ELEMENT head (atom)>
<!ELEMENT body (atom*)>
<!ELEMENT atom (predicate,term*)>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT function (#PCDATA)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT const (#PCDATA)>
<!ELEMENT query (atom*)>

Rules (2)

<body>
  <atom><predicate>p</predicate>
  <term><var>X</var></term>
  <term> <const>a</const> </term>
</atom>
<atom><predicate>q</predicate>
  <term> <var>Y</var></term>
  <term> <const>b</const></term>
</atom>
</body>

The Alternative Data Model of RuleML

- RuleML is an important standardization effort in the area of rules
- RuleML is at present based on XML but uses RDF-like “role tags,” the position of which in an expression is irrelevant
  - although they are different under the XML data model, in which the order is important
**Our DTD vs. RuleML**

<table>
<thead>
<tr>
<th>program</th>
<th>rulebase</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>imp</td>
</tr>
<tr>
<td>head</td>
<td>_head</td>
</tr>
<tr>
<td>body</td>
<td>_body</td>
</tr>
<tr>
<td>atom*</td>
<td>and</td>
</tr>
<tr>
<td>predicate</td>
<td>rel</td>
</tr>
<tr>
<td>const</td>
<td>ind</td>
</tr>
<tr>
<td>var</td>
<td>var</td>
</tr>
</tbody>
</table>

**Lecture Outline**

1. Introduction
2. Monotonic Rules: Example
3. Monotonic Rules: Syntax & Semantics
4. Nonmonotonic Rules: Syntax
5. Nonmonotonic Rules: Example
6. A DTD For Monotonic Rules
7. A DTD For Nonmonotonic Rules

**Changes w.r.t. Previous DTD**

- There are no function symbols
  - The term structure is flat
- Negated atoms may occur in the head and the body of a rule
- Each rule has a label
- Apart from rules and facts, a program also contains priority statements
  - We use a `<strong>` tag to represent priorities, and an ID label in rules to denote their name

**An Example**

r1: p(X) ⇒ s(X)
r2: q(X) ⇒ ¬s(X)
p(a)
q(a)
r1 > r2
**Rule r1 in XML**

```
<rule id="r1">
  <head>
    <atom>
      <predicate>s</predicate>
      <term><var>X</var></term>
    </atom>
  </head>
  <body>
    <atom>
      <predicate>p</predicate>
      <term><var>X</var></term>
    </atom>
  </body>
</rule>
```

**Fact and Priority in XML**

```
<fact>
  <atom>
    <predicate>p</predicate>
    <term><const>a</const></term>
  </atom>
</fact>

<stronger superior="r1" inferior="r2"/>
```

**A DTD**

```
<!ELEMENT program ((rule|fact|stronger)*)>
<!ELEMENT fact (atom|neg)>
<!ELEMENT neg (atom)>
<!ELEMENT rule (head,body)>
<!ATTLIST rule id ID #IMPLIED>
<!ELEMENT head (atom|neg)>
<!ELEMENT body ((atom|neg)*)>
```

**A DTD (2)**

```
<!ELEMENT atom (predicate,(var|const)*)>
<!ELEMENT stronger EMPTY>
<!ATTLIST stronger
  superior IDREF #REQUIRED>
  inferior IDREF #REQUIRED>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT const (#PCDATA)>
<!ELEMENT query (atom*)>
```
Summary

- Horn logic is a subset of predicate logic that allows efficient reasoning, orthogonal to description logics.
- Horn logic is the basis of monotonic rules.
- Nonmonotonic rules are useful in situations where the available information is incomplete.
- They are rules that may be overridden by contrary evidence.
- Priorities are used to resolve some conflicts between rules.
- Representation XML-like languages is straightforward.