Chapter 5

Logic and Inference: Rules

Based on slides from Grigoris Antoniou and Frank van Harmelen

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

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

The Importance of Logic (2)

- 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

Specializations of Predicate Logic: RDF and OWL

- 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

Specializations of Predicate Logic: Horn Logic

- A rule has the form: **A1, . . ., An → B**
  - **Ai** and **B** are atomic formulas
- There are 2 ways of reading such a rule:
  - Deductive rules*:* If **A1,..., An** are known to be true, then **B** is also true
  - Reactive rules: If the conditions **A1,..., An** are true, then carry out the action **B**

Description Logics vs. Horn Logic

- 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:

    studies(X,Y), lives(X,Z), loc(Y,U), loc(Z,U) → 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

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

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

Rules

B1, . . . , Bn → A

- **A, B1, ... , Bn** are atomic formulas
- **A** is the head of the rule
- **B1, ... , Bn** are the premises (body of the rule)
- The commas in the rule body are read conjunctively
- Variables may occur in **A, B1, ... , Bn**
  - **loyalCustomer(X), age(X) > 60 → discount(X)**
  - Implicitly universally quantified

Facts and Logic Programs

- A fact is an atomic formula
- E.g. **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 **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: **B1, . . . , Bn →**
- If n = 0 we have the empty goal □

First-Order Interpretation of Goals

- $\forall$**X1 . . . $\forall$Xk (¬B1 $\vee$ . . . $\vee$ ¬Bn)**
  - Where **X1, ... , Xk** are all variables occurring in **B1, ..., Bn**
  - Same as **pl(r)**, with the rule head omitted
- Equivalently: **¬$\exists$X1 . . . $\exists$Xk (B1 $\wedge$ . . . $\wedge$ Bn)**
  - Suppose we know **p(a)** and we have the goal **p(X)** $\rightarrow$
  - 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)

¬$\exists$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 $\exists$**X p(X)** follows from **p(a)**

Monotonic Rules – Predicate Logic Semantics

- Given a logic program P and a query

  B1, . . . , Bn $\rightarrow$
- with the variables **X1, ... , Xk** we answer positively if, and only if,

  pl(P) |= $\exists$X1 . . . $\exists$Xk(B1 $\wedge$ ... $\wedge$ Bn) (1)
- or equivalently, if

  pl(P) $\cup$ **{¬$\exists$X1 . . . $\exists$Xk (B1 $\wedge$ ... $\wedge$ 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 consists of:
  - a domain 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 dom(A) for every function symbol
  - a concrete relation on dom(A) for every predicate

The Semantics of Predicate Logic (2)

- The meanings of the logical connectives $\neg, \vee, \wedge, \rightarrow, \forall, \exists$ 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 $\mathbf{A} \models \varphi$
- A formula φ follows from a set M of formulas if φ is true in all models A in which M is true

Motivation of First-Order Interpretation of Goals

p(a)
p(X) → q(X)
q(X) →

- **q(a)** follows from **pl(P)**
- **∃X q(X)** follows from **pl(P)**,
- Thus, **pl(P)∪{¬∃ Xq(X)}** is unsatisfiable, and we give a positive answer

Motivation of First-Order Interpretation of Goals

p(a)
p(X) → q(X)
q(b) →

- We must give a negative answer because **q(b)** does not follow from **pl(P)**

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)** →
  - 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

add(X,0,X)
add(X,Y,Z) → add(X,s(Y),s(Z))
add(X, s^8(0),Z) →

- Possible ground witnesses:
  - **{X/0,Z/s^8(0)}, {X/s(0),Z/s^9(0)}** . . .
- The parameterized witness **Z = s^8(X)** is the most general answer to the query:
  - **∃X ∃Z add(X,s^8(0),Z)**
- The computation of most general witnesses is the primary aim of SLD resolution

Lecture Outline

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) → q(X)**
  - **r(X) → ¬q(X)**

Defeasible Rules

p(X) ⇒ q(X)
r(X) ⇒ ¬q(X)

- Given also the facts **p(a)** and **r(a)** we conclude neither **q(a)** nor **¬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) ⇒ q(X)
r2: r(X) ⇒ ¬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 : L1, ..., Ln ⇒ L

- **r** is the label
- {**L1, ..., Ln**} the body (or premises)
- **L** the head of the rule
- **L, L1, ..., Ln** are positive or negative literals
- A literal is an atomic formula **p(t1,...,tm)** or its negation **¬p(t1,...,tm)**
- 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: ⇒ acceptable(X)

r2: bedrooms(X,Y), Y < 2 ⇒ ¬acceptable(X)

r3: size(X,Y), Y < 45 ⇒ ¬acceptable(X)

r4: ¬pets(X) ⇒ ¬acceptable(X)

r5: floor(X,Y), Y > 2,¬lift(X) ⇒ ¬acceptable(X)

r6: price(X,Y), Y > 400 ⇒ ¬acceptable(X)

r2 > r1, r3 > r1, r4 > r1, r5 > r1, r6 > r1

Formalization of Carlos's Requirements – Rules

r7: size(X,Y), Y ≥ 45, garden(X,Z), central(X) ⇒ offer(X, 300 + 2*Z + 5*(Y − 45))

r8: size(X,Y), Y ≥ 45, garden(X,Z), ¬central(X) ⇒ offer(X, 250 + 2*Z + 5(Y − 45))

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

r9 > r1

Representation of Available Apartments

bedrooms(a1,1)
size(a1,50)
central(a1)
floor(a1,1)
¬lift(a1)
pets(a1)
garden(a1,0)
price(a1,300)

Representation of Available Apartments

Flat	Bedrooms	Size	Central	Floor	Lift	Pets	Garden	Price
a1	1	50	yes	1	no	yes	0	300
a2	2	45	yes	0	no	yes	0	335
a3	2	65	no	2	no	yes	0	350
a4	2	55	no	1	yes	no	15	330
a5	3	55	yes	0	no	yes	15	350
a6	2	60	yes	3	no	no	0	370
a7	3	65	yes	1	no	yes	12	375

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(rent(x)) = {¬rent(x)} ∪ {rent(y) | y ≠ 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>
  </atom>
</fact>
```

Rules

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

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>
</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 term (const|var|(function,term*))>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT function (#PCDATA)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT const (#PCDATA)>
<!ELEMENT query (atom*))>
```

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

program	rulebase
rule	imp
head	_head
body	_body
atom*	and
predicate	rel
const	ind
var	var

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 **<stronger>** 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