Chapter 16

Logic Programming

Topics

- Introduction
- Predicate Calculus
- Propositions
- Clausal Form
- Horn Clauses

Logic Programming Paradigm

- AKA Declarative Paradigm
  - The programmer
    - Declares the goal of the computation (specification of results are stated).
    - Does not declare a detailed algorithm by which these goals are to be achieved.
- Application domain
  - Database design
  - Natural language processing
  - Artificial Intelligence
    - Automatic theorem proving
- Example language: Prolog

Logic Programming

- Instead of providing implementation, execute specification.
  - Relieves the programmer of specifying the implementation.
  - Express programs in a form of symbolic logic.
- Declarative specification:
  - Given an element x and a list L, to prove that x is in L, proceed as follows:
    - Prove that L is [x].
    - Otherwise, split L into L_1 and L_2 and prove one of the following:
      - x is in L_1 or
      - x is in L_2

Logic Programming

- Less effort to write, but implementation may be very inefficient.
  - Requires that the execution engine be more complex.
  - Use a logical inferencing (INFERENCE ENGINE) process to produce results

Introduction to Predicate Calculus

- Symbolic logic can be used for the basic needs of formal logic:
  - Express propositions
  - Express relationships between propositions
  - Describe how new propositions can be inferred from other propositions
- Particular form of symbolic logic used for logic programming is called *first-order predicate calculus*

Introduction to Predicate Calculus

- **Proposition**: a logical statement that may or may not be true.
  - Consists of objects and relationships of objects to each other.
    - Can either assert truth ("john speaks Russian") or query existing knowledge base ("does john speak Russian").
    - Can contain variables, which can become bound speaks(x,Russian).

Introduction to Predicate Calculus

- Example (English statements – Predicate Calculus)
  - 0 is a natural number
    - `natural(0).`
  - 2 is a natural number
    - `natural(2).`
  - For all x, if x is a natural number, then so is the successor of x.
    - `For all x, natural(x) → natural(successor(x)).`
  - -1 is a natural number
    - `natural(-1).`

Introduction to Predicate Calculus

- First and third logical statements are axioms for the natural numbers.
  - Statements that are assumed to be true and from which all true statements about natural numbers can be proved.
- Second logical statement can be proved from the previous axioms.
  - `2 = successor(successor(0)).`
  - `natural(0) → natural(successor(successor(0))).`
- Fourth logical statement cannot be proved from the axioms and so can be assumed to be false.

Predicate Calculus: statements

- Predicate calculus classifies the different parts of statements as:
  1. ***Constants***. These are usually number or names. Sometimes they are called *atoms*, since they cannot be broken down into subparts.
     - Example: `1, 0, true, false`
  2. **Predicates**. These are names for functions that are true or false, like Boolean functions in a program.
     - Can take any number of arguments.
     - Example: the predicate `natural` takes one argument.

Predicate Calculus: statements

3. ***Functions***. Predicate calculus distinguishes between functions that are true or false – these are predicates – and all other functions, which represent non-Boolean values.
   - Example: `successor`
4. **Variables**. Variables stand for as yet unspecified quantities.
   - Example: x
5. **Connectives**. These include the operations `and`, `or`, and `not`, just like the operations on Boolean data in programming languages. Additional connectives are implication and equivalence

Predicate Calculus: table of connectives

Name	**Symbol**	**Example**	**Meaning**	**Notes**
Negation	¬	¬a	not a	*True* if a is false; otherwise *false*
Conjunction	∩	a ∩ b	a and b	*True* if a and b are both true
Disjunction	∪	a ∪ b	a or b	*True* if either a or b (or both) is true
Equivalence	≡	a ≡ b	a is equivalent to b	*True* if a and b are both true or both false
Implication	⊃ ⊂	a ⊃ b a ⊂ b	a implies b b implies a	Logically equivalent to ¬a∪b

Predicate Calculus: connectives

- By convention, negation has highest precedence. Conjunctions, disjunctions, and equivalence have higher precedence than implication (in that order).
  - Example: $p \cup q \cap r \supset \neg s \cup t$ is equivalent to
    $((p \cup (q \cap r)) \supset ((\neg s) \cup t))$

Predicate Calculus: quantifiers

6. ***Quantifiers***. These are operations that introduce variables.
   - **Universal Quantifier:** `"for all"`
   - **Existential Quantifier**: `"there exists"`
   - A variable introduced by a quantifier is said to be **bound** by the quantifier.
   - It is possible for variables also to be **free** (not bound by any quantifier).
   - Quantifiers have higher precedence than any of the operators.

Predicate Calculus: table of quantifiers

Name	Symbol	Example	Meaning
Universal	∀	∀ X P	For all X, P is true
Existential	∃	∃ X P	There exists a value of X such that P is true

Predicate Calculus: quantifiers

- Examples:
  - `∀x(speaks(x,Russian))`
    - True if everyone on the planet speaks Russian; false otherwise.
  - `∃x(speaks(x,Russian))`
    - True if at least one person on the planet speaks Russian; false otherwise.
  - `∀x ∃y(speaks(x,y))`
    - True if every person x speaks some language y; false otherwise.
  - `∃ x ∀ y(speaks(x,y))`
    - True if at least one person on the planet speaks every language y; false otherwise.

Predicate Calculus: statements

7. ***Punctuation Symbols***. These include left and right parentheses the coma, and the period. Parentheses can be left out based on common conventions about the precedence of connectives.

- Arguments to predicates and functions can only be terms, that is, combinations of variables, constants, and functions. Terms cannot contain predicates, quantifiers, or connectives.

Predicate Calculus: examples

- `prime(n)`
  - *True* if the integer value of n is a prime number.
- `0 ≤ x + y`
  - *True* if the real sum of x and y is nonnegative.
- `speaks(x,y)`
  - *True* if the person x speaks language y.
- `0 ≤ x ∩ + x ≤ 1`
  - *True* if x is between 0 and 1, inclusive.
- `speaks(x,Russian)∩speaks(y,Russian) ⊃ talkswith(x,y)`
  - *True* if the fact that both x and y speak Russian implies that x talks with y
- `∀x(¬literate(x) ⊃ (¬writes(x)∩ ∃y(reads(x,y)∩ book(y))))`
  - *True* if every illiterate person x does not write and has not read any book y.

Predicate Calculus: tautologies

- *Tautologies*: Propositions that are true for all possible values of their variables.
  - Example: q ∪ ¬q
- Predicates that are *true* for some particular assignment of values to their variables are called *satisfiable*.
  - Example: `speaks(x,Russian)`
    - If at least one person in the planet speaks Russian.
- Predicates that are *true* for all possible assignments of values to their variables are *valid*.
  - Example: `even(y) ∪ odd(y)`
    - It is true for all integers

Propositions: Summary

- Objects in propositions are represented by simple terms: either constants or variables
- ***Constant***: a symbol that represents an object
- ***Variable***: a symbol that can represent different objects at different times
  - Different from variables in imperative languages

Propositions: Summary

- Simplest propositions are called *atomic propositions* which consist of compound terms
- A *compound term* is composed of two parts
  - *Functor*: function symbol that names the relationship.
  - Ordered list of *parameters* (tuple)

Propositions: Summary

- Examples:

```
student(jon)
like(seth, OSX)
like(nick, windows)
like(jim, linux)
```

- Propositions can be stated in two forms:
  - *Fact*: proposition is assumed to be true
  - *Query*: truth of proposition is to be determined
- Compound proposition:
  - Have two or more atomic propositions
  - Propositions are connected by operators

Clausal Form

- Problem of predicate calculus:
  - Too many ways to state the same thing
- Solution: use a standard form for propositions
- All propositions can be expressed in ***clausal form***:

$$B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$$

  - means if all the As are *true*, then at least one B is *true*
- Characteristics of clausal form:
  - Existential quantifiers are not required.
  - Universal quantifiers are implicit with use of variables.
  - No operator other than conjunctions and disjunctions.

Clausal Form

- *Antecedent*: right side of proposition.
- *Consequent*: left side of the proposition.

$$\underbrace{\texttt{likes(bob,mary)}}_{\text{consequent}} \subset \underbrace{\texttt{likes(bob,redheads)} \cap \texttt{redhead(mary).}}_{\text{antecedent}}$$

- A proposition with zero or one terms in the consequent is called a *Horn clause*.

Horn Clauses

- A Horn clause has a head h, which is a predicate, and a body, which is a list of predicates $p_1, p_2, \ldots p_n$

$$\underbrace{p_1, p_2, \ldots p_n}_{\text{body}} \rightarrow \underbrace{h}_{\text{head}}$$

  - In a Horn clause the head is true if every predicate of the body is true (simultaneously).

Horn Clauses: facts and queries

- *Fact*: a Horn clause without body.
  - They are called *headless* Horn clauses.
    - → h or just h
  - It means that h is always *true*.
    - Example: → `mammal(human).`
- *Query:* a Horn clause without a head.
  - The "opposite" of a fact.
    - Example: `mammal(human)` →

From Predicates to Horn Clauses

- There is a limited correspondence between Horn clauses and predicates.
  - Horn clauses can be written equivalently as a predicate
    - HC: `snowing(C)` ← `precipitation(C),freezing(C).`
    - PC: `snowing(C)` ⊂ `precipitation(C)`∩`freezing(C).`
  - Not all predicates can be translated into Horn clauses.

Properties of Predicate Logic Expressions

Property	Meaning	
Commutativity	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distributivity	$p \vee q \wedge r \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Idempotence	$p \vee p \equiv p$	$p \wedge p \equiv p$
Identity	$p \vee \neg p \equiv true$	$p \wedge \neg p \equiv false$
deMorgan	$\neg (p \vee q) \equiv \neg p \wedge \neg q$	$\neg (p \wedge q) \equiv \neg p \vee \neg q$
Implication	$p \supset q \equiv \neg p \vee q$	
Quantification	$\neg \forall x\, P(x) \equiv \exists x\, \neg P(x)$	$\neg \exists x\, P(x) \equiv \forall x\, \neg P(x)$

From Predicates to Horn Clauses

- Six-step procedure that will, whenever possible, translate a predicate p into a Horn clause.

1. Eliminate implications from p, using the implication property.
2. Move negation inward in p, using the deMorgan and quantification properties, so that only individual terms are negated.
3. Eliminate existential quantifiers from p, using a technique called *skolemization*. Here, the existentially quantified variable is replaced by a unique constant.
   - For example, the expression $\exists x P(x)$ is replaced by $P(c)$, where c is an arbitrarily chosen constant in the domain of x.

From Predicates to Horn Clauses

4. Move all universal quantifiers to the beginning of p; as long as there are no naming conflicts, this step does not change the meaning of p. Assuming that all variables are universally quantified, we can drop the quantifiers without changing the meaning of the predicates.
5. Use the distributive, associative, and commutative properties to convert p to *conjunctive normal form*. In this form, the conjunction and disjunction operators are nested no more than two level deep, with conjunctions at the highest level.

From Predicates to Horn Clauses

6. Convert the embedded disjunctions to implications, using the implication property. If each of these implications has a single term on its right, then each can be rewritten as a series of Horn clauses equivalent to p.

- Example:

  $\forall x(\neg \texttt{literate}(x) \supset (\neg \texttt{writes}(x) \wedge \neg \exists y(\texttt{reads}(x,y) \wedge \texttt{book}(y))))$

- Example:

  $\forall x(\texttt{literate}(x) \supset \texttt{reads}(x) \vee \texttt{writes}(x))$