CS603 Programming Language Organization

Lecture 23

Spring 2005

Department of Computer Science

Overview

- Questions
- Some final points about ML
- Introduction to Prolog

Welcome to the ML Realm

- Suppose we want to represent all inhabitants of a kingdom by their class:
  - King – simply himself
  - Peer – has a degree, territory and number
    - ex.: "the 7th Earl of Carlisle"
  - Knight or Peasant – has a name

How might we represent this in μ-Scheme?

The Realm in μ-Scheme

- `"king"`
- `("Earl" "Carlisle" 7) ("Duke" "Norfolk" 9)`
- `("Knight" "Gawain") ("Knight" "Galahad")`
- `("Peasant" "Jack Cade") ("Peasant" "Wat Tyler")`
- But this doesn't work in ML!

Why?

More About ML

- Strings are supported with "abc", concatenation is caret, ^
  - example: "abc" ^ "def" yields "abcdef"
- Pattern matching can be done on datatypes
  - ```
    datatype iTree = IntNode of int * iTree * iTree
                   | Empty;
    fun sum (IntNode(val,L,R)) = val + sum L + sum R
      | sum Empty = 0;
    ```
    ```

Programs

- Write a datatype for representing a person
  - ```
    datatype person = King
                    | Peer of ...
                    | Knight of ...
                    | Peasant of ... ;
    ```
    ```

Programs (cont.)

- Write a function `title` that behaves as follows:
  `title King` yields `"His Majesty the King"`
  `title Peer("Earl", "Carlisle",7)` yields `"The Earl of Carlisle"`
  `title (Knight "Gawain")` yields `"Sir Gawain"`
  `title (Peasant "Joel")` yields `"Joel"`
- Write a function `sirs` that returns the name of all Knights in a list of `persons`, as a list of strings:
  `sirs [King, Peasant("Joel"), Knight("Gawain")]` yields `["Gawain"]`

Solution

- ```
  datatype person = King
                  | Peer of string * string * int
                  | Knight of string
                  | Peasant of string;

  fun title King                 = "His Majesty the King"
    | title (Peer(deg,terr,_)) = "The " ^ deg ^ " of " ^ terr
    | title (Knight name)        = "Sir " ^ name
    | title (Peasant name)       = name;

  fun sirs []                    = []
    | sirs ((Knight(s) :: ps)) = s :: (sirs ps)
    | sirs (_::ps)               = sirs ps;
  ```
  ```

ML Resources

- "Programming in Standard ML"
  - http://www-2.cs.cmu.edu/~rwh/smlbook/offline.pdf
- Source code from above book
  - http://www-2.cs.cmu.edu/~rwh/smlbook/examples/
- Moscow ML
  - http://www.dina.dk/~setsoft/mosml.html

Prolog

- PROgramming in LOGic
- Algorithm = Logic + Control
- Logic = relation R(I,O) between input I and output O
- Control = method of searching for O that satisfies R(I,O), given input I
- E.g. Find X and Y such that
  - 3*X+2*Y=1
  - X-Y=4
- E.g. find array B such that
  - elements in B are the same as those in A
  - elements of B are in non-descending order

What is Prolog

- Prolog is a ‘typeless’ language with a very simple syntax.
- Prolog is declarative: you describe the relationship between input and output, not how to construct the output from the input (“specify what you want, not how to compute it”)
- Prolog uses a subset of first-order logic

Classical First-Order Logic

- simplest form of logical statements is an atomic formula. e.g.
  - man(tom)
  - woman(mary)
  - married(tom,mary)
- More complex formulas can be built up using logical connectives: $\wedge, \vee, \neg, \rightarrow, \forall X, \exists X$

Pair Up:
- Define each of these symbols

Examples of First Order Logic

- smart(tom) $\vee$ dumb(tom)
- smart(tom) $\vee$ tall(tom)
- $\neg$ dumb(tom)
- $\exists$ X married(tom,X)
- $\exists$ X loves(tom,X)
- $\exists$ X [married(tom,X) $\wedge$ female(X) $\wedge$ human(X)]
- rich(tom) $\vee \neg$ smart(tom)
- $\exists$ X mother(john,X)
- $\forall$ X $\forall$ Y [mother(john,X) $\wedge$ mother(john,Y) $\rightarrow$ Y=X]

Note: $A \rightarrow B \equiv B \vee \neg A$

Horn Rules

- Logic programming is based on formulas called Horn rules. These have the form

$$\forall x_1,\ldots,x_k\,[A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_j]$$

- Examples:

$\forall$ X,Y[A(X) $\leftarrow$ B(X,Y) $\wedge$ C(Y)]

$\forall$ X[A(X) $\leftarrow$ B(X)]

$\forall$ X[A(X,d) $\leftarrow$ B(X,e)]

A(c,d) $\leftarrow$ B(d,e)

$\forall$ X A(X)

$\forall$ X A(X,d)

A(c,d)

Horn Rules (cont.)

- Note that atomic formulas are also Horn rules, often called facts.
- A set of Horn rules is called a Logic Program.

Logical Inference with Horn Rules

- Logic programming is based on a simple idea: From rules and facts, derive more facts.
- Example 1. Given the facts and rules:

  1. A
  2. B
  3. C
  4. $E \leftarrow A \wedge B$
  5. $F \leftarrow C \wedge E$
  6. $G \leftarrow E \wedge F$

- From 1, derive E; from 2, derive F; from 3, derive G.

Logical Inference

- Example 2: Given these facts:
  - man(plato)
  - man(socrates)
- and this rule:
- $\forall X$ [mortal(X) $\leftarrow$ man(X)]
- derive:
- mortal(plato), mortal(socrates).