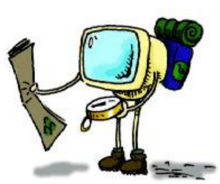
Programming Languages

Tevfik Koşar

Lecture - XIX
March 28th, 2006

Roadmap

- Logic Languages
  - Predicate Calculus
- Prolog Programming Language
  - Basics
  - Queries
  - Resolution
  - Unification
  - Lists
  - Running Prolog Programs



Logic Programming

- Based on predicate calculus
  - (Functional lang. were based on Lambda Calculus)
- Predicates - building-blocks $P(a_1,a_2,...,a_K)$
  - Define relationships between entities
- Eg:
  - enrolled(john, cs4101)
  - taught_by(cs4101, dr_kosar)

3

Logic Programming

- Axioms: define logic rules between predicates
- $H \leftarrow B_1, B_2, B_3, ... B_n$
- Eg:
  - takes_class_from(X,Y) ← enrolled (X,Z), taught_by(Z,Y)

  - enrolled(john, cs4101)
  - taught_by(cs4101, dr_kosar)

  ➔ takes_class_from(john, dr_kosar)

4

Logic Programming

- Eg:
  - green (X) ← rainy (X)

  - rainy (baton_rouge)

  → green (baton_rouge)

5

Prolog

- Atoms (constants):
  - foo   my_Const   +   'Baton Rouge'

- Variables:
  - Foo   My_var   X

  - No variable declarations
  - Variables can be instantiated to arbitrary values at runtime
  - Type checking occurs only when a variable/value is used at runtime
  - The scope of each variable is limited to the clause in which it appears

6

Prolog

- Prolog predicates/facts:
  - rainy(baton_rouge).
  - teaches(dr_kosar, csc4101).

  - Predicates/clauses end with '.'
  - Predicates are called facts, and axioms are called rules

- Prolog axioms/rules:
  - snowy(X) :- rainy(X), cold(X).

  - ':-' is the implication symbol
  - ',' indicates 'and'.

7

Prolog Queries

rainy(seattle).
rainy(newyork).
?- rainy(X).

X = seattle;
X = newyork;
no

8

Prolog Queries

```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).

?- snowy(X).
X = rochester;
no
```


Resolution

- C ← A, B
- D ← C

- Resolution: D ← A, B

- takes (jane, csc4101).
- classmates(X,Y) :- takes(X,Z), takes(Y,Z).

- Resolution: classmates(jane, Y) :- takes(Y, csc4101)

Unification

- Pattern-matching process used to associate variables with their values
  - student(joe).
  - :- student(X).
  - X = joe;

- Unification Rules:
  - A constant unifies only with itself
  - Two structures unify iff same functor and same # of arguments, and corresponding arguments unify recursively
  - A variable unifies with anything.

Unification

```
?- a = a.
yes
?- a = b.
no
?- foo(a, b) = foo(a, b).
yes
?- X = a.
X = a;
no
?- foo(a, b) = foo(X b).
X=a;
No
?- A = B.
A = _123
B = _123
```

Unification

- takes_lab(S) :- takes(S, C), has_lab(C).
- has_lab(D) :- meets_in(D, R), is_lab(R).

➔ takes_lab(S) :- takes(S, C), meets_in(C, R), is_lab(R).

13

Lists

- [a, b, c] can be expressed as (using vertical bar notation):
  - [a | [b, c]]
  - [a, b | [c]]
  - [a, b, c | []]

- member(X, [X|T]).
- member(X, [H|T]) :- member(X, T).

- sorted([]).
- sorted([X]).
- sorted([A, B | T]) :- A <= B, sorted([B | T]).

14

Lists

```
append([], A, A).
append(H | T), A, [H | L]) :- append(T, A, L).

?- append([a, b, c], [d, e], L).
L = [a , b, c, d, e]
?- append(X, [d, e], [a, b, c, d, e]).
X = [a, b, c]
?- append([a, b, c], Y, [a, b, c, d, e]).
Y = [d, e]
```

➔ Prolog predicates do not have a clear distinction between input and output arguments!

Arithmetic

- Built-in is predicate: unifies its first argument with the arithmetic value of its second argument.

```
?- is(X, 1+2).
X = 3
?- X is 1+2.
X  = 3
?- 3 is 1+2.
yes
?- 1+2 is 3.
no
?- X is Y.
<error>
?- Y is 1+2, X is Y.
X = 3
Y = 3
```

Running Prolog Programs

- GNU Prolog is already installed on byte:

```
-bash-2.05b$ gprolog
GNU Prolog 1.2.16
By Daniel Diaz
Copyright (C) 1999-2002 Daniel Diaz
| ?-
```

- Starts in Query mode
- You need to do consult(filename) or consult(user) to enter the facts and rules into the knowledgebase.