The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2002

Lecture 27: Prolog's Resolution and Programming Techniques

Felix Hernandez-Campos

March 27

SWI-Prolog

- We will use SWI-Prolog for the Prolog programming assignments
  - http://www.swi-prolog.org/
- After the installation, try the example program

```
?- [likes].
% likes compiled 0.00 sec, 2,148 bytes
Yes
?- likes(sam, curry).
No
?- likes(sam, X).
X = dahl ;
X = tandoori ;
X = kurma ;
```

Load example likes.pl

This goal cannot be proved, so it assumed to be false (This is the so called *Close World Assumption*)

Asks the interpreter to find more solutions

SWI-Prolog

- The editor shipped as part of SWI-Prolog supports coloring and context-sensitive indentation
  - Try "Edit" under "File"

Prolog Programming Model

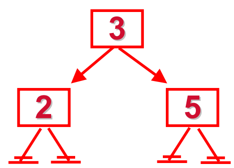
- A program is a *database of (Horn) clauses*
- Each clauses is composed of *terms*:
  - *Constants* (atoms, that are identifier starting with a lowercase letter, or numbers)
    - » *E.g.* `curry, 4.5`
  - *Variables* (identifiers starting with an uppercase letter)
    - » *E.g.* `Food`
  - *Structures* (predicates or data structures)
    - » *E.g.* `indian(Food), date(Year,Month,Day)`

Data Structures

- Data structures consist of an atom called the *functor* and a list of arguments
  - *E.g.* `date(Year,Month,Day)`
  - *E.g.*

Functors

```
T = tree(3, tree(2,nil,nil), tree(5,nil,nil))
```



Principle of Resolution

- Prolog execution is based on the *principle of resolution*
  - If C_1 and C_2 are Horn clauses and the head of C_1 matches one of the terms in the body of C_2, then we can replace the term in C_2 with the body of C_1
- For example,

C_1: `likes(sam,Food) :- indian(Food), mild(Food).`
C_2: `indian(dahl).`
C_3: `mild(dahl).`

  - We can replace the first and the second terms in C_1 by C_2 and C_3 using the principle of resolution (after *instantiating* variable `Food` to `dahl`)
  - Therefore, `likes(sam, dahl)` can be proved

Unification

- Prolog associates variables and values using a process known as *unification*
  - Variable that receive a value are said to be *instantiated*
- Unification rules
  - A constant unifies only with itself
  - Two structures unify if and only if they have the same functor and the same number of arguments, and the corresponding arguments unify recursively
  - A variable unifies to with anything

Equality

- Equality is defined as *unifiability*
  - An equality goal is using an infix predicate =
- For instance,

```
?- dahl = dahl.
Yes
?- dahl = curry.
No
?- likes(Person, dahl) = likes(sam, Food).
Person = sam
Food = dahl ;
No
?- likes(Person, curry) = likes(sam, Food).
Person = sam
Food = curry ;
No
```

Equality

- What is the results of

```
?- likes(Person, Food) = likes(sam, Food).

Person = sam
Food = _G158 ;

No
```

Internal Representation for an uninstantiated variable
Any **instantiation proves the equality**

Execution Order

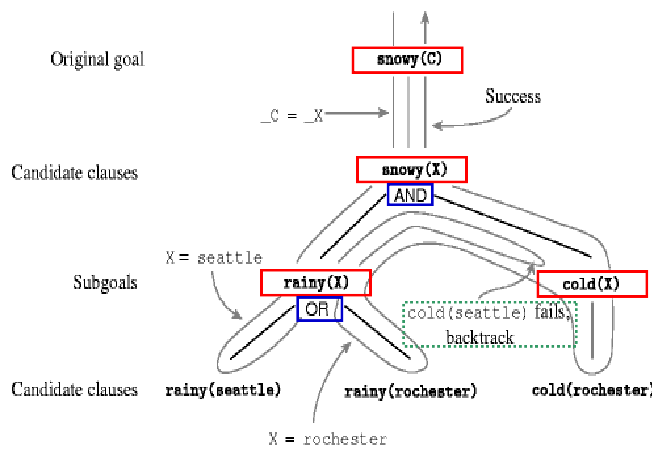
- Prolog searches for a resolution sequence that satisfies the goal
- In order to satisfy the logical predicate, we can imagine two search strategies:
  - *Forward chaining*, derived the goal from the axioms
  - *Backward chaining*, start with the goal and attempt to resolve them working backwards
- Backward chaining is usually more efficient, so it is the mechanism underlying the execution of Prolog programs
  - Forward chaining is more efficient when the number of facts is small and the number of rules is very large

Backward Chaining in Prolog

- Backward chaining follows a classic depth-first backtracking algorithm
- Example
  - Goal:

    Snowy(C)

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



Depth-first backtracking

- The search for a resolution is ordered and depth-first
  - The behavior of the interpreter is predictable
- Ordering is fundamental in recursion
  - Recursion is again the basic computational technique, as it was in functional languages
  - Inappropriate ordering of the terms may result in non-terminating resolutions (infinite regression)
  - For example: Graph

```
edge(a,b). edge(b, c). edge(c, d).
edge(d,e). edge(b, e). edge(d, f).
path(X, X).
path(X, Y) :- edge(Z, Y), path(X, Z).
```

Correct

Depth-first backtracking

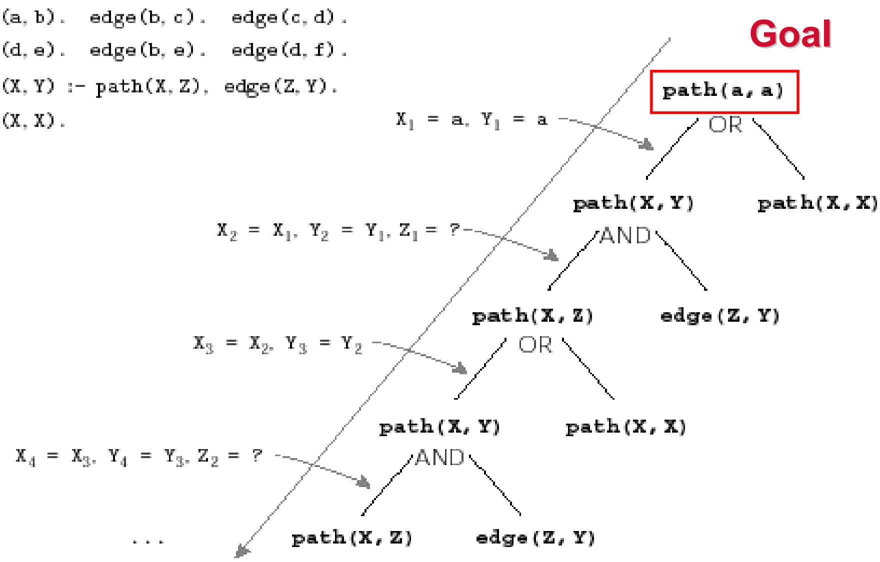
- The search for a resolution is ordered and depth-first
  - The behavior of the interpreter is predictable
- Ordering is fundamental in recursion
  - Recursion is again the basic computational technique, as it was in functional languages
  - Inappropriate ordering of the terms may result in non-terminating resolutions (infinite regression)
  - For example: Graph

```
edge(a,b). edge(b, c). edge(c, d).
edge(d,e). edge(b, e). edge(d, f).
path(X, Y) :- path(X, Z), edge(Z, Y).
path(X, X).
```

Incorrect

Infinite Regression

```
edge(a, b).  edge(b, c).  edge(c, d).
edge(d, e).  edge(b, e).  edge(d, f).
path(X, Y) :- path(X, Z), edge(Z, Y).
path(X, X).
```



Examples

- Genealogy
  - http://ktiml.mff.cuni.cz/~bartak/prolog/genealogy.html
- Data structures and arithmetic
  - Prolog has an arithmetic functor `is` that unifies arithmetic values
    - » *E.g.* `is (X, 1+2), X is 1+2`
  - Dates example
    - » http://ktiml.mff.cuni.cz/~bartak/prolog/genealogy.html

Reading Assignment

- Read
  - Scott Sect. 11.3.1
- *Guide to Prolog Example*, Roman Barták
  - Go through the first two examples
  - http://ktiml.mff.cuni.cz/~bartak/prolog/learning.html