CMSC 330: Organization of Programming Languages

Logic Programming with Prolog
Background

- 1972, University of Aix-Marseille
- Original goal: Natural language processing
- At first, just an interpreter written in Algol
  - Compiler created at Univ. of Edinburgh
More Information On Prolog

- Various tutorials available online
- Links on webpage
Logic Programming

- At a high level, logic programs model the relationship between “objects”
  1. Programmer specifies relationships at a high level
  2. Language builds a database
  3. Programmer then queries this database
  4. Language searches for answers
Features of Prolog

- Declarative
  - Specify what goals you want to prove, not how to prove them (mostly)

- Rule based

- Dynamically typed

- Several built-in datatypes
  - Lists, numbers, records, … but no functions

- Several other logic programming languages
  - Datalog is simpler; CLP and λProlog more feature-ful
  - Erlang borrows some features from Prolog
/* A small Prolog program */

female(alice).
male(bob).
male(charlie).
father(bob, charlie).
mother(alice, charlie).

% "X is a son of Y"
son(X, Y) :- father(Y, X), male(X).
son(X, Y) :- mother(Y, X), male(X).

Lowercase logically terminates

Program consists of facts and rules

Uppercase denotes variables

Use /* */ for comments, or % for 1-liners

Period ends statements
Running Prolog (Interactive Mode)

Navigating location and loading program at top level

?- working_directory(C,C).  Find current directory
C = 'c:/windows/system32/'.

?- working_directory(C,'c:/Users/me/desktop/p6').  Set directory
C = 'c:/Users/me/desktop/'.

?- ['01-basics.pl'].  Load file 01-basics.pl
% 01-basics.pl compiled 0.00 sec, 17 clauses
true.

?- make.  Reload modified files; replace rules
true.
Running Prolog (Interactive Mode)

Listing rules and entering queries at top level

?- listing(son).
son(X, Y) :-
    father(Y, X),
    male(X).
son(X, Y) :-
    mother(Y, X),
    male(X).
true.

?- son(X,Y).
X = charlie,
Y = bob;
X = charlie,
Y = alice.

List rules for son

User types ; to request additional answer

Multiple answers

User types return to complete request
Style

One predicate per line

blond(X) :-
  father(Father, X),
  blond(Father), % father is blond
  mother(Mother, X),
  blond(Mother). % and mother is blond

Descriptive variable names

Inline comments with % can be useful
Outline

- Syntax, terms, examples
- Unification
- Arithmetic / evaluation
- Programming conventions
- Goal evaluation
  - Search tree, clause tree
- Lists
- Built-in operators
- Cut, negation
Prolog Syntax and Terminology

Terms

- **Atoms**: begin with a lowercase letter
  - horse  underscores_ok  numbers2

- **Numbers**
  - 123  -234  -12e-4

- **Variables**: begin with uppercase or _
  - X  Biggest_Animal  _the_biggest1  _

- **Compound terms**: functor(arguments)
  - bigger(horse, duck)
  - bigger(X, duck)
  - f(a, g(X, _), Y, _)

No blank spaces between functor and (arguments)
Clauses

- **Facts**: define predicates, terminated by a period
  
  bigger(horse, duck).
  
  bigger(duck, gnat).
  
  Intuitively: “this particular relationship is true”

- **Rules**: Head :- Body
  
  is_bigger(X,Y) :- bigger(X,Y).
  
  is_bigger(X,Y) :- bigger(X,Z), is_bigger(Z,Y).
  
  Intuitively: “Head if Body”, or “Head is true if each of the subgoals can be shown to be true”

- **A program is a sequence of clauses**
Prolog Syntax and Terminology (cont.)

Queries

• To “run a program” is to submit queries to the interpreter
• Same structure as the body of a rule
  ➢ Predicates separated by commas, ended with a period
• Prolog tries to determine whether or not the predicates are true

?- is_bigger(horse, duck).
?- is_bigger(horse, X).

“Does there exist a substitution for X such that is_bigger(horse,X)?”
Unification – The Sine Qua Non of Prolog

- Two terms unify if and only if
  - They are identical
    ```prolog
    ?- gnat = gnat.
    true.
    ```
  - They can be made identical by substituting variables
    ```prolog
    ?- is_bigger(X, gnat) = is_bigger(horse, gnat).
    X = horse.  \{ This is the substitution: what \textit{X} must be for the two terms to be identical. \}
    ```
    ```prolog
    ?- pred(X, 2, 2) = pred(1, Y, X)
    false
    ```
    Sometimes there are multiple possible substitutions; Prolog can be asked to enumerate them all
    ```prolog
    ?- pred(X, 2, 2) = pred(1, Y, _)
    X = 1,
    Y = 2.
    ```
The = Operator

- For unification (matching)
- \texttt{?- 9 = 9.}
  \texttt{true.}
- \texttt{?- 7 + 2 = 9.}
  \texttt{false.}
- Why? Because these terms do not match
  - 7+2 is a compound term (e.g., +(7,2))
- Prolog does not evaluate either side of =
  - Before trying to match
The Is Operator

- For arithmetic operations
- “LHS is RHS”
  - First evaluate the RHS (and RHS only!) to value V
  - Then match: LHS = V

Examples

?- 9 is 7+2.      ?- 7+2 is 9.
  true.            false.

?- X = 7+2.       ?- X is 7+2.
  X = 7+2.        X = 9.
No Assignment

- and is operators do not perform assignment

Example

- foo(...,X) :- ... X = 1,... % true only if X = 1
- foo(...,X) :- ... X = 1, ..., X = 2, ... % always fails
- foo(...,X) :- ... X is 1,... % true only if X = 1
- foo(...,X) :- ... X is 1, ..., X is 2, ... % always fails

X can’t be unified with 1 & 2 at the same time
Function Parameter & Return Value

Code example

increment(X,Y) :-
    Y is X+1.
?- increment(1,Z).
  Z = 2.
?- increment(1,2).
true.
?- increment(Z,2).
ERROR: incr/2: Arguments are not sufficiently instantiated to int
Function Parameter & Return Value

- Code example

```
addN(X,N,Y) :-
    Y is X+N.

?- addN(1,2,Z).
Z = 3.
```

Parameters

Return value

Query

Result
Recursion

Code example

\[ \text{addN}(X,0,X). \]  \hspace{1cm} \text{Base case}

\[ \text{addN}(X,N,Y) :\]  \hspace{1cm} \text{Inductive step}
\[ \quad X1 \text{ is } X+1, \]
\[ \quad N1 \text{ is } N-1, \]
\[ \quad \text{addN}(X1,N1,Y). \]  \hspace{1cm} \text{Recursive call}

?- \text{addN}(1,2,Z).
Z = 3.
Factorial

- Code

```prolog
factorial(0, 1).
factorial(N, F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, F1),
    F is N * F1.
```
Tail Recursive Factorial w/ Accumulator

Code

tail_factorial(0,F,F).
tail_factorial(N,A,F) :-
    N > 0,
    A1 is N*A,
    N1 is N -1,
    tail_factorial(N1,A1,F).
And and OR

- **And**
  - To implement X && Y (use , in body of clause)
  - Example
    \[ Z :- X,Y. \]

- **OR**
  - To implement X || Y (use two clauses)
  - Example
    \[ Z :- X. \]
    \[ Z :- Y. \]
Goal Execution

- When submitting a query, we ask Prolog to substitute variables as necessary to make it true.
- Prolog performs **goal execution** to find a solution:
  - Start with the goal
  - Try to unify the head of a rule with the current goal
  - The rule hypotheses become subgoals
    - Substitutions from one subgoal constrain solutions to the next
  - If it reaches a dead end, it **backtracks**
    - Tries a different rule
  - When it can backtrack no further, it reports **false**
- More advanced topics later – cuts, negation, etc.
Goal Execution (cont.)

Consider the following:

• “All men are mortal”
  mortal(X) :- man(X).

• “Socrates is a man”
  man(socrates).

• “Is Socrates mortal?”
  ?- mortal(socrates).
  true.

How did Prolog infer this?

1. Sets mortal(socrates) as the initial goal
2. Sees if it unifies with the head of any clause:
   mortal(socrates) = mortal(X).
3. man(socrates) becomes the new goal (since X=socrates)
4. Recursively scans through all clauses, backtracking if needed …
Clause Tree

- Clause tree
  - Shows (recursive) evaluation of all clauses
  - Shows value (instance) of variable for each clause
  - Clause tree is true if all leaves are true

Factorial example

```prolog
factorial(0,1).
factorial(N,F) :-
    N > 0,
    N1 is N-1,
    factorial(N1,F1),
    F is N*F1.
```

```
factorial(3,6)
  3>0  2 is 3-1  factorial(2,2)  6 is 3*2
    2>0  1 is 2-1  factorial(1,1)  2 is 2*1
       1>0  0 is 1-1  factorial(0,1)  1 is 1*1
          true
```