COMP219: Artificial Intelligence

Lecture 3: Introduction to Prolog

Notice

- Practical classes for the module begin next week.
- Check your timetable to see which class you have been allocated to.
- Exercise sheets (non-assessed) will be posted on the module web page.
- Demonstrators will be on hand to provide assistance and feedback.

Overview

- Last time
  - Techniques, applications and state-of-the-art in AI
- Today:
  - a brief *introduction* to Prolog and its uses
  - describe the use of *facts* and *rules* in Prolog programs
  - explain how Prolog evaluates *queries*.
  - There will be four subsequent Prolog lectures
- Learning outcome covered today:
  Understand and write Prolog code to solve simple knowledge-based problems.

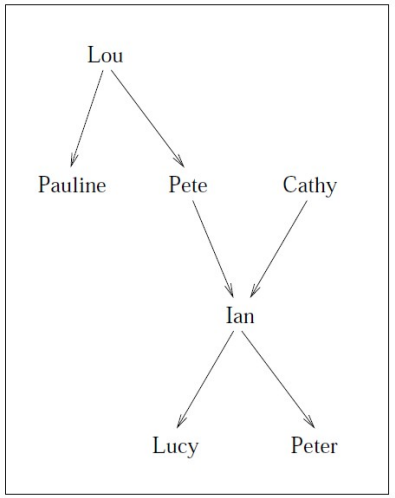
Prolog

- Short for Programming in Logic
- Based on an idea to use logic as a programming language (we'll see the connection later)
- Developed in the 1970s in Edinburgh and Marseilles
- Often thought of as an AI programming language
- Useful for solving problems involving objects and relations between them
- Not suitable for problems involving a lot of numeric calculation or GUIs
- "Deductive Database" with respect to which one can make deductive inferences. Can be used as the underlying logic of web applications

Books

- *Prolog Programming for Artificial Intelligence*, Ivan Bratko, third edition, Addison Wesley
- *The art of Prolog: advanced programming techniques*, Leon Sterling, Ehud Shapiro, MIT Press 1986
- *Introduction to programming in Prolog* Danny Crookes Prentice-Hall, 1988
- SWI Prolog Users' Manual available at
  - http://www.swi-prolog.org/pldoc/index.html
- SWI Prolog Web site
  - http://www.swi-prolog.org

Example: A Family Tree



- Lou is parent of Pete
- Pete is parent of Ian
- Ian is parent of Lucy

Facts

- To represent the previous family tree in Prolog we use *facts*, which are true without conditions
- Facts are clauses (a relation and arguments)
  - parent(pete,ian).
  - parent(ian,peter).
  - parent(ian,lucy).
- Note the full stop at the end of every line
- Relation names and arguments (constants or atoms) start with a lower case
- Compare to a database record: table *parent* with attributes for *parent* and *child*

More Facts

- We could store information about the gender of people in the family tree as follows
  - female(cathy).
  - female(lucy).
  - male(ian).
- Any kind of facts we want
  - lives_in(lucy,walford).
- Any number of arguments
  - lived_from(lou,1890,1992).
- No data-type declaration is needed (e.g. for characters, digits, strings)

The Family Tree Program

```
/* Family Tree */

parent(pete,ian).          % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(lou,pete).
parent(lou,pauline).
parent(cathy,ian).

female(cathy).             % Cathy is female.
female(lucy).
female(pauline).
female(lou).

male(ian).                 % Ian is male.
male(pete).
male(peter).
```

Comment symbols

Relations grouped together

Comments

- As with all programming languages, it is a good idea to include comments
- In Prolog there are two ways to do this

  % Ian is male

  % is a comment on this line only
- Extended comment

  /*

  Family Tree

  Describes family relationships

  */

  where

  /* denotes the beginning of the comment and

  */ denotes the end of the comment

Querying the Family Tree Program

- Assume the previous facts have been stored as a Prolog program and interpreted. A Prolog query is a clause
  - Is Ian a parent of Lucy?
    ?- parent(ian,lucy).
    true.
  - Is Ian a parent of Pauline?
    ?- parent(ian,pauline).
    false.
  - Is Cathy a parent of Peggy? *NB: Peggy does not appear at all in our program*
    ?- parent(cathy,peggy).
    false.
- 'true' here means 'have found a fact in the program that matches the query'
- 'false' here means 'have not found a fact in the program that matches the query'

Querying the Family Tree Program

- Can use variables as well to check facts
- Who is Lucy's parent?

  ?- parent(X,lucy).

  X = ian.

- Starting from top of program, Prolog matches parent relation with lucy as second argument, finds parent(ian,lucy), binds X to ian, and returns the result
- Variables begin with upper case letters (here X)

Querying the Family Tree Program

- If there are several answers, the next solution can be requested by typing a semi-colon

Who are Lou's children?

?- parent(lou,X).

X = pete ;

X = pauline .

- The order of the answers follows the order of facts in the program

Querying a program: summary

- Relations and arguments
  - Relations, e.g. parent (lower case)
  - concrete objects and constants, known as *atoms*, e.g. ian or cathy (lower case)
  - *variables*, e.g. X, Father, or Child (upper case)
- Questions to the system are *goals* (go and find the answer)
  - If a positive answer to the question is found, we say the goal was satisfiable, and the goal succeeded
  - If we get a negative answer, the goal was unsatisfiable, and the goal failed

Rules

- Rules describe what holds depending on a certain condition: conclusion if condition satisfied
- With rules, we can define complex relations in terms of simpler relations
- Mothers are female parents
  mother(X,Y) :- parent(X,Y),female(X).
- For all X and Y, X is the mother of Y if X is the parent of Y and X is female - implicit universal quantification
- The left side of the rule (the conclusion part) is known as the head of the clause. The right side of the rule (the condition part) is known as the body of the clause
- Commas between clauses are understood as conjunction

Evaluating Rules

- Is Cathy the mother of Ian?
  ?- mother(cathy,ian).
  true.
- There are no facts about mothers in the program so we must use the rule about mothers
  mother(X,Y) :- parent(X,Y),female(X).
- We work from what we can satisfy to see if we can get a conclusion
  - Start with a body clause, parent(X,Y), match to a fact, bind variables, use the bound variables for female(X)
  - If both clauses are satisfied, then we get a conclusion
  - Otherwise, try binding the variables with respect to another fact
  - Continue until satisfaction or failure

How It Works

```
/* Family Tree */

parent(pete,ian).     % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(lou,pete).
parent(lou,pauline).
parent(cathy,ian).

female(cathy).        % Cathy is female.
female(lucy).
female(pauline).
female(lou).

male(ian).            % Ian is male.
male(pete).
male(peter).
```

?- mother(X,Y).

Find a value for X and a value for Y which satisfies mother(X,Y)

mother(X,Y):-parent(X,Y),female(X).

Considers the goals in order (left to right) with respect to the facts (top down)

parent(X,Y)

Matches X = pete, Y = ian. Binding of variables is applied to other clauses

mother(pete,ian) :- parent(pete,ian),female(pete).

First body clause is OK, but female(pete) fails (not a fact)

Backtracks to try another way of satisfying the body clauses

parent(ian,peter) is no good as female(ian) fails

parent(ian,lucy) is no good as female(ian) fails (again)

parent(lou,pete) succeeds as female(lou) succeeds

mother(X,Y) succeeds where X=lou, Y=pete. Others?

More queries

```
/* Family Tree */

parent(pete,ian).     % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(lou,pete).
parent(lou,pauline).
parent(cathy,ian).

female(cathy).        % Cathy is female.
female(lucy).
female(pauline).
female(lou).

male(ian).            % Ian is male.
male(pete).
male(peter).
```

Can instantiate a variable:

?-mother(cathy,Child).

This will bind the other variables in the body:

mother(cathy,Child):-
parent(cathy,Child),female(cathy).

parent(cathy,ian) succeeds, binds Child to ian, Child=ian, and female(cathy) matches. So the following succeeds:

mother(cathy,ian).

Others:

?-mother(pete,Y).

Y=ian. But, fails since female(pete) fails

?-mother(lou,pauline).

Succeeds

?-mother(lucy,Y).

Fails since parent(lucy,Y) fails as it is never a fact for any binding of Y

?-mother(Parent,ian).

Succeeds with Parent=cathy. Will try Parent=pete first, but fail

More Rules – complex relations from facts or defined relations

- Grandparents are parents of parents
  grandparent(X,Z) :- parent(X,Y),parent(Y,Z).
- Siblings are people with the same parent
  sibling(X,Y) :- parent(Z,X),parent(Z,Y).
  sister(X,Y) :- female(X), parent(Z,X),parent(Z,Y).
- Careful on interpretation of clauses – what do grandparent(X,Z) or sister(X,Y) mean? X has grandparent Z, or X is grandparent of Z? Is it X is sister of Y, or X has sister Y?
- Can use relations defined in rules in other rules
  grandmother(X,Z) :- mother(X,Y),parent(Y,Z).
  sister(X,Y) :- sibling(X,Y),female(X).

Exercise

Running Prolog under UNIX/LINUX

- On a unix/linux system enter the word swipl

```
$ swipl
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```


Loading Programs

- Create the program (facts and rules) using any text editor
- If the program is saved in the same directory, type

```
?- consult(familyTree).
% familyTree compiled 0.00 sec, 15 clauses
true.
?-
```

- An alternative is to type

```
?- ['familyTree.pl'].
% familyTree compiled 0.00 sec, 15 clauses
true.
?-
```

Now you can type in your queries.
To exit the program type

```
?- halt.
```


Running Prolog under Windows

- Choose SWI-Prolog from

  Start -> All Programs -> Programming Resources -> SWI-Prolog -> Prolog

- To load a program, in the console type

  ['h:\\familyTree']

  making sure you specify the correct path to the location of the familyTree program within your filestore.

Summary (I)

- A Prolog program is constructed from facts and rules
- Facts describe things that are true without conditions
  - like data in a database
- Rules describe things that hold depending on certain conditions. The conditions are defined in rules and facts
  - Ultimately all rules unfold to facts
- Prolog programs can be queried
- Prolog clauses are facts, rules, and queries
- Queries are answered by solving each goal in turn by matching clauses (perhaps in the body of a rule) against facts (whether given or derived), binding variables, and backtracking where necessary until either the query is satisfied or it fails

Summary (II)

- Today
  - Introduction to Prolog
    - Facts, rules, queries
    - Running Prolog

- Next time
  - A brief history of AI
  - AI and intelligent agents