Prolog programming hints

COMP9414 - 2008 Semester 1

Ronnie Taib (ronniet@cse.unsw.edu.au)

About these slides

- Practical notes on using Prolog
- Some hints for better code
- LECTURE NOTES ALWAYS PREVAIL!
- The Prolog dictionary also contains a mine of information

Process

- Understand the problem (paper & pen)
- Break things down: sub-tasks (paper & pen)
- State them as rules that should be true (paper & pen)
- Write your comments (code)
- Start coding now, not before (code)
- Test / debug (code)
- Submit / get a good mark (yay!)

How does Prolog work?

- There is no AI: programmer = intelligence
- Prolog
  - Prover system: can only prove something is true or fail to do so
  - Unification: Make equal things that should be equal
  - Backtracking: explore the search tree, backtrack on failure
  - Try all possible combinations until satisfied (pretty much)
  - (Then try again: read topic on More? in class forum)
  - Respect code order: Prolog reads code from top to bottom
  - Some optimisations (don’t worry about them for now)
Recursion

- Basic skill you need to master in this course
- Needed for data of non fixed length (lists, trees, tables...)

Stop condition(s): base or trivial case
Recursive case(s)

- Pay attention to predicate order
- Run it in your head (not so easy, but rewarding)

Recursion

- Designing the recursive case:
  - “Pretend” the problem has been solved for the tail
  - Solve the problem for the head only

stop_condition([1, 2, 3, 4, 5], Result) :-
  stop_condition([2, 3, 4, 5], TailResult), % Mentally pretend tail solved,
  % i.e. TailResult = 14
  Result is Head + TailResult. % Use it and process
  % head, to generate result, i.e.
  % Result is 1 + 14 = 15

- Add stop condition(s) and that’s it!

Cuts!

- Can slash your mark
- Prolog experts use them a lot, so why not you?
  - Have side effects novices may overlook
  - Hinder legibility
  - Make mental debugging really hard
- Luckily, you can often do without

Cuts!

isEven(N) :-
  0 is N mod 2,
  write("It is even").

isEven(N) :-
  write("It is not even").

isEven(N) :-
  0 is N mod 2,
  write(N),
  write(" is even").

% Will not try second predicate if this one succeeds

isEvenCut(N) :-
  write(N),
  write(" is not even").

Works well, but it's a sin…

* Don't forget to prevent M Inspector by adding this at the top of these examples:
  % set_prolog_flag(prompt_alternatives_on, groundness).
Cuts!

- isEvenNoCut(N) :-
  0 is N mod 2,
  write(N),
  write(' is even').

- isEvenNoCut(N) :-
  \\+ 0 is N mod 2,
  write(N),
  write(' is not even').

- Predicates are mutually exclusive on their first rule: works well without sinning.

Hints: compact forms

- Make your code more compact and legible
- Any 2 variables you want equal should share the same name
- Beware how instructions are given
- Example of dangerous subject:
  Write a predicate add_if_equal(X, Y, Result) that binds the sum of X and Y to Result, iff X and Y are equal.

Hints: compact forms

- addIfEqual(X, Y, Result) :-
  X = Y,
  Result is X + Y.

- Any ‘X = Y’ in your code is a bad sign and should be addressed by using the same name for both X and Y

- addIfEqual(X, X, Result) :-
  Result is X + X.

- This code does exactly the same thing, but is more compact, using only X everywhere we wanted it.

Hints: comments

- Commenting your code will
  – Make a real difference in all your subjects
  – Attract good marks in industrial placements
  – Get good recommendations
  – Ease your life (hard to believe but true)
Hints: comments

- Help you understand the problem better by explaining it to (a virtual) someone
- Crucial to debug and re-use by others or you
- Can clear you from plagiarism suspicion: they show you know what the code is doing
- Take time and space in your code
- But, not so much time, after offsetting savings

Hints: comments

- Code header (see assignment specs)
- Each predicate (about 3 lines)
  - What does it do?
    - Quick overview, insist on tricky aspects
  - What are the expected parameters
    - State limitations, e.g. “should be a list containing integers only”.
  - “Return” values
    - What does the predicate return, under what form (can be a success/failure or some variable being set)

Hints: code layout

% COMP9414 - Assignment 1, 2008, Semester 1
% Ronnie Taib (<studentID>)
%
% This assignment covers blah, blah, blah…
%
% isEven(N)
% This predicate returns true if N is even, and
% fails otherwise. N must be an integer value.
isEven(N) :-
  0 is N mod 2,  % “mod” must be right of “is” to be evaluated
  [...]
Hints: naming

- Use meaningful names for predicates and variables
- mas(L,M,N) % What could this be doing? vs.
- merge_and_sort(List1, List2, MergedList)

You will not be assessed on this, but non meaningful names will invariably cause trouble!

Hints: Google is evil

- Beware code on the net
  - Expert code is complex
  - Contains cuts
  - May have side effects
  - Is plagiarism!
  - Will kill your skills (remember the final exam)

- Lucky enough: you won’t find much help applicable to the assignments

Hints: testing

- All test cases provided in spec must work correctly
- But you need more
- Create your own tests, be creative
- Insist on borderline cases
  - Empty list
  - One element list
  - Special values: 0, -1, -2... (if applicable)
  - Vary element order in lists

Hints: testing

- Write a test suite to save typing the tests again and again
  - Separate file, e.g. hintsTest.pl
  - Test suites for each predicate, for both expected success and failure
  - Also place all test facts and rules provided by lecturer
  - DO NOT SUBMIT THE TEST SUITE FILE!
Hints: testing

- Once in Prolog:
  - Consult test suite (this will in turn consult your code)
  - Type "test."
  - If it fails, something went wrong.
  - So that's 2 simple commands to run all your tests as many times as required until you fix all your code.
  - You may add outputs for each test.

```
[:...]
% The test suite
test_descendant :-
descendant(peter, james),
    not(descendant(james, peter)),
    [...],
    write('descendant passed.').

test_my_other_predicate :-
    [...],
    write('my_other_predicate passed').

test :-
    % This one calls all individual %tests
test_descendant,
test_my_other_predicate.
```

You MUST NOT submit the test suite file

- You will not be assessed on it, nor will you get a better mark for doing it.
- However, it will probably save you time and make your code more bullet proof!

Hints: debugging

- Compare mental execution (what you expect) and the actual execution (Prolog trace)
- Trace mode (see lecture notes)
  - Space bar / Enter: keep going
  - 's': skip (obtain result of a predicate without showing all steps it contains)
  - 'a': abort (stop tracing)
  - Pay attention to "redo" and operation number
  - Watch for exit or fail on left

- Consider using print statements (single quote for text) (remove when debugging complete)
  `write(N),
  write(" is even"`).