Nondeterministic Programming

(Sterling and Shapiro - the Art of Prolog)

Nondeterminism is powerful for defining and implementing algorithms.

Intuitively, a nondeterministic machine can choose its next operation correctly when faced with several alternatives.

Nondeterminism can be simulated/approximated by Prolog's sequential search and backtracking. Nondeterminism cannot *truly* be achieved.

Examples of nondeterministic programs (mostly for NP-complete problems):

- generate-and-test
- N-queens
- Map colouring
- AI planning
- Towers of Hanoi
- etc.

Towers of Hanoi

Setup: 3 pegs ("left", "centre", "right"). In the initial state one peg (let's say the "left" peg) has N rings on it, stacked from largest to smallest.

Task: Move N disks from the left peg to the right peg using the centre peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk.

Solution:

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Towers of Hanoi (cont.)

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Execution for N=3:

```
?-  move(3,left,right,centre).
Move top disk from left to right
Move top disk from left to centre
Move top disk from right to centre
Move top disk from left to right
Move top disk from centre to left
Move top disk from centre to right
Move top disk from left to right

Yes
```

Towers of Hanoi (cont.)

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Advice for Writing Prolog

To minimize bugs, especially with cut and negation:

- Use cut or negation as necessary to avoid wrong answers.
- Always use ";" when testing to check all possible answers.
- Use cut to avoid duplicate answers.
- Use cut where possible for efficiency.
- Use "_" where possible for efficiency.
- Follow the safety guidelines for negation.
- Test with variables in every combination of positions.
- Use a precondition to state where variables are disallowed.

Prolog Review

- Logic Programming
- Prolog vs. Scheme (relational vs. functional)
- Logic Programming vs. Prolog (nondeterministic vs. deterministic, etc.)
- Prolog Syntax (Horn Clauses (w/ variables), Facts, translating from english to Prolog)
- Writing Recursive Predicates (e.g., family relations)
- Lists (internal representation (dot predicate), head/tail)
- Recursive Predicates for List Manipulation (including accumulators)
- Other Structures (functions, e.g., resistor, parse tree, double examples)
- How Prolog Works
  - Unification
  - Goal-Directed Reasoning
  - Rule Ordering
  - Backtracking DFS

- Improving Efficiency
  - Anonymous Variables
  - Accumulators
  - CUT
- Negation as Failure (NAF) (safety conditions, etc.)
- Arithmetics
- Cut (!)
- univ, call, functor, arg, assert, retract
- Nondeterministic Programs