Nondeterministic Programming

Nondeterministic is powerful for defining and implementing algorithms.

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

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):

- generation
- search
- map coloring
- AI planning
- Towers of Hanoi
- ...

Towers of Hanoi

Setup: 3 pegs ('left', 'centre', 'right'). In the initial state, one peg (let's say the 'left' peg) has N disks 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:

\[ m(n, 1, X, Y, Z) \]

\[ \text{move}(1, \text{top disk from } X, \text{to } Y) ; \]
\[ \text{move}(n-1, X, Z, Y) ; \]
\[ \text{move}(1, \text{top disk from } Z, \text{to } Y) ; \]
\[ m(n-1, X, Y, Z) \]

Exercise for N3:

To \[ m(3, x, y, z) \]:

\[ \text{move}(1, \text{top disk from left to right}) ; \]
\[ \text{move}(3, \text{top disk from left to right}) ; \]
\[ \text{move}(2, \text{top disk from right to center}) ; \]
\[ \text{move}(1, \text{top disk from right to center}) ; \]
\[ \text{move}(3, \text{top disk from center to right}) ; \]
\[ \text{move}(2, \text{top disk from center to right}) ; \]

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 (declarative vs. functional)
- Logic Programming vs. Prolog (unification/rewriting vs. backtracking, etc.)
- Prolog Syntax (lists, variables, backtracking, etc.)
- Writing Recursive Predicates (\( m(n, x, y, z) \))
- Lists (\( \text{reverse} \))
- Recursive Predicates for List Manipulation (\( \text{size} \))
- Other Predicates (\( \text{factorial} \))
- How Prolog Works
  - Unification
  - Goal Directed Reasoning
  - Rule Ordering
  - Backtracking

- Improving Efficiency
  - Anonymous Variables
  - Accumulators
  - CUT
- Negation as Failure (NAF) (\( \text{not}(\text{goal}) \))
- Arithmetic
- Cut (!)
- unify, cons, functor, and, assert, retract
- Nondeterministic Predicates
- Non-deterministic Programs