Logic Programming

- Programming paradigm based on symbolic logic
  - First order predicate calculus using constants, predicates, functions, variables, connectives, quantifiers, punctuation
- Start with axioms
  - Prove theorems
  - Proof is a kind of computation
- A program in the logic paradigm
  - A collection of statements is assumed to be correct
  - A desired fact is derived by some automatic application of inference rules

Example of Inference

- Assumptions
  - A horse is a mammal.
  - A human is a mammal.
  - Mammals have four legs and no arms, or two legs and two arms.
  - A horse has no arms.
  - A human has arms.
- A theorem
  - A horse has four legs.
As Horn Clauses

- Horn clauses are implications:
  - head ← body, where body is a collection of simple statements
- Examples
  - mammal(horse). *An axiom is a head with empty body.*
  - mammal(human).
  - legs(x,2) ← mammal(x), arms(x,2)
  - legs(x,4) ← mammal(x), arms(x,0)
  - arms(horse,0) .
- Query (theorem to be proved)
  - ← legs(horse,4)

Resolution and Unification

- Resolution inference rule:
  - If we have two clauses and head of first matches statement in the body of the second, then we can replace it by the body of the first
- Example: suppose we have two clauses
  - b ← a
  - c ← b
  - Then we can infer
    - c ← a
  - Essentially, we combine heads and bodies, then cancel matching statements on both sides
- Unification – pattern matching to make statements identical
  - Variables set to patterns, i.e., variables are instantiated
Using Resolution and Unification

- Given
  - mammal(horse).
  - arms(horse,0).
  - legs(x,2) ← mammal(x), arms(x,2)
  - legs(x,4) ← mammal(x), arms(x,0)
- Query
  - legs(horse,4)
- Inference
  - Use 4th rule to get: legs(x,4) ← mammal(x), arms(x,0), legs(horse,4)
  - Match x to horse: legs(horse,4) ← mammal(horse), arms(horse,0), legs(horse,4)
  - Cancel to get: ← mammal(horse), arms(horse,0)
  - Use 1st rule: mammal(horse) ← mammal(horse), arms(horse,0)
  - Cancel to get: ← arms(horse,0)
  - Use 2nd rule: arms(horse,0) ← arms(horse,0)
  - Cancel to get: ←, which shows the query is true.

A more program like example

- Greatest common divisor
- Given
  - gcd(u,v,w). (the gcd of a number and zero is that number)
  - gcd(u,v,w) ← not zero(v), gcd(v, u mod v, w). (the gcd of two numbers is the same as the gcd of the second and the mod of the two)
- Query
  - gcd(15,10,x)
- Inference
  - 2nd rule, unification: gcd(15,10,x) ← not zero(10), gcd(10,15 mod 10,x),gcd(15,10,x)
  - Cancel to get: ← not zero(10), gcd(10,15 mod 10,x)
  - Use arithmetic, basic properties: ← gcd(10,5,x)
  - 2nd again: gcd(10,5,x) ← not zero(5), gcd(5,10 mod 5,x),gcd(10,5,x)
  - Cancel to get: ← not zero(5), gcd(5,10 mod 5,x)
  - Simplify again: ← gcd(5,0,x)
  - This matches first rule (with x=5): gcd(5,0,5) ← gcd(5,0,5), so answer x=5 is true
Prolog

- Widely used logic programming language
  - Based on Horn clauses
  - Uses linear depth first strategy
- Interpreted
- Syntax
  - Use :- for implications
  - Variables capitalized
  - Builtin arithmetic
  - Comparison dicey – must force evaluation using 'is'
  - Statements terminated with period
  - 'consult' used to read assertions

Prolog Example

- File 'gcd' contains
  \[
gcd(U, 0, U) \cdot\n\]
  \[
gcd(U, V, W) \ :- \ (V=\not=0), \text{R is } U \mod V, gcd(V, R, W).\n\]
- A Prolog session
  \[
| \ ?- \ consult('gcd'). \\n\{\text{consulting } /\text{nfs/home/faculty/datkins/prolog/gcd...}\} \\nyes \\n| \ ?- \ gcd(15, 10, X). \\n\text{X = 5} ? \\nyes \\n| \ ?- \ gcd(96, 5, X). \\n\text{X = 1} ? \\nyes \\n| \ ?- \ gcd(96, 60, X). \\n\text{X = 12} ? \\nyes \\n| \ ?- \n\]

Another Prolog Example

- File 'append' contains
  
  ```prolog
  append([], Y, Y).
  append([H|X], Y, [H|Z]) :- append(X, Y, Z).
  ```

- Prolog session

```prolog
| ?- consult('append').
{consulting /nfs/home/faculty/datkins/prolog/append...} yes
| ?- append([a,b],[c,d,e], [a,b,c,d,e]).
yes
| ?- append([a,b],[c,d,e],X).
X = [a,b,c,d,e] ?
yes
| ?- append([a,b],X, [a,b,c,d]).
X = [c,d] ?
yes
| ?- append(X, [d,c], [a,b,c,d]).
no
```