CSE 3302
Programming Languages

Logic Programming: Prolog

Chengkai Li
Fall 2007

SWI-Prolog

- http://www.swi-prolog.org/

Logic Programming

- Program
  Axioms (facts): true statements
- Input to Program
  query (goal): statement true (theorems) or false?
- Logic programming systems = deductive databases
  datalog

Example

- Axioms (facts):
  0 is a natural number.
  For all x, if x is a natural number, then so is the successor of x.
- Query (goal).
  2 is natural number? (can be proved by facts)
  -1 is a natural number? (cannot be proved)

Another example

- Facts:
  The factorial of 0 is 1.
  If m is the factorial of n - 1, then n * m is the factorial of n.
- Query:
  The factorial of 2 is 3?
  (How do we ask "What is the factorial of 2?")

First-Order Predicate Calculus

- Logic used in logic programming:
  First-order predicate calculus
  First-order predicate logic
  Predicate logic
  First-order logic

  $\forall x\ (x \neq x+1)$
- Second-order logic
  $\forall S\ \forall x\ (x \in S \vee x \notin S)$

First-Order Predicate Calculus: Example

- `natural(0)`
  `∀ X, natural(X) → natural(successor(x))`
- `∀X and Y, parent(X,Y) → ancestor(X,Y).`
  `∀A, B, and C, ancestor(A,B) and ancestor(B,C) → ancestor(A,C).`
  `∀X and Y, mother(X,Y) → parent(X,Y).`
  `∀X and Y, father(X,Y) → parent(X,Y).`
  `father(Bill,Jill).`
  `mother(Jill,Sam).`
  `father(Bob,Sam).`
- `factorial(0,1).`
  `∀ N and M, factorial(N-1,M) → factorial(N,N*M).`

First-Order Predicate Calculus: statements

Symbols in statements:

- *Constants (a.k.a. atoms)*
  numbers (e.g., `0`) or names (e.g., `Bill`).
- *Predicates*
  Names for Boolean functions (true/false) . Can have arguments. (e.g. `parent(X,Y)`).
- *Functions*
  non-Boolean functions (`successor(X)`).
- *Variables*
  e.g., `X`.
- *Connectives (operations)*
  `and, or, not`
  `implication (→):a→b (b or not a)`
  `equivalence (↔): a↔b (a→b and b→a)`

First-Order Predicate Calculus: statements (cont'd)

- *Quantifiers*
  universal quantifier "for all" ∀
  existential quantifier "there exists" ∃
  bound variable (a variable introduced by a quantifier)
  free variable
- *Punctuation symbols*
  parentheses (for changing associativity and precedence.)
  comma
  period
- Arguments to predicates and functions can only be ***terms***:
  - Contain constants, variables, and functions.
  - Cannot have predicates, qualifiers, or connectives.

Horn Clause

- First-order logic too complicated for an effective logic programming system.
- Horn Clause: a fragment of first-order logic
  `b ← a1 and a2 and a3 … and an.`

  head body no "or" and no quantifier

  `b ←.` fact
  `←b.` query

- Variables in head: universally quantified
  Variables in body only: existentially quantified
- Need "or" in head? Multiple clauses

Horn Clauses: Example

- First-Order Logic:
  `natural(0).`
  `∀X, natural(X) → natural(successor(x)).`

⇒

- Horn Clause:
  `natural(0).`
  `natural(successor(x)) ← natural(X).`

Horn Clauses: Example

- First-Order Logic:
  `factorial(0,1).`
  `∀ N and ∀ M, factorial(N-1,M) → factorial(N,N*M).`

⇒

- Horn Clause:
  `factorial(0,1).`
  `factorial(N,N*M) ← factorial(N-1,M).`

Horn Clauses: Example

- Horn Clause:

```
ancestor(X,Y) ← parent(X,Y).
ancestor(A,C) ← ancestor(A,B) and ancestor(B,C).
parent(X,Y) ← mother(X,Y).
parent(X,Y) ← father(X,Y).
father(Bill,Jill).
mother(Jill,Sam).
father(Bob,Sam).
```


Horn Clauses: Example

- First-Order Logic:

```
∀ X, mammal(X) → legs(X,2) or legs(X,4).
```

⇒

- Horn Clause:

```
legs(X,4) ← mammal(X) and not legs(X,2).
legs(X,2) ← mammal(X) and not legs(X,4).
```


Prolog syntax

- :- for ←
  , for and

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```


Problem Solving

- Program = Data + Algorithms
- Program = Object.Message(Object)
- Program = Functions Functions
- Algorithm = Logic + Control

Programmers: facts/axioms/statements

Logic programming systems: prove goals from facts

- The holy grail: we specify the logic itself, the system proves.
  - Not totally realized by logic programming languages. Programmers must be aware of how the system proves, in order to write efficient, or even correct programs.
- Prove goals from facts:
  - Resolution and Unification

Resolution

- Resolution: Using a clause, replace its head in the second clause by its body, if they "match".

- $a \leftarrow a_1, \ldots, a_n.$
  $b \leftarrow b_1, \ldots, b_i, \ldots, b_m.$

  if b_i matches a;
  $b \leftarrow b_1, \ldots, a_1, \ldots, a_n, \ldots, b_m.$

Resolution: Another view

- Resolution: Combine two clauses, and cancel matching statements on both sides.

- $a \leftarrow a_1, \ldots, a_n.$
  $b \leftarrow b_1, \ldots, b_i, \ldots, b_m.$

  $\cancel{a}, b \leftarrow a_1, \ldots, a_n, b_1, \ldots, \cancel{b_i}, \ldots, b_m.$

Problem solving in logic programming systems

- Program:
  - Statements/Facts (clauses).
- Goals:
  - Headless clauses, with a list of subgoals.
- Problem solving by resolution:
  - Matching subgoals with the heads in the facts, and replacing the subgoals by the corresponding bodies.
  - Cancelling matching statements.
  - Recursively do this, till we eliminate all goals. (Thus original goals proved.)

Example

- Fact:
  `mammal(human).`
- Goal:
  `← mammal(human).`
- Proving:
  $\texttt{mamma}\cancel{\texttt{l}}\texttt{(human)} \leftarrow \texttt{mamma}\cancel{\texttt{l}}\texttt{(human).}$
  `←.`

Example

- Fact:
```
legs(X,2) ← mammal(X), arms(X,2).
legs(X,4) ← mammal(X), arms(X,0).
mammal(horse).
arms(horse,0).
```
- Goal:
  `← legs(horse,4).`
- Proving: ?

Unification

- Unification: Pattern matching to make statements identical (when there are variables).
- Set variables equal to patterns: instantiated.
- In previous example:
  legs(X,4) and legs(horse,4) are unified.
  (X is instantiated with horse.)

Unification: Example

- Euclid's algorithm for greatest common divisor
- Facts:
```
gcd(U,0,U).
gcd(U,V,W) ← not zero(V), gcd(V, U mod V, W).
```
- Goals:
  `← gcd(15,10,X).`

Things unspecified

- The order to resolve subgoals.
- The order to use clauses to resolve subgoals.
- Possible to implement systems that don't depend on the order, but too inefficient.
- Thus programmers must know the orders used by the language implementations. (Search Strategies)

Example

- Facts:

```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Queries:

```
?- ancestor(bill,sam).
?- ancestor(X,bob).
```