Programming Languages: Logic Paradigm

Onur Tolga Şehitoğlu

Computer Engineering,METU

8 May 2008

Outline

Logic Programming Paradigm

- Based on logic and declarative programming
- 60's and early 70's
- Prolog (**Pro**gramming in **log**ic, 1972) is the most well known representative of the paradigm.
- Prolog is based on Horn clauses and SLD resolution
- Mostly developed in fifth generation computer systems project
- Specially designed for theorem proof and artificial intelligence but allows general purpose computation.
- Some other languages in paradigm: ALF, Frill, Gödel, Mercury, Oz, Ciao, λProlog, datalog, and CLP languages

Constraint Logic Programming

- Clause: disjunction of universally quantified literals,

$$\forall(L_1 \vee L_2 \vee \ldots \vee L_n)$$

- A logic program clause is a clause with exactly one positive literal

$$\forall(A \vee \neg A_1 \vee \neg A_2 \ldots \vee \neg A_n) \equiv$$
$$\forall(A \Leftarrow A_1 \wedge A_2 \ldots \wedge A_n)$$

- A goal clause: no positive literal

$$\forall(\neg A_1 \vee \neg A_2 \ldots \vee \neg A_n)$$

- Proof by refutation, try to unsatisfy the clauses with a goal clause G. Find $\exists(G)$.
- Linear resolution for definite programs with constraints and selected atom.

What does Prolog look like?

```
father(ahmet,ayse).
father(hasan,ahmet).
mother(fatma,ayse).
mother(hatice,fatma).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
grandparent(X,Y) :- parent(X,Z),parent(Z,Y).
```

- CLP on first order terms. (Horn clauses).
- Unification. Bidirectional.
- Backtracking. Proof search based on trial of all matching clauses.

Prolog Terms

- Atoms:
  1. Strings with starting with a small letter and consist of
     `[a-zA-Z_0-9]*`
     `a aDAM a1_2`
  2. Strings consisting of only punctuation
     `* *** .+. .<.>.`
  3. Any string enclosed in single quotes (like an arbitrary string)
     `'ADAM' 'Onur Sehitoglu' '2 * 4 < 6'`
- Numbers

  `1234 12.32 12.23e-10`

- Variables:
  1. Strings with starting with a capital letter or _ and consist of `[a-zA-Z_0-9]*`
     Adam _adam A093
  2. _ is the universal match symbol. Not variable
- Structures:
  - starts with an atom head
  - has one or more arguments (any term) enclosed in paranthesis, separated by comma
  - structure head cannot be a variable or anything other than atom.
    `a(b) a(b,c) a(b,c,d) ++(12) +(*) *(1,a(b)) 'hello world'(1,2)`
    √
    `X(b) 4(b,c) a() ++() _(3)` ×
  - some structures defined as infix:
    `+(1,2)` ≡ `1+2` , `:-(a,b,c,d)` ≡ `a :- b,c,d`
    `is(X,+(Y,1))` ≡ `X is X + 1`

Syntactic Sugars

- Prolog interpreter automatically maps some easy to read syntax into its actual structure.
- List: `[a,b,c]` ≡ `.(a,.(b,.(c,[])))`
- Head and Tail: `[H|T]` ≡ `.(H,T)`
- String: `"ABC"` ≡ `[65,66,67]` (ascii integer values)
- use `display(Term).` to see actual structure of the term.

Unification

Bi-directional (both actual and formal argument can be instantiated)

1. if S and T are atoms or number, unification successfull only if $S = T$
2. if S is a variable, S is instantiated as T, if it is compatible with current constraint store (S is instantiated to another term, they are unified)
3. if S and T are structures, successfull if:
   - head of S = head of T
   - they have same arity
   - unification of all corressponding terms are successfull

$\mathcal{A}$: list of atoms, $\mathcal{N}$: list of numbers, $\mathcal{V}$: list of variables,
$\mathcal{S}$: list of structures, $\mathcal{P}$ current constraint store
$s \in \mathcal{S}, arity(s)$: number of arguments of structure,
$s \in \mathcal{S}, head(s)$: head atom of the structure,
$s \in \mathcal{S}, arg_i(s)$: i^{th} argument term of the structure,
$p \vDash \mathcal{P}$: p is consistent with current constraint store.

$$S \equiv T; \mathcal{P} \quad =$$

$$(S, T \in \mathcal{A} \quad \vee \quad S, T \in \mathcal{N}) \quad \wedge \quad S = T \quad \rightarrow \quad true; \mathcal{P}$$
$$S \in \mathcal{V} \quad \wedge \quad S \equiv T \models \mathcal{P} \quad \rightarrow \quad true; S \equiv T \quad \wedge \quad \mathcal{P}$$
$$T \in \mathcal{V} \quad \wedge \quad S \equiv T \models \mathcal{P} \quad \rightarrow \quad true; S \equiv T \quad \wedge \quad \mathcal{P}$$
$$S, T \in \mathcal{S} \quad \wedge \quad head(S) = head(T) \quad \wedge \quad arity(S) = arity(T) \quad \rightarrow$$
$$\forall i, arg_i(S) \equiv arg_i(T); P$$

Unification Examples

- `X = a` $\rightarrow$ $\surd$with $X = a$
- `a(X,3) = a(X,3,2)` $\rightarrow$ $\times$
- `a(X,3) = b(X,3)` $\rightarrow$ $\times$
- `a(X,3) = a(3,X)` $\rightarrow$ $\surd$with $X = 3$
- `a(X,3) = a(4,X)` $\rightarrow$ $\times$
- `a(X,b(c,d(e,f))) = a(b(c,Y),X)` $\rightarrow$ $X = b(c, d(e, f)),\ Y = d(e, f)$

Declarations

Two types of clauses:

- `p1(arg1, arg2, ...) :- p2(args,...) , p3(args,...) .`
  means if p2 and p3 true, then p1 is true. There can be arbitrary number of (conjunction of) predicates at right hand side.
- `p(arg1, arg2, ...) .`
  sometimes called a fact. It is equivalent to:
  `p(arg1, arg2, ...) :- true.`
- `p(args) :- q(args) ; s(args) .`
  Is disjunction of predicates. q or s implies p. Equivalent to:
  `p(args) :- q(args).`
  `p(args) :- s(args).`
- A prolog program is just a group of such clauses.

List Examples

```
% list membership
memb(X, [X|Rest]) .
memb(X, [_|Rest]) :- memb(X, Rest).

% concatenation
conc([],L,L).
conc([X|R] , L , [X|R_and_L]) :- conc(R, L, R_and_L).

% second list starts with first list
prefixof([],_).
prefixof([X|Rx], [X|Ry]) :- prefixof(Rx, Ry).

% second list contains first list
sublist(L1, L2) :- prefixof(L1, L2).
sublist(L, [_|R]) :- sublist(L, R).
```

Procedural Interpretation

- For goal clause all matching head clauses (LHS of clauses) are kept as backtracking points (like a junction in maze search)
- Starts from first match.
- To prove head predicate, RHS predicates need to be proved recursively.
- If all RHS predicates are proven, head predicate is proven.
- When fails, prolog goes back to last backtracking point and tries next choice.
- When no backtracking point is left, goal clause fails.
- All predicate matches go trough unification so goal clause variables can be instantiated.

Arithmetics and operators

- `X = 3+1` is not an arithmetic expression!
- operators (`is`) force arithmetic expressions to be evaluated
- all variables of the operations needs to be instantiated
  `12 is 3+X` does not work!
- Comparison operators force LHS and RHS to be evaluated:
  `X>Y, X<Y, X>=Y, X =< Y, X =:= Y, X == Y`
- is operator forces RHS to be evaluated: `X is Y+3*Y` Y needs to have a numerical value when search hits this expression.
- note that `X is X+1` is never successful in Prolog. Variables are instantiated once.

Greatest Common Divisor (Euler's)

$$gcd(m,n) = gcd(n, m-n) \text{ if } n < m$$
$$gcd(m,n) = gcd(n, m) \text{ if } m < n$$

```
gcd(X,X,X) .
gcd(X,Y,D) :- X < Y, Y1 is Y-X, gcd(X,Y1,D).
gcd(X,Y,D) :- Y < X, gcd(Y, X, D).
```