Resources

• Download:
  http://www.swi-prolog.org/dl-stable.html

• Documentation:
  (You don’t necessarily need to read. But good for reference when you have questions.)
  http://www.swi-prolog.org/dl-doc.html
  Menu “Help -> Online Manual” (HTML files in directory "doc")

Query Prompt

• query prompt
  ?- (Enter goals after “?-”)
  Example: ?- help().

• Load a file with facts
  ?-[swi('myprogram/test.pl')].
  or
  ?-[swi('myprogram/test')].
  (myprogram must be a subdirectory in the swi-prolog program directory)

User Interaction

?- parent(bob,sam).
   (a query must end with .)
   Yes
   ?- parent(bob,jill).
   No
   ?- parent(bob, X),
      |  parent(X,sam)
      |
   (user can use multiple lines to write a query, using |
   No
   ?- parent(X, sam).
   X = jill ;
   X = bob
   (user typed ; to ask for more answers.)

Debugging

?- trace, parent(X, sam).
   Call: (8) parent(_G944, sam) ? creep
   Call: (9) mother(_G944, sam) ? creep
   Exit: (9) mother(jill, sam) ? creep
   Exit: (8) parent(jill, sam) ? creep
   X = jill ;
   Redo: (9) parent(_G944, sam) ? creep
   Call: (9) father(_G944, sam) ? creep
   Exit: (9) father(bob, sam) ? creep
   Exit: (8) parent(bob, sam) ? creep
   X = bob
   More details in section 2.9 and 4.2.8 of the manual
Prolog Syntax

Basic Syntax

- `<clause>` ::= `<fact>` | `<rule>`
- `<fact>` ::= `<term>` .
- `<rule>` ::= `<term>` :- `<termlist` .
- `<termlist>` ::= `<term>` | `<term>` , `<termlist`.
- `<term>` ::= `<variable>` | `<constant>` | `<compound-term`.
- `<constant>` ::= `<number>` | `<atom`.
- `<compound-term>` ::= `<atom` ( `<termlist`).

Arithmetic

- Arithmetic operation can use prefix or infix notations.
  - `+(3,4)`
  - `3+4`
- Value is not immediately evaluated.
  - `?- write(3+5).
  - ?- X = 3+5.`
  - `[is a predicate that evaluates 3+5]`
  - `?- X = 4+3.`
  - `[these are two different terms]`
  - `No.`
- `?- X = 3+6, Y = 4+3, X = Y.`
  - `[unification]`
  - `X=7, Y=7`

Unification

- The semantics of `=` is determined by unification, i.e., `=` forces unification.
  - `(See unification algorithm in Page 556)`

  - `?- me = me.`
    - Yes
  - `?- me = you.`
    - No
  - `?- me = X.`
    - `X=me`
  - `?- f(a,X) = f(Yb).`
    - `X=bt`
  - `?- f(X) = g(X).`
    - No

List

  - `?- [H|T]=[1,2,3].`
    - `H = 1, T = [2,3]`
  - `?- [H1,H2|T]=[1,2,3].`
    - `H1 = 1, H2 = 2, T = [3]`
  - `?- [H1,H2,H3|T]=[1,2,3,4,5].`
    - `H1 = 1, H2 = 2, H3 = 3, T = [4,5]`
List Operations

- Concatenation:
  \[- X = [0,1|[2,3,4]].
  X = [0,1,2,3,4]

- Get elements, or tail:
  \[- [H1,H2|[3,4]] = [0,1|[2,3,4]].
  What do we get?

Define List Operation Predicates

- \text{cons}(X,Y,L) \iff L = [X|Y].
- \text{cons}(0,\{1,2,3\},A).
- \text{cons}(X,Y,\{1,2,3\}).

- Rewrite \text{cons}:
  \text{cons}(X,Y,[X|Y]).

Define List Operation Predicates

- \text{append}(X,Y,Z) \iff X = [\ ], Y = Z.
- \text{append}(X,Y,Z) \iff X = [A|B], Z = [A|W], \text{append}(B,W).

- Another definition
  \text{append}([],Y,Z).
  \text{append}([A|B],Y,[A|W]) \iff \text{append}(B,W).
- \text{append}(X,Y,\{1,2\}).

Resolution and Unification

- Order matters:
  - The order to resolve subgoals.
  - The order to use clauses to resolve subgoals.

- Thus programmers must know the orders used by the language implementations, in order to write efficient or even correct program. (Search Strategies)
Prolog’s Strategy

• Depth-first search
  – The order to resolve subgoals.
    (left to right)
  – The order to use clauses to resolve subgoals.
    (top to bottom)

• Backtrack:
  try another clause when it fails.

Example 1

• Facts:
  \( \text{ancestor}(X, Y) :- \text{ancestor}(X, Z), \text{parent}(Z, Y). \)
  \( \text{ancestor}(X, Y) :- \text{parent}(X, Y). \)
  \( \text{parent}(X, Y) :- \text{mother}(X, Y). \)
  \( \text{parent}(X, Y) :- \text{father}(X, Y). \)
  \( \text{father}(\text{bill}, \text{jill}). \)
  \( \text{father}(\text{bill}, \text{jill}). \)
  \( \text{mother}(\text{jill}, \text{sam}). \)
  \( \text{father}(\text{bob}, \text{sam}). \)

• Queries:
  \( ?- \text{ancestor}(\text{bill}, \text{sam}). \)

Example 1

• Facts:
  \( X_1 = \text{bill}, Y_1 = \text{sam} \)

Example 1

• Facts:
  \( \text{ancestor}(X, Y) :- \text{ancestor}(X, Z), \text{parent}(Z, Y). \)
  \( \text{ancestor}(X, Y) :- \text{parent}(X, Y). \)
  \( \text{parent}(X, Y) :- \text{mother}(X, Y). \)
  \( \text{parent}(X, Y) :- \text{father}(X, Y). \)
  \( \text{father}(\text{bill}, \text{jill}). \)
  \( \text{father}(\text{bill}, \text{jill}). \)
  \( \text{mother}(\text{jill}, \text{sam}). \)
  \( \text{father}(\text{bob}, \text{sam}). \)

• Queries:
  \( ?- \text{ancestor}(\text{bill}, \text{sam}). \)

Example 1

• Facts:
  \( \text{ancestor}(X_1, Y_1) :- \text{ancestor}(X_2, Y_2), \text{parent}(Z_2, Y_2). \)
  \( \text{ancestor}(X_2, Y_2) :- \text{parent}(X_2, Z_2), \text{parent}(Z_2, Y_2). \)

• Queries:
  \( ?- \text{ancestor}(\text{bill}, \text{sam}). \)

Resulting in an infinite loop.
Original order was bad
Example 2

• Facts:
  ancestor(X,Y) :- parent(X,Y).
  ancestor(X,Y) :- ancestor(X,Z), parent(Z,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).

What will happen?
Note that we change the order of the first two clauses in facts.

Example 3

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

• Queries:
  ?- ancestor(X,bob).

What will happen?
Note that we change the order of the two subgoals in clause (2).

Example 4

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

• Queries:
  ?- ancestor(X,bob).

What will happen?
Note that we change the order of the two subgoals in clause (2).

Loops and Control:
fail and cut (!)
fail

- Loops:
  Enforce backtrack even when an answer is found (using built-in predicate fail)

Example

- Print all solutions of appending.
  \[
  \text{printpieces}({L}) \leftarrow \text{append}(X,Y,L), \\
  \text{write}(X), \\
  \text{write}(Y), \\
  \text{nl}, \\
  \text{fail}.
  \]

? - printpieces([1,2]).
  
  [[1,2]]
  [1][2]
  [1,2][1]
  No

Example

- num(0).
  num(X) :- num(Y), X is Y+1.
  ?- num(X).

- writenum(I,J) :- num(X),
  I <= X,
  X <= J,
  write(X),
  nl,
  fail.
  ?- writenum(1,10).

Example

- writenum(I,J) :- num(X),
  I <= X,
  X <= J,
  write(X),
  nl,
  X >= I, !,
  fail.
  ?- writenum(1,10).

cut

- Cut (using built-in predicate !) branches in the search tree (to avoid infinite loop).

  "prunes" the search tree of all other siblings to the right of the node containing !.