### How Programming in Prolog Works

- **algorithm = logic + control**
  - logic refers to the rules and queries in a logic program
  - control refers to how the language computes a response to a query

### Control in Prolog

- Control in Prolog is characterized by two decisions:
  - **Goal Order**
    - Choose the leftmost subgoal
  - **Rule Order**
    - Select the first applicable rule

### Highly Simplified Control Algorithm

1. start with a query as the current goal;
2. while the current goal is nonempty do
3.   choose the leftmost subgoal;
4.   if a rule applies to the subgoal then
5.     select the first applicable rule;
6.     form a new current goal
7.   else
8.     backtrack
9. end if
10. end while;
11. succeed

### Sample Prolog Program

```prolog
female(mary).
pARENT(C,M,F) :- mother(C,M),father(C,F).
mother(john,ann).
mother(mary,ann).
father(mary,fred).
father(john,fred).
```
Internal Prolog DB

<table>
<thead>
<tr>
<th>female(mary).</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent(C,M,F) :- mother(C,M),father(C,F).</td>
</tr>
<tr>
<td>mother(john,ann).</td>
</tr>
<tr>
<td>mother(mary,ann).</td>
</tr>
<tr>
<td>father(mary,fred).</td>
</tr>
<tr>
<td>father(john,fred).</td>
</tr>
</tbody>
</table>

Internal Representation of Query

?- parent(mary,M,F),parent(john,M,F).

\[
\begin{array}{c}
\text{parent(mary,M,F)} \\
\text{parent(john,M,F)}
\end{array}
\]

The Flow of Satisfaction

- Satisfying a goal involves:
  1) Searching the database for a matching clause (and then)
  2) Marking the place in the database (and then)
  3) Satisfying any subgoals
- Failure can occur because all the alternative clauses for a goal have been tried, or the user typed a semicolon. In this case, Prolog backtracks in order to resatisfy goals
- Resatisfying a goal involves:
  1) Uninstantiating all variables that had been instantiated
  2) Resume searching from the place marker
  3) If another match is found, things continue as before ...

Matching

- Prolog attempts to match its search goals to facts or the head of rules (all variables are initially uninstantiated)
- Uninstantiated variables will match any object
- Integers and atoms match themselves
- Structures match other structures with the same functor and number of arguments, and all the corresponding arguments must match
- When two uninstantiated variables are matched together, they are said to “share”. i.e. as soon as one is instantiated, so is the other and with the same value
Tracing Model for Prolog Programs

- There are 4 kinds of events that occur:
  - CALL: A CALL event occurs when Prolog starts trying to satisfy a goal
  - EXIT: An EXIT event occurs when some goal has just been satisfied
  - REDO: A REDO event occurs when the system comes back to a goal, trying to resatisfy it
  - FAIL: A FAIL event occurs when a goal fails

```
(trace) ?- parent(mary,M,F),parent(john,M,F).
Call: (8) parent(mary, _G284, _G285) ? creep
   Call: (9) mother(mary, _G284) ? creep
Exit: (9) mother(mary, ann) ? creep
   Call: (9) father(mary, _G285) ? creep
Exit: (9) father(mary, fred) ? creep
Exit: (8) parent(mary, ann, fred) ? creep
   Call: (8) parent(john, ann, fred) ? creep
Exit: (9) mother(john, ann) ? creep
   Call: (9) father(john, ann) ? creep
Exit: (9) father(john, fred) ? creep
Exit: (8) parent(john, ann, fred) ? creep
M = ann
F = fred ;
No
```

Prefix, Suffix and Append

```
append([], Y, Y).
append([H|X],Y,[H|Z]) :- append(X,Y,Z).
prefix(X,Z) :- append(X,Y,Z).
suffix(Y,Z) :- append(X,Y,Z).
```

Example: Sublist

- We can specify a relation for a sublist $S$ of $Z$ in two different ways:
  - $\text{prefix } X \text{ of } Z \text{ and } \text{suffix } S \text{ of } X$
  - $\text{suffix } S \text{ of } X \text{ and } \text{prefix } X \text{ of } Z$

- If $S$ is a sublist of $Z$, everything is fine. But if not, we run into trouble
  ```
  ?- prefix(X,[a,b,c]), suffix([e],X).
  no
  ?- suffix([e],X), prefix(X,[a,b,c]).
  [infinite computation]
  ```
The “Cut” mechanism

- The “cut” is a special mechanism that allows you to tell Prolog not to consider certain choices again when it backtracks through the chain of satisfied goals
- Syntactically, a use of cut in a rule looks just like the appearance of a goal which has the predicate “!” and no arguments
- As a goal, this succeeds immediately and it cannot be resatisfied
- Encountering a cut has a side-effect, which alters how normal backtracking works
How “Cut” works

• When a cut is encountered as a goal, the system thereupon becomes committed to all choices made since the parent goal was invoked. All other alternatives are discarded. Hence an attempt to resatisfy any goal between the parent goal and the cut goal will fail.
• One can say that the choices are cut or frozen, that the system commits itself to choices made or that alternatives are discarded.

Operation of “cut”

• One can also look at the cut symbol as being like a fence that separates goals.
• Consider the conjunction of goals:
  foo :- a, b, c, !, d, e, f.
• Prolog will backtrack among goals a, b, and c, until the success of c causes the “fence” to be crossed to the right to reach goal d. Then backtracking can occur among d, e, and f, perhaps satisfying the entire conjunction several times.
• If d fails, causing the “fence” to be crossed to the left, then no attempts will be made to resatisfy goal c; the entire conjunction of goals, and the goal foo will also fail.