Procedural Prolog

- Prolog combines procedural and non-procedural programming techniques.
- Prolog’s control strategy – based on simple depth-first search.

Conditional Execution

- Prolog procedures can have multiple definitions (clauses) – each applying under different conditions.
- Conditional execution (if or case statements) – expressed with alternative definitions of procedures.

Example - Java switch/case statement

```java
public static void printNum(int X) {
    switch(X) {
        case 1:
            System.out.println(" One");
            break;
        case 2:
            System.out.println(" Two");
            break;
        case 3:
            System.out.println(" Three");
            break;
    }
}
```
Conditional Execution

In Prolog, `printNum` has three definitions:

```
printNum(1):- write('One').
printNum(2):- write('Two').
printNum(3):- write('Three').
```

Example

```
Pascal
procedure a(X:integer);
begin
  b;
  if X=0 then c else d;
  e
end;
```

```
Prolog
a(X):- b,
    cd(X),
e.
cd(0):- c.
    cd(X):= \+ X = 0, d.
```

Conditional Execution

Common mistakes - inefficient:

```
printNum(X): -
  X=1, write('One').
printNum(X): -
  X=2, write('Two').
printNum(X): -
  X=3, write('Three').
```

Gives correct results but waste time - Execute each clause, perform test, and backtrack out.

The "IF-THEN-ELSE" Structure

Can be implemented in Prolog as:

```
Goal1 -> Goal2 ; Goal3
"if Goal1 then Goal2 else Goal 3"
```

Meaning:

Test whether Goal1 succeeds, and if so, execute Goal2, otherwise execute Goal3.

Conditional Execution

Effective programming in Prolog:

- Make each logical unit of the program into a separate procedure.
- Each if or case statement should become a procedure call – decisions are made by procedure-calling process – choosing the right clause.

Example (simple if-then-else):

```
writeNum(X): - X=1 -> write('one') ; write('Not one').
```

Meaning:

If X = 1 the ‘One’ will be written, if not (else) ‘Not one’ will be written.
The “IF-THEN-ELSE” Structure

- Example (nested if-then-else):

```prolog
writeNum(X):-
  (X=1 -> write(one)
   ; X=2 -> write(two)
   ; X=3 -> write(three)
   ; write('out of range'))).
```

The “IF-THEN-ELSE” Structure

- If-then-else structure – for making decision without calling procedures.

- Discouraged

  - Looks like ordinary structured programming
  - Prolog clauses are supposed to be logical formulas.

Controlling Backtracking

- Uncontrolled backtracking may cause inefficiency in a program.

- Control using ‘cut’ facility.

  - The symbol is ‘!’.  
  
  - Function – prevent backtracking.

  - Useful – relieves the programmer of the burden of programming backtracking explicitly.

---

Backtracking

- Prolog will automatically backtrack – for satisfying a goal.

- In console, Prolog will backtrack automatically after we press “;”.

- To force backtracking use fail/0.
Controlling Backtracking

Example: (Prolog)

writename(1):- write('one'), !.
writename(2):- write('two'), !.
writename(3):- write('three'), !.
writename(_):- write('out of range').

Because, anonymous variable is used in the last clause. This variable will match with any value.

Controlling Backtracking

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

Why?? Prolog will automatically backtrack to the second rule, right??

Correct... but in this case its different. Cut "!" will prevent Prolog from backtrack. So, there is no attempt to look for the second alternative.

Controlling Backtracking

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

So, I guess this example can be modified this way, right??

writename(1):- write('one'), !.
writename(2):- write('two'), !.
writename(3):- write('three'), !.
writename(_):- write('out of range').

Fast learner... excellent...

Controlling Backtracking

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

Cut "!" operator will tell the computer to ignore other alternatives.

Controlling Backtracking

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

Meaning that, it will prevent backtracking...

Cut "!" operator will tell the computer to ignore other alternatives.

Controlling Backtracking

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

I don’t understand...

Now, look at this example

Controlling Backtracking - Discussion

Example: (Prolog)

b:- c, d, l, e, f.
b:- g, h.

Given that:

max(X,Y,Max).

Where Max = X if X is greater than or equal to Y, and Max = Y if X is less than Y.

max(X,Y,XX):- X>=Y.
mmax(X,Y,YY):- X<Y.
Controlling Backtracking - Discussion

max(X,Y,X):- X>=Y.
max(X,Y,Y):- X<Y.

These rules are mutually exclusive.
- If the first one succeeds then the second one will fail.
- If the first one fails then the second must succeed.

Making a goal deterministic without cuts

Example:
?- food(A).
A = rice ;
A = ice_cream ;
A = banana

?- once(food(A)).
A = rice

More economical formulation

instead of:
if X>=Y then Max = X
if X<Y then Max = Y
else Max = Y
max(X,Y,X):- X>=Y.
max(X,Y,Y):- X<Y.

Goal Always Succeed or Always Fail

In order to control the program flow, there is a need:
- to guarantee that a goal will succeed - regardless of the results.
- to guarantee that a goal will always fail.

Making a goal deterministic without cuts

Instead of creating deterministic predicates, we can define nondeterministic predicates in the ordinary manner and then block backtracking when we call them.

Special built-in predicate once/1.

To define once/1 as:

once(Goal):- call(Goal), !.
Recursion

- A procedure that calling itself to perform the tasks inside its tasks until the stopping condition is reached.

- Must have at least two clauses:
  - Basic clause – to stop the recursion.
  - Recursive clause – the one that call and reference to itself.

Example

```prolog
display_num(0).

display_num(X):-
write(X),
NewX is X - 1,
display_num(NewX).
```

Rule

Example

```prolog
in(city_plaza, alor_star).
in(alor_star, kedah).
in(kedah, malaysia).
in(malaysia, south_east).
in(south_east, asia).
in(asia, world).
```

Recursive Rule
Recursive Rule

Example (step-by-step)

\(? - \text{is\_in(city\_plaza, world).}\)

Working memory:
- \(X = \text{city\_plaza}\)
- \(Y = \text{world}\)
- Fact \(\text{is\_in(city\_plaza, world)}\) not exist

\(\text{is\_in(O,Y)}\)
\(\text{is\_in(O,Y)}\)
\(\text{is\_in(T,Y)}\)

Call is\_in(alor\_star, world)