UUM
Universiti Utara Malaysia

STIN2024
Pengaturcaraan Logik | Programming Logic

The Eminent Management University

UUM
Universiti Utara Malaysia

September
1st Session 2013/2014 (A131)

Wan Hussain Wan Ishak

School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia

(P) 04-9284786 (F) 04-9284753
(E) hussain@uum.edu.my
(U) http://wanhussain.com

STIN2024
Pengaturcaraan Logik | Programming Logic

The Eminent Management University

UUM
Universiti Utara Malaysia

Lecture notes

Expressing Procedural Algorithm

- Conditional execution
- “if-then-else” structure
- repetition through backtracking
- recursion

STIN2024
Pengaturcaraan Logik | Programming Logic

The Eminent Management University

Procedural Prolog

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

The Eminent Management University

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.

The Eminent Management University

Conditional Execution

- Example - Java switch/case statement

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

The Eminent Management University

Conditional Execution

- In Prolog, **printNum** has three definitions:

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

Conditional Execution

- 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.

The "IF-THEN-ELSE" Structure

- 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):

```
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.

Backtracking

- Prolog will automatically backtrack – for satisfying a goal.
- In console, Prolog will backtrack automatically after we press “
- To force bactracking use fail/0.

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.

Controlling Backtracking

Example (Pascal)

```
procedure writename(X:integer);
begin
    case X of
        1: write(‘one’);
        2: write(‘two’);
        3: write(‘three’);
    else
        write(‘out of range’)
    end
end;
```

```
writename(1):- write(‘one’).
writename(2):- write(‘two’).
writename(3):- write(‘three’).
writename(X):- X < 1,
            write(‘out of range’).
writename(X):- X > 3,
            write(‘out of range’).
```

Correct but lack of *conciseness*

Controlling Backtracking

```
writename(1):- write(‘one’).
writename(2):- write(‘two’).
writename(3):- write(‘three’).
writename(X):- X < 1,
        write(‘out of range’).
writename(X):- X > 3,
        write(‘out of range’).
```

Can be re-written as

```
writename(1):- write(‘one’).
writename(2):- write(‘two’).
writename(3):- write(‘three’).
writename(_):- write(‘out of range’).
```

Controlling Backtracking

Example: (Pr

But why???

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, !, 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.

The Eminent Management University

Controlling Backtracking

Meaning that, it will prevent backtracking …

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

The Eminent Management University

Controlling Backtracking

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

So, I guest 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…

The Eminent Management University

Controlling Backtracking

- Example: (Prolog)

I don't understand…

b:- c, d, !, e, f.

b:- g, h.

Now, look at this example

The Eminent Management University

Controlling Backtracking - Discussion

- Given that:

max(X,Y,Max).

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

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

The Eminent Management University

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:

Facts:
```
food(rice).
food(ice_cream).
food(banana).
```

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

All alternatives were taken out

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

Only the first fact return.

Controlling Backtracking - Discussion

- More economical formulation

instead of:

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

Goal Always Succeed or Always Fail

- Always succeed:
  - Used true/0.
  - Example:

    ?- eat(ahmad,fish); true.
- Always fail:
  - Used fail/0.
  - Example:

    writeNum(X):- X> 0, write('more than 0'), fail.

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.

The Eminent Management University

Rule

- Example

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

is_in(X,Y):-
in(X, T),
in(T,T2),
in(T2,T3).
in(T3, T4),
...

?

? – is_in(city_plaza, world).

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

Recursive rule

The Eminent Management University

Recursion

- Example

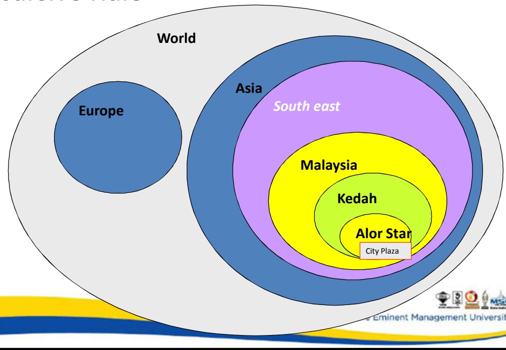
display_num(0). → Basic clause

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

Recursive clause

The Eminent Management University

Recursive Rule



Eminent Management University

Rule

- Example

in(city_plaza, alor_star).
in(alor_star, kedah).

? – is_in(city_plaza, kedah).

is_in(X,Y):-
in(X, T),
in(T,Y).

in(city_plaza, alor_star).
in(alor_star, kedah).
in(kedah, malaysia).

? – is_in(city_plaza, malaysia).

is_in(X,Y):-
in(X, T),
in(T,T2),
in(T2,Y).

The Eminent Management University

Recursive Rule

- Example

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

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

The Eminent Management University

Recursive Rule

- Example (step-by-step)

? – is_in(city_plaza, world).

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

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

Working memory:

X = city_plaza
Y = world

** Fact in(city_plaza, world) not exist*

Recursive Rule

- Example (step-by-step)

? – is_in(city_plaza, world).

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

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

Working memory:

X = alor_star
Y = world
T = kedah

Call is_in(kedah, world)

Recursive Rule

- Example (step-by-step)

? – is_in(city_plaza, world).

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

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

Working memory:

X = city_plaza
Y = world
T = alor_star

Call is_in(alor_star, world)

Recursive Rule

- Example (step-by-step)

? – is_in(city_plaza, world).

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

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).

Working memory:

X = alor_star
Y = world

** Fact in(alor_star, world) not exist*