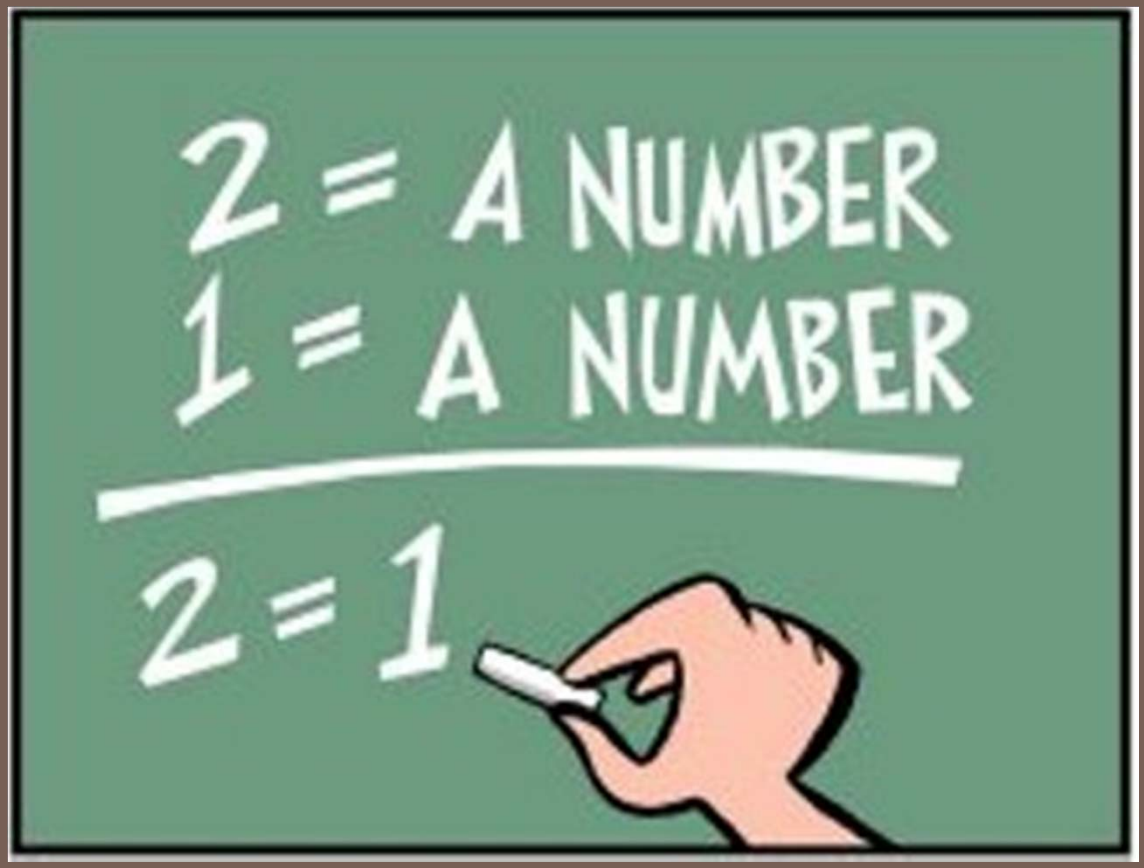
LOGIC PROGRAMMING



Lecture 4: Backtracking and Cut

Ahmed Sallam

Backtracking and Cut

- Review Backtracking
- Cut
- Cut uses
- Problems with Cut

Backtracking

Example

```
pair(X,Y):-
   boy(X),girl(Y).
boy(johm).
boy(marmaduke).
boy(bertram).
boy(charles).
girl(griselda).
girl(ermitrude).
girl(brunhilda).
```

```
?- pair(X,Y).
X=john, Y=griselda ;
X=john, Y=ermintrude ;
X=john, Y=brunhilde ;
X=marmaduke, Y=griselda ;
X=marmaduke, Y=ermintrude ;
X=marmaduke, Y=brunhilde ;
X=bertram, Y=griselda ;
...
```

12 solutions.

Backtracking and Cut

- Review Backtracking
- Cut
- Cut uses
- Problems with Cut

The "Cut"

Cut (written "!") tells the system which previous choices need not to be considered again when it backtracks.

Advantages:

- The program will run faster. No time wasting on attempts to re-satisfy certain goals.
- The program will occupy less memory. Less backtracking points to be remembered.

Example of Cut

- sum_to is a predicate to compute he summation from N to 1.

```
1. sum_to(1,1).
2. sum_to(N, R):-
3.          N1 is N-1,
4.           sum_to(N1,R1),
5.           R is N+R1.
```

- The program will recursively call sum_to and each time N reduced by one until reaching **sum_to (1, R)**
- The problem is that, when prolog tries to match **sum_to(1,R)** both predicates in line 1 and 2 are a valid match. So the program will enter infinite loop after the first solution.

Example of Cut

- Possible solution is to add line 3 to the program

```
1. sum_to(1,1).
2. sum_to(N, R):-
3.          N1 > 1, % or \+ ( N1=<1)
4.          N1 is N-1,
5.           sum_to(N1,R1),
6.           R is N+R1.
```

(\+) is a built in predicate in prolog used as (Not) operation

- Even this solution will solve the problem, the prolog still backtrack to **sum_to(N, R)** as an alternative solution for **sum_to(1, R)**.

Example of Cut

- A perfect solution is to add "!" in line 1.

```
1. sum_to(1,1):-  ! .
2. sum_to(N, R):-
3.          N1 is N-1,
4.           sum_to(N1,R1),
5.           R is N+R1.
```

- Now, after prolog matches **sum_to(1, R)** with the first alternative **sum_to(1,1)**. The Cut "!" will succeed and prevent he prolog to backtrack for other alternatives.

Backtracking and Cut

- Review Backtracking
- Cut
- Common uses of Cut
- Problems with Cut

Common Uses of Cut

Three main cases:

1. To tell the system that it found the right rule for a particular goal. *Confirming the choice of a rule*.
2. To tell the system to fail a particular goal without trying for alternative solutions. *Cut-fail combination.*
3. To tell the system to terminate the generation of alternative solutions by backtracking. *Terminate a "generate-and-test."*

Confirming the Choice of a Rule (use 1)

- Exatlly like the previous example

```
1. sum_to(1,1):-  ! .
2. sum_to(N, R):-
3.          N1 is N-1,
4.           sum_to(N1,R1),
5.           R is N+R1.
```

Fail


```
fail.
```

- Built-in predicate.
- No arguments.
- Always fails as a goal and causes backtracking.

The "Cut-fail" Combination (Use 2)

`fail` after `cut`:

- The normal backtracking behavior will be altered by the effect of cut.
- Quite useful combination in practice.

The Fees discount

- Suppose there is a discount for 3rd grade students on the college fees this year. However, this discount is not applied on Foreign students.
- So if we have a foreign student x, the discount predicate should fail. We can write this as following

```
1. grade3_fees(X, Discount,Fees):- foreigner(X), fail.
2. grade3_fees(X,Discount, Fees):- Fees is 200-Discount.
```

- The problem is: Even, 'fail' will cause line one to fail it will also force backtracking to the second line which will apply the discount for the foreign student.

The Fees discount

- A good solution is to combine '!' with fail to prevent the backtracking. This means that predicate in line one will fail (no discount for foreigners) , at the same time, prolog will not backtrack to other alternatives.

```
1. grade3_fees(X, Discount,Fees):- foreigner(X), !, fail.
2. grade3_fees(X,Discount, Fees):- Fees is 200-Discount.
```

- Another readable solution for this is:

```
grade3_fees(X,Discount,Fees):- \+foreigner(X), Fees is 200-Discount.
```

Terminating a "Generate-and-Test" (Use 3)

"Generate-and-Test":

- One of the simplest AI search techniques.
- Generate: Generate all possible solutions to a problem.
- Test: Test each to see whether they are a solution.
- A possible solution is generated and then tested.
- If the test succeeds a solution is found.
- otherwise, backtrack to next possible solution.

Example 1: Tic-Tac-Toe

Tic-Tac-Toe game: Get three in a raw, column, or diagonal:

X		O
O	O	
X	X	X

X	X	O
O	X	
O	X	

O		O
X	O	
X	X	O

Representation:

1	2	3
4	5	6
7	8	9

Tic-Tac-Toe

- We can represent the successful lines in the board as following:

```
line(b(X,Y,Z,_,_,_,_,_,_), X, Y, Z).
line(b(_,_,_,X,Y,Z,_,_,_), X, Y, Z).
line(b(_,_,_,_,_,_,X,Y,Z), X, Y, Z).
line(b(X,_,_,Y,_,_,Z,_,_), X, Y, Z).
line(b(_,X,_,_,Y,_,_,Z,_), X, Y, Z).
line(b(_,_,X,_,_,Y,_,_,Z), X, Y, Z).
line(b(X,_,_,_,Y,_,_,_,Z), X, Y, Z).
line(b(_,_,X,_,Y,_,Z,_,_), X, Y, Z).
```

Tic-Tac-Toe

- Consider the player o will have a threating if player x completed two cells in a successful line, and the third is empty.

```
threatening(e, x, x).
threatening(x, e, x).
threatening(x, x, e).
```

- Threatening here means that player o have no choice other than filling the empty cell.

Tic-Tac-Toe

- Now player o will provide our small program with the board status and want to make sure there is no threatening before his move.

```
forced_move(Board) :-
line(Board, X, Y, Z), %genrate
threatening(X, Y, Z), %test
!.
```

Tic-Tac-Toe

- Here, a complete view for this part of the program:

```
line(b(X,Y,Z,_,_,_,_,_,_), X, Y, Z).
line(b(_,_,_,X,Y,Z,_,_,_), X, Y, Z).
line(b(_,_,_,_,_,_,X,Y,Z), X, Y, Z).
line(b(X,_,_,Y,_,_,Z,_,_), X, Y, Z).
line(b(_,X,_,_,Y,_,_,Z,_), X, Y, Z).
line(b(_,_,X,_,_,Y,_,_,Z), X, Y, Z).
line(b(X,_,_,_,Y,_,_,_,Z), X, Y, Z).
line(b(_,_,X,_,Y,_,Z,_,_), X, Y, Z).

threatening(e, x, x).
threatening(x, e, x).
threatening(x, x, e).

forced_move(Board) :-
line(Board, X, Y, Z), %genrate
threatening(X, Y, Z), %test
!.
```

Tic-Tac-Toe

- If the player o provide the following goal:
  `forced_move(b(x,x,e,e,o,x,o,e,e)).`
- The program will return True and the cut will terminate the backtracking process because one threatening is enough for player o to know where he should play.

Example 2: Divide two numbers.

- Simply this program generate an integer starting from 0 and test the condition line 5.

```
1. divide(N1, N2, Result) :-
2.      is_integer(Result),
3.      Product1 is Result * N2,
4.      Product2 is (Result + 1) * N2,
5.      Product1 =< N1, Product2 > N1,
6.      !.
7. is_integer(0).
8. is_integer(X) :- is_integer(Y), X is Y + 1.
```

- If the condition met, then there is no need to find another number and the cut will terminate the backtracking process.
- Note that the backtracking happen because of **is_integer** predicate.

Example 2: Divide two numbers.

- The predicate is_integer(). Is responsible for generating an infinite number of integer numbers starting from 0.

```
7. is_integer(0).
8. is_integer(X) :- is_integer(Y), X is Y + 1.
```

- To understand this, you can ask prolog the following

```
?- is_integer(X).
X=0 ;
X=1 ;
X=2 ;
...
```

Problems with the Cut

- Consider the following adapted version of the append.

```
append([], X, X):-!.
append([A|B], C, [A|D]) :- append(B, C, D).
```

- Now, if we ask the following:

```
try append(X, Y, [a,b,c]).
```

- Normally, it should have many solutions. However, the cut operation '!' will terminate the backtracking after the first solution.