CS370

Symbolic Programming
Declarative Programming

LECTURE 7: Controlling Backtracking

Jong C. Park

park@cs.kaist.ac.kr

Computer Science Department

Korea Advanced Institute of Science and Technology

http://nlp.kaist.ac.kr/~cs370

Controlling Backtracking

- Preventing backtracking
- Examples using cut
- Negation as failure
- Problems with cut and negation

Preventing Backtracking

- Sample program

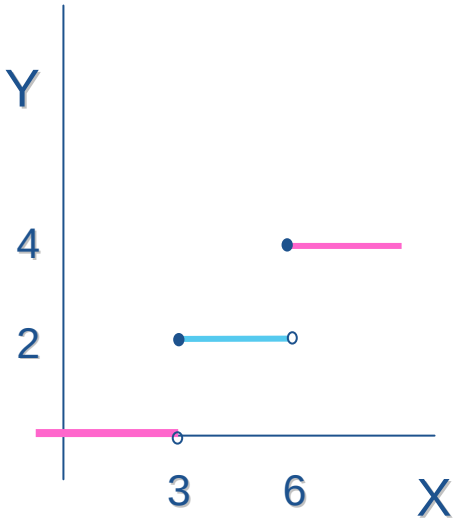
```
C :- P, Q, R, S, T, U.
A :- B, C, D.
P.
R :- true.
D.

?- A.
```

```
C :- V.
B.
Q :- true.
V.
```

Preventing backtracking

- A double-step function



A double-step function

- Rule specification

Rule 1: if X < 3 then Y = 0

Rule 2: if 3 <= X and X < 6
then Y = 2

Rule 3: if 6 <= X then Y = 4

Preventing backtracking

- Use the procedure f(X,Y)

```
f(X,0) :- X < 3.
f(X,2) :- 3 =< X, X < 6.
f(X,4) :- 6 =< X.
```

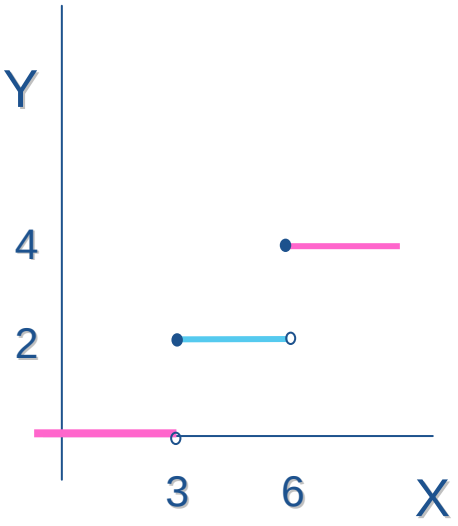
- Experiment 1

```
?- f(1,Y), 2 < Y.
```

  - Problem?

  - How do we fix it?

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.
```



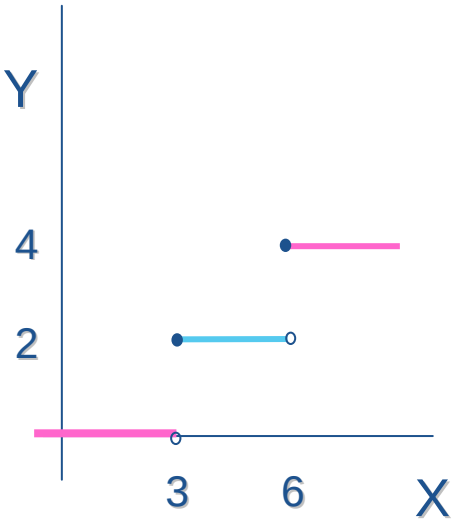
A double-step function

Preventing backtracking

- Experiment 2

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.

?- f(7,Y).
Y = 4
```



A double-step function

- Problem?
  -

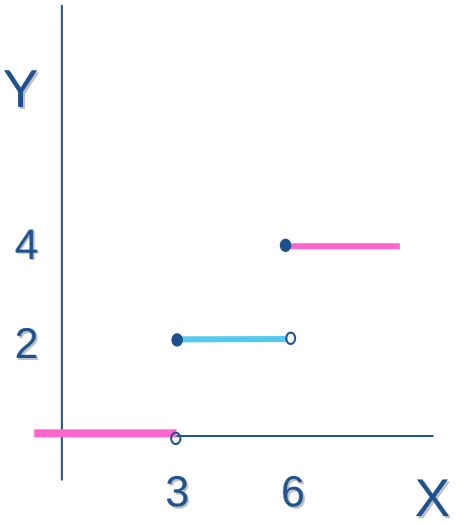
```
f(X,0) :- X < 3, !.
f(X,2) :- X < 6, !.
f(X,4).
```

Preventing backtracking

⊙ Experiment 3

```
f(X,0) :- X < 3.
f(X,2) :- X < 6.
f(X,4).

?- f(1,Y).
Y = 0;
Y = 2;
Y = 4
```



A double-step function

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.
```

Preventing backtracking

- H :- B_1, …, B_m, !, B_n, …, B_o.
  - Backtracking may happen within B_1 through B_m.
  - Once the cut is reached, alternatives for B_1 through B_m, as well as for H, are discarded.
  - Backtracking may happen within B_n through B_o.
  - Example

```
C :- P, Q, R, !, S, T, U.        C :- V.
A :- B, C, D.                    B.
P.                          Q :- … .
R :- … .                                 V.
D.
?- A.
```

Examples using cut

- Computing maximum
- Single-solution membership
- Adding an element to a list without duplication
- Classification into categories

Examples using cut

Computing maximum

- The procedure for finding the larger of two numbers: max(X,Y,Max)

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

- Improvements?

```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y).
```

- Problems?

```
?- max(3,1,1).
yes
max(X,Y,Max) :- X >= Y, !, Max = X; Max = Y.
```

Examples using cut

⊙ Single-solution membership

- Example

```
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).
```

```
member(X,[X|L]) :- !.
member(X,[Y|L]) :- member(X,L).

?- member(X,[a,b,c]).
X = a;
```

Examples using cut

- Adding an element to a list without duplicates
  - Example: add(X, L, L1)

```
add(X,L,L) :- member(X,L), !.          add(X,L,[X|L]).
?- add(a,[b,c],L).
L = [a,b,c]
?- add(X,[b,c],L).
L = [b,c]                              member(X,[X|L]) :- !.
X = b                                  member(X,[Y|L]) :- member(X,L).
?- add(a,[b,c,X],L).
?- add(X,[a],[a,a]).
```

  - Is the cut necessary?

```
add(X,L,L) :- member(X,L).             add(X,L,[X|L]).
?- add(a,[a,b,c],L).
L = [a,b,c];
L = [a,a,b,c]
```

Examples using cut

⊙ Classification into categories

- Example: a database of game results

```
beat(tom,jim).  beat(ann,tom). beat(pat,jim).
```

winner: every player who won all his or her games

fighter: any player who won some games and lost some

sportsman: any player who lost all his or her games

```
class(X,fighter) :- beat(X,_), beat(_,X), !.
class(X,winner) :- beat(X,_), !.
class(X,sportsman) :- beat(_,X).
?- class(tom,C).
C = fighter
?- class(tom,sportsman).
```

Negation as failure

⊙Example

- Mary likes all animals.

```
likes(mary,X) :- animal(X).
```

- Mary likes all animals but snakes.

```
likes(mary,X) :- snake(X), !, fail.
likes(mary,X) :- animal(X).
likes(mary,X) :- snake(X), !, fail; animal(X).

different(X,X) :- !, fail.  different(X,Y).
different(X,Y) :- X = Y, !, fail; true.
```

Negation as failure

- The unary predicate not

```
not(P) :- P, !, fail; true.

likes(mary,X) :- animal(X), not snake(X).

different(X,Y) :- not(X = Y).

class(X,fighter) :- beat(X,_),beat(_,X).
class(X,winner) :- beat(X,_), not beat(_,X).
class(X,sportsman) :- beat(_,X), not beat(X,_).
```

Problems with cut and negation

- Advantages of using the cut facility
  - We can often improve the efficiency of the program.
  - We can specify mutually exclusive rules, enhancing the expressive power of the language.
- But we may lose the correspondence between the declarative and procedural meaning of programs.

```
p :- a, b.        p :- c.

p :- a, !, b.     p :- c.

p :- c.           p :- a, !, b.
```

Problems with cut and negation

- Types of cut
  - green cuts
    - no effect on the declarative meaning
  - red cuts
    - changes the declarative meaning
- *cut-fail* combination vs. not

Summary

- Preventing backtracking
- Examples using cut
- Negation as failure
- Problems with cut and negation