Controlling Backtracking

- Backtracking:
  - does normally exactly what one expects in terms of logic programming
  - control primitives offer extra control over backtracking

Enforcing Backtracking: Fail

- ?- fail.
  no (no match)
- Program:
  p(a).
  p(b).

  Query:
  ?- p(X). (match)
  X = a
  yes

Fail - no Recursion

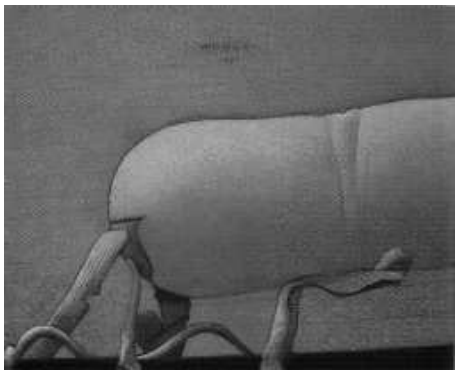
- Program:
  p(a).
  p(b).
  p(X) :- q(X).
  q(c).
- Query:
  ?- p(X), write(X), nl, fail.
  a
  b
  c
  no

backtracking

Fail - with Recursion

- Program:
  /*1*/ member(X, [X|_]).
  /*2*/ member(X, [_|Y]) :-
  member(X,Y).
- Query/call:
  ?- member(Z,[a,b]), write(Z), nl, fail.
  Z = X, X = a match 1
  ?- write(a), nl, fail.
  a **backtracking**
  ?- member(Z,[a,b]), write(Z), nl, fail.
  Z = X, Y = [b] match 2

Finger Cut by Grass



Controlling Backtracking: !

- Procedural meaning of the cut !:

A :- B1, B2, !, B3, B4.

Search for alternatives

Search for alternatives

!

Stop searching

Cut

Program:

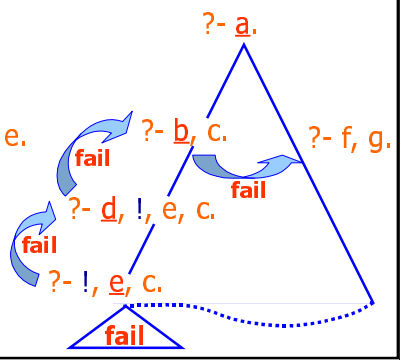
```
p(a).
p(b).
q(X) :- p(X), r(X).
r(Y) :- !, t(Y).
r(a).
t(c).
```

Execution:

```
?- q(Z).
     Z = X
?- p(X), r(X).
     X = a
?- r(a).
     Y = a
?- t(a).
     fail, no
     backtracking
     to r(a).
     Try X = b
```

Search Space and !

```
a :- b, c.
a :- f, g.
  ...
b :- d, !, e.
b :- ... .
  ...
d.
```



Various Applications of !

- Cut as commitment operator:

if X < 3 then Y = 0
if X ≥ 3 and X < 6 then Y = 2
if X ≥ 6 then Y = 4

- Prolog:

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

Commitment Operator

- Cut as commitment operator:

```
/*1*/  t(X, 0) :- X < 3.
/*2*/  t(X, 2) :- X >= 3, X < 6.
/*3*/  t(X, 4) :- X >= 6.
```

- Execution trace:

```
?- t(1, Y), Y > 2.                match 1
?- 1 < 3, 0 > 2.
?- 0 > 2.                         fail 1
?- 1 >= 3, 1 < 6, 1 > 2.          match 2
?-    ...                         fail 2
?- 1 >= 6, 4 > 2.   match 3, fail 3   ☹
```

Commitment Operator

- Cut as commitment operator:

```
/*1*/  t(X, 0) :- X < 3, !.
/*2*/  t(X, 2) :- X >= 3, X < 6, !.
/*3*/  t(X, 4) :- X >= 6.
```

- Execution trace:

```
?- t(1, Y), Y > 2.                match 1
?- 1 < 3, !, 0 > 2.
?- !, 0 > 2.                      fail 1
no                                  ☺
```

Various Applications of !

- Cut used for removal of conditions:

  min(X, Y) is X if X ≤ Y
  min(X, Y) is Y if X > Y

- Prolog:

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

- Execution:

```
?- min(3, 5, Z).
?- 3 =< 5.          match 1
Z = 3               yes
```

Removal of Conditions

- Cut used for removal of conditions:

```
min(X, Y, Z) :-
      X =< Y, !,
      Z = X.
min(X, Y, Y).
```

- Execution:

```
?- min(3, 5, W).
?- 3 =< 5, !, W = 3.      match 1
W = 3                     yes
```

Removal of Conditions

- Cut used for removal of conditions:

```
min(X, Y, Z ⇒ X) :-
      X =< Y, !.
      Z = X.          why included?
min(X, Y, Y).
```

- Execution:

```
?- min(3, 5, 5).
                        fail 1, match 2
yes
```

Change in Meaning?

- Cut used for removal of conditions:

```
min(X, Y, Z) :-
      X =< Y,       (! omitted)
      Z = X.
min(X, Y, Y).
```

- Execution:

```
?- min(3, 5, W), W = 5.
?- 3 =< 5, 5 = 3, W = 5.   match 1
?- 5 = 3, W = 5.           fail
?- W = 5.  match 2 (with Y = 5 = W)
W = 5
yes
```

Cut-fail Combination

- When a certain condition is satisfied, failure must be returned

```
b :- !, fail
```

- **Example:**

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

?- different(3, 3).
no
```

Negation by Failure

- Simulation of negation: not(p) is true if p is false (fails):

```
not(X) :- call(X), !, fail.
not(X).
```

- **Example:**

```
p(a).
q(X) :- p(X), not(r(X)).
r(c).
?- q(Y).
yes
```

Single Solution

- Circumvention of double search:

```
/*1*/  member(X, [X|_]) :- !.
/*2*/  member(X,[_|Y]) :-
          member(X,Y).
```

- **Example:**

```
?- member(a, [a, b, a]).
yes

?- member(X, [a, b]).
X = a;
no
```

Green and Red Cuts

- **Green cut:**
  - when omitted, does not change declarative (logical) meaning of program
  - used to increase efficiency
- **Red cut:**
  - when omitted, declarative meaning of program is changed
  - used for efficiency
  - used to enforce termination

Green and Red Cuts

- **Green cut:**
  - commitment operator
- **Red cut:**
  - removal of conditions
  - cut-fail combination
  - single solution