Logic and Functional Programming

Controlling Backtracking in Prolog

Dr. Cristian-Paul Bara

Computer Science Department
Faculty of Mathematics and Computer Science,
Babeş Bolyai University,
Cluj-Napoca, Romania

Table of Contents

Types of cuts

Green When used to prevent backtracking on branches known not to lead to a solution.

```
f(X,0):- X < 3.          f(X,0):- X < 3, !.
f(X,1):- 3 =< X, X < 6.  f(X,1):- 3 =< X, X < 6, !.
f(X,2):- 6 =< X.         f(X,2):- 6 =< X.
```

Red When used to change the logic of the program. Try to avoid.

```
f(X,0):- X < 3.   f(X,0):- X < 3, !.
f(X,1):- X < 6.   f(X,1):- X < 6, !.
f(X,2):-.         f(X,2):-.
```

Case instructions

Cuts are used to prevent checking other values once a previous goal is satisfied.

```
r(1):- !, a, b, c.
r(2):- !, d.
r(3):- !, e.
r(_):-write("default case").
```

```
r(X):- X=1, !, a, b, c.
r(X):- X=2, !, d.
r(X):- X=3, !, e.
r(_):-write("default case").
```

Whare will the output be?

```
p(1).
p(2).
q(3).
q(4).
r1(X, Y) :- !, p(X), q(Y).
r2(X, Y) :- p(X), !, q(Y).
r3(X, Y) :- p(X), q(Y), !.
```

Whare will the output be?

```
p(1).
p(2).
q(3).
q(4).
r1(X, Y) :- !, p(X), q(Y).
r2(X, Y) :- p(X), !, q(Y).
r3(X, Y) :- p(X), q(Y), !.
```

```
?- r1.
X=1 Y=3
X=1 Y=4
X=2 Y=3
X=2 Y=4
```

Whare will the output be?

```
p(1).
p(2).
q(3).
q(4).
r1(X, Y) :- !, p(X), q(Y).
r2(X, Y) :- p(X), !, q(Y).
r3(X, Y) :- p(X), q(Y), !.
```

```
?- r1.
X=1 Y=3
X=1 Y=4
X=2 Y=3
X=2 Y=4
?- r2.
X=1 Y=3
X=1 Y=4
```

Whare will the output be?

```
p(1).
p(2).
q(3).
q(4).
r1(X, Y) :- !, p(X), q(Y).
r2(X, Y) :- p(X), !, q(Y).
r3(X, Y) :- p(X), q(Y), !.
```

```
?- r1.
X=1 Y=3
X=1 Y=4
X=2 Y=3
X=2 Y=4
?- r2.
X=1 Y=3
X=1 Y=4
?- r3.
X=1 Y=3
```

Whare will the output be?

```
p([], 0).
p([H|T], S) :-
        H > 0,
        !,
        p(T, S1),
        S is S1 + H.
p([_|T], S) :- p(T, S).
```

Whare will the output be?

```
p([], 0).
p([H|T], S) :-
        H > 0,
        !,
        p(T, S1),
        S is S1 + H.
p([_|T], S) :- p(T, S).
```

```
?- p([1, 2, 3, 4], S).
S=10.
?- p([1, -1, 2, -2], S).
S=3.
```

Whare will the output be?

```
p([], 0).
p([H|T], S) :-
        !,
        H > 0,
        p(T, S1),
        S is S1 + H.
p([_|T], S) :- p(T, S).
```

Whare will the output be?

```
p([], 0).
p([H|T], S) :-
        !,
        H > 0,
        p(T, S1),
        S is S1 + H.
p([_|T], S) :- p(T, S).
```

```
? p([1, 2, 3, 4], S).
S=10.
? p([1, -1, 2, -2], S).
false
```

The "fail" predicate

```
p(a,b).
p(c,d).
p(e,f).
all :-
        p(X,Y),
        write(X),
        write(Y), nl,
        fail.
all1 :-
        p(X,Y),
        write(X),
        write(Y), nl.
```

- Forces backtracking.
- Same effect as an impossible predicate, e.g., 2=3
- No statement in a rule after fail will be evaluated
- Similar to a while loop:

```
while:-
        < condition >,
        < statements >,
        fail.
```

The "fail" predicate

```
p(a,b).
p(c,d).
p(e,f).
all :-
        p(X,Y),
        write(X),
        write(Y), nl,
        fail.
all1 :-
        p(X,Y),
        write(X),
        write(Y), nl.
```

```
?- all.
ab
cd
ef
false.
```

The "fail" predicate

```
p(a,b).
p(c,d).
p(e,f).
all :-
        p(X,Y),
        write(X),
        write(Y), nl,
        fail.
all1 :-
        p(X,Y),
        write(X),
        write(Y), nl.
```

```
?- all.
ab
cd
ef
false.
?-all1.
ab
true;
cd
true;
ef
true.
```

The "fail" predicate

```
p(a,b).
p(c,d).
p(e,f).
all :-
        p(X,Y),
        write(X),
        write(Y), nl,
        fail.
all1 :-
        p(X,Y),
        write(X),
        write(Y), nl.
```

```
?- all.
ab
cd
ef
false.
?-all1.
ab
true;
cd
true;
ef
true.
```

```
?-p(X,Y).
X = a,
Y = b;
X = c,
Y = d;
X = e,
Y = f.
```