Contents

The Cut

- Backtracking is a characteristic feature of Prolog
- But backtracking can lead to inefficiency:
  - Prolog can waste time exploring possibilities that lead nowhere
  - It would be nice to have some control
- The cut predicate !/0 offers a way to control backtracking

```
p(X) :- b(X), c(X), !, d(X), e(X).
```

- Cut is a goal that *always* suceeds*
- Commits Prolog to the choices that where made since the parent goal was called

What cut does

- The cut only commits us to choices made since the parent goal was unified with the left-hand side of the clause containing the cut
- For example, in a rule of the form

```
q :- p1, .., pn. !, r1, ... rn.
```

- When we reach the cut it commits us:
  - To this particular clause of q
  - To the choices made by p1, …, pn
  - NOT to choices made by r1, …, rn

Using Cut

- Consider the following predicate max/3 that succeeds if the third argument is the maximum of the first two

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

?- max(2, 3, 3).
true
?- max(7, 3, 7).
true
?- max(2, 3, 2).
false
?- max(2, 3, 5).
false
```

- What's the problem?
- There is a potential inefficiency
  - Suppose it is called with `?- max(3, 4, Y).`
  - It will correcrlt unify Y with 4
  - But when asked for more solutions, it will try to satisfy the second clause
  - This is completely pointless

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

- If the `X <= Y` succeeds, the cut commits us to this choice, and the second clause of max/3 is not considered
- If the `X <= Y` fails, Prolog goes on to the second clause

Green Cuts

- Cuts that do not change the meaning of a predicate are called *green cuts*
- The cut in max/3 is an example of a green cut
  - The new code gives exactly the same answers as the old version
  - But its more efficient

Red Cuts

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

- Cuts that change the meaning of a predicate are called *red cuts*
- The cut in the revised max/3 is an example of a red cut
  - If we take out the cut, we don't get an equivalent program
- Programs containing red cuts
  - Are not fully declarative
  - Can be heard to read
  - Can lead to subtle programming mistakes

Failure

- Another predicate in prolog is the fail/0
- This is a goal that will immediately fail when Prolog tries to proof it
- When Prolog fails, it tries to back track

Negation as Failure

- The cut-fail combination offers us some form of negation
- It is called *negation as failure(and defined as follows

```
neg(Goal) :- Goal, !, fail.
neg(Goal).
```

- Necause negation as failure is so often used, there is no need to define it
- In standard Prolog the prexfix operator `\+` means negation as failure
- It is not logicial negation, so changing the order of the goals gives different behaviour