Lecture notes 14.1

Prolog: the cut

COMP 2411, Session 1, 2004

1 How the cut works

The cut "!" is a special type of atom used to control Prolog's backtracking behaviour.

Suppose that a program contains a rule

$$\mathtt{H} :- \mathtt{A}_1, \ldots, \mathtt{A}_n, !, \mathtt{B}_1, \ldots, \mathtt{B}_m.$$

- When first encountered during resolution, the cut succeeds.
- When the cut is reached during backtracking, all subsequent branches descended from the goals whose leftmost subgoals are instances of $\mathtt{A}_1, \ldots, \mathtt{A}_n$ and H, are omitted from the search.

In other words, bactracking over the cut forces a *fail* on all atoms between the cut and the head of the rule it occurs in, and shifts the focus of backtracking to the point just before H was called.

No further ways of satisfying $\mathtt{A}_1, \ldots, \mathtt{A}_n$ are considered.

Nor are any further rules for H considered.

As a first example, consider the following program.

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

b(1).
b(2).

c(3).
c(4).

d(3).
d(4).

e(5).
```

Here are a few answers to to queries.

```
?- a(X).

X = 1 ;

X = 2 ;

X = 3 ;

No
?- a(4).

Yes
?- a(5).

Yes
```

The cut has destroyed the logical interpretation of the program: `X = 4` and `X = 5` are not among the solutions to the query `a(X)?` This cut is a *red* cut.

The cut does not restrict bactracking on goals "higher up" in the proof tree, as exemplified in the next example.

```
a(Y) :- b(Y),!.
a(2).

b(1).
b(2).
```

```
?- b(X),a(Y).

X = 1
Y = 1;

X = 2
Y = 1;

No
```

2 Applications

We can use the cut to eliminate the part of the search that is bound to fail.

For instance, there is only one way to reverse a list. Thus, once we have found a match, we can discard all attempts of finding alternative matches.

```
reverse_order(X,Y) :- reverse_order(X,[],Y).
reverse_order([], L, L) :- !.
reverse_order([X|Y], Z, T):- reverse_order(Y, [X|Z], T), !.
```

```
?- reverse_order([2,3,4],X).

X = [4, 3, 2] ;

No
?- reverse_order(X,[2,3,4]).

X = [4, 3, 2] ;

No
```

But SWI-Prolog uses a technique known as *indexing*: no attempt is made to match a goal G with the head H of a clause such that:

- either G and H and built from different predicate symbols, or
- the first arguments of G and H cannot be unified.

So if the previous example is only used with the first kind of query, there is no gain in efficiency; it is better to leave the cut out. On the other hand, the cut is useful if the previous example is to be used with the second kind of query.

Sometimes, the cut can be used to select a unique condition when two or more apply.

For instance, consider the following situation.

> Australian citizens are to pay 10% GST.
> US Citizens are exempt from GST, i.e., pay 0% GST.
> People with dual citizenship of Australia and the USA are to be treated as Australian citizens, i.e., they pay 10% GST.

Here the cut is used to take care of citizens who have dual citizenship:

```
gst(Person,10) :- citizen(Person,australia), !.
gst(Person,0)  :- citizen(Person,usa).

citizen(jones,australia).
citizen(jones,usa).

?- gst(jones,X).

X = 10 ;

No
```

We can use the cut to restrict the set of answers returned. For instance, consider the following program for membership.

```
member_of(X,[X|_]).
member_of(X,[_|R]) :- member_of(X,R).
```

We then get the behaviour:

```
?- member_of(2,[2,3,2,2,5]), X=1.

X = 1 ;

X = 1 ;

X = 1 ;

No
```

Now let us use the cut.

```
  member_of(X,[X|_]) :- !.
  member_of(X,[_|R]) :- member_of(X,R).
```

We then get:

```
member_of(2,[2,3,2,2,5]), X=1.

X = 1 ;

No
?- member_of(X,[4,5,6,7]).

X = 4 ;

No.
```

The last query shows that we have lost the ability to generate all possible answers...

A common mistake is to misuse the cut. Consider the following two programs for computing the minimum of two numbers.

```
min1(X,Y,X) :- X < Y, !.
min1(X,Y,Y).

min2(X,Y,X) :- X < Y, !.
min2(X,Y,Y) :- X >= Y.
```

Only the second program is logically correct (consider the query `min1(2,3,3)`).

As another example, consider the problem of coloring Australia's states with four different colors.

```
color(WA,NT,SA,QLD,NSW,VIC,TAS) :-
      next(WA,NT),
      next(WA,SA),
      next(NT,QLD),
      next(NT,NSW),
      next(NT,SA),
      next(SA,QLD),
      next(SA,NSW),
      next(SA,VIC),
      next(QLD,NSW),
      next(NSW,VIC),
      next(VIC,TAS),
      !.

next(X,Y) :-
      choose_color(X),
      choose_color(Y),
      X \== Y.

choose_color(red).
choose_color(green).
choose_color(blue).
choose_color(yellow).
```

Only one solution is generated:

```
?- color(WA,NT,SA,QLD,NSW,VIC,TAS).

WA = red
NT = green
SA = blue
QLD = red
NSW = yellow
VIC = red
TAS = green
```