Common use of CUT

Horst Reichel

Winter Term 2010/11

Common use of CUT [1]:

1. If you get this far, you have picket the correct rule for this goal.
2. If you get to here, you should stop trying to satisfy the goal.
3. If you get to here, you have found the only (interesting) solution to the problem, and there is no point in ever looking for alternatives.

[1] W.F. Clocksin and C.S. Mellish: Programming in Prolog, Springer–Verlag, 1981, 1984, 1987, 1994, 2003

```
sum(X, Y, Z) :- X1 is X-1, Z1 is Z-1, sum(X1, Y, Z1) .
sum(0, X, X) .

?- sum(1, 2, X) .        divergence
```

```
sum(0, X, X) .
sum(X, Y, Z) :- X1 is X-1, Z1 is Z-1, sum(X1, Y, Z1) .

?- sum( -1, 2, X) .        divergence
```

Problem: The usual oder on integers is not well founded, since there are infinite decreasing chains.

In which way we can inductively define a predicate on integers?

SOLUTION:
One considers the two alternative cases $X >= 0$ and $X < 0$.

```
sum(0,X,X) .
sum(X,Y,Z) :- X >= 0, X1 is X-1, Z1 is Z-1, sum(X1, Y, Z1).
sum(X,Y,Z) :- X < 0, X1 is X+1, Z1 is Z-1, sum(X1, Y, Z1).
```

Since the last two rules are alternative, the better solution is

```
sum(0,X,X) .
sum(X,Y,Z) :- X >= 0, !, X1 is X-1, Z1 is Z-1, sum(X1,Y,Z1).
sum(X,Y,Z) :- X < 0, X1 is X+1, Z1 is Z-1, sum(X1, Y, Z1).
```

and one gets an example for the first application area of CUT.

The second application area of CUT will be illustrated by the specification of an average taxpayer – in this case, the calculation might be quite simpler and not have to involve considering lots of special cases.

In this application area the cut is used in conjunction with the built–in `fail` predicate. It is defined in such a way that as a goal it always fails and causes backtracking to take place. This is just like what happens if we try to satisfy a goal for a predicate with no facts or rules.

```
average_taxpayer(X) :-      foreigner(X), !, fail.
average_taxpayer(X) :-      spouse(X,Y), gross_income(Y,Inc),
                            Inc > 3000, !, fail.
average_taxpayer(X) :-      gross_income(X,Inc),
                            2000 < Inc, 20000 > Inc.
gross_income(X,Y) :-        receives_pension(X,P),
                            P < 5000, !, fail.
gross_income(X,Y) :-        gross_salary(X,Z),
                            investment_income(X,W), Y is Z+W.
investment_income(X,Y) :-   ...
```

The differences between a rule with and without a cut one can demonstrate by the first rule. Without the cut a foreigner with a gross income between 2000 and 20000 would be accepted as an average taxpayer, and with a cut each foreigner is not accepted.

In case of the forth rule one can see that for a person with a pension, limited by 5000, the gross salary and the investment income ist not relevant.

An intersting application of the cut–fail combination is in the definition of the predicate `not`.

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

The third application area of CUT will be illustrated by searching for a solution of a linear equation, like $X = 2 * X - 4$.

```
num(1).
num(N) :- num(N), N is N+1.

solution (X) : - num(X)=, X is 2*X - 4, !.
```

Since we know that a linear equation has at most one solution, there is no need to search further after a solution is found.