Cut in prolog

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

! = a cut

can backtrack on subgoals before the cut
If ever fail on a subgoal after the cut symbol then not only do we fail at this particular rule, but fail on the goal a.

Example

```
not(x) :- call(x), !, fail.
not(_).                    _ = anonymous variable
```

Can_fly(albatross).

1? not(can_fly(penguins)).

The first rule will fail because the call (can_fly(penguins)) will fail, so we look at the second rule, will print yes because the second rule is satisfied

1? – not(can_fly(albatross)).
will evaluate call(x) to true, and cross the cut returning fail

Example:

When we know there is only one solution, so don't want to even attempt to backtrack.

```
head_of_state(usa, bush) :- !.
head_of_state(russia, putin) :- !.
head_of_state(mexico, fox) :-!.
```

|? – head_of_state(usa, X)
yes
x = bush ; <return>
then we don't look at any more rules, and fails (saves time by preventing backtracking)

Example: (Add numbers 1 to n)

```
sum_up(1, 1) :- !.
        first slot, up till which number to sum (second slot indicates sum)
sum_up(N, X) :- N1 is N – 1,
                sum_up(N1, X1),
                X is X1 + N.
```

The cut in this problem forces us to just have 1 solution

|? – sum_up(3, x)
yes x = 6
If we don't use the cut in this problem, we would have an infinite loop

Simple game example:

```
repeat.
repeat :- repeat.

game :- initialize,
        repeat,
        do_game,
        again?,          fails if the person wants to go again
        !,
        shutdown.        we cross the cut if we don't want to play again, and shutdown
```

PARSING OF ENGLISH IN PROLOG

Example: AfterS represents what comes after the first sentence

```
sentence (input, AfterS) :-     noun_phrase(Input, AfterNP) ,
                                verb_phrase(AfterNP, AfterS).
noun_phrase(Input, AfterNP) :-      determiner(Input, AfterDet),
                                    noun(AfterDet, AfterNP).
verb_phrase(Input, AfterVP) :-      verb(Input, AfterVP).
verb_phrase(Input, AfterVP) :-      verb(Input, AfterVerb),
                                    noun_phrase(AfterVerb, AfterVP).
determiner ([the | AfterDet], AfterDet).
determiner ([a | AfterDet], AfterDet).
noun([cat | AfterN], AfterN).
noun([milk | AfterN], AfterN).
verb([drinks | AfterVerb], AfterVerb).
verb([licks | AfterVerb], AfterVerb).

|? – sentence([the, cat, licks], [ ]).
        yes
```

Notice all rules have two slots in 1st goal.
Prolog has a built in mechanism for simplifying writing such rules.

Example:

```
sentence --> noun_phrase, verb_phrase        [ this rule is the same as the first rule above ]
```