CUT OPERATOR

Prolog has a feature of backtracking, but sometimes we want to control that backtracking mechanism. Though we can control that mechanism via changing the goal orders. But there is one more way to do so. Prolog provides the built-in predicate 'Cut operator' for that, which always succeeds. Cut operator ' **!** ' helps us control over the way prolog looks back for the solution - thus we can prevent it from unwanted backtracking. "When executed in the body of a clause, the cut always succeeds and removes backtracking points set before it in the current clause"[c]. Cut basically gives order to prolog to freeze the decisions made so far in this predicate and this way we can save time, improve the speed, performance and memory usage. You can write cut in the prolog rule (right hand side) or on prolog query.

Let's see simple examples to understand the behaviour of the

Cut. **Cut Operator Example 1 :**

Program :

```
s(c).
s(m).
s(d).

solve(X) :- s(X).
solve(other_solution).
```

Query Prompt :

```
?- solve(X).
X = c;
```

Without Using Cut

```
X = m;
X = d;
X =
```

other_solution.

Using Cut

Program :	**Query Prompt :**
s(c). s(m). s(d). solve(X) :- s(X), **!**. solve(other_solution).	?- solve(X). X = c.

Cut Operator Example 2 with illustration:

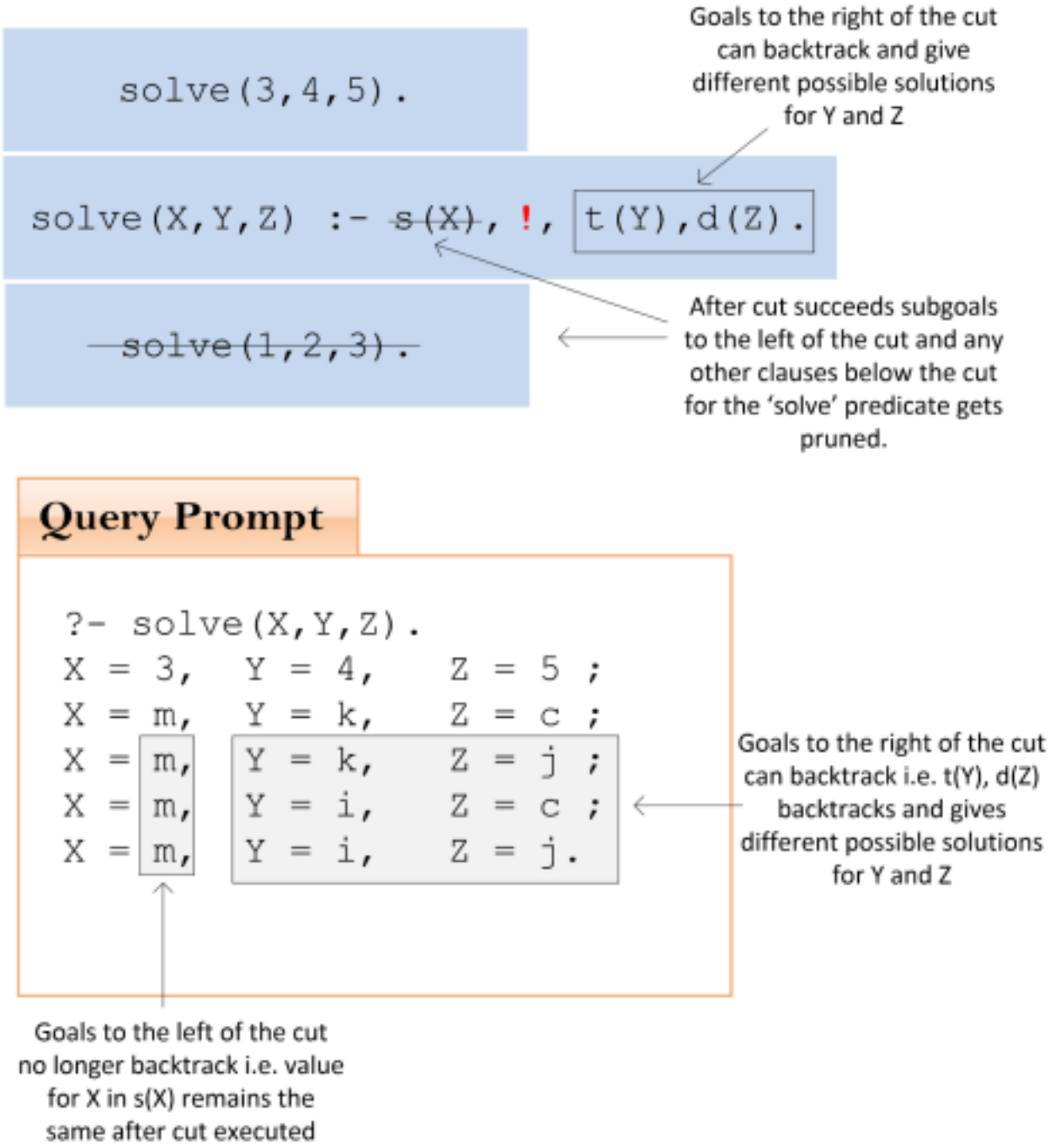
Without Using

Cut

Program :	**Query Prompt :**
s(m). s(d). t(k). t(i). d(c). d(j). solve(3,4,5). solve(X,Y,Z) :- s(X), t(Y), d(Z). solve(1,2,3).	?- solve(X, Y, Z). X = 3, Y = 4, Z = 5 ; X = m, Y = k, Z = c ; X = m, Y = k, Z = j ; X = m, Y = i, Z = c ; X = m, Y = i, Z = j ; X = d, Y = k, Z = c ; X = d, Y = k, Z = j ; X = d, Y = i, Z = c ; X = d, Y = i, Z = j ; X = 1, Y = 2, Z = 3.

Using Cut

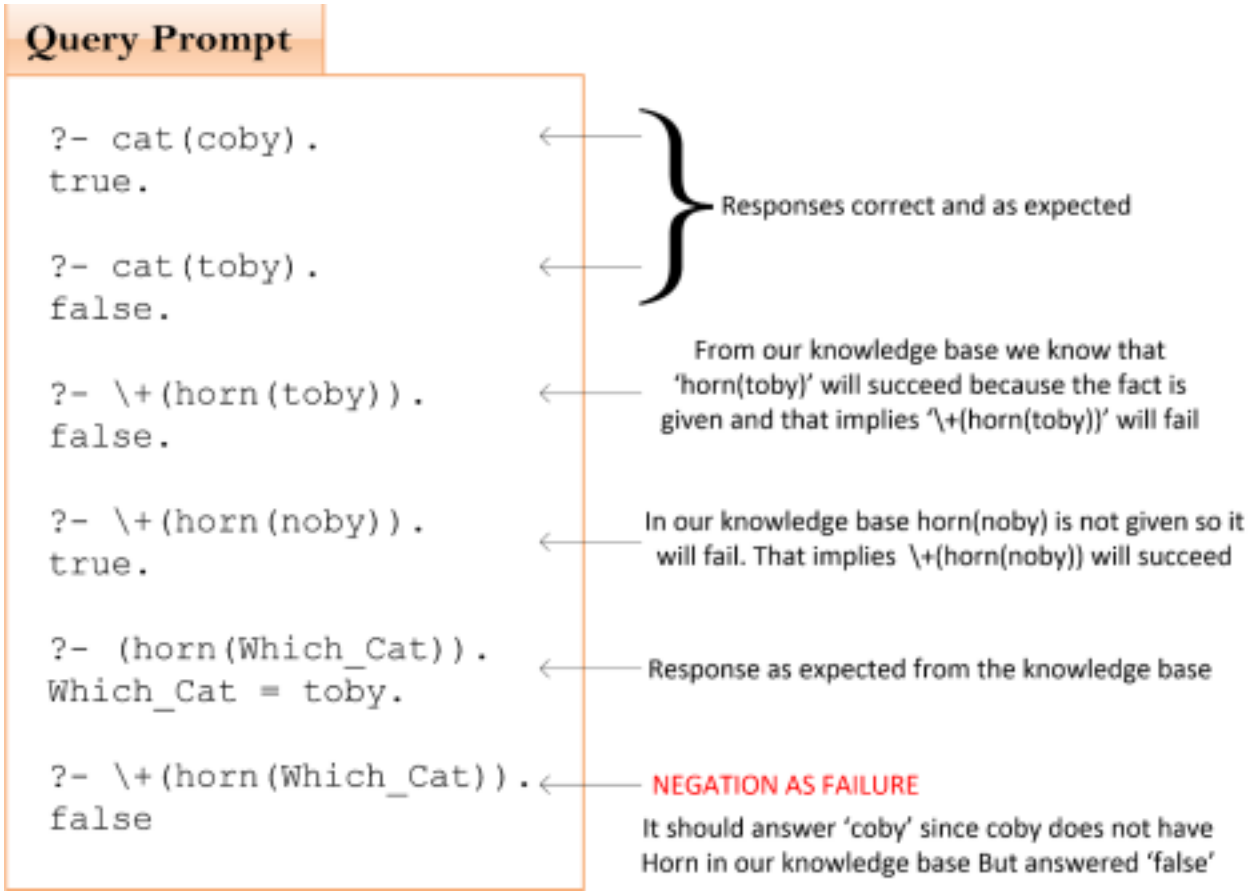
Program :	**Query Prompt :**
s(m). s(d). t(k). t(i). d(c). d(j). solve(3,4,5). solve(X,Y,Z) :- s(X), **!**, t(Y),d(Z). solve(1,2,3).	?- solve(X,Y,Z). X = 3, Y = 4, Z = 5 ; X = m, Y = k, Z = c ; X = m, Y = k, Z = j ; X = m, Y = i, Z = c ; X = m, Y = i, Z = j.
Example 2 illustration below	



NEGATION AS FAILURE

The negation symbol is written as '**not**' or '**\+**' in Prolog. Negation cannot be written in LHS of the rule in prolog.

Let' see an **example below** to understand the behaviour of negation. In below example, left box contains the program and right box represents the query executions and illustrations of the query responses.



Closed world assumption in Prolog : Whatever you have entered in your program, is only considered true and everything else is considered false.