The 'fail' predicate

- It's a built in predicate without any argument.
- It always fails. It is an explicit way of forcing backtracking.

Example:

1.
```
go :- test, write ("Will you get here ?").
test :- fail.
```

```
goal : go
False
```

2.
```
go :- write ("You will get here "), test.
test :- fail.
```

```
goal : go
You will get here
False
```

3.
```
predicates
    location (string, string)
    go
clauses
    go :- writef ("%-10  %5\n", "CITY", "STATE"),
          fail.
```

```
go :-  location (City, State),
       writef ("%-10 %5 \n", City, State),
       fail.

go.

location ("Jackson", "MS").
location ("Washington", "DC").
location ("Raleigh", "NC").
```

Goal: go ↵

```
CITY        STATE
Jackson     MS
Washington  DC
Raleigh     NC
Yes.
```

(1) If we remove fail ①, the o/p would be

```
CITY        STATE
Yes.
```

(2) With no fail in the program, the output is same as (1).

(3) With fail in 1st go clause and removing it from the 2nd, the o/p would be

```
CITY      STATE
Jackson   MS
Yes.
```

(No variable in the goal. So, once a goal succeeds, there is no backtracking)

predicates

location(string, string)
go(string, string)

clauses

go(_,_) :- writef("%-10 %5\n", "CITY", "STATE"), fail.

go(X,Y) :- location(X,Y).
location("Jackson", "MS").
location("Washington", "DC").
location("Raleigh", "NC").

goal: go(Location, City).

CITY STATE

Location = Jackson, City = MS
Location = Washington, City = DC
Location = Raleigh, City = NC
3 solns

If we remove fail in the first go clause, then the answer is:

Location = _, City = _,
+ original 3 solutions

4 Solutions

predicates

```
location (string, string)
go
chkstate (string)
```

clauses

```
go :- writef ("%-10 %5\n", "CITY",
                "STATE"), fail.

go :- location (City, State), chkstate (State),
      writef ("%-10 %2\n", City, State), fail.

go.

chkstate ("DC") :- fail.
chkstate (_).

location ("Jackson", "MS").
location ("Washington", "DC").
location ("Raleigh", "NC").
```

Goal: go.

```
CITY       STATE

Jackson    MS
Washington DC
Raleigh    NC
Yes
```

To print only state MS & NC, (without DC),

The Cut (!)

- Its symbol is '!'.
- Its a built in predicate without any argument.
- It always succeeds.
- It is used for 'preventing backtracking'.
- It is mainly used for:
  i) Reducing the search space.
  ii) Retaining values of variables once they are bound.

```
go :- premise1, premise2, !, premise3, premise4.
```

'Cut' acts as a fence. All possibilities in premise3 & 4 would be tried out. Whether it succeeds or fails, no more go clause or premise 1 or premise 2 can be tried.

```
chkstate ("DC") :- !, fail.
chkstate (_).
```

'!' will not allow PROLOG to try alternate clause. Thus, go clause will fail with City & State variable of location bound to "Washington" and "DC". So, PROLOG will try with alternate values of these variables.

{ we can also get the desired result like :

```
chkstate("MS").
chkstate("NC").
```

Thus, for chkstate("DC"), there is no matching and it fails.

We don't want to do that.

}

predicates

```
l(string, string)
go
cs(string)
```

clauses

```
go:- writef("%-10 %5\n", "CITY", "STATE"), fail.
go:- l(City, State), cs(State), writef("%-10 %5\n",
                                City, State),
                                fail.

go.

l("JK", "MS").
l("WS", "DC").
l("RL", "NC").
cs("DC") :- !.
cs(_).
```

Goal: go.

O/p

CITY	STATE
JK	MS
WS	DC
RL	NC

Yes.

{ : cs("DC") would succeed with "!" as a condition. So, first two cities would be printed. Now l(City, State) would be tried for the 3rd possibility also.

}

Imp. cs(State) would not be tried out any more once "!" comes. But, once it succeeds or fails, other possibilities to the left of ~~out~~ premise in which '!' is present could be tried.

→ If we remove "fail" from 2nd go clause, then the O/p would be:

CITY	STATE
JK	MS

{ Here chkstate clauses are:

chkstate(State) :- !

→ 8 Let's change 2nd go clause & chkstate clauses:

```
go :- l(City, State), cs(State), writef("  "),
      !.

cs("DC") :- !, fail.
cs(_).
```

Output: CITY STATE

JK MS

Yes

(As no more "go" clauses can be tried).

→ With 2nd go clause as:

```
go :- l(City, State), cs(State), writef("  "),
      !, fail.

cs("DC") :- !, fail.
cs(_).
```

O/p CITY STATE

JK MS

No

(∵ No more go clauses can be tried).

→ Let's check output for the following "go" clause :

```
go :- writef ("                ") , fail.
go :- l (City, State), !, cs (State),
      writef ("     ").
go.
chkstate ("DC") :- !, fail.
chkstate (_).
```

Goal: go

```
CITY      STATE
JK        MS

Yes
```

The O/P is so because '!' in the "go" clause is different from the cut in the chkstate clause. In all previous examples, cut was present in the chkstate clause, whereas it is present here in the go clause. So, no more possibility in the go clause can be tried.

→ If we ~~remove~~ our go clause is

```
go :- l (City, State), !, cs (State), writef ("  "
      fail.
```

O/P

(10)

CITY STATE
JK MS
NO

```
a(0).
```

Change ① `a(X) :- !, b(X), !`

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

Change ② `b(X) :- c(X), !`

```
b(3).
c(4).
c(5).
c(X) :- !, d(X).
c(6).
d(7).
```

Goal: 1) `d(X)`

```
X = 7
1 solution.
```

2) Goal: `c(X)`

```
X = 4
X = 5
X = 7
3 soln
```

3) Goal: `b(X)`.

```
X = 1
X = 2
X = 4
3 soln
```

4) Goal: `a(X)`

```
X = 0
X = 1
X = 2
X = 4
4 solution.
```

① → If we change `a(X)` as: (No change in 2)

```
a(X) :- b(X), !
```

Then o/p for goal `a(X)` is:

```
X = 0
X = 1
2 soln
```

② → If `b(X)` is changed as (No change in (1))

```
b(X) :- c(X)   (i.e. cut is removed)
```

o/p is for the goal `a(X)` is:

```
a(0)
a(X) :- !, b(X).
b(1).
b(2).
b(X) :- c(X), !, fail.  /* left has changed */
b(3).
c(4).
c(5).
c(X) :- !, d(X).
c(6).
```

goal: b(X)

X = 1

X = 2

2 soln

["!" in the b(X) clause does not allow Other b(X) clause to be tried out]

→ New b(X) clause is:

```
b(X) :- !, c(X), fail.
```

Then o/p is same for the same reason.

```
a(0).
a(X) :- !, b(X).
b(1).
b(2).
b(X) :- c(X)   /* This has changed */
b(3).
c(4).
c(5).
c(X) :- !, d(X), fail.
c(6).
d(7).
```

goal: b(X).

X = 1

X = 2

X = 4

X = 5

X = 3 ; 5th sol^n

5 solution

If we change the c(X) clause as (other things unchanged)

```
c(X) :- fail, !, d(X).
```

Then the o/p is for the goal b(X):

X = 1

X = 2

X = 4

X = 5

X = 6

X = 3