Prolog
(Rekurens, Cut, Fail, List)

IF1221 Computational Logic
Semester II - 2024/2025

Informatics Engineering Study Program
School of Electrical Engineering and Informatics ITB

Rekurens

- Kalkulasi sederhana dengan deklaratif → memanfaatkan analisis kasus
- Program menjumlahkan dua buah bilangan integer dalam domain yang dibatasi [1 – 9] dengan hasil dalam domain yang sama

  /*suksesorkecil(X,Y) benar artinya Y adalah suksesor dari X, dengan Y < 10 dan 1 ≤ X ≤ 8 */ → Fakta

  /*jumlahkecil(X,Y,Z) benar artinya Z adalah jumlah dari X + Y, dan Z lebih kecil dari 10. */ → Aturan

Rekurens (2)

- Yang dimanipulasi bukanlah ‘integer murni’, tapi simbolik dari bilangan integer
- Aturan yang definisinya berdasarkan aturan itu sendiri → rekursif
- Analisis rekurens dalam pemrograman deklaratif:
  - Penalaran berdasarkan definisi predikat yang rekursif, atau
  - Berdasarkan tipe (simbolik) yang juga terdefinisi secara rekursif
- Analisis rekurens harus mendefinisikan:
  - aturan basis → jaminan bahwa pencocokan berhenti
  - aturan rekurens → harus menuju basis
- Fakta tidak mungkin rekursif → ‘instans’ dunia nyata
- Query tidak mungkin rekursif → pertanyaan pada program

Rekurens (3)

- Manipulasi dengan analisis rekurens dapat menjumlahkan bilangan integer positif tak terbatas
- Contoh penjumlahan dua buah bilangan integer positif

```
/*plus(X,Y,Z) benar artinya X ditambah dengan Y hasilnya adalah Z
  jika x+y = z
  maka (x+1) + y = z + 1
  Basis 0: bilangan ditambah 0 hasilnya adalah bilangan itu sendiri */
       plus(0,X,X).
/*Rekurens: plus(X,Y,Z) menyatakan x + y = z
  plus((X+1),Y,(Z+1)) harus dilinearisasi
  suksesor(X,X') benar jika X' adalah suksesor dari X*/
       plus(X', Y, Z') :- suksesor(X,X'),
                          plus(X, Y, Z),
                          suksesor(Z,Z').
```

Cut

- Cut adalah salah satu predikat sistem yang disediakan Prolog.
- Cut digunakan untuk memberikan kontrol atas backtracking.
- Digunakan untuk merealisasikan analisa kasus pada Prolog.
- Kenapa program Prolog memanfaatkan Cut?
  - agar eksekusi program lebih efisien
  - tanpa cut dapat dibuat program Prolog yang dapat dieksekusi dengan benar.
- Kapan cut harus dipakai?
  - Jika dengan penggunaan cut proses pencarian solusi dapat dibatasi
  - Hal ini dimungkinkan dengan cara menghilangkan proses backtrack ke cabang tersebut.

Cara kerja Cut

Predicate H:

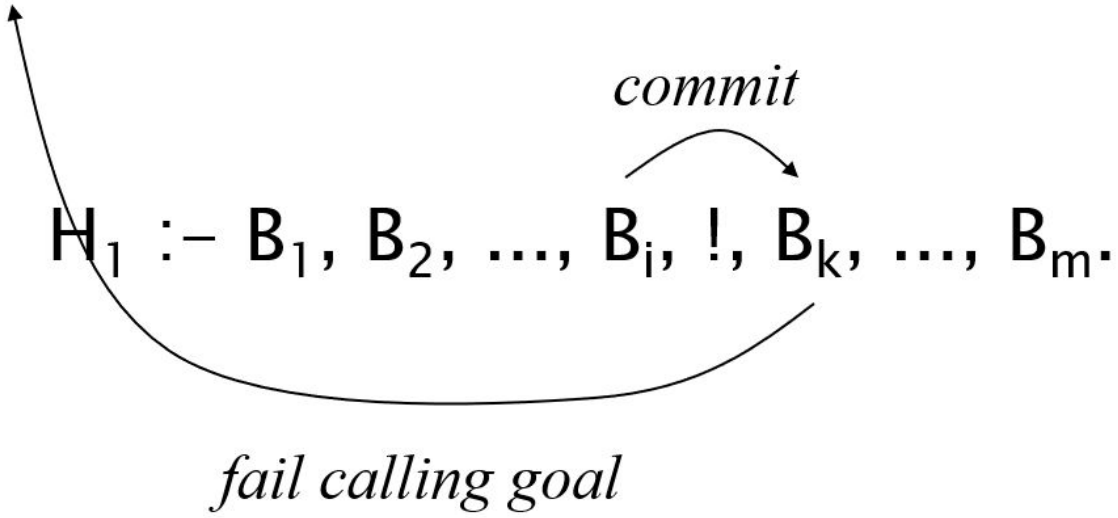
H_1 :- B_1, B_2, ..., B_i, !, B_k, ..., B_m.

H_2 :- B_n, ... B_p.

- Jika H_1 cocok, B_1... Bi mungkin backtrack di antara mereka sendiri. Jika B_1 gagal, H_2 akan dicoba. Tapi begitu 'cut' dilewati, Prolog akan memilih pilihan saat ini. Semua pilihan lain dibuang.
- B_k, ..., B_m dapat saling backtrack satu sama lain, tetapi jika B_k gagal, maka predikatnya gagal (klausa berikutnya tidak cocok).

Cara mudah memahami Cut

- Pikirkan ‘cut' sebagai 'pagar' yang, ketika dilintasi sebagai sukses, menegaskan sukses sebagai solusi saat ini.
- Namun, ketika sebuah kegagalan mencoba melewati pagar, seluruh predikat akan gagal.



Cut

- Terdapat 2 (dua) tujuan penggunaan cut yang menyebabkannya menjadi jenis yang berbeda:
  - Green Cut: cut digunakan untuk efisiensi eksekusi, karena jika cut dihilangkan arti program tidak berubah.
  - Red Cut: cut digunakan untuk kebenaran eksekusi, karena jika cut dihilangkan arti program berubah.

CONTOH KASUS 1

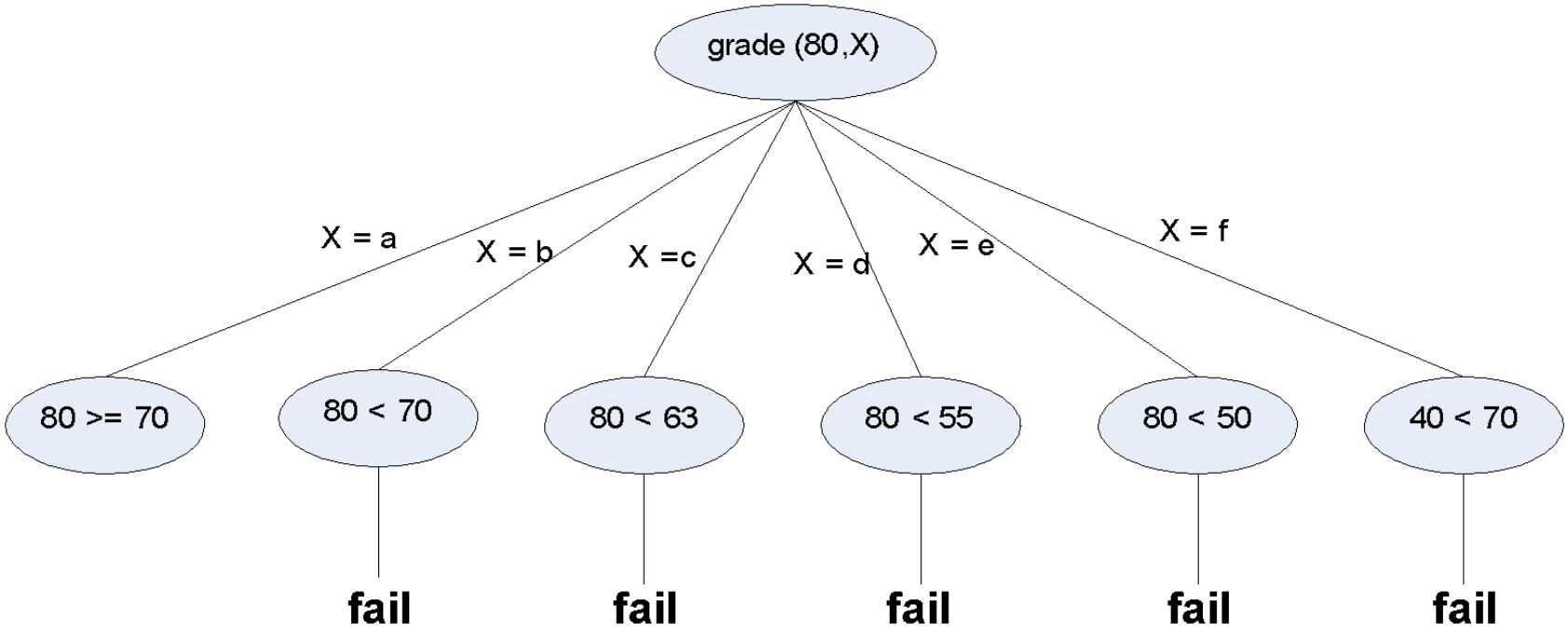
- **Rule yang diberikan**

grade(Mark, a) :- Mark>=70.
grade(Mark, b) :- Mark<70, Mark>=63.
grade(Mark, c) :- Mark<63, Mark>=55.
grade(Mark, d) :- Mark<55, Mark>=50.
grade(Mark, e) :- Mark<50, Mark>=40.
grade(Mark, f) :- Mark<40.

- **Query**

grade(80, X).

Pohon Eksekusi (1)



Sehingga menghasilkan X = a

Karakter Cut (1)

- Rule pada kasus sebelumnya tidak efisien, karena Prolog akan kembali mencari apakah ada solusi lain.
- Penggunaan karakter Cut(!) dapat meningkatkan efisiensi pada kasus tersebut.

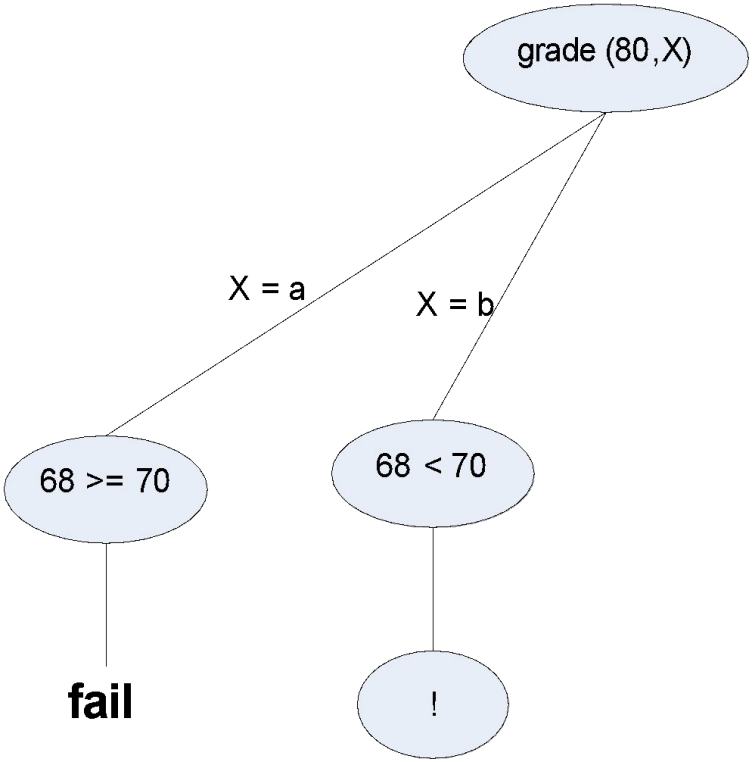
Karakter Cut (1)

```
grade(N, a) :- N>=70, ! .
grade(N, b) :- N>=63, ! .
grade(N, c) :- N>=55, ! .
grade(N, d) :- N>=50, ! .
grade(N, e) :- N>=40, ! .
grade(N, f) :- N<40.
```

- **Query :**

  grade(68,X).

Pohon Eksekusi (2)



- Program menghasilkan nilai X = b.
- Setelah bertemu karakter cut(!), program tidak akan melakukan backtrack. Sehingga tidak perlu diperiksa untuk nilai

  X = d, e, f.

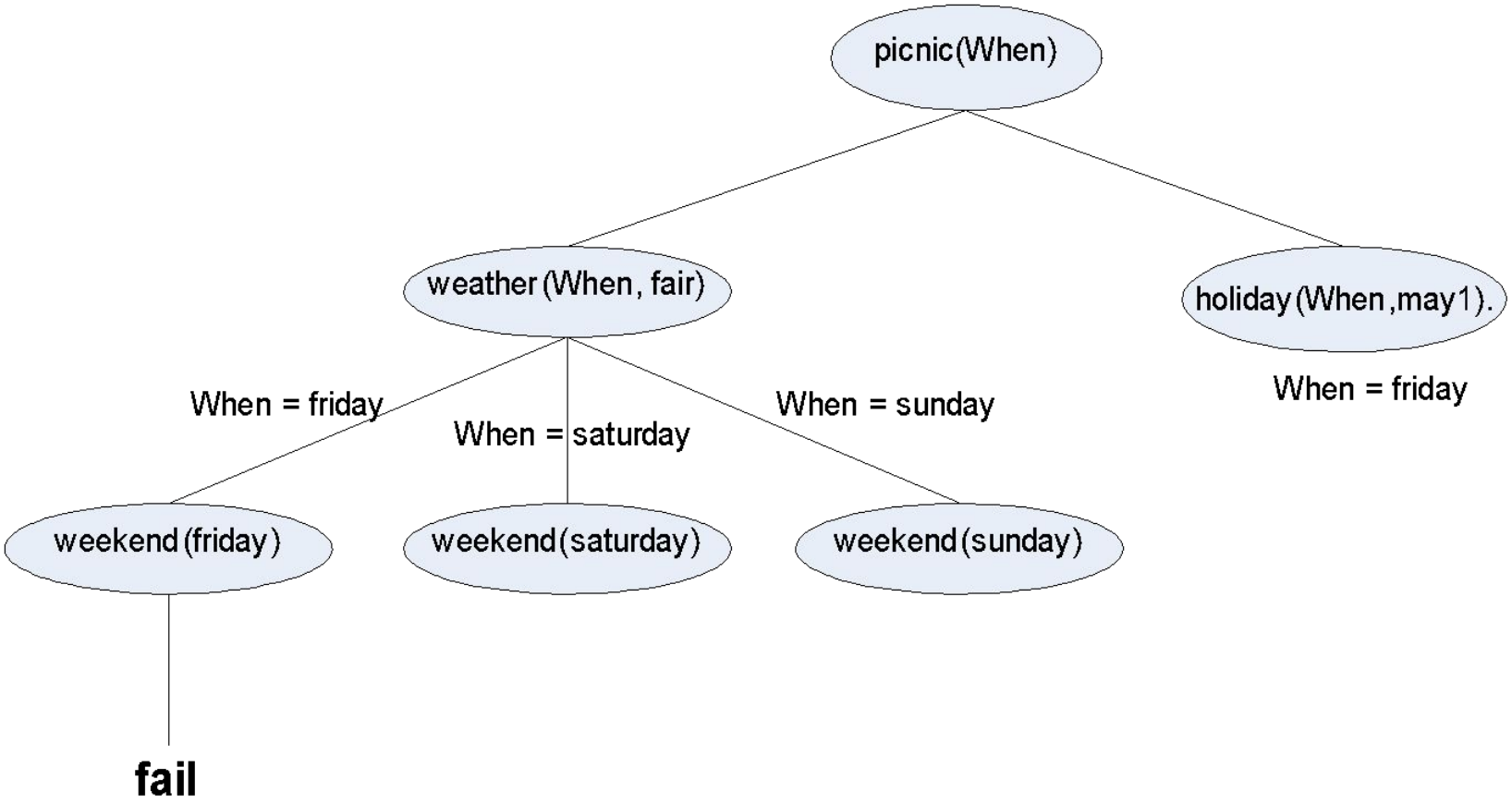
CONTOH KASUS 2

- Fakta yang diberikan :
  - holiday(friday, may1).
  - weather(friday, fair).
  - weather(saturday, fair).
  - weather(sunday, fair).
  - weekend(saturday).
  - weekend(sunday).

- Mary akan pergi piknik pada weekend dengan weather fair, atau hari libur tanggal 1 Mei. Maka rule yang diperlukan :
  - picnic(Day) :- weather(Day,fair), weekend(Day).
  - picnic(Day) :- holiday(Day,may1).

- Query :
  - picnic(When).

Pohon Eksekusi (3)

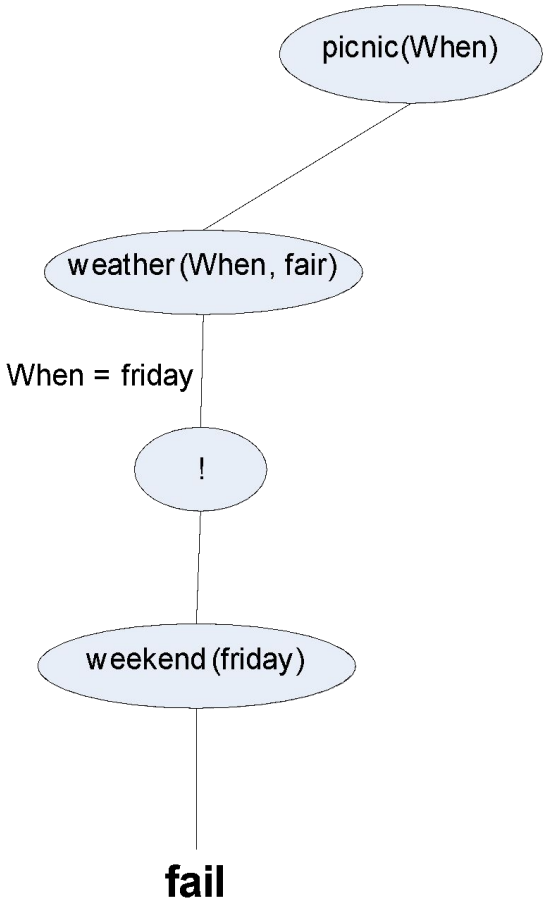


Karakter Cut (2)

- Untuk contoh kasus 2, Program akan menghasilkan nilai `When = friday, saturday, sunday.` Program melakukan backtrack untuk memeriksa semua kemungkinan.
- Perhatikan rule berikut :
  - picnic(Day) :- weather(Day,fair), !, weekend(Day).
  - picnic(Day) :- holiday(Day,may1).

Pohon Eksekusi (4)

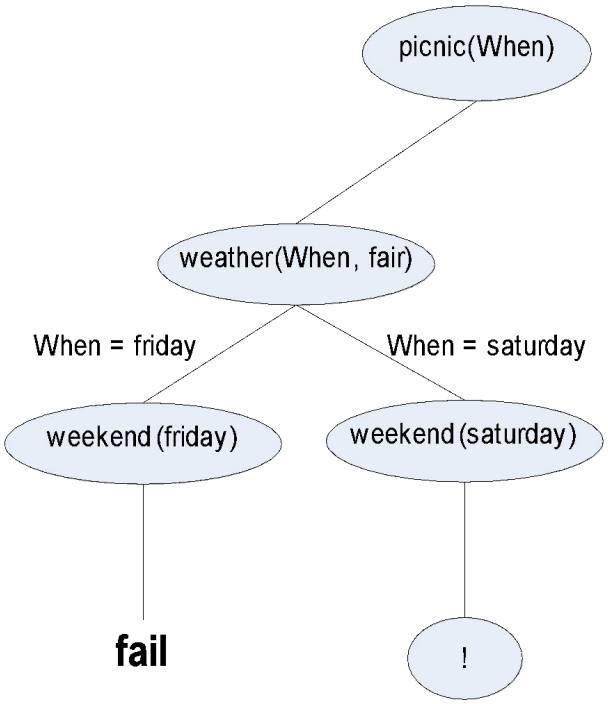
- Setelah melewati karakter cut(!) program tidak akan melakukan backtrack.
- Program tidak kembali pada `weather(When,fair)` untuk mencari kemungkinan lain dari nilai `When`.



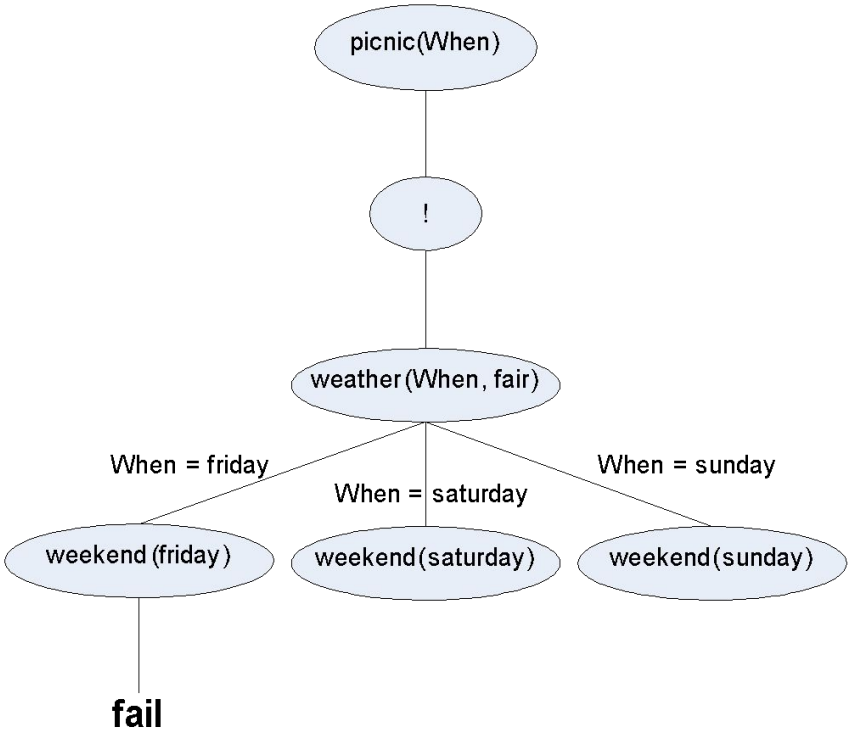
Karakter Cut (3)

- Apa yang terjadi jika karakter cut (!) pada contoh kasus di atas diubah-ubah letaknya?
  - Contoh 1 :
    - picnic(Day) :- weather(Day,fair), weekend(Day), !.
    - picnic(Day) :- holiday(Day,may1).
  - Contoh 2 :
    - picnic(Day) :- !, weather(Day,fair), weekend(Day).
    - picnic(Day) :- holiday(Day,may1).

Pohon Eksekusi (4)



Contoh 1 : `When = Saturday`



Contoh 2 :
`When = Saturday, Sunday`

Contoh Program 2

```
buy_car(Model,Color):-
   car(Model,Color,Price),colors(Color,sexy),!,Price < 25000.
car(maserati,green,25000).
car(corvette,black,24000).
car(corvette,red,26000).
car(corvette,red,23000).
car(porsche,red,24000).
colors(red,sexy).
colors(black,mean).
colors(green,preppy).
```

Query : buy_car(corvette,Color).

Backtrack

- Prolog akan mengatur redo point untuk mencari seluruh solusi yang mungkin
- Karakter cut (!) untuk meminta prolog tidak melakukan backtrack

Backtrack

- Untuk query `buy_car(corvette,Color)` proses eksekusinya sebagai berikut :
  - `car(corvette,_1,_2)`
  - `car(corvette,black,24000)`
  - `colors(black,sexy)` (fail)
  - `car(corvette,red,26000)` (hasil backtrack)
  - `colors(red,sexy)`
  - `!` (bahwa jika telah selesai, tidak perlu melakukan backtrack)
  - `26000 < 25000` (fail)
- Maka pemrosesan query berhenti di sini dengan tidak ada fakta yang memenuhi
- Bahwa ada fakta `car(corvette,red,23000)` tidak diperhatikan karena adanya `!`.

Contoh

- Program 1:

```
max2(X,Y,X) :- X >= Y, !.
max2(X,Y,Y) :- X < Y.
```

- Program 2:

```
max2(X,Y,X) :- X >= Y, !.
max2(X,Y,Y).
```

Fail

- Predikat FAIL juga salah satu predikat sistem yang disediakan Prolog.
- Secara konsep, FAIL digunakan untuk memanipulasi program sehingga suatu proses tertentu harus dilakukan. Tetapi, jika dikombinasikan cut dan fail, secara konsep, digunakan untuk merealisasikan ***STOP Statement*** pada Prolog, artinya untuk memaksa program Prolog berhenti pada satu kasus tertentu.

Fail

- Umumnya digunakan untuk merealisasikan *Error Message*.
- Dengan FAIL, Prolog justru memaksa proses backtrack ke suatu cabang tertentu.
- Contoh:

0. Z :- A.
1. A :- B, fail.
2. A :- C.

Jika predikat Z dieksekusi, maka selanjutnya akan memproses predikat A.

Pada saat A dieksekusi dengan menelusuri kalimat (1), jika B bernilai true maka (1) akan digagalkan (bernilai false). Selanjutnya Prolog akan menelusuri kalimat (2), dan hasilnya tergantung pada C.

Cut dan Fail

- Dengan kombinasi CUT yang dilanjutkan dengan FAIL, Prolog dapat memaksa proses backtrack ke suatu cabang tertentu untuk selanjutnya berhenti di cabang tersebut (tidak melanjutkan proses backtrack ke cabang lain).
- Contoh:

0. Z :- A.
1. A :- B, !, ErrorMsg, fail.
2. A :- C.
3. ErrorMsg :- write('Error Message').

- Jika predikat Z dieksekusi, maka selanjutnya akan memproses predikat A. Pada saat A dieksekusi dengan menelusuri kalimat (1), jika B bernilai true maka (1) akan digagalkan (bernilai false). Selanjutnya Prolog akan menelusuri (3), tetapi tanpa menelusuri kalimat (2).

Contoh

```
/*Fact*/
bird(sparrow).
bird(eagle).
bird(duck).
bird(crow).
bird(ostrich).
bird(puffin).
bird(swan).
bird(albatross).
bird(starling).
bird(owl).
bird(kingfisher).
bird(thrush).

/*Rule*/
can_fly(X):-bird(X).
```

Ostrich can't fly

Without cut:

?- can_fly(duck).
yes
?- can_fly(ostrich).
yes

With failure exception:

```
/*Rule*/
can_fly(ostrich):-fail.
can_fly(X):-bird(X).

?- can_fly(duck).
yes
?- can_fly(ostrich).
yes
```

With cut and failure exception

```
/*Rule*/
can_fly(ostrich):-!,fail.
can_fly(X):-bird(X).

?- can_fly(duck).
yes
?- can_fly(ostrich).
no
```

THANK YOU