Rekurens, Cut & Fail, List, File Eksternal, Loop

Informatics Engineering Study Program
School of Electrical Engineering and Informatics

Institute of Technology Bandung

Rekurens

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 & Fail

Cut

- Cut adalah salah satu predikat sistem yang disediakan Prolog.
- Secara konsep, cut 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.

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

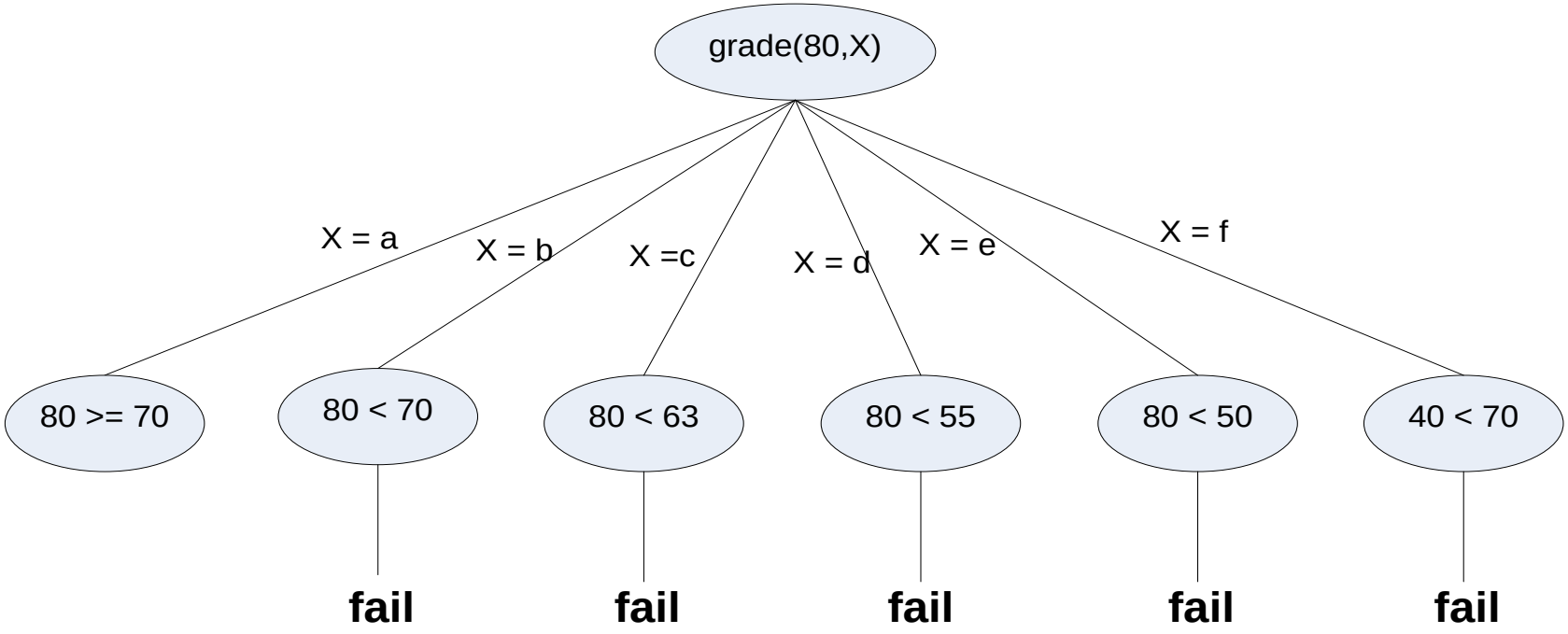
- **Rule yang diberikan**

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)

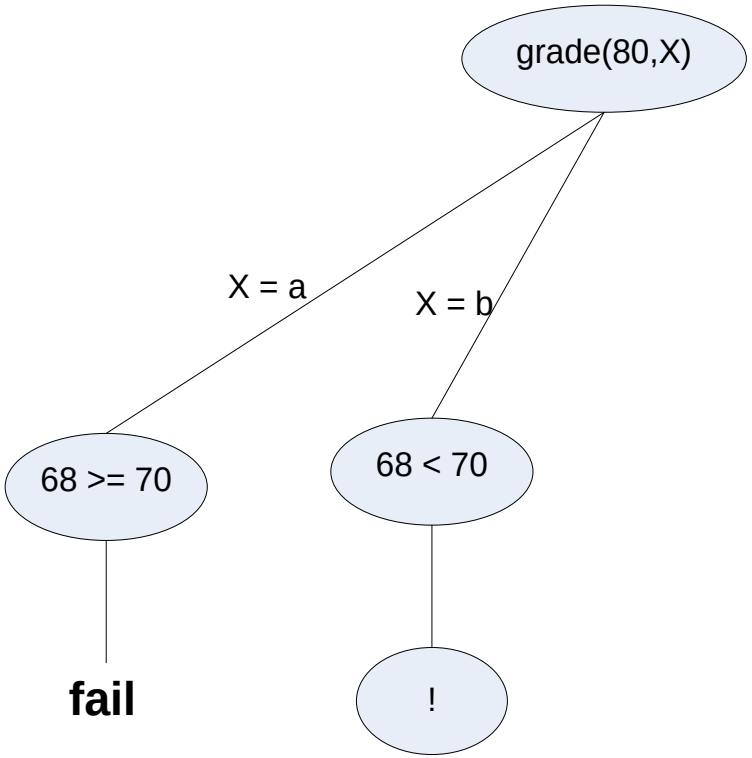
- Rule pada kasus sebelumnya tidak efisien, karena Prolog akan kembali mencari apakah ada solusi lain.
- Penggunaan karakter Cut(!) dapat meningkatkan efisiensi pada kasus tersebut.

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.

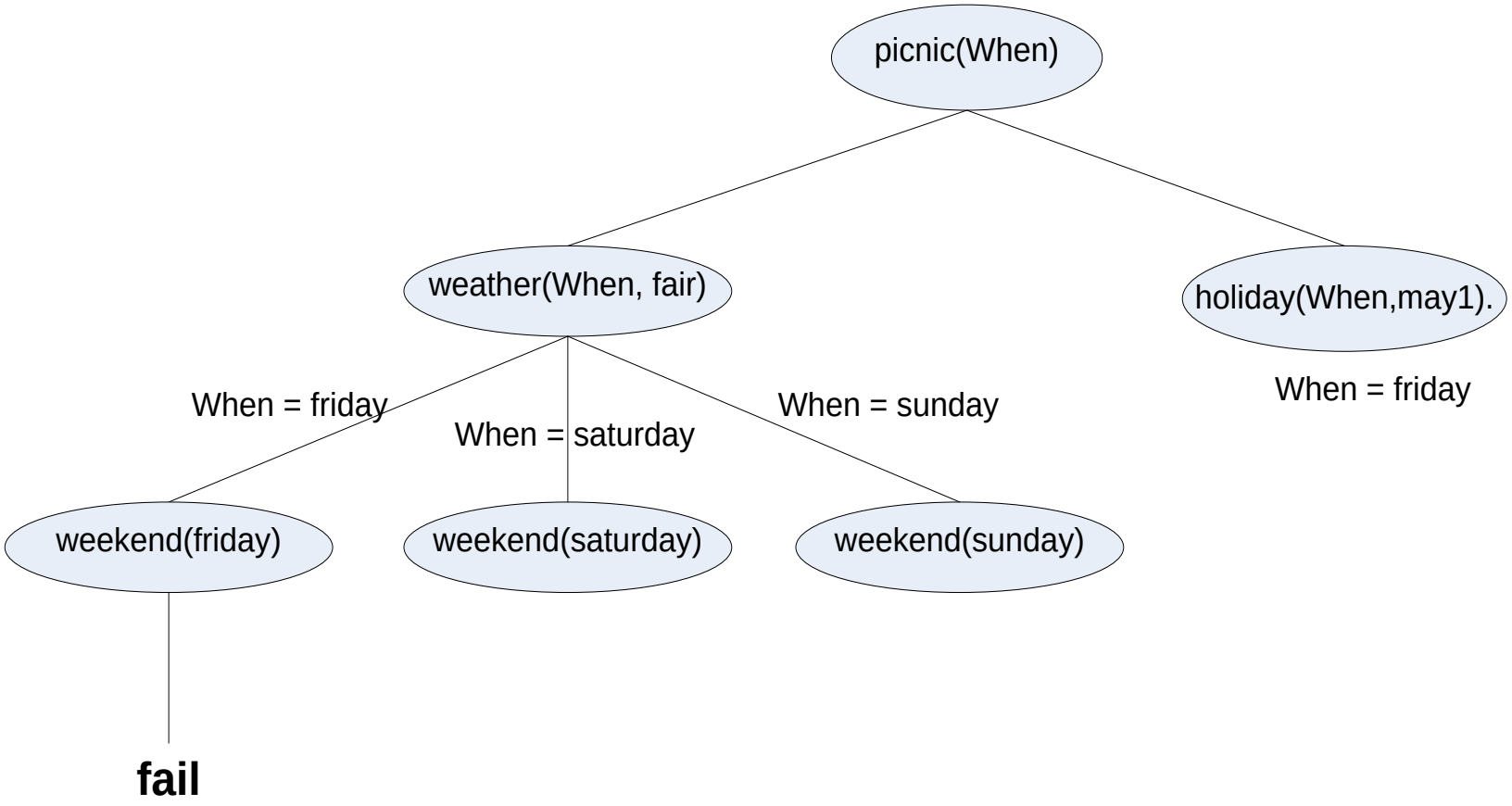
CONTOH KASUS 2

- 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)

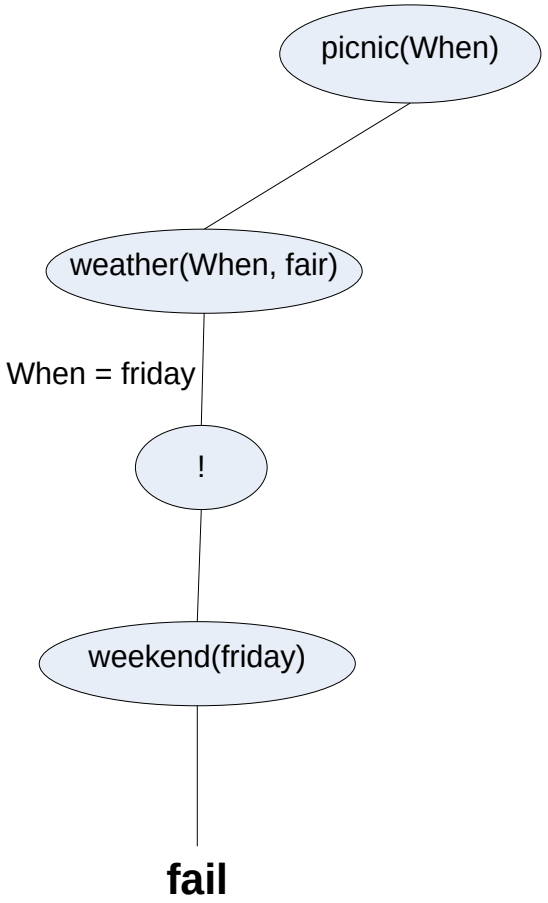


Karakter Cut (2)

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

Pohon Eksekusi (4)

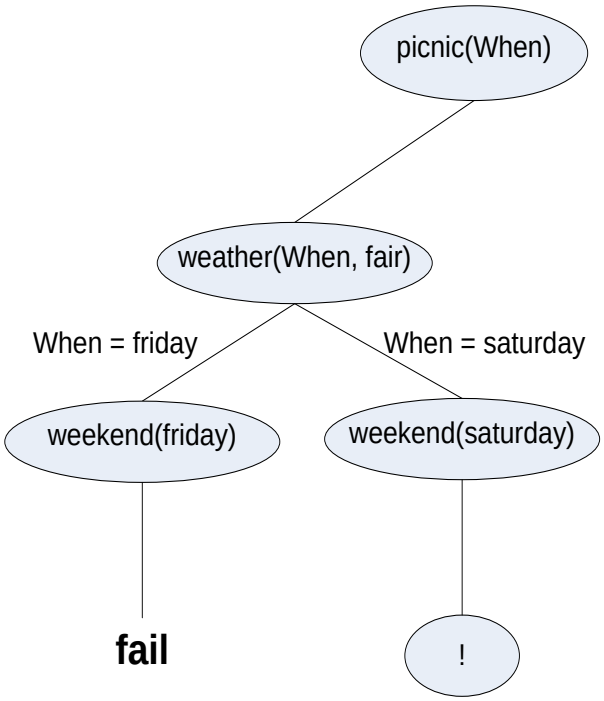
- Setelah melewati karakter cut(!) program tidak akan melakukan backtrack.
- Program tidak kembali pada `weather(When,fair)` untuk mencari kemungkinan lain dari nilai `When`.



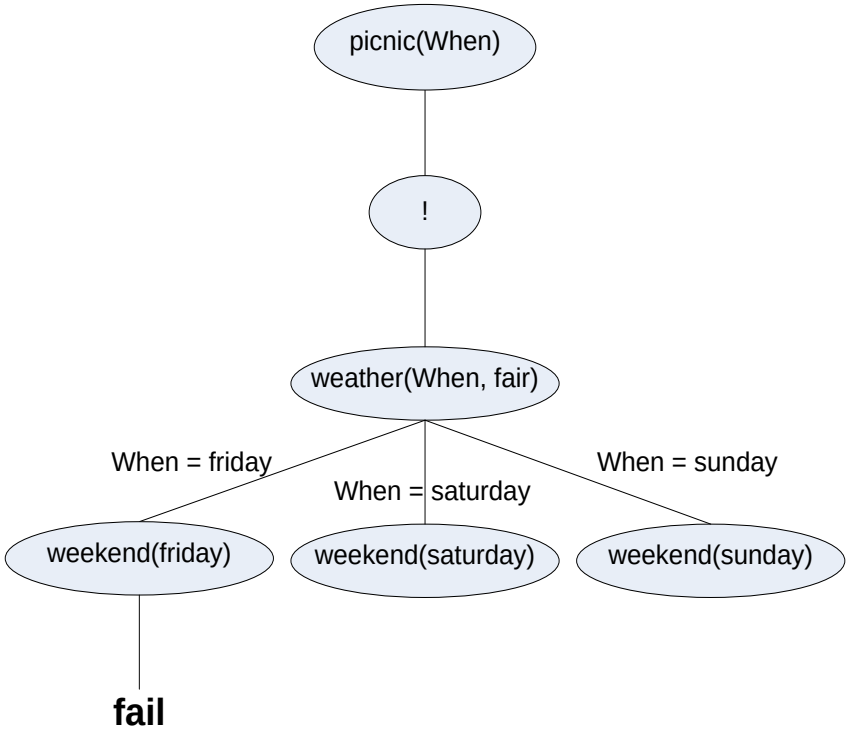
Karakter Cut (3)

- 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
```

List

Pengenalan List

- Definisi List → sekumpulan elemen dengan keterurutan tertentu yang diketahui elemen pertamanya dan setiap elemen boleh muncul lebih dari satu kali
- Definisi rekursif
  - Basis → list kosong adalah sebuah list
  - Elemen list tidak kosong terdiri atas **head** dan **tail**
  - Head adalah elemen list; tail adalah list
- Elemen list → TERM yang bisa berupa konstanta, variabel,struktur yang mungkin juga berupa list
- Pemanfaatan List → representasi tree, *grammar*, mathematical entity, dsb
- Penulisan list dalam pemrograman Deklaratif
  - Dituliskan di antara tanda kurung siku
  - Pemisahan antara head dan tail menggunakan pipe (|)
- Contoh list sederhana
  - List dengan elemen integer

listint([1,2,3]).
listint([X|Y]). → apa yang dihasilkan?

Exercise

- Unifikasi adalah proses pengikatan suatu variabel dengan konstanta tertentu. Untuk setiap kasus di bawah ini, tentukan apakah unifikasi bisa dilakukan terhadap pasangan list. Jika unifikasi bisa dilakukan, tuliskanlah hasil substitusinya.

Catatan: variabel dinyatakan dengan simbol yang diawali huruf kapital.

a) [a,d,z,c] dan [H|T]
b) [a,b,X] dan [a,b,c,d]
c) [apple,pear,grape] dan [A,pear|Rest]
d) [a,[]] dan [A,B|Rest]
e) [a,b,c] dan [b|T]

Answer Exercise

a) H = a, T = [d,z,c]
b) Tidak bisa
c) A = apple, Rest=[grape]
d) A = a, B = [], Rest = []
e) Tidak bisa.

Contoh

- Terdapat program Prolog sebagai berikut.

```
append([ ], X, X) :- !.
append([A|B], C, [A|D]) :- append(B, C, D).
```

- Tentukan hasil dari query berikut, yang diterapkan pada program di atas

a) `append([a, b, c], [d, e], X).`
b) `append(X, Y, [a, b, c]).`
c) `append(X, [ ], Y).`

Contoh

a) X = [a,b,c,d,e]

   yes

b) X = []

   Y = [a,b,c] ?

   yes

c) X = []

   Y = [] ?

   yes

Latihan 1

Dalam domain peternakan kuda, diketahui bahwa kuda yang berharga adalah seekor induk kuda yang memiliki anak yang dapat berlari cepat. Dalam suatu peternakan terdapat empat ekor kuda bernama Comet, Prancer, Dasher, dan Thunder. Comet adalah induk dari Dasher dan Prancer. Dasher adalah induk dari Thunder. Dari pengalaman balap kuda, terlihat bahwa Prancer dan Thunder adalah kuda yang cepat.

- Buatlah program sederhana dalam gprolog berdasarkan informasi di atas, menggunakan predikat:

is_a_horse/1, is_fast/1, is_parent_of/2, dan valuable/1.

- Tuliskan query untuk mencari kuda yang berharga.
- Tuliskan jawaban dari program anda ketika mendapatkan query seperti pada butir (b).

Latihan 2

- Terdapat program prolog sederhana sebagai berikut. Program dibuat untuk menyatakan bahwa Pete menyukai semua jenis burger kecuali Big Kahuna Burger.

```
big_mac(a).
big_mac(c).
big_kahuna_burger(b).
whopper(d).
enjoys(pete,X)  :-  big_kahuna_burger(X),fail.
enjoys(pete,X)  :-  burger(X).
burger(X)  :-  big_mac(X).
burger(X)  :-  big_kahuna_burger(X).
burger(X)  :-  whopper(X).
```

Latihan 2 (2)

a) Jawablah query di bawah ini jika diterapkan pada program tersebut.

- (i) enjoys(pete,a).
- (ii) enjoys(pete,b).
- (iii) enjoys(pete,c).
- (iv) enjoys(pete,d).
- (v) enjoys(pete,X).

b) Tentukan apakah program tersebut sudah benar atau belum berdasarkan jawaban butir (a). Jika sudah benar, cukup tuliskan sudah benar. Jika belum benar, tuliskan bagaimana seharusnya.

Latihan 3

- Buatlah predikat yang dapat dijalankan dalam GNU Prolog, untuk menghitung nilai faktorial dari suatu bilangan integer positif (termasuk nol). Predikat yang digunakan adalah faktorial(X,Y); di mana X adalah bilangan yang dicari nilai faktorialnya, dan Y adalah nilai faktorialnya.

Latihan 4

Buatlah program prolog sederhana untuk operasi list sederhana bilangan integer (tidak mengandung list di dalam list), sebagai berikut.

- Predikat ukuranlist(list, integer) untuk menghitung banyaknya elemen list; contoh query size([1,2,3,4],N) menghasilkan N=4.
- Predikat jumlahlist(list, integer) untuk menjumlahkan elemen-elemen dalam list; contoh query sumlist([1,2,3,4],N) menghasilkan N=10.
- Predikat buangganjil(list1,list2) untuk menghasilkan list baru yang tidak mengandung bilangan ganjil di list1, list1 mungkin kosong; contoh query buangganjil([-1,0,1,2,3,4],X) menghasilkan X = [0,2,4].
- Predikat buangnegatif(list1,list2) untuk menghasilkan list baru yang tidak mengandung bilangan negatif di list1, list1 mungkin kosong; contoh query buangnegatif ([-1,0,2,-4],X) menghasilkan X = [0,2].

Latihan 5

Terdapat sebuah program prolog sederhana sebagai berikut.

predikat1(L):- predikat2(L,L).

predikat2([],[]).

predikat2([H|T],R):- predikat2(T,T1),append(T1,[H],R).

Apakah hasil query berikut ini, yang diterapkan pada program di atas.

- predikat1([ ]).
- predikat1([a]).
- predikat1([a,b]).
- predikat1([a,b,c]).
- predikat1([a,b,a]).

Jelaskan dengan singkat, apakah yang dilakukan oleh program ini.

Loop

Loop in Prolog

- We can use 'repeat' in Prolog
- Example:

```
command_loop:-
      repeat,
      write('Enter command (end to exit): '),
      read(X),
      write(X), nl,
      X == end.
```

- Example taken from: http://www.amzi.com/AdventureInProlog/a14cntrl.php

Loop in Prolog (2)

Other example:

```
command_loop:-
        write('Welcome to Nani Search'), nl,
        repeat,
        write('>nani> '),
        read(X),
        do(X), nl,
        end_condition(X).

end_condition(end).
end_condition(X) :- have(X),!,
                        write('Congratulations').

do(X):- have(X),!.
do(end).
do(_):- write('Invalid Command').

have(X):- X==nani,!.
```

Program stop when user input 'end' or having nani.

Example taken from: http://www.amzi.com/AdventureInProlog/a14cntrl.php

File Eksternal

Writing to Files

- In order to write to a file we have to open a stream
- To write the string 'Hogwarts' to a file with the name hogwarts.txt we do:

…

```
open(‘hogwarts.txt‘, write, Stream),
write(Stream, 'Harry'),
close(Stream),
```

…

- Predicates:
  - open(filename, mode, stream) → write (overwrite), append (add the existing one), read
  - write(Stream, argument) → notes: different from writing to screen
  - tab(Stream, argument)
  - nl(Stream)
  - format(Stream, control, argument)
  - close (Stream)

Reading from Files

- Example file 'houses.txt'

gryffindor.
hufflepuff.
ravenclaw.
slytherin.

Example Prolog program:

```
main:-
        open('houses.txt',read,S),
        read(S,H1),
        read(S,H2),
        read(S,H3),
        read(S,H4),
        close(S),
        write([H1,H2,H3,H4]), nl.
```

Reading from Files (2)

- Predicates:
  - Open/3 (3 terms)
  - Read/2 (2 terms)
  - Close/1 (1 term)
- Only works for Prolog term
- Problem 1: the end of file
- The built-in predicate **at_end_of_stream/1** checks whether the end of a stream has been reached
- It will succeed when the end of the stream (given to it as argument) is reached, otherwise if will fail
- We can modify our code for reading in a file using this predicate

Reading from Files (3)

- Modified program:

```
main:-
        open('houses.txt',read,S),
        readHouses(S,Houses),
        close(S),
        write(Houses), nl.

readHouses(S,[]):-
        at_end_of_stream(S), !.

readHouses(S,[X|L]):-
        \+ at_end_of_stream(S), !,
        read(S,X),
        readHouses(S, L).
```

Reading from Files (4)

- Problem 2: reading arbitraty input
- The predicate **get_code/2** reads the next available character from the stream
  - First argument: a stream
  - Second argument: the character code

Reading from Files (5)

- Example: a predicate **readWord/2** that reads atoms from a file

```
main2:-
        open('coba.txt',read,InStream),
        readWord(InStream,W),
        close(InStream),
        write(W).

readWord(InStream,W):- get_code(InStream,Char),
                checkCharAndReadRest(Char,Chars,InStream),
                atom_codes(W,Chars).

checkCharAndReadRest(10,[],_):- !./*return*/
checkCharAndReadRest(-1,[],_):- !./*end_of_stream*/
checkCharAndReadRest(end_of_file,[],_):- !.
checkCharAndReadRest(Char,[Char|Chars],InStream):-
                get_code(InStream,NextChar),
                checkCharAndReadRest(NextChar,Chars,InStream).
```

THANK YOU