Logikai programozás

3.

ISMÉTLÉS

Elágazás-szervezés:

a/ $B :- A_{11}, A_{12}, \ldots, A_{1n}.$
$B :- A_{21}, A_{22}, \ldots, A_{2k}.$
…
$B :- A_{m1}, A_{m2}, \ldots, A_{mr}.$

b/ $B :- (A_{11}, A_{12}, \ldots, A_{1n})$
;
$(A_{21}, A_{22}, \ldots, A_{2k})$
;
…
;
$(A_{m1}, A_{m2}, \ldots, A_{mr}).$

c/

feltétel -> akció.

ISMÉTLÉS: ELÁGAZÁS (példa)

```
proba1(A, B) :- A < B, writeln('Az első szám kisebb').
proba1(A, B) :- A > B, writeln('A második szám kisebb').
proba1(_,_)   :- writeln('Egyformák').


proba2(A,B) :- A < B, writeln('Az első paraméter kisebb')
               ;
               A > B, writeln('A második paraméter kisebb')
               ;
               writeln('Egyformák').
```

ISMÉTLÉS: ELÁGAZÁS (példa)

```
proba3(A,B) :- A < B -> writeln('Az első szám kisebb')
               ;
               A > B -> writeln('A második szám kisebb')
               ;
               writeln('Egyformák').


proba4(A,B) :- A < B -> writeln('Az első paraméter kisebb')
               ;
                writeln('Nem kisebb az első').
```

KITÉRŐ: CUT (vágás)

„Normál” működés:

B :- A_{11}, A_{12}, … A_{1k-1}, A_{1k}, A_{1k+1}, …, A_{1n}.

B :- A_{21}, A_{22}, …, A_{2m}.

…

Cut-tal (jele: !):

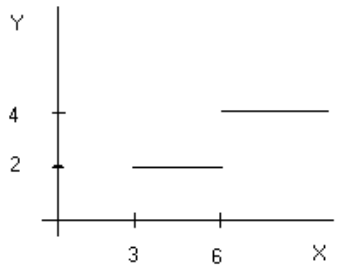
B :- A_{11}, A_{12}, … A_{1k-1}, !, A_{1k}, A_{1k+1}, …, A_{1n}.

B :- A_{21}, A_{22}, …, A_{2m}.

…

CUT – KICSIT RÉSZLETEZVE

Példa:



ha X < 3 $\Rightarrow$ Y = 0,

ha 3<=X és X < 6 $\Rightarrow$ Y = 2,

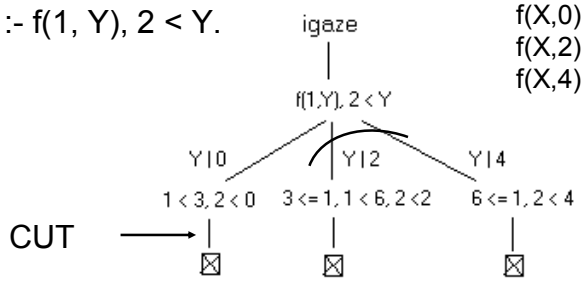
ha 6 <= X $\Rightarrow$ Y = 4

f(X,0) :- X < 3.
f(X,2) :- 3 =< X, X < 6.
f(X,4) :- 6 =< X.

igaze :- f(1, Y), 2 < Y.

CUT – KICSIT RÉSZLETEZVE

igaze :- f(1, Y), 2 < Y.



f(X,0) :- X < 3.
f(X,2) :- 3 =< X, X < 6.
f(X,4) :- 6 =< X.

f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.

vagy:

igaze :- f(1, Y),!, 2 < Y.

A „cut” növelte a program hatékonyságát, de nem befolyásolta a deklaratív jelentését ⇒ **green (zöld) cut**

CUT – KICSIT RÉSZLETEZVE

Új cél: ?:- f(7, Y)

f(X,0) :- X < 3.
f(X,2) :- 3 =< X, X < 6.
f(X,4) :- 6 =< X.

7 < 3 hamis. ⇒ a 3 =< 7 feltételt fölösleges még egyszer megvizsgálni a második szabály kiértékelésekor.
A program ügyesebb átfogalmazása:

f(X,0) :- X < 3, !.
f(X,2) :- X < 6, !.
f(X,4).

Ennek a programnak ugyanaz az eredménye, mint az eredetinek, de hatékonyabb mindkét előző verziónál.

CUT – KICSIT RÉSZLETEZVE

Mi a helyzet cut nélkül?

```
f(X,0) :- X < 3, !.          f(X,0) :- X < 3.
f(X,2) :- X < 6, !.          f(X,2) :- X < 6.
f(X,4).                      f(X,4).
```

Az alternatívák végigvizsgálása miatt többszörös megoldást nyújthat, melyek közül némelyik helytelen.

```
?:- f(1,Y) :
Y=0
Y=2
Y=4
```

Megváltozott a program deklaratív jelentése is ⇒ **red (vörös) cut**

CUT – PÉLDA

Példa:

```
fovaros(franciao, parizs).
fovaros(usa, washington).
fovaros(magyaro, budapest).

uzemel(parizs, hp).          elad(hp, 120).
uzemel(parizs, dior).        elad(dior, 200).
uzemel(parizs, chanel).      elad(chanel, 300).
uzemel(budapest, kinoin).    elad(kinoin, 130).
uzemel(budapest, mom).       elad(mom, 140).

nagy(C) :- fovaros(C, X), uzemel(X, Y), elad(Y, Z), Z > 100.

nagy(C) :- fovaros(C,X), valami(X).
valami(X) :-  uzemel(X,Y), elad(Y, Z), Z > 100, ! .
```

„CIKLUSSZERVEZÉS”: REKURZIÓ (ism.)

A rekurzió részei:

1. rekurzív szabály(ok)

   (szabályfejben, szabálytörzsben ugyanaz a predikátumszimbólum szerepel)

2. megállító szabály(ok)

   (szabályfejben, szabálytörzsben nem szerepel ugyanaz a predikátumszimbólum)

REKURZIÓ – PÉLDA

Határozzuk meg n! értékét!

Vagyis:

1. rekurzív szabály:

   ha N >1, akkor számoljuk ki N-1 faktoriálisát, és szorozzuk meg N-nel. (Azaz n! = n*(n-1)!, n>1)

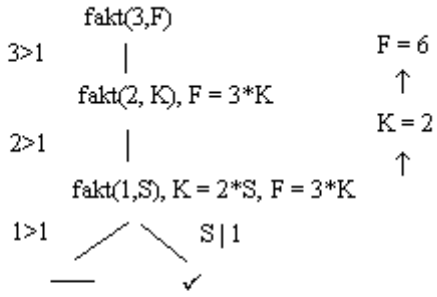
2. megállító feltétel:

   1 faktoriális értéke 1.

```
fakt(N,F) :- N>1, M is N-1, fakt(M,K), F is N*K.
fakt(1,1).
```

KIÉRTÉKELÉS

fakt(N,F) :- N>1, M is N-1, fakt(M,K), F is N*K.
fakt(1,1).



FAKTORIÁLIS MÁSKÉPP

Nem rekurzív megadás: N! = 1*2*3*… * I * … * N.

```
I = 1; P = 1;
ciklus amíg I <= N
        P = P * I;
        I  = I + 1;
ciklusvég

Fakt = P
```

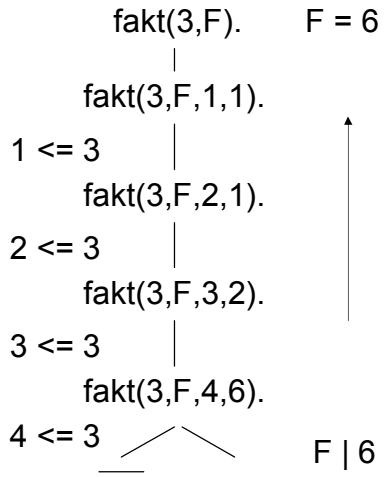
```
fakt(N,F) :- fakt(N,F,1,1).

fakt(N,F,I,P) :- I =< N,
                 UjP is P*I,
                 UjI is I + 1,
                 fakt(N,F,UjI, UjP).

fakt(_,F,_,F).
```

KIÉRTÉKELÉS

fakt(N,F,I,P) :- I =< N, UjP is P*I, UjI is I + 1, fakt(N,F,UjI, UjP).



Jobb-rekurzió

(tail recursion)

Hatékonyabb, mint a másik.

ÚTKERSÉSI PÉLDA

ut(Start,Cel) :- repulo(Start,Cel).
ut(Start,Cel) :- repulo(Start,Uj_varos), ut(Uj_varos,Cel).

Problémák:

Megoldás (előkészítése):

Tárolni kell az adatokat!

(Legalább a kiértékelés idejéig.)

LISTÁK

Lista: elemek felsorolása, pl:

[1, 3, 5, 4, 12, 27]

[alma, körte, barack]

[alma, 200, banán, 350, narancs, 270]

[(alma, 200), (banán, 350), (narancs, 270)]

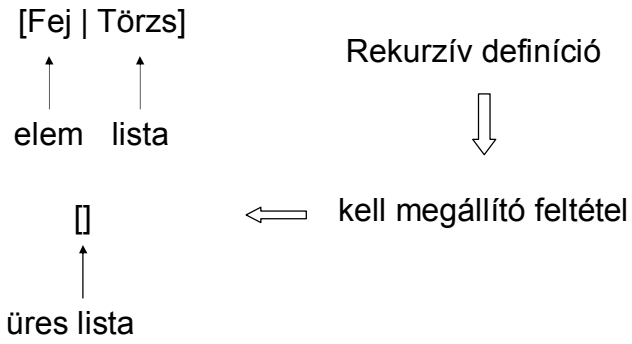
[[alma, 200], [banán, 350], [narancs, 270]]

[piaci(alma, 200), piaci(banán, 350), piaci(narancs, 270)]

stb…

LISTÁK

Lista definiálása:



LISTÁK – NÉHÁNY EGYSZERŰ PÉLDA

– egy elem benne van-e egy listában?
– hány elemű a lista?
(„sima” rekurzióval és jobb-rekurzióval)
– egy elem törlése egy listából
– a elemek szétválogatása

stb…