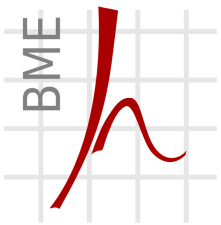
Kód optimalizálás

Adatáram analízis



Híradástechnikai Tanszék

Izsó Tamás

2012. szeptember 20.

Section 1

Kód optimalizálás

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = a * a;$$
$$d = b + c;$$
$$e = b + b;$$

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = a * a;$$
$$d = b + c;$$
$$e = b + b;$$

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$

$$c = a * a;$$

$$d = b + c;$$

$$e = b + b;$$

Ismétlődő kifejezések kiszűrése

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + c;$$
$$e = b + b;$$

Ismétlődő kifejezések kiszűrése

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + c;$$
$$e = b + b;$$

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + c;$$
$$e = b + b;$$

Másolat terjedés

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + b;$$
$$e = b + b;$$

Másolat terjedés

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + b;$$
$$e = b + b;$$

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + b;$$
$$e = b + b;$$

Ismétlődő kifejezések kiszűrése

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + b;$$
$$e = d;$$

Ismétlődő kifejezések kiszűrése

Példa lokális optimalizálás végrehajtása

$$b = a * a;$$
$$c = b;$$
$$d = b + b;$$
$$e = d;$$

Lokális optimalizálás alkalmazása

A műveleteket addig kell folytatni, amíg a szabályok közül valamelyiket alkalmazni lehet. Mivel a műveletek a eredményt *egy irányba* változtatják, ezért az eljárás mindig leáll.

1. ismétlődő kifejezések kiszűrése;
2. másolat továbbterjedésének a megakadályozása;
3. felesleges utasítások törlése.

Optimalizálás és analízis kapcsolata

Az optimalizálás során szükségünk van a program viselkedésének a megértésére. Az ismétlődő kifejezések elhagyása és a másolat terjedésének az optimalizálásához a **kifejezés elérhetőségének** (available expression) a vizsgálatára van szükség.

Definició: Elérhető kifejezés

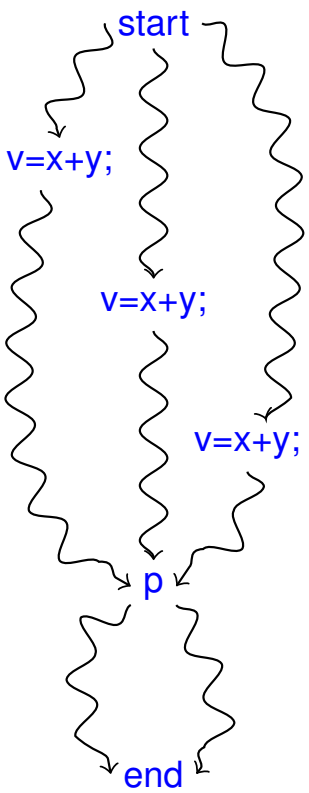
A program adott pontján egy kifejezés akkor érhető el, ha van olyan változó, ami hordozza a kifejezés értékét.

Elérhető kifejezések előállítása

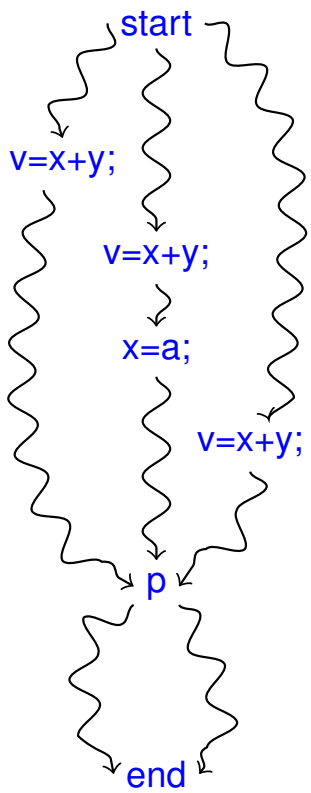
- Az alapblokk elején az *elérhető kifejezés*ek halmaza üres.
- Amikor egy **a = b + c** kifejezéshez érünk:
  - Minde kifejezést, amely az **a**-t tartalmazza, ki kell venni a halmazból.
  - Az **a = b + c** kifejezést be kell tenni az elérhető kifejezések halmazába.
- Mit kell tenni, ha az **a = a + b** kifejezéssel találkozunk?

Elérhető kifejezés előállítása

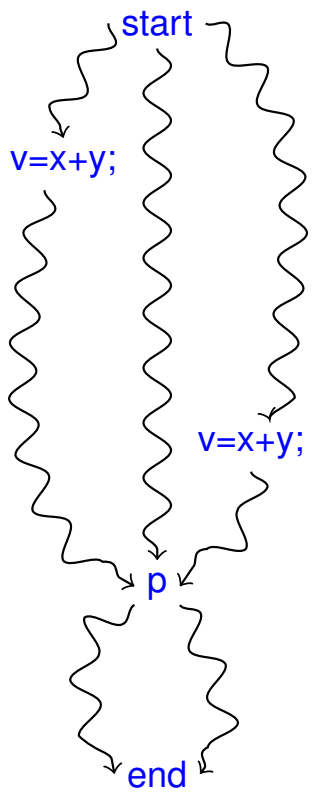
v=x+y a p pontban elérhető



v=x+y a p pontban nem elérhető



v=x+y a p pontban nem elérhető



Elérhető kifejezés előállítása

```
a = b;

c = b;

d = a + b;

e = a + b;

d = b

f = a + b;
```

Elérhető kifejezés előállítása

{}

a = b;

c = b;

d = a + b;

e = a + b;

d = b

f = a + b;

Elérhető kifejezés előállítása

{}

a = b;

{ a = b }

c = b;

d = a + b;

e = a + b;

d = b

f = a + b;

Elérhető kifejezés előállítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

e = a + b;

d = b

f = a + b;

Elérhető kifejezés előállítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = a + b;

d = b

f = a + b;

Elérhető kifejezés előállítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = a + b;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

f = a + b;

Elérhető kifejezés előállítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = a + b;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

Elérhető kifejezés előállítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = a + b;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = a + b;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

{}

a = b;

{ a = b }

c = b;

{ a = b, c = b }

d = a + b;

{ a = b, c = b, d = a + b }

e = a + b;

{ a = b, c = b, d = a + b , e = a + b }

d = b

{ a = b, c = b, d = b , e = a + b }

f = a + b;

{ a = b, c = b, d = b , e = a + b , f + a + b}

Azonos kifejezések eltávolítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = d;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = d;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = d;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = a + b;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

{}

a = b;

{ a = b }

c = b;

{ a = b, c = b }

d = a + b;

{ a = b, c = b, d = a + b }

e = d;

{ a = b, c = b, d = a + b , e = a + b }

d = b

{ a = b, c = b, d = b , e = a + b }

f = e;

{ a = b, c = b, d = b , e = a + b , f + a + b}

Azonos kifejezések eltávolítása

$\{\}$

a = b;

$\{ a = b \}$

c = b;

$\{ a = b, c = b \}$

d = a + b;

$\{ a = b, c = b, d = a + b \}$

e = d;

$\{ a = b, c = b, d = a + b , e = a + b \}$

d = b

$\{ a = b, c = b, d = b , e = a + b \}$

f = e;

$\{ a = b, c = b, d = b , e = a + b , f + a + b\}$

Azonos kifejezések eltávolítása

a = b;

c = b;

d = a + b;

e = d;

d = b

f = e;

Elérhető kifejezés előállításásának a formalizálása

- Jelöljük D-vel a kifejezések halmazát, $s_k \in D$-vel az egyes kifejezéseket, és legyen s_1 az első kifejezés a program-ban.
- $f_s(x) : x \in D \rightarrow D$ függvény a program s pontjában az elérhető kifejezések halmazához hozzátesz és/vagy elvesz elemeket.
- Gen_s az s kifejezést hozzáadja a halmazhoz
- $Kill_s$ minden kifejezést kivesz az x halmazból, amely tartalmazza az s bal oldalán lévő változót.
- $In_n \subseteq D$ az $f_n(x)$ lehetséges bemenő értéke
- $Out_n = f(In_n)$ ahol $f_n(x) = Gen_n \cup (x - Kill_n)$
- $In_n \begin{cases} s_1 \text{esetén} \emptyset, \text{különben az összes kifejezés} & \text{inicializálásnál} \\ \cap_{p \in pred(n)} Out_p, & \text{egyébként} \end{cases}$

Elérhető kifejezés előállításásának a formalizálása

A D halmazt reprezentálhatjuk logikai értékek vektorával jelőljük L-lel.

bit pozició	1	2	3	4	5	6
kifejezés	a=b	c=b	d=a+b	e=a+b	d=b	f=a+b

Például: $\{a = b, d = a + b, f = a + b\} = \langle 101001 \rangle$
Ekkor $f_n(x) = Gen_n \vee (x \wedge \sim Kill_n)$

$$f_3 = f_{d=a+b} = \begin{cases} Gen_3, & \langle 001000 \rangle \\ Kill_3, & \langle 000010 \rangle \end{cases}$$

$$\begin{aligned} f_3(\langle 110010 \rangle) &= \langle 001000 \rangle \vee (\langle 110010 \rangle \wedge \sim \langle 000010 \rangle) \\ &= \langle 001000 \rangle \vee (\langle 110010 \rangle \wedge \langle 111101 \rangle) \\ &= \langle 001000 \rangle \vee \langle 110000 \rangle \\ &= \langle 111000 \rangle \end{aligned}$$

Alapblokk $f_B(x)$ függvényének a kiszámítása

Minden alapblokk $s_1 \ldots s_n$ utasítások szekvenciális sorozatát tartalmazza, ezért az f_B függvény az f_i függvényekből egyszerűen előállítható.

$$\begin{aligned} f_B(x) &= f_n \circ f_{n-1} \ldots f_2 \circ f_1(x) \\ &= f_n(f_{n-1}(\ldots f_2(f_1(x)) \ldots)); \end{aligned}$$

Két utasításra:

$$\begin{aligned} f_B(x) &= f_2 \circ f_1(x) \\ &= Gen_2 \vee (Gen_1 \vee (x \wedge \sim Kill_1) \wedge \sim Kill_2) \\ &= Gen_2 \vee ((Gen_1 \wedge \sim Kill_2) \vee (x \wedge \sim (kill_1 \vee kill_2)) \end{aligned}$$

Adatfolyam analízis keretalgoritmusa

```
procedure worklist_iteration( N, entry, F, dfin, Init )
    N : in set of Node;
    entry : in Node;
    F : in Node x L → L;
    dfin : out Node → L;
    Init: in L;
begin
    B, P : Node;
    Worklist : set of Node;
    totaleffect : L;
    dfin(entry):= Init;
    Worklist := N - {entry};
    for each B ∈ N - {entry} do
        dfin(B) := ⊤;
    od;
```

Adatfolyam analízis keretalgoritmusa

```
repeat
    B := ♦ Worklist;
    Worklist -= {B};
    totaleffect := ⊤;
    for each P ∈ Pred(B) do
        totaleffect ⊓= F(P,dfin(P));
    od;
    if dfin(B) ≠ totaleffect then
        dfin(B) := totaleffect;
        Worklist ∪= Succ(B);
    fi;
until Worklist = ∅ ;
end
```

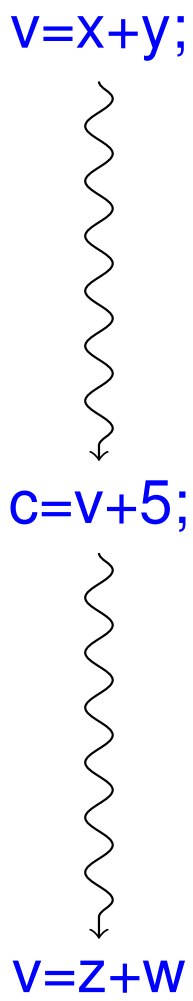
Aktív változók

- Angol irodalomban a Live Variables kifejezést használják, míg az azt felhasználó elemzést Liveness Analysis-nek hívják.
- Az aktív változók ismeretében lehet a felesleges utasításokat megszüntetni.
- Egy változó a program egy pontján aktív, ha később az értéke felhasználásra kerül.

Aktív változók előállítása

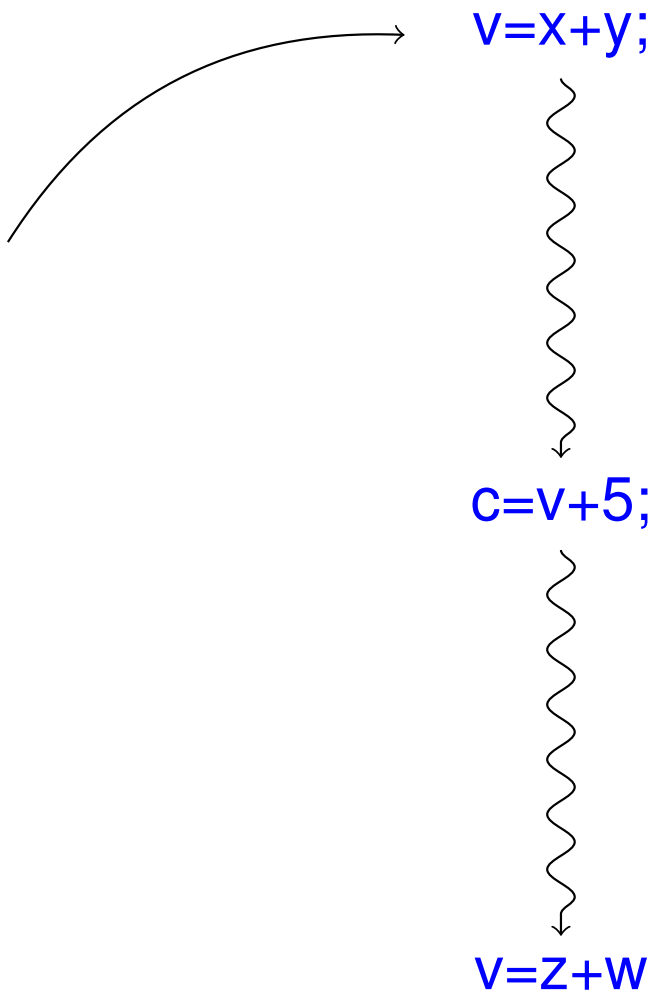
- A feldolgozás az alapblokk végétől az eleje felé halad.
- A blokk végén lévő aktív változók halmazát az algoritmus alapján mi határozzuk meg.
- Amikor az **a = b + c** kifejezéshez érünk:
  - az **a** változót ki kell venni;
  - a **b** és **c** változókat be kell tenni az az aktív változók halmazába.

Aktív változó



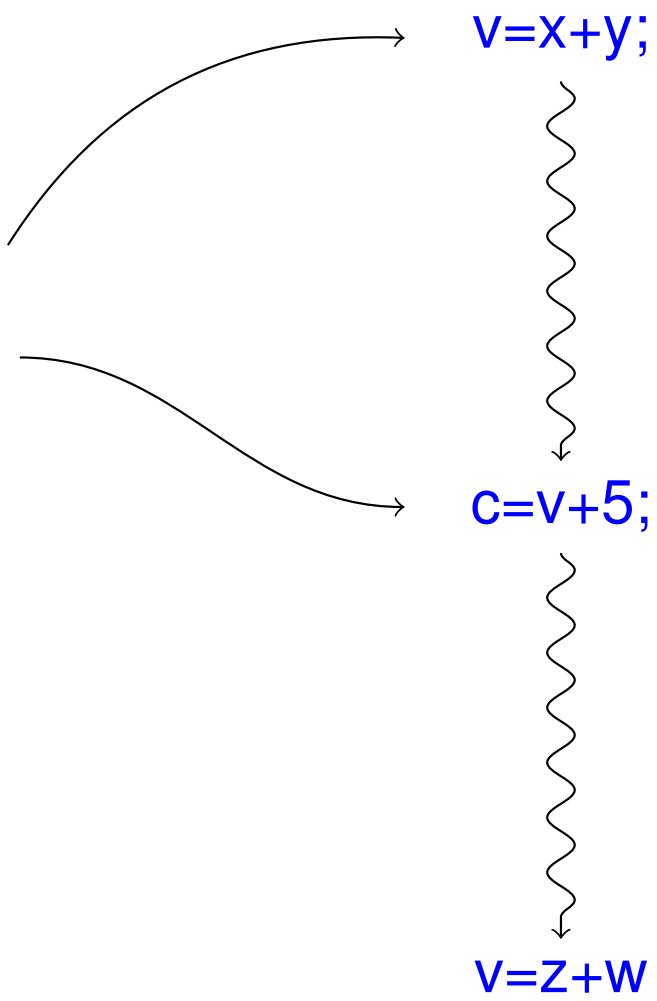
Aktív változó

- *v* változó definiálása



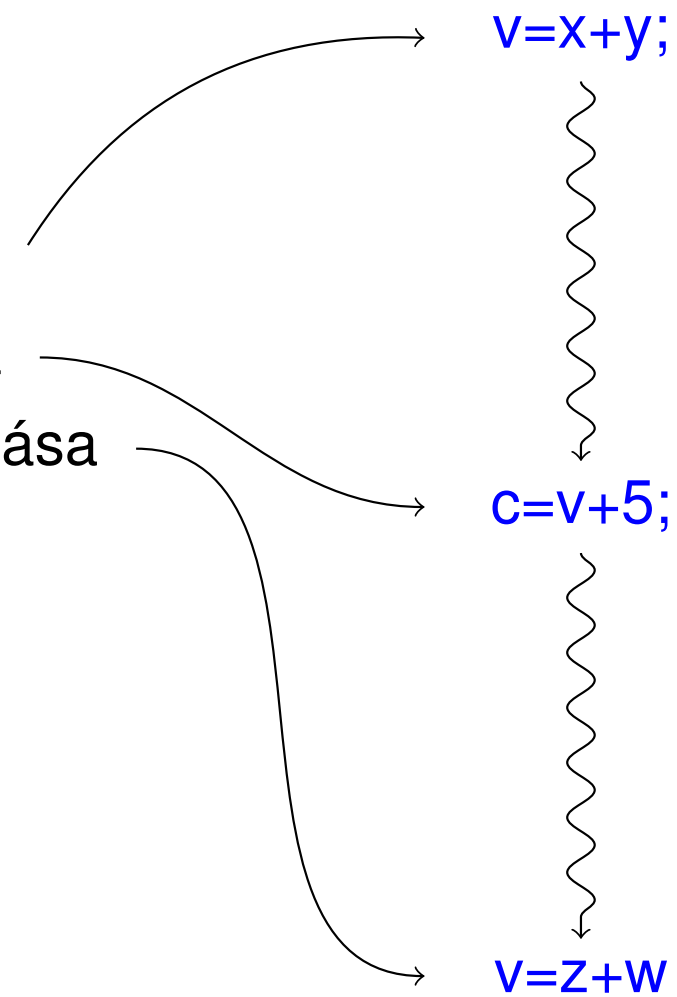
Aktív változó

- *v* változó definiálása
- *v* változó használata



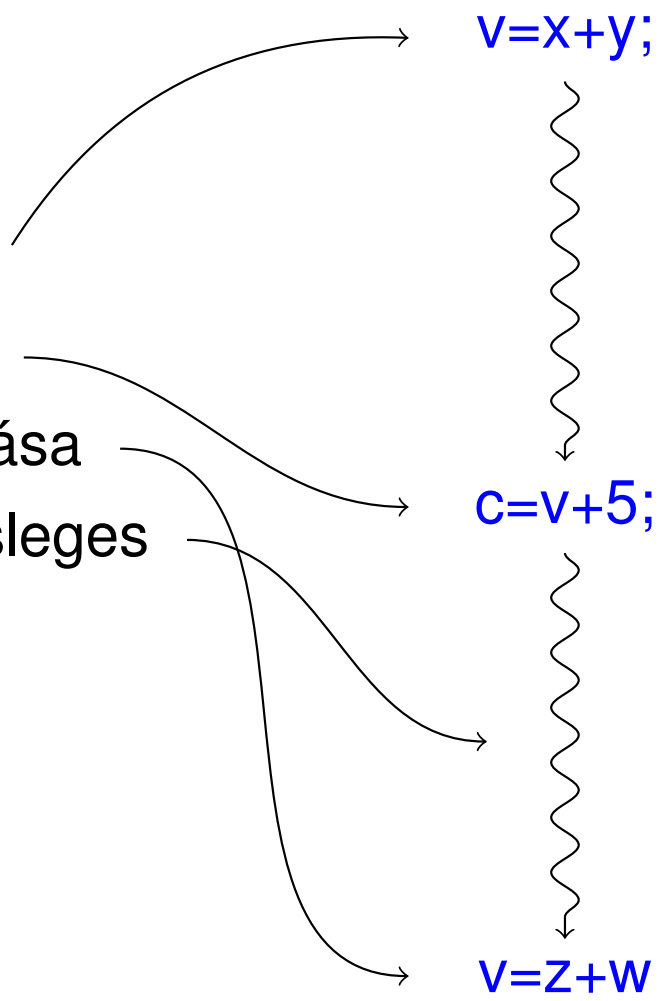
Aktív változó

- *v* változó definiálása
- *v* változó használata
- *v* változó újradefiniálása



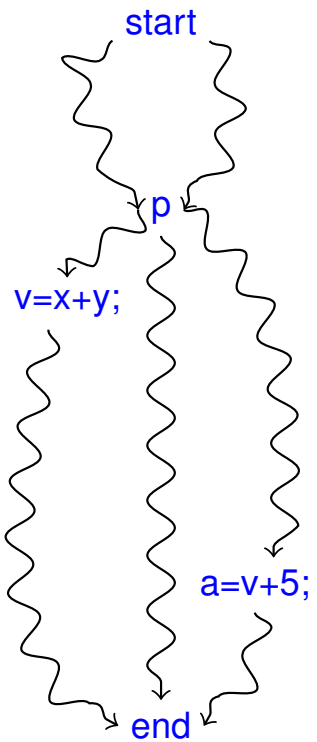
Aktív változó

- *v* változó definiálása
- *v* változó használata
- *v* változó újradefiniálása
- *v* változó értéke felesleges

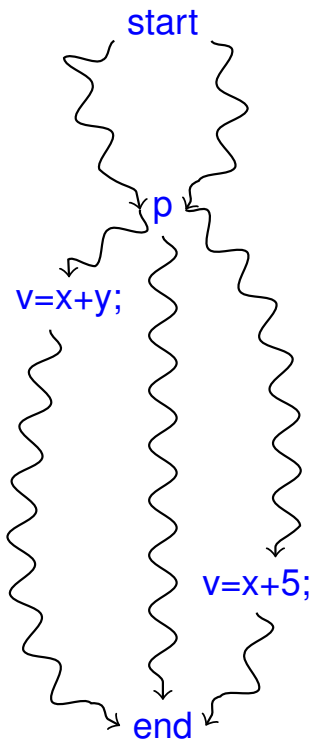


Aktív változó globális analízise

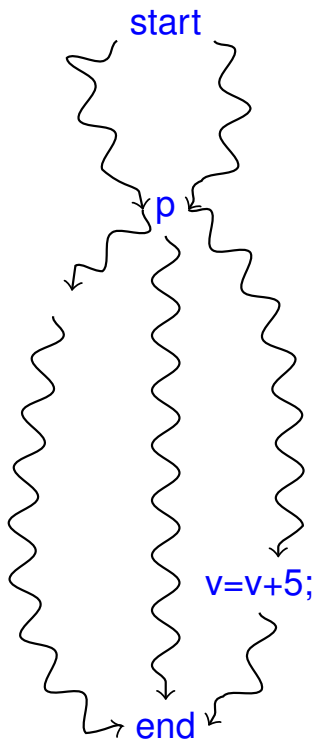
v változó aktív
a p pontban



v változó nem aktív
a p pontban



v változó aktív
a p pontban



Aktív változók előállítása

a = b;

c = a;

d = a + b;

e = d;

d = a;

f = e;

Aktív változók előállítása

a = b;

c = a;

d = a + b;

e = d;

d = a;

f = e;

{b , d }

Aktív változók előállítása

a = b;

c = a;

d = a + b;

e = d;

d = a;
{ b, d, e}
f = e;
{b , d }

Aktív változók előállítása

a = b;

c = a;

d = a + b;

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Aktív változók előállítása

a = b;

c = a;

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Aktív változók előállítása

a = b;

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Aktív változók előállítása

a = b;
{ a, b }
c = a;
{ a, b }
d = a + b;
{ a, b, d }
e = d;
{ a, b, e}
d = a;
{ b, d, e}
f = e;
{b , d }

Aktív változók előállítása

{ b }

a = b;

{ a, b }

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Felesleges kód törlése

{ b }

a = b;

{ a, b }

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Felesleges kód törlése

{ b }

a = b;

{ a, b }

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

f = e;

{b , d }

Felesleges kód törlése

{ b }

a = b;

{ a, b }

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

{b , d }

Felesleges kód törlése

{ b }

a = b;

{ a, b }

c = a;

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

{b , d }

Felesleges kód törlése

{ b }

a = b;

{ a, b }

{ a, b }

d = a + b;

{ a, b, d }

e = d;

{ a, b, e}

d = a;

{ b, d, e}

{b , d }

Felesleges kód törlése

```
a = b;

d = a + b;

e = d;

d = a;
```

Aktív változók előállítása II

```
a = b;

d = a + b;

e = d;

d = a;
```

Aktív változók előállítása II

a = b;

d = a + b;

e = d;

d = a;

{b , d }

Aktív változók előállítása II

a = b;

d = a + b;

e = d;

{a , b }

d = a;

{b , d }

Aktív változók előállítása II

a = b;

d = a + b;

{a , b, d }

e = d;

{a , b }

d = a;

{b , d }

Aktív változók előállítása II

a = b;

{a , b }

d = a + b;

{a , b, d }

e = d;

{a , b }

d = a;

{b , d }

Aktív változók előállítása II

{ b }

a = b;

{a , b }

d = a + b;

{a , b, d }

e = d;

{a , b }

d = a;

{b , d }

Felesleges kód törlése

{ b }
a = b;

{a , b }
d = a + b;
{a , b, d }
e = d;
{a , b }
d = a;

{b , d }

Felesleges kód törlése

{ b }
a = b;

{a , b }
d = a + b;
{a , b, d }
e = d;
{a , b }
d = a;

{b , d }

Felesleges kód törlése

{ b }
a = b;

{a , b }
d = a + b;
{a , b, d }

{a , b }
d = a;

{b , d }

Felesleges kód törlése

```
a = b;
```

```
d = a + b;
```

```
d = a;
```

Aktív változók előállítása III

a = b;

d = a + b;

d = a;

Aktív változók előállítása III

a = b;

d = a + b;

d = a;

{b , d }

Aktív változók előállítása III

a = b;

d = a + b;

{a , b }
d = a;

{b , d }

Aktív változók előállítása III

a = b;

{a , b }
d = a + b;

{a , b }
d = a;

{b , d }

Aktív változók előállítása III

{ b }
a = b;

{a , b }
d = a + b;

{a , b }
d = a;

{b , d }

Felesleges kód törlése

{ b }
a = b;

{a , b }
d = a + b;

{a , b }
d = a;

{b , d }

Felesleges kód törlése

{ b }
a = b;

{a , b }
d = a + b;

{a , b }
d = a;

{b , d }

Felesleges kód törlése

$\{ b \}$

a = b;

$\{a , b \}$

$\{a , b \}$

d = a;

$\{b , d \}$

Felesleges kód törlése

```
a = b;
```

```
d = a;
```

Globális analízis – példa elérhető kifejezésre

```
1:   i = 0
2:   a = n * 3
3:   t1 = i < a
4:   IfZ t1 Goto End
5:   loop: b = i * 4
6:   c = p + b
7:   d = M[c]
8:   e = d * 2
9:   f = i * 4
10:  g = p + f
11:  M[g] = e
12:  i = i +1
13:  a = n * 3
14:  t2 = i < a
15:  IfZ t2 Goto end
16:  Goto loop
17:  end:
```

i	$pred_i$	gen_i	$kill_i$
1		1	3,5,9,12,14
2	1	2	3,14
3	2	3	
4	3		
5	4,16	5	6
6	5	6	7
7	6	7	8
8	7	8	
9	8	9	10
10	9	10	
11	10		7
12	11		1,3,5,9,14
13	12	2	2,3,14
14	13	14	
15	14		
16	15		
17	4,15		

Fontos 2 és 13 azonos, ezért a 13-ik utasításra is 2-vel hivatkozunk!

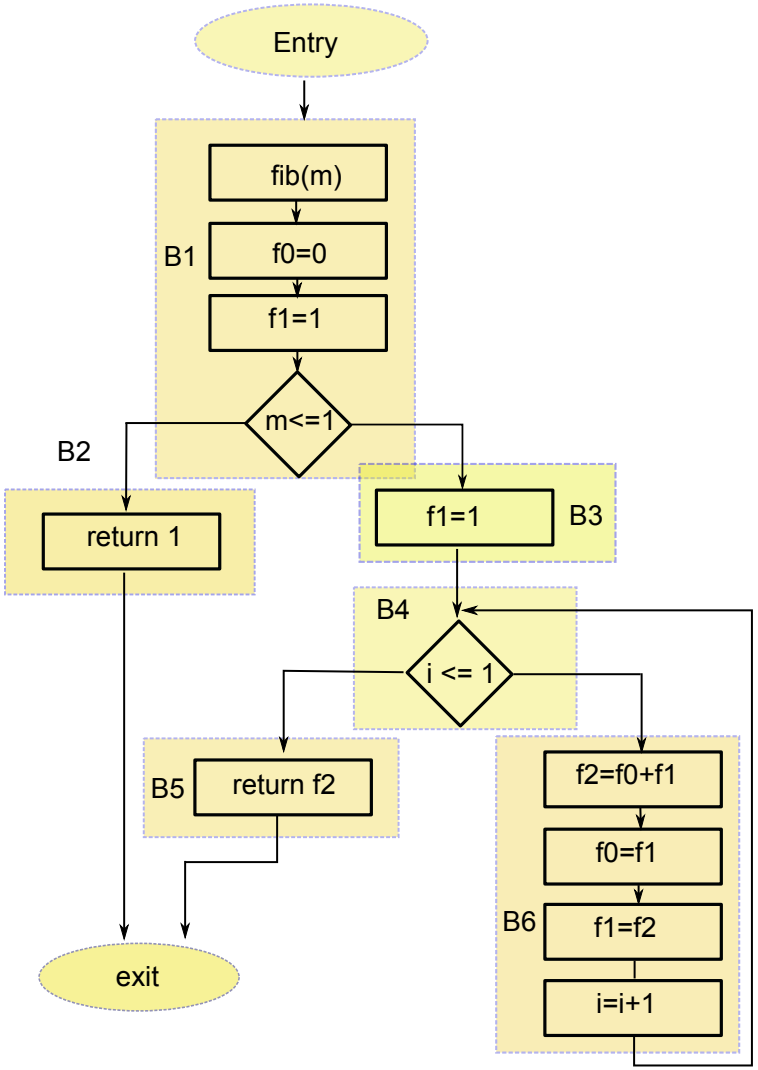
Globális analízis – példa elérhető kifejezésre

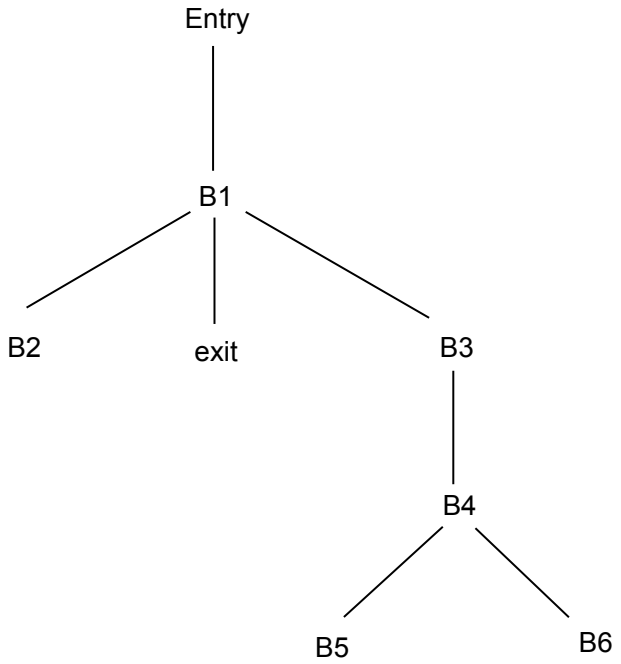
	Init		Iter 1	
i	in_i	out_i	in_i	out_i
1		1,2,3,5,6,7,8,9,10,14		1
2	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1	1,2
3	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2	1,2,3
4	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3	1,2,3
5	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3	1,2,3,5
6	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5	1,2,3,5,6
7	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6	1,2,3,5,6,7
8	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7	1,2,3,5,6,7,8
9	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8	1,2,3,5,6,7,8,9
10	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9	1,2,3,5,6,7,8,9,10
11	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10	1,2,3,5,6,8,9,10
12	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,8,9,10	2,6,8,10
13	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	2,6,8,10	2,6,8,10
14	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	2,6,8,10	2,6,8,10,14
15	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	2,6,8,10,14	2,6,8,10,14
16	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	2,6,8,10,14	2,6,8,10,14
17	1,2,3,5,6,7,8,9,10,14	1,2,3,5,6,7,8,9,10,14	2	2

Globális analízis – példa elérhető kifejezésre

	Iter2		Iter 3	
i	in_i	out_i	in_i	out_i
1		1		1
2	1	1,2	1	1,2
3	1,2	1,2,3	1,2	1,2,3
4	1,2,3	1,2,3	1,2,3	1,2,3
5	2	2,5	2	2,5
6	2,5	2,5,6	2,5	2,5,6
7	2,5,6	2,5,6,7	2,5,6	2,5,6,7
8	2,5,6,7	2,5,6,7,8	2,5,6,7	2,5,6,7,8
9	2,5,6,7,8	2,5,6,7,8,9	2,5,6,7,8	2,5,6,7,8,9
10	2,5,6,7,8,9	2,5,6,7,8,9,10	2,5,6,7,8,9	2,5,6,7,8,9,10
11	2,5,6,7,8,9,10	2,5,6,8,9,10	2,5,6,7,8,9,10	2,5,6,7,8,9,10
12	2,5,6,8,9,10	2,6,8,10	2,5,6,7,8,9,10	2,6,8,10
13	2,6,8,10	2,6,8,10,13	2,6,8,10	2,6,8,10,13
14	2,6,8,10,13	2,6,8,10,13,14	2,6,8,10,13	2,6,8,10,13,14
15	2,6,8,10,14	2,6,8,10,14	2,6,8,10,13,14	2,6,8,10,14
16	2,6,8,10,14	2,6,8,10,14	2,6,8,10,14	2,6,8,10,14
17	2	2	2	2

Struktúrális analízis – dominancia

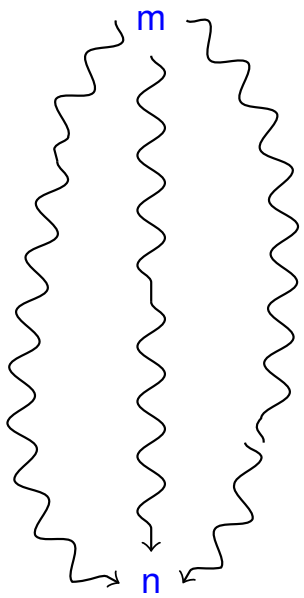




Domináns alapblokk

- A B_i alapblokk akkor domináns B_j felett, ha az *e*ntry pontból a j-edik blokkig minden lehetséges úton eljutva érinteni kell a B_i blokkot.
- Minden blokk domináns önmagával.
- $(a \text{ dom } b) \wedge (b \text{ dom } c) \rightarrow (a \text{ dom } c)$.
- Közvetlen, vagy szomszédos dominanciáról akkor beszélünk, ha a dom b és $a \neq b$ esetén nem létezik olyan c, hogy $(a \text{ dom } c) \wedge (c \text{ dom } b)$.
- Posztdominanciáról akkor beszélünk, ha a B_j blokkból az *e*xit, blokk fele haladva minden úton érintjük a B_k blokkot.

Domináns alapblokk



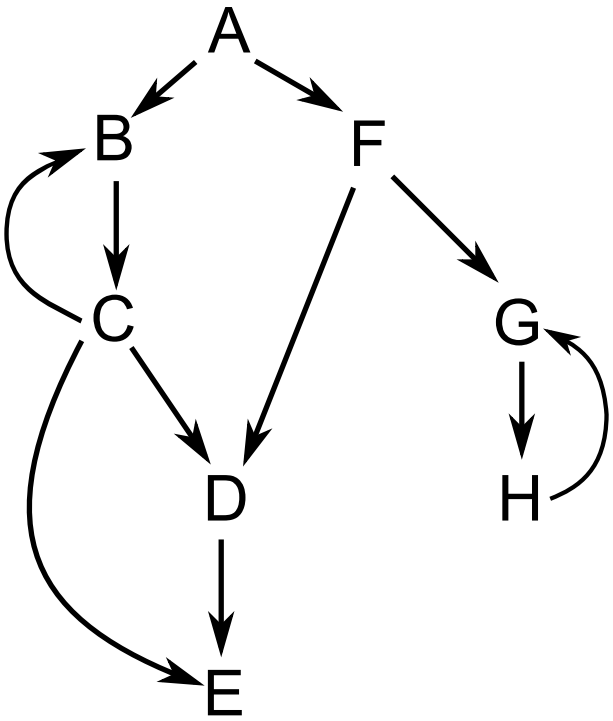
- $D(n)$ az n-re domináns alapblokkok halmaza
- $Pred(n)$ az irányított vezérlési folyamatgráf alapján az n-be menő élek forrás csomópontjai
- $$D(n) = \{n\} \cup \bigcap_{p \in pred(n)} D(p)$$

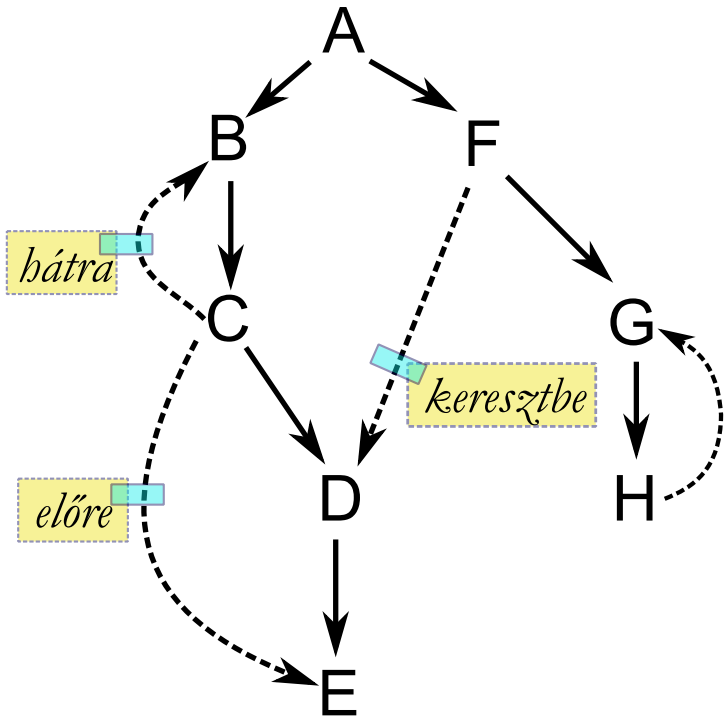
Dominancia meghatározása

```
procedure Dom_Comp( N, Pred, r )
    N : in set of Node;
    Pred : in Node → set of Node;
    r : in Node;
begin
    D, T : set of Node;
    n, p : Node;
    change:=true: boolean;
    Domin : Node → set of Node;
    for each n ∈ N - {r} do Domin(n):=N;
    repeat
        change := false;
        for each n ∈ N - {r} do
            T:=N;
            for each p ∈ Pred(n) do
                T ∩ = Domin(p) ;
            od;
            D := { n } ∪ T;
            if D ≠ Domin(n) then
                change := true;
                Domin(n) := D;
            fi;
        od;
    until !change
return Domin
```

Depth-First Search

Mélységi keresés





A B C D E D C B A F G H G F A

A F D E D F G H G F A B C B A

DFS algoritmus

```
N : in set of Node;
r, x : Node
i:=1, j:=1 : integer;
Pre, Post : Node → integer;
Visit: Node → boolean;
Etype: (Node x Node )
    → enum{ tree, forward, back, cross };

procedure Deap_First_Search( N, Succ, x )
    N : in set of Node;
    Succ : in Node → set of Node;
    x : in Node;
begin
    y : Node;
    Visit(x) := true;
    Pre(x) := j;
    j = j + 1;
    for each y ∈ Succ(x) do
        if !Visit(y) then
          Deep_First_Search(N,Succ,y);
          EType(x → y) := tree;
        elif Pre(x) < Pre(y) then
          EType(x → y) := forward;
        elif Post(y) = 0 then
          EType(x → y) := back;
        else
          EType(x → y) := cross;
        fi;
    od;
    Post(x) := i; i :+= 1;
end

begin
    for each x ∈ N do;
        Visit(x) := false;
        Pre(x) := Post(x) := 0;
    od
    Deep_First_Search(N,Succ,r);
end
```