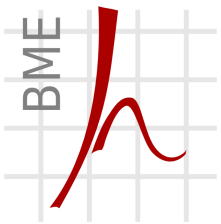
Fordítás – Kódoptimalizálás

Kód visszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2016. október 20.

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.

Példa *aktív változóra*

Példa:

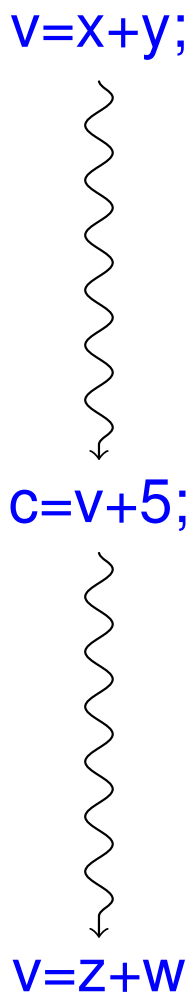
```
[x:=2]1;
[y:=4]2;
[x:=1]3;
if [y>x]4do
  then [z:=y]5
  else [z:=y*y]6
fi
[x:=z]7;
```

Az 1-es címkéjű `x` változó nem aktív az első utasítás után (semelyik utasítás sem használja fel az értékét), ezért ez felesleges.

Aktív változók előállítása

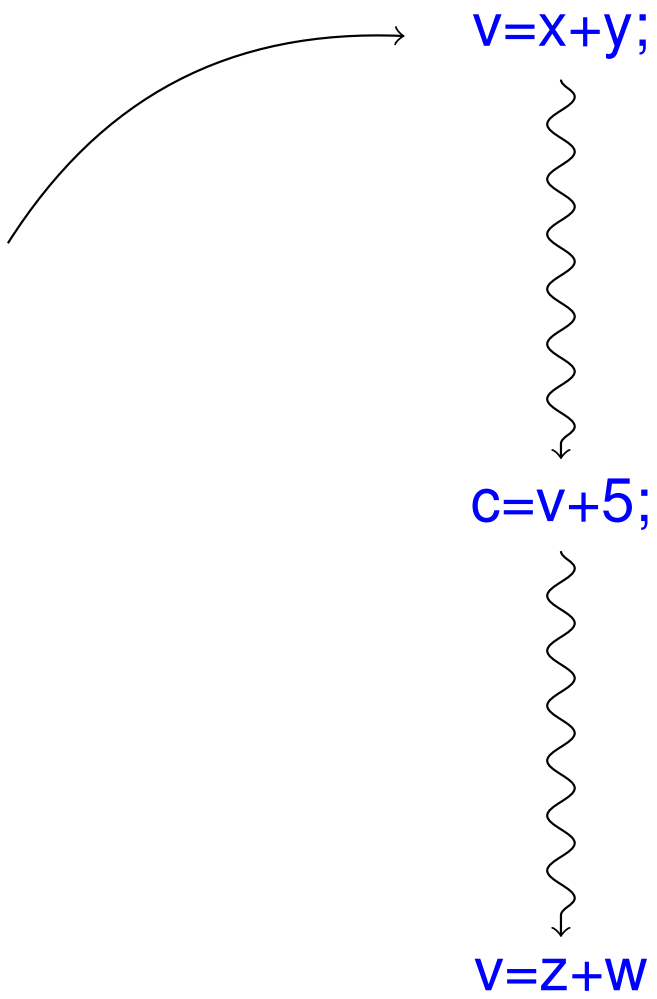
- A feldolgozás az alapblokk végétől az eleje felé halad.
- A blokk végén lévő aktív változók halmazát az algoritmus alapján mi határozzuk meg.
- Amikor az **a = b + c** kifejezéshez érünk:
  - az **a** változót ki kell venni;
  - a **b** és **c** változókat be kell tenni az az aktív változók halmazába.

Aktív változó



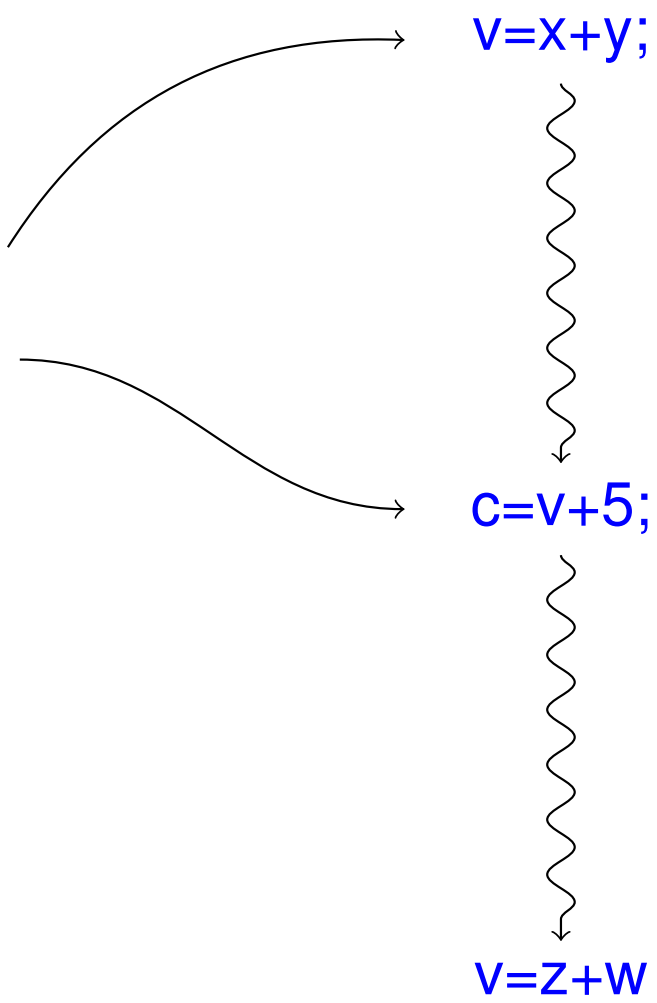
Aktív változó

- *v* változó definiálása



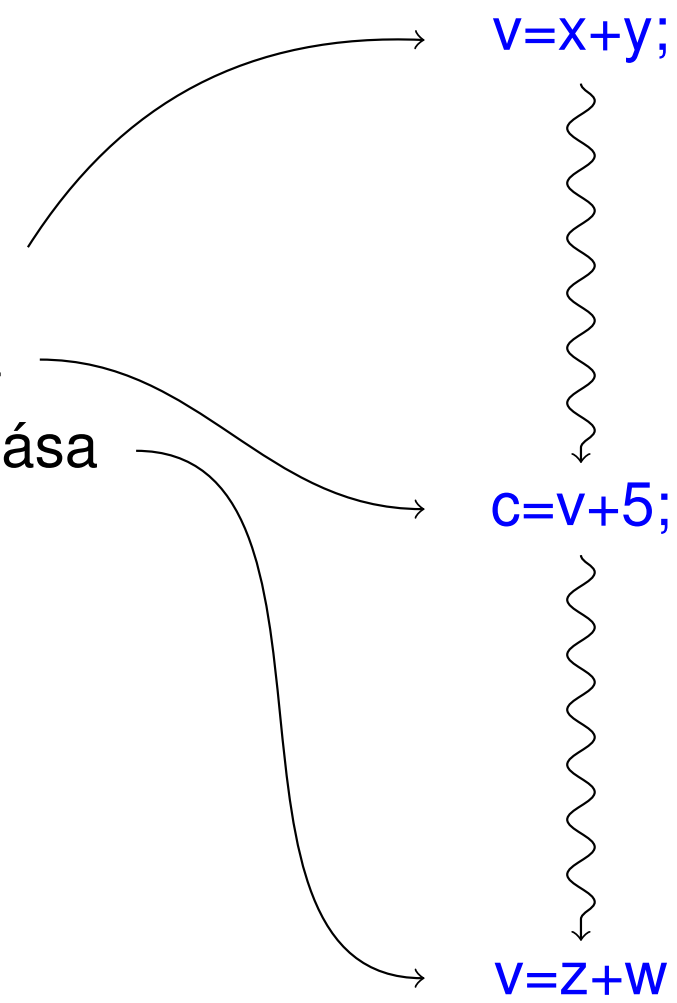
Aktív változó

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



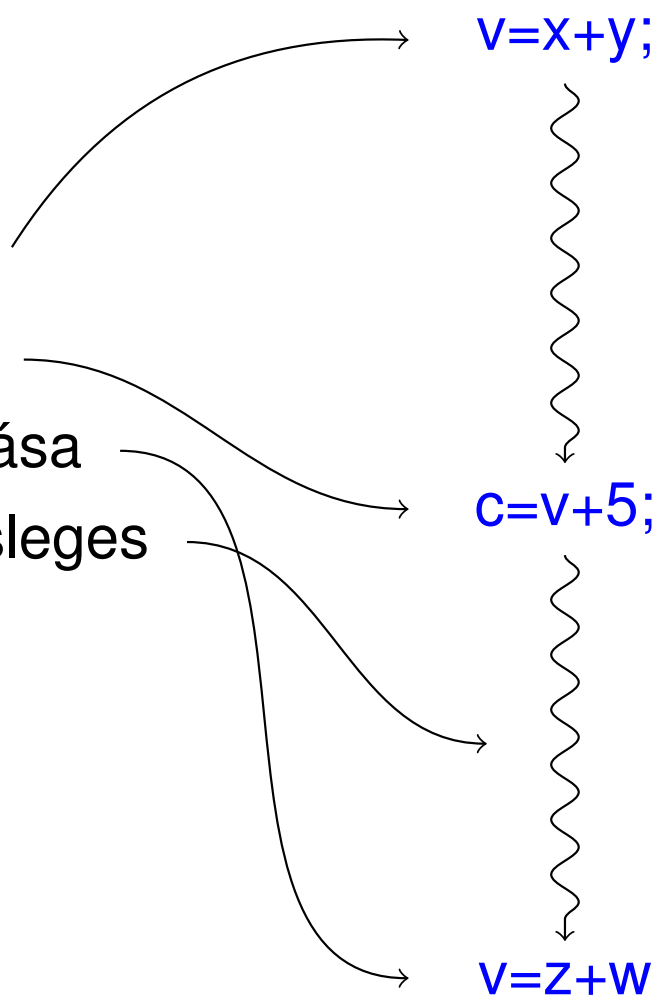
Aktív változó

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



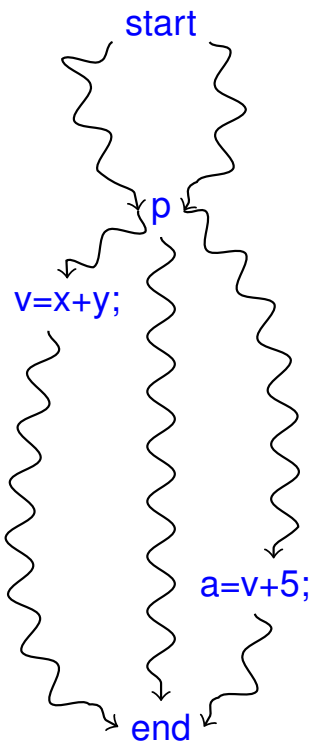
Aktív változó

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

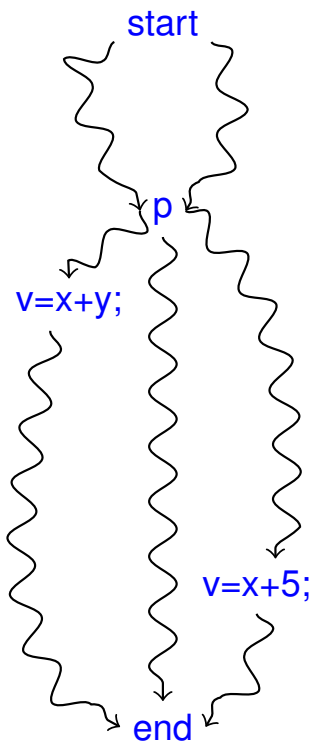


Aktív változó globális analízise

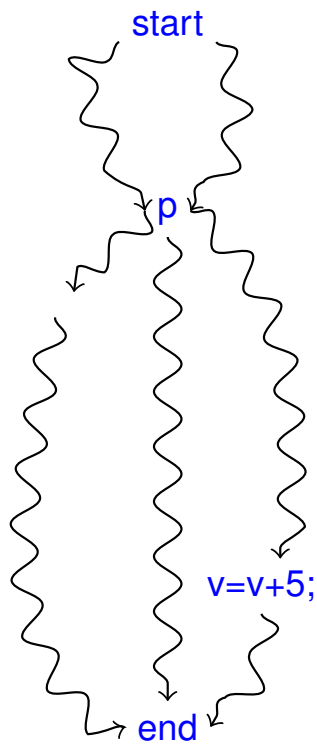
v változó aktív
a p pontban



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;

Aktív változók előállításának a formalizálása

$$
\begin{aligned}
In_n &= (Out_n - Kill_n) \cup Gen_n \\
Out_n &= \begin{cases} BI & n \text{ az utolsó blokk végén .} \\ \cup_{x \in succ(n)} In_x & \text{máskülönben.} \end{cases}
\end{aligned}
$$

- A kimenet van összehuzalozva az n után következő bemenetek uniójával.
- $\cup$-ból következik, hogy az inicializálás az üres halmazzal történik, kivéve a kezdő állapotot.
- Out_n alapján számítjuk az In_n-t, tehát a számítás visszafele halad.

Reaching Definition analízis

Az analízis megadja, hogy a program egy adott pontján az x változó hol kapott értéket. Elágazások és ciklusok miatt több ponton is definiálva[1] lehet.

$$
\begin{aligned}
RD_{entry}(p) &= \begin{cases} RD_{init} & \text{kezdetben .} \\ \cup_{p'} \in pred(p)\ RD_{exit}(p') & \text{máskülönben.} \end{cases} \\
RD_{exit}(p) &= (RD_{entry}(p) \setminus Kill_{RD}(p)) \cup Gen_{RD}(p)
\end{aligned}
$$

[1] Itt a definíciót bővebb értelemben használjuk. Minden pont, ahol egy változó értéket kap definíciós pontnak hívjuk.

Reaching Definition jelölések

Ha az x változót a p pontban definiálva van, akkor ezt a tényt a (x, p) párral fejezzük ki. Az $RD_{entry}(p)$ és az $RD_{exit}(p)$ ezeknek a pároknak a halmazát tartalmazza.

Inicializálás

$$RD_{init} = \{(x, ?) | x \in \text{programváltozó.}\}$$

A kérdőjel azt jelenti, hogy még nem lett inicializálva a változó.

Példa Reaching Definition-ra

```
[x := 5]^1;
[y := 1]^2;
while [x > 1]^3 do
  [y := x * y]^4;
  [x := x - 1]^5
od
```

$$
\begin{aligned}
RD_{entry}(1) &= \{(x,?),(y,?)\} \\
RD_{entry}(2) &= RD_{exit}(1) \\
RD_{entry}(3) &= RD_{exit}(2) \cup RD_{exit}(5) \\
RD_{entry}(4) &= RD_{exit}(3) \\
RD_{entry}(5) &= RD_{exit}(4)
\end{aligned}
$$

$$
\begin{aligned}
RD_{exit}(1) &= (RD_{entry}(1) \setminus \{(x,?),(x,1),(x,5)\}) \cup (x,1) \\
RD_{exit}(2) &= (RD_{entry}(2) \setminus \{(y,?),(y,2),(y,4)\}) \cup (y,2) \\
RD_{exit}(3) &= RD_{entry}(3) \\
RD_{exit}(4) &= (RD_{entry}(4) \setminus \{(y,?),(y,2),(y,4)\}) \cup (y,4) \\
RD_{exit}(5) &= (RD_{entry}(5) \setminus \{(x,?),(x,1),(x,5)\}) \cup (x,1)
\end{aligned}
$$

RD végeredmény

	RD_{entry}	RD_{exit}
1	{(x,?),(y,?)}	{(x,1),(y,?)}
2	{(x,1),(y,?)}	{(x,1),(y,2)}
3	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,2),(y,4),(x,5)}
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(y,5)}

Konstans kiértékelés fordítási időben I

$$RD \vdash [x := a]^{\ell} \rhd [x := a[y \mapsto n]]^{\ell}$$

$$\text{ha} \begin{cases} y \in FV(a) \ \wedge \ (y, ?) \notin RD_{entry}(\ell) \wedge \\ \forall (y', \ell') \in \ RD_{entry}(\ell) \wedge \\ y' = y \Rightarrow [\ldots]^{\ell'} = [y := n]^{\ell'} \end{cases}$$

$$RD \vdash [x := a]^{\ell} \rhd [x := n]^{\ell}$$

$$\text{ha} \begin{cases} FV(a) = \emptyset \ \wedge \ a \text{ nem konstans} \wedge \\ a \text{ kifejezés értéke } n \end{cases}$$

Megj: $FV(a)$ megadja, hogy az a aritmetikai kifejezés tartalmaz-e szabad változót.

Konstans kiértékelés fordítási időben II

$$\frac{RD \vdash S_1 \rhd S_1'}{RD \vdash S_1 ; S_2 \rhd S_1' ; S_2} \qquad \frac{RD \vdash S_2 \rhd S_2'}{RD \vdash S_1 ; S_2 \rhd S_1 ; S_2'}$$

$$\frac{RD \vdash S_1 \rhd S_1'}{RD \vdash \textbf{if}[b^\ell]\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \rhd \textbf{if}[b^\ell]\ \textbf{then}\ S_1'\ \textbf{else}\ S_2}$$

$$\frac{RD \vdash S_2 \rhd S_2'}{RD \vdash \textbf{if}[b^\ell]\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \rhd \textbf{if}[b^\ell]\ \textbf{then}\ S_1\ \textbf{else}\ S_2'}$$

$$\frac{RD \vdash S \rhd S'}{RD \vdash \textbf{while}[b^\ell]\ \textbf{do}\ S \rhd \textbf{while}[b^\ell]\ \textbf{do}\ S'}$$

Konstans kiértékelés példa I.

Program:

$$[x := 10]^1; \quad [y := x + 10]^2; \quad [z := y + 10]^3;$$

RD analízis eredménye:

$$\begin{aligned}
RD_{entry}(1) &= \{(x, ?), (y, ?), (z, ?)\} \\
RD_{exit}(1) &= \{(x, 1), (y, ?), (z, ?)\} \\
RD_{entry}(2) &= \{(x, 1), (y, ?), (z, ?)\} \\
RD_{exit}(2) &= \{(x, 1), (y, 2), (z, ?)\} \\
RD_{entry}(3) &= \{(x, 1), (y, 2), (z, ?)\} \\
RD_{exit}(2) &= \{(x, 1), (y, 2), (z, 3)\}
\end{aligned}$$

Konstans kiértékelés példa II.

$$
\begin{aligned}
RD \vdash\ & [x := 10]^1;\ [y := x + 10]^2;\ [z := y + 10]^3; \\
\rhd\ & [x := 10]^1;\ [y := 10 + 10]^2;\ [z := y + 10]^3; \\
\rhd\ & [x := 10]^1;\ [y := 20]^2;\ [z := y + 10]^3; \\
\rhd\ & [x := 10]^1;\ [y := 20]^2;\ [z := 20 + 10]^3; \\
\rhd\ & [x := 10]^1;\ [y := 20]^2;\ [z := 30]^3;
\end{aligned}
$$

Konstans kiértékelés példa II.

Visual Studio C eredménye

```
1  int funcion()  {
2     int x,y,z;
3     x=1;
4     y=100*x;
5     z=100*y-y;
6     while( x < 4 ) {
7        z = z +y;
8        y = 2*x;
9        x++;
10    }
11    return z;
12 }
```

Lefordított kód:

```
_funcion:
   00000000:      mov        eax,2710h
   00000005:      ret
```

Adatfolyam analízis algoritmusok

- Az adatfolyam algoritmus előre halad, ha
  1. a kimenetet *out*() számoljuk ki az alapblokkba bemenő *in*() értékek alapján
  2. a bemenet az előző alapblokkok kimenetének a kombinációja
- Az adatfolyam algoritmus visszafele halad, ha
  1. a bemenetet *in*() számoljuk ki az alapblokk kimenetelén lévő *out*() értékek alapján
  2. a kimenet a következő alapblokkok bemenetének a kombinációja

A következő táblázat a lehetséges adatfolyam algoritmusokat ábrázolja:

	Előre haladó	Visszafele haladó
Legalább egy úton	$Out(B_i) = Gen(B_i) \bigcup (In(B_i) - Kill(B_i))$ $In(B_i) = \bigcup_{p \in Pred(B_i)} Out(p)$	$In(B_i) = Gen(B_i) \bigcup (Out(B_i) - Kill(B_i))$ $Out(B_i) = \bigcup_{s \in Succ(B_i)} In(s)$
Minden egy úton	$Out(B_i) = Gen(B_i) \bigcup (In(B_i) - Kill(B_i))$ $In(B_i) = \bigcap_{p \in Pred(B_i)} Out(p)$	$In(B_i) = Gen(B_i) \bigcup (Out(B_i) - Kill(B_i))$ $Out(B_i) = \bigcap_{s \in Succ(B_i)} In(s)$

Változó definíció-használat lánc

Definíció: definíció-használat láncolat (du-chain)

megadja, hogy a program egy pontján definiált változó, regiszter vagy feltétel kód értékére a programban található utasítások közül kik hivatkoznak.

Definíció: használat-definíció láncolat (ud-chain)

megadja, hogy a program egy pontján hivatkozott (használt) változó, regiszter vagy feltétel kód a programban hol kaphattak értéket.

u-d chain analízis

Az u-d chain analízis megadja, hogy a program ℓ címén az *elérhető definíciós pontok analízis* (Reaching definition analysis) szerint az ℓ' címke alatt definiált x változó értékét használjuk.

$$UD_{entry}(x, \ell) = \begin{cases} \ell' \mid (x, \ell') \in RD_{entry}(\ell) & \text{ha } x \in gen_{LV}(B^{\ell}) \\ \emptyset & \text{máskülönben.} \end{cases}$$

Példa u-d és d-u láncra

```
[x := 0]1;
[x := 3]2;
if [z = x]3 then
  [z := 0]4;
else
  [z := x]5
[y := x]6;
[x := y+z]7;
```

$ud(\chi, \ell)$	x	y	z
1	$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$	$\emptyset$
3	$\{2\}$	$\emptyset$	$\{?\}$
4	$\emptyset$	$\emptyset$	$\emptyset$
5	$\{2\}$	$\emptyset$	$\emptyset$
6	$\{2\}$	$\emptyset$	$\emptyset$
7	$\emptyset$	$\{6\}$	$\{4,5\}$

$du(\chi, \ell)$	x	y	z
1	$\emptyset$	$\emptyset$	$\emptyset$
2	$\{3,5,6\}$	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$	$\{7\}$
5	$\emptyset$	$\emptyset$	$\{7\}$
6	$\emptyset$	$\{7\}$	$\emptyset$
7	$\emptyset$	$\emptyset$	$\emptyset$
?	$\emptyset$	$\emptyset$	$\{3\}$