Analiza przepływu danych

- Analiza oferuje globalną informację o tym, jak program manipuluje danymi
- Różne rodzaje optymalizacji potrzebują różnej informacji (definicje osiągające, żywotność zmiennych)
- Analiza może być wykonywana zarówno na kodzie źródłowym jak i kodzie pośrednim
- Analiza lokalna (bloki bazowe)

a := b + c

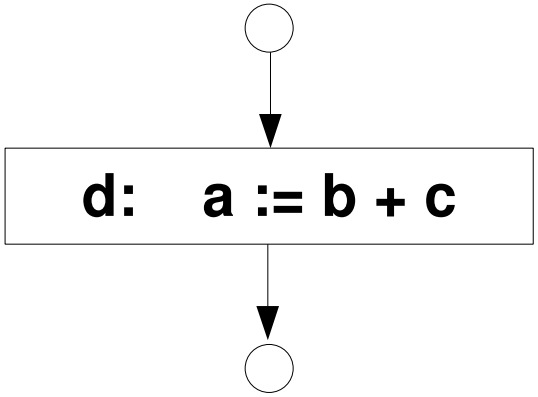
  - używa zmiennych **b** i **c**
  - generuje nowe podstawienie dla **a** (nowa wartość)
  - usuwa starą wartość **a** (zabija)

- Analiza globalna (między blokami)

  Rozwiązujemy układy równań opisujących zależności pomiędzy danymi wejściowymi, wyjściowymi oraz modyfikowanymi wewnątrz bloku

Definicje osiągające (reaching definitions)

- Punkty i ścieżki w grafie przepływu
- Definicja zmiennej x - instrukcja, która wiąże zmienną z wartością
  - pewna - podstawienie, instrukcja wejścia
  - niepewna - wywołanie procedury (parametr, widoczność)
- Definicja ***d*** osiąga punkt ***p*** w grafie, jeżeli istnieje ścieżka z punktu za ***d*** do ***p*** taka, że ***d*** nie jest zabijane
- Zbiory IN, OUT, GEN, KILL - definicje wyjściowe są zależne od definicji wejściowych



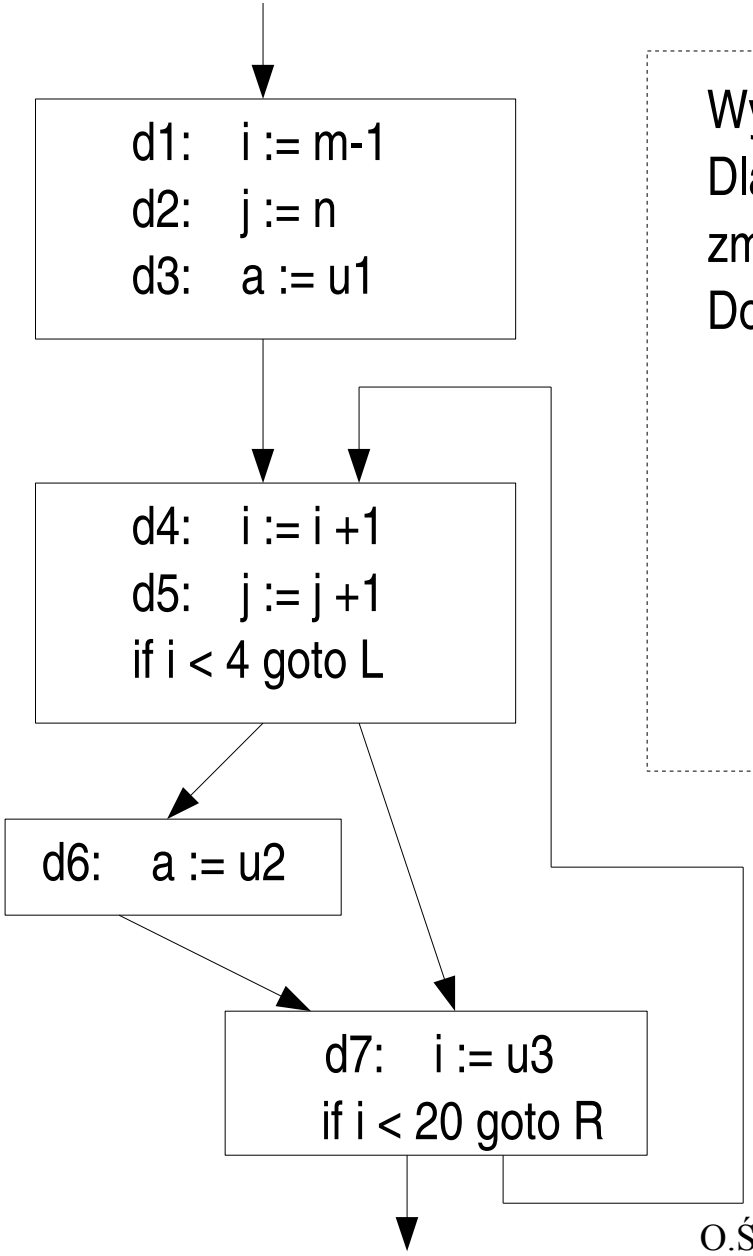
GEN(X) = { d }

KILL(X) = wszystkie def. a - {d}

OUT(X) = GEN(X) + (IN(X) - KILL(X))

złożenie, instrukcja warunkowa, pętla

Definicje osiągające dla bloków



Wyznacz zbiory gen i kill dla wszystkich bloków bazowych w grafie
Dla każdego bloku B inicjalizuj out(B) = gen(B) oraz in(B) jest puste
zmiana = TRUE
Dopóki zmiana wykonuj:
    zmiana = FALSE
    dla każdego B:
        in(B) = suma wszystkich zbiorów out(X), X - poprzednik B
        t = out(B)
        out(B) = gen(B) + (in(B) - kill(B))
        jeżeli t jest różne od out(B), to zmiana = TRUE

GEN(B1) = {1,2,3}	KILL(B1) = {4,5,6,7}
GEN(B2) = {4,5}	KILL(B2) = {1,2,7}
GEN(B3) = {6}	KILL(B3) = {3}
GEN(B4) = {7}	KILL(B4) = {1,4}

Żywotność zmiennych

- Zmienna jest żywa w punkcie ***p***, jeżeli jej wartość jest używana na ścieżce idącej z ***p***
- Zbiory - tym razem dane wejściowe zależne są od wyjściowych
  - USE - zbiór zmiennych używanych w bloku
  - DEF - zbiór zmiennych definiowanych w bloku
  - IN - zbiór zmiennych żywych przed wejściem do bloku
  - OUT - zbiór zmiennych żywych po wyjściu z bloku

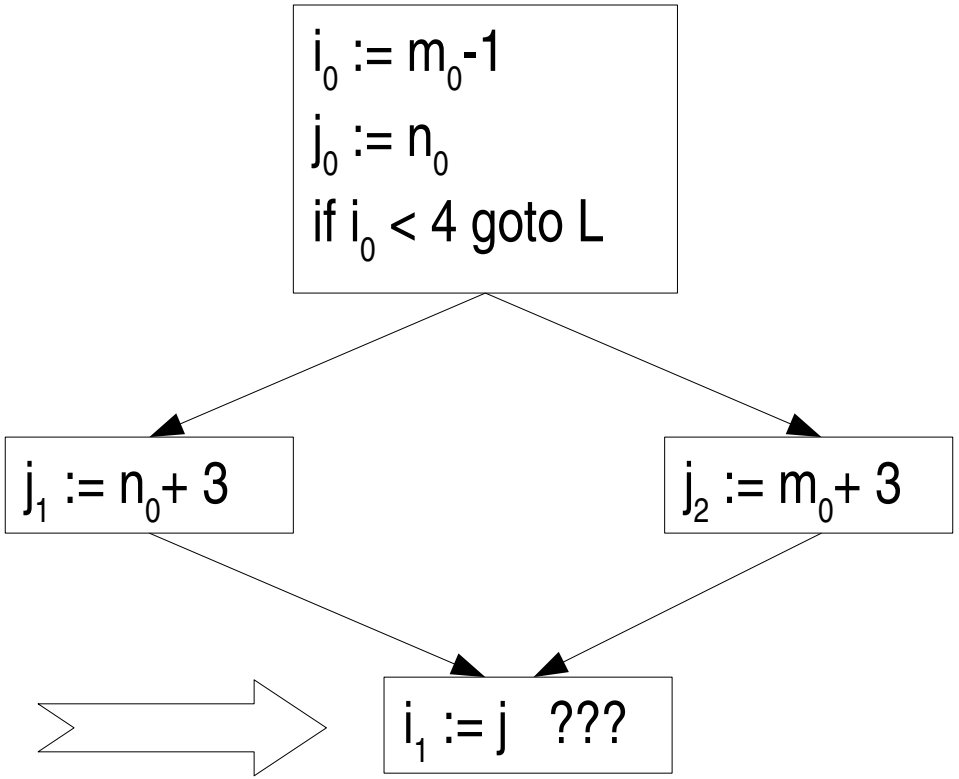
```
Dla każdego bloku B inicjalizuj IN(B) =  puste
zmiana = TRUE
Dopóki zmiana wykonuj:
        zmiana = FALSE
        dla każdego B:
            OUT(B) = suma wszystkich zbiorów IN(X), X - następnik B
            t = IN(B)
            IN(B) = USE(B) + ( OUT(B) - DEF(B) )
            jeżeli t jest różne od IN(B), to zmiana = TRUE
```

Static Single Assignment

Forma zapisu kodu pośredniego, w której **każda zmienna w programie ma tylko jedną definicję**

$i_0 := m_0 - 1$
$j_0 := n_0$
$a_0 := u_0$
$i_1 := i_0 + 1$
$j_1 := j_0 + 1$
if $i_1 < 4$ goto L

kolejna wersja zmiennej ***i***



Która zmienna ***j*** ?
Określamy $\Phi(j_1, j_2)$

Przekształcanie do formy SSA

[1991] R.Cytron, J.Ferrante, B.K.Rosen, M.N.Wegman, F.K.Zadeck

1. Wyznaczanie granic dominacji

- Succ(X) - zbiór bezpośrednich następników X w grafie przepływu sterowania
- Idom(Y) - bezpośredni dominant nad Y
- Przeglądamy drzewo dominatorów w porządku: dzieci - rodzic

Dla każdego kolejnego X z drzewa dominatorów wykonaj:

DF(X) - pusty

Dla każdego Y ze zbioru *Succ*(X) wykonaj:

Jeżeli X nie jest *Idom*(Y), to do zbioru DF(X) dodaj {Y}

Dla każdego Z będącego dzieckiem X wykonaj:

Dla każdego Y ze zbioru DF(Z) wykonaj:

Jeżeli X nie jest *Idom*(Y), to do zbioru DF(X) dodaj {Y}

Przekształcanie do formy SSA c.d.

2. Wstawianie Φ – *funkcji*

- W - zbiór aktualnie przetwarzanych węzłów
- HA(X), WK(X) - flagi dla poszczególnych węzłów
- Początkowo: zbiór W jest pusty, HA(X)=0,WK(X)=0 dla każdego X

ic=0;
Dla każdej zmiennej V wykonaj:
ic ++;
Dla każdego X, w którym zdefiniowano przypisanie dla V wykonaj:
WK(X) = ic; Do zbioru W dodaj {X};
Dopóki W nie jest pusty wykonaj: Weź X ze zbioru W
Dla każdego Y należącego do DF(X) wykonaj:
Jeżeli HA(Y) < ic to:
wpisz V := Φ(V,...,V) do bloku Y
HA(Y) = ic;
Jeżeli WK(Y) < ic to:
WK(Y)=ic; Do zbioru W dodaj {Y}

Przekształcanie do formy SSA c.d.

3. Zmiana nazw zmiennych

- C(V) - licznik dla zmiennej V S(V) - stos dla zmiennej V
- Początkowo: wszystkie liczniki ustawione na 0 i wszystkie stosy puste

procedura RENAME(X):

Dla każdego przypisania A w X wykonaj:

Dla każdej zmiennej V używanej po prawej stronie A zamień jej nazwę na V_i, gdzie i=Top(S(V))

Dla każdej zmiennej V po lewej stronie A wykonaj:

i = C(V); zamień V na V_i ; połóż i na stos S(V) ; C(V) = i + 1

Dla każdego Y ze zbioru Succ(X) wykonaj:

j = numer określający, którym poprzednikiem Y jest X

Dla każdej funkcji Φ w Y:

Zamień j argument w tej funkcji na V_i, gdzie i = Top(S(V))

Dla każdego Y - dziecka X wykonaj:

RENAME(X)

Dla każdego przypisania A w X

Dla każdej zmiennej z lewej(oryginalnej) strony A: Pop(S(V))

Przekształcanie do formy SSA c.d.

Przykład:

```
        x := 0
        y := 0
        z := read
        if z < 4 goto L7
        x := 4
        y := x
        goto L8
L7:     x := 5
        y := 6
L8:     z := x - 1
        y := y + 1
```