Compilatoare

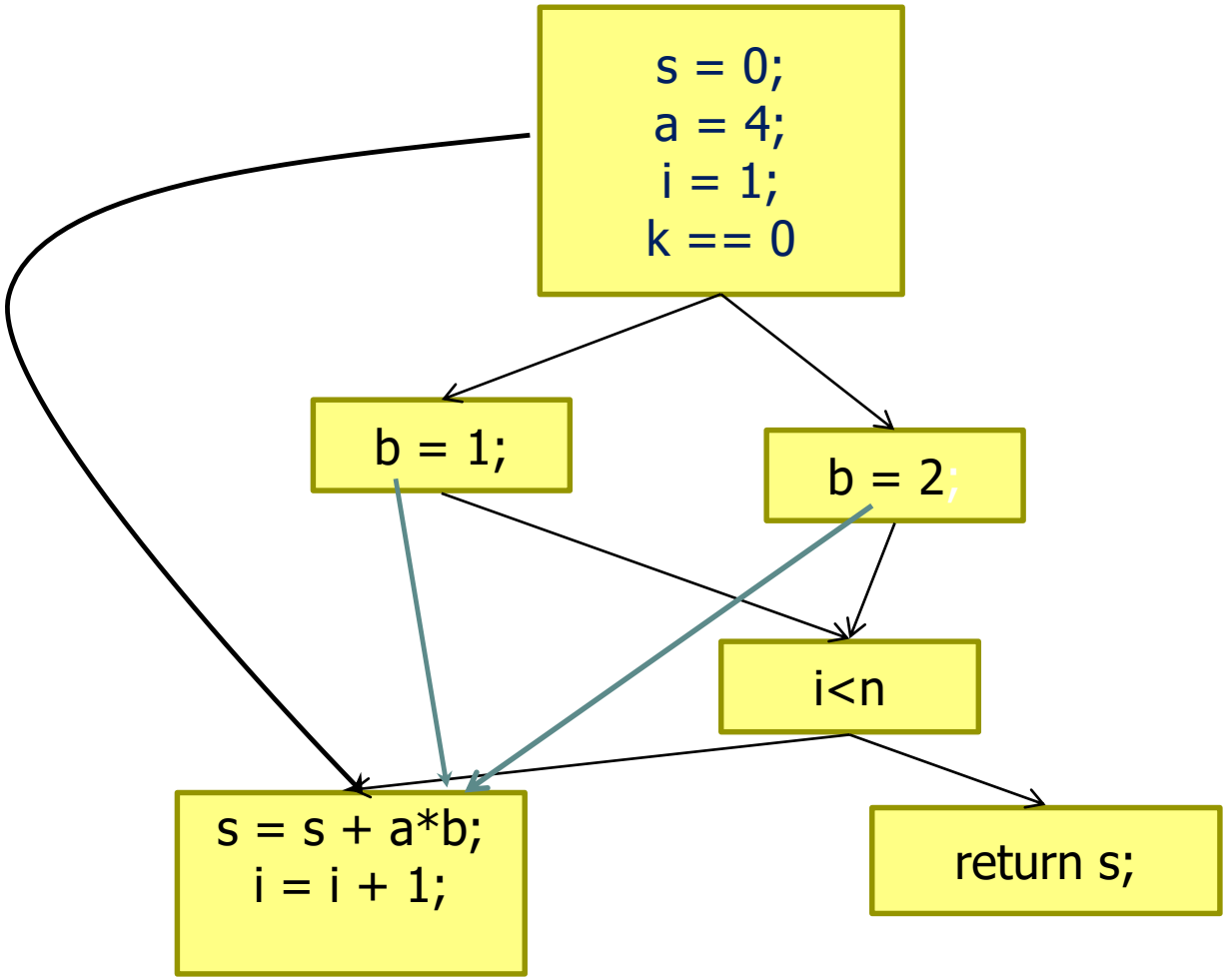
Analiza fluxului de date
Forma SSA

CPL
IR crunch

Review

- **Basic Block** – un set de instructiuni care se executa intotdeauna consecutiv
- **CFG** – Control Flow Graph
  - Fiecare muchie corespunde unui posibil flux de control al programului
- **Optimizare locala** – optimizare la nivelul unui singur basic block
- **Analiza globala** – analiza la nivelul CFG-ului

Reaching Definitions & Constant Propagation – exemplu



Is ***a*** constant in
s = s + a*b ?

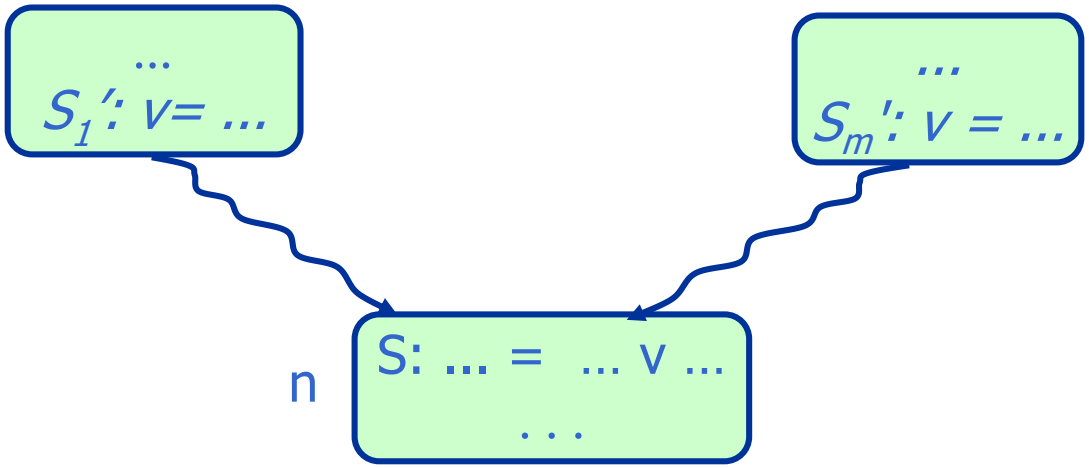
YES (4)

Is ***b*** constant in
s = s + a*b ?

NO

UD Chain

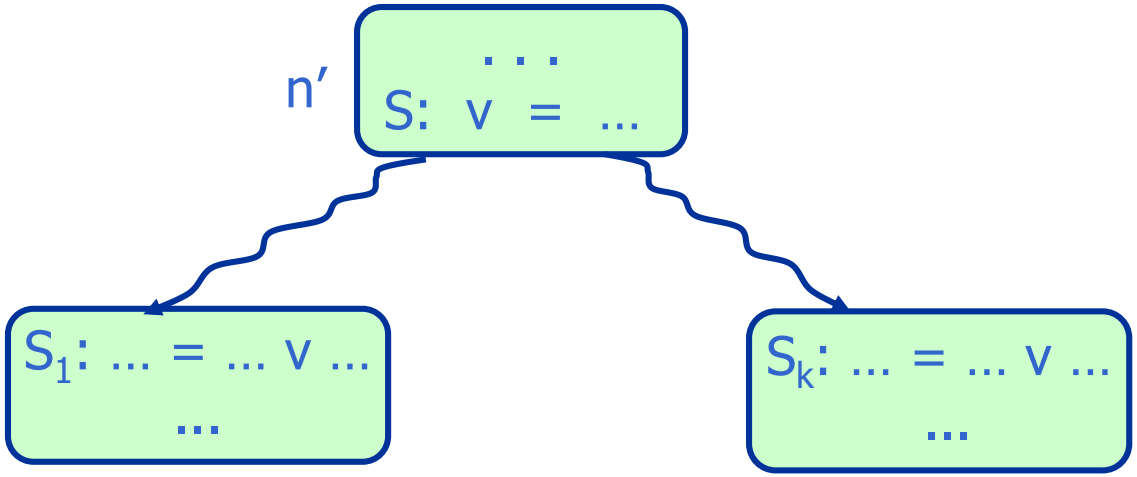
Un lant use-def(UD chain) e o lista a tuturor definirilor care pot ajunge la o folosire a unei variabile.



UD chain: UD(n, v) = (S_1', ..., S_m').

DU Chain

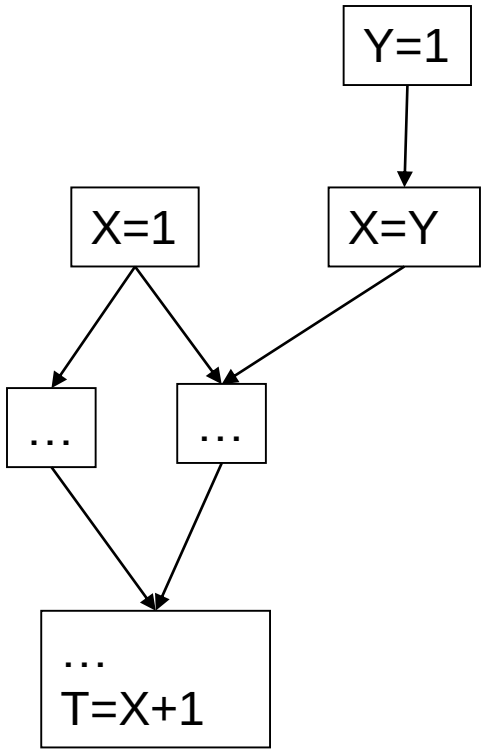
Un lant def-use (DU chain) e o lista a tuturor folosirilor la care se poate ajunge de la o anumita definire a unei variabile.

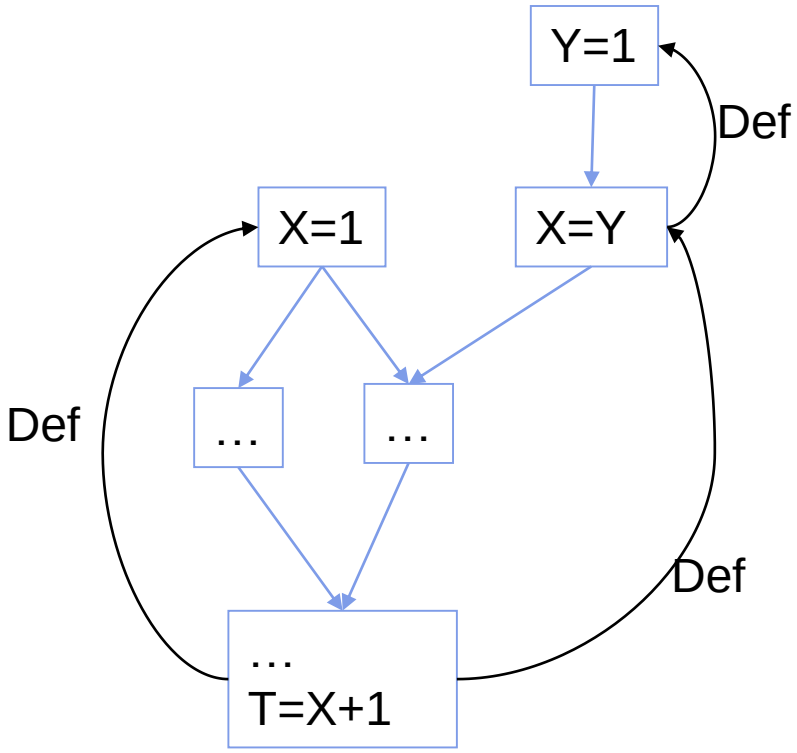


DU chain: DU(n', v) = (S_1, …, S_k).

Folosirea Def-Use Chains

- Propagarea constantelor – poate fi iterata pe CFG sau pe UD-chain (“sparse constant propagation”)





Aplicaţiile analizei def/use

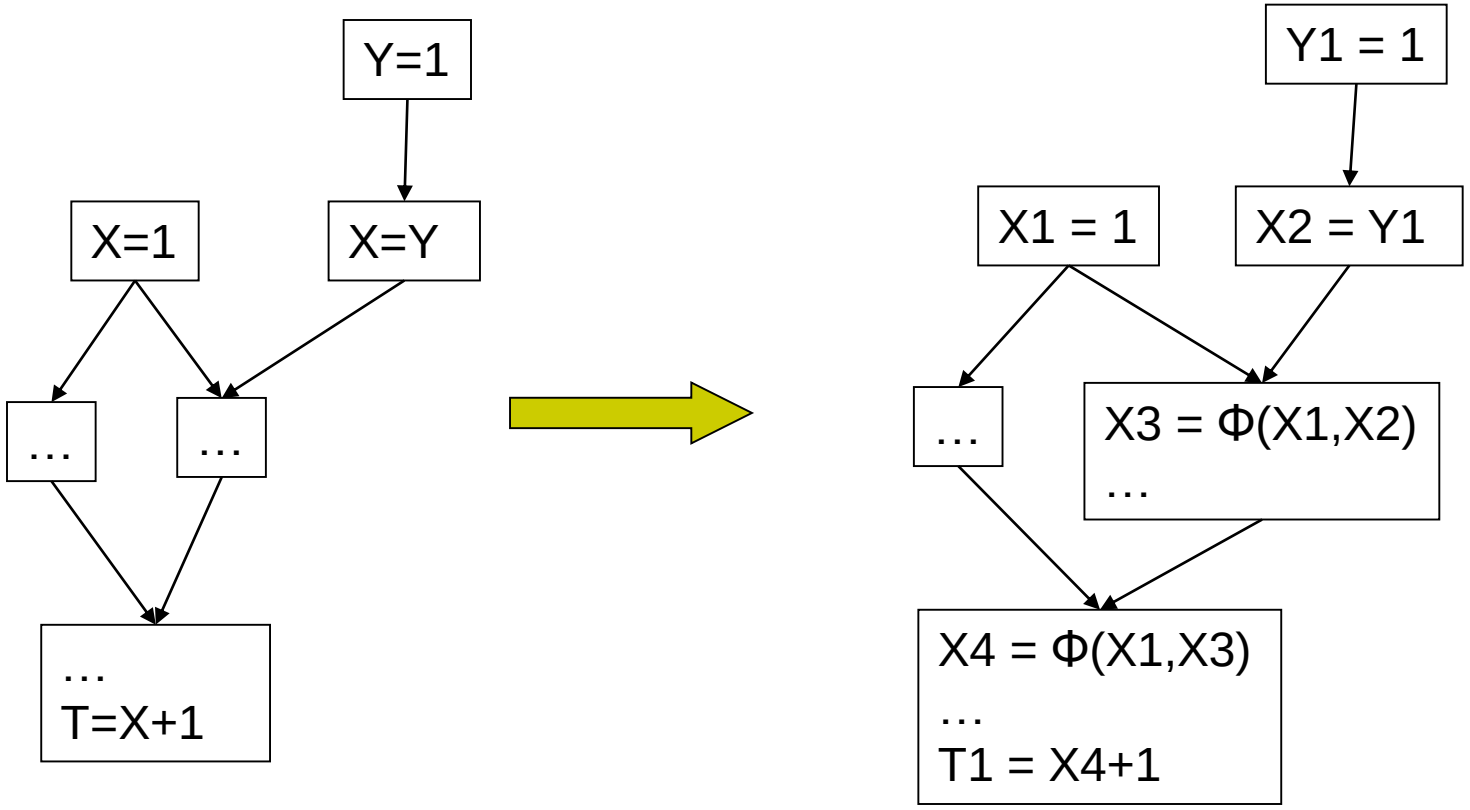
DU-chain - structura de date principala pentru optimizari

- Propagarea constantelor, copierilor
- Analiza variabilelor in viata si eliminarea codului inutil (“dead” code)
- Eliminarea calculelor redundante
- Detectia variabilelor de inductie (index de buclă)
- Descoperirea dependentelor, planificarea si paralelizarea instructiunilor.
- Analiza de alias
- …si multe alte analize pentru diverse transformari

… dar consuma memorie: M – definiri, N folosiri => O(M x N)
si trebuie recalculata dupa fiecare transformare

Forma SSA

- Reprezentare intermediara ce permite optimizari mai rapide decat folosind DU-chain:



CPL
IR crunch

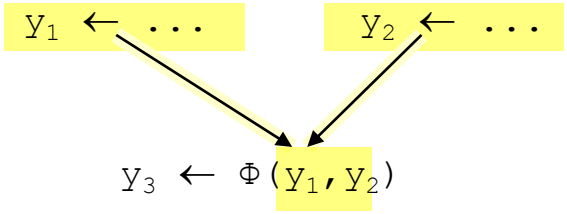
SSA

- SSA: Un program e in forma SSA daca si numai daca
  - Fiecare variabila e definita (static) exact o singura data, si
  - Fiecare folosire a unei variabile e dominata de definirea acelei variabile

Aceasta unica definire statica poate fi intr-o bucla si deci executata de multe ori. Astfel, chiar si intr-un program SSA o variabila poate fi definita in mod dinamic de mai multe ori.

Functia Φ

- Functia Φ e o copiere speciala care selecteaza unul din parametri.
- Alegerea parametrului e guvernata de arcul din CFG pe care s-a ajuns la blocul curent



- Procesoarele reale nu implementeaza functii Φ direct in hardware (este posibil?)

CPL
IR crunch

SSA: Motivatia

- Ofera o baza uniforma la nivel de IR pentru a rezolva o gama larga de probleme de flux de date
- Codifica informatie de flux de date + control
- Forma SSA poate fi construita si intretinuta eficient
- Multi algoritmi de analiza de date/optimizare pe forma SSA sunt mai eficienti (au complexitate mai scazuta) decat varianta pe CFG.

Forma SSA – exemplu

Forma SSA

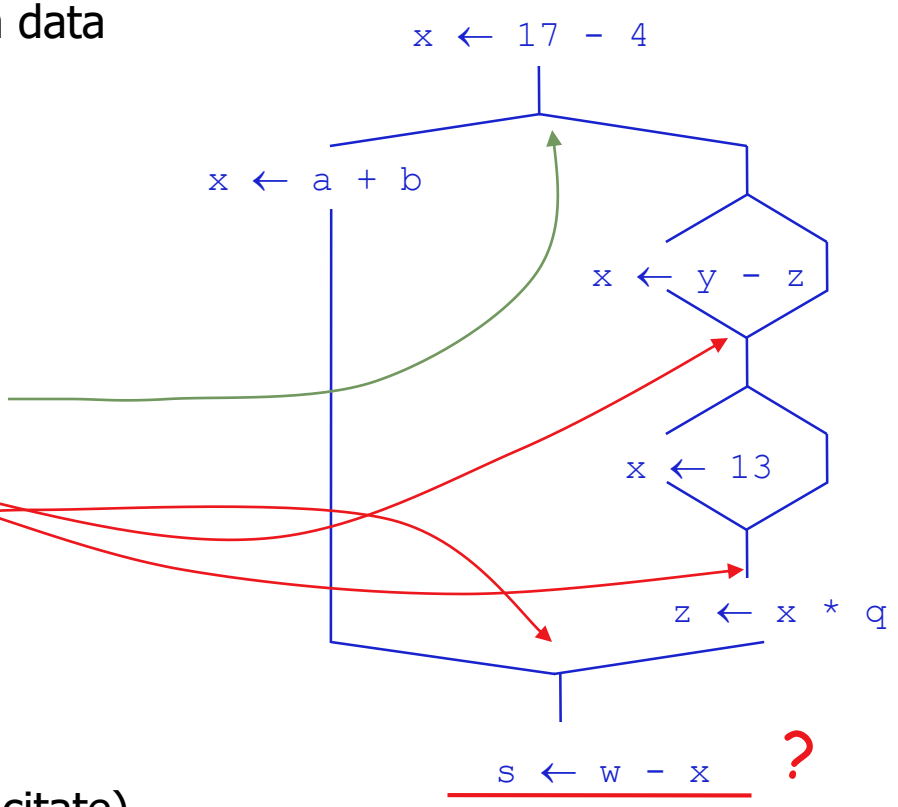
- Fiecare nume e definit exact o singura data
- Fiecare folosire refera un singur nume

Ce e mai greu

- Codul liniar e usor de transformat
- 'Splits' in CFG - usor de transformat
- 'Joins' in CFG - sunt mai problematice

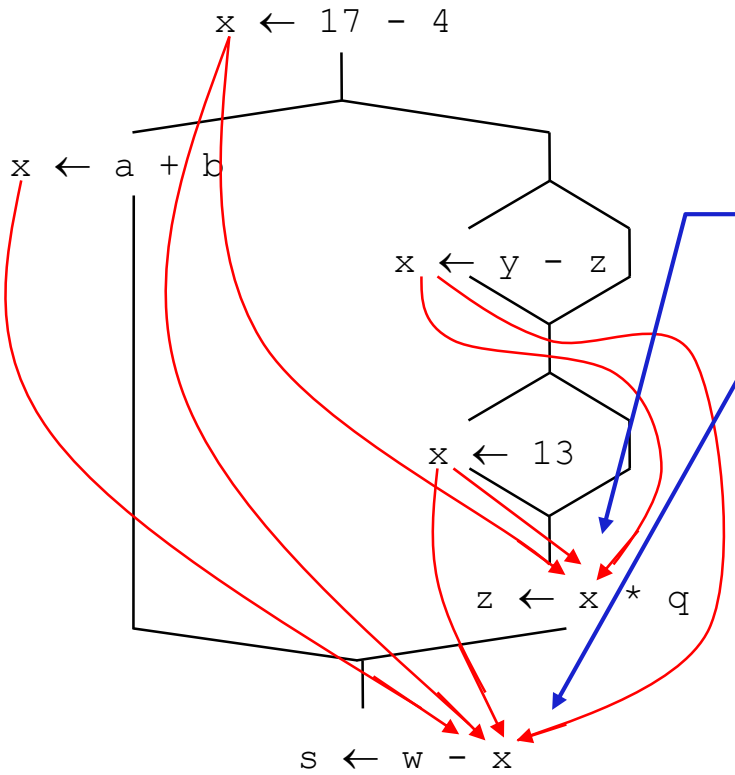
Construirea formei SSA

- Insereaza functii Ø in 'birth points'
- Redenumeste toate variabilele (pt. unicitate)



Birth Points

- Sa luam exemplul urmator:



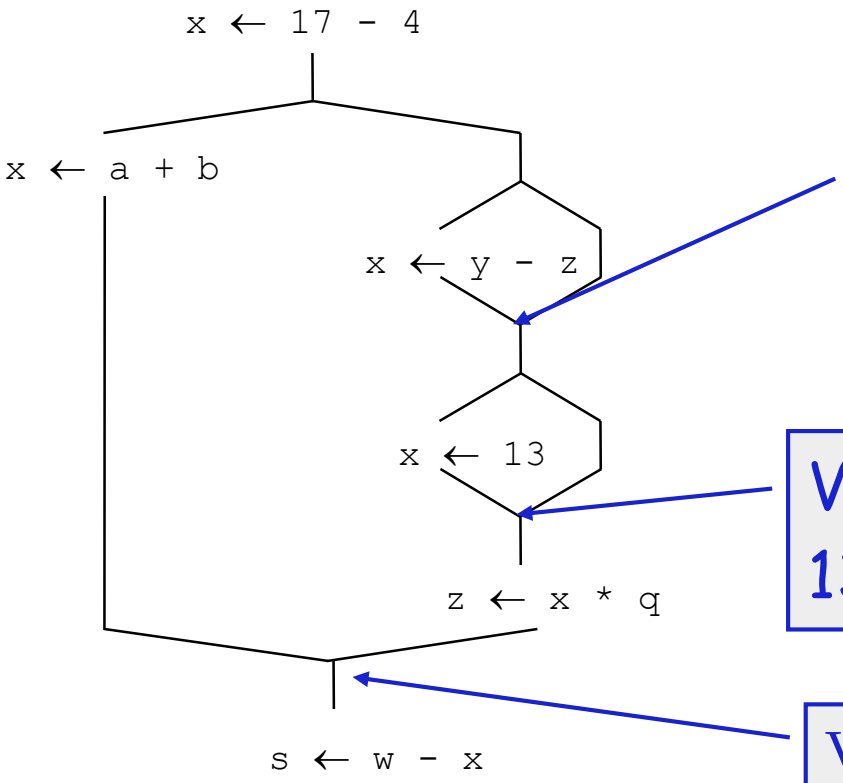
Variabila x apare peste tot si ia mai multe valori

- Aici, x poate fi 13, y-z, sau 17-4
- Aici, ar putea fi si a+b

Daca fiecare valoare are numele ei…

- Avem nevoie de o modalitate de a "uni" valorile distincte
- Valorile 'unite' se "nasc" la punctele de "join" din CFG

Birth Points

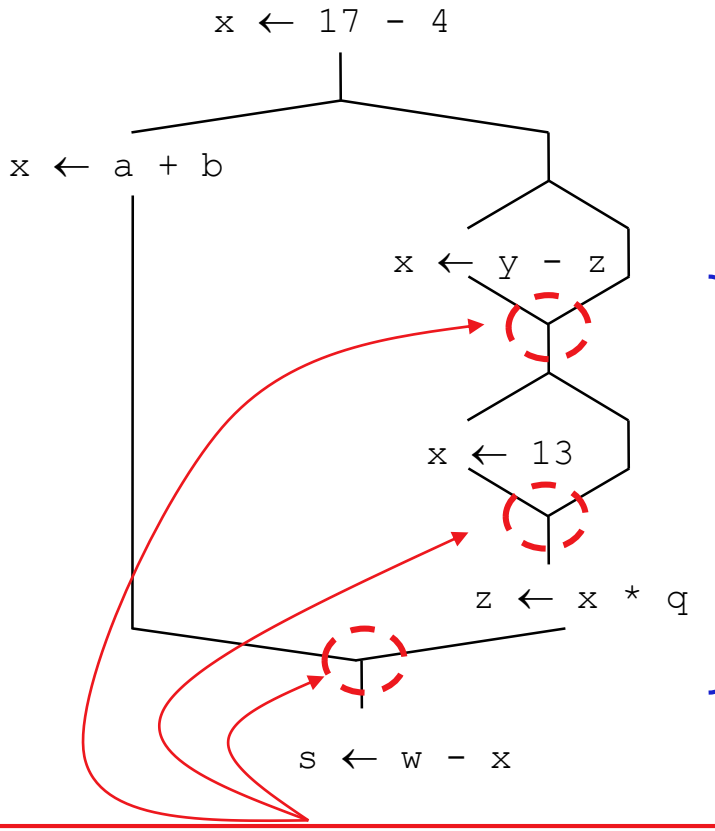


Valoare noua pt. x aici
17 - 4 sau y - z

Valoare noua pt. x aici
13 sau (17 - 4 sau y - z)

Valoare noua pt. x aici
a+b sau ((13 sau (17-4 sau y-z))

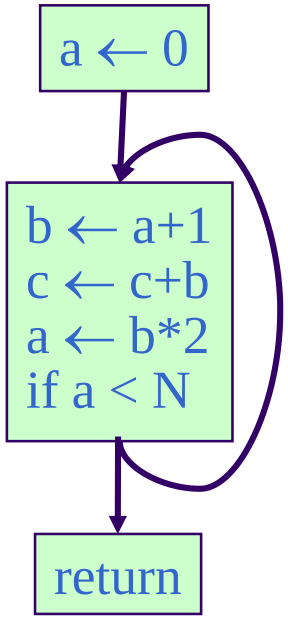
Birth Points

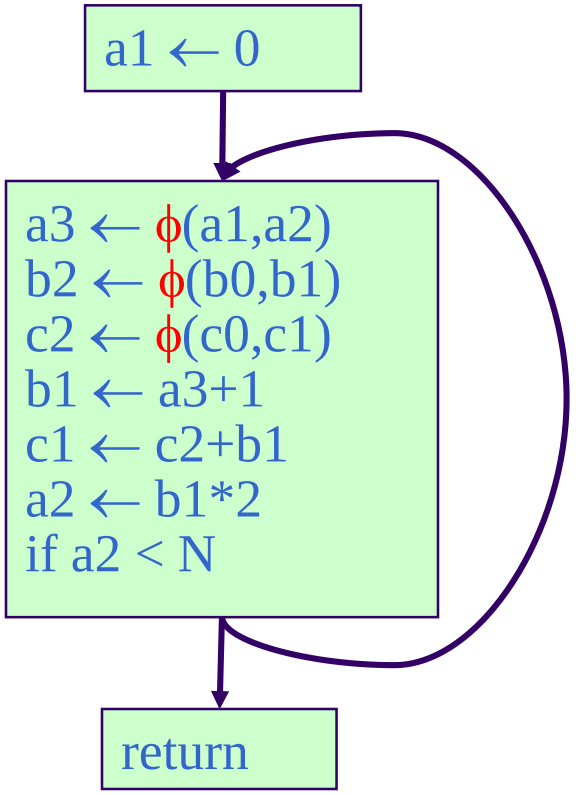


- Toate 'birth points' sunt join-uri pe CFG
- Nu toate join-urile sunt birth points
- Birth points sunt specifice fiecarei variabile …

birth points pentru valorile lui x

Un exemplu cu bucla





ϕ(b0,b1) nu e necesar, caci b1 nu e folosit. Dar faza care genereaza functiile ϕ nu stie asta. Functiile nenecesare sunt eliminate de dead code elimination.

Nota: doar a, c sunt folosite in bucla inainte de a fi de a fi redefinite.

Criterii pt inserarea functiilor ϕ

Am putea insera o functie ϕ pt fiecare variabila la fiecare *join* (punct cu mai mult de un predecesor). Dar ar fi un numar exagerat de mare.

Adaugam functii ϕ la 'birth point' – dar cum stim care sunt?

Intuitiv, adaugam o functie ϕ daca 2 definiri ajung la un punct z pe cai diferite

Criteriul convergentei cailor

Insereaza o functie ϕ pt. o variabila *a* la nodul *z* daca urmatoarele conditii sunt adevarate:

1. Exista un bloc *x* care defineste *a*
2. Exista un bloc $y \neq x$ care defineste *a*
3. Exista o cale nevida P_{xz} de la *x* la *z*
4. Exista o cale nevida P_{yz} de la *y* la *z*
5. Caile P_{xz} si P_{yz} nu au noduri in comun in afara de z
6. Nodul *z* nu apare si in P_{xz} si in P_{yz} inainte de sfarsit, (dar poate aparea in una dintre cai).

Blocul de start (Entry) contine o definire implicita a tuturor variabilelor.

Criteriul convergentei cailor

Functia ϕ e ea insasi o definitie.
De aceea, criteriul de mai sus trebuie iterat:

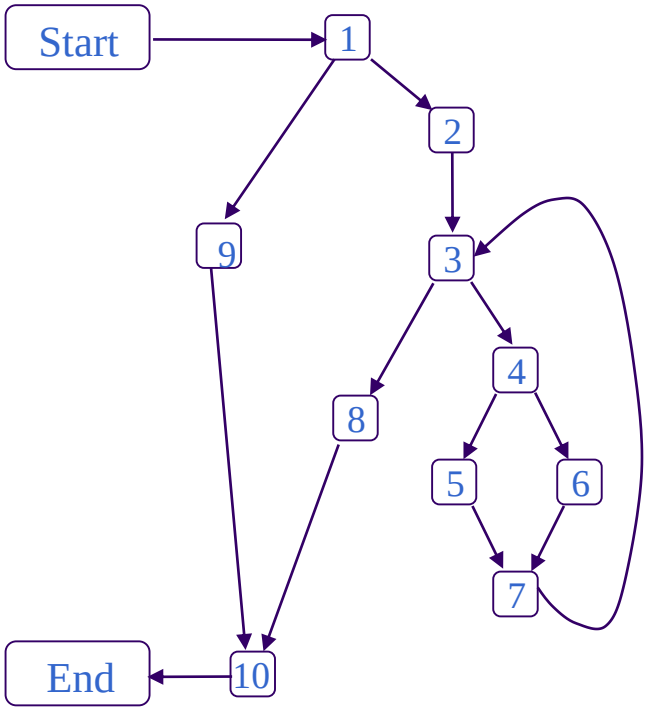
Cat timp exista noduri *x*, *y*, *z* indeplinind conditiile 1-5
si *z* nu contine o functie ϕ pentru *a*
adauga $a \leftarrow \phi(a, a, \ldots, a)$ la nodul *z*

Acest algoritm e extrem de costisitor, deoarece necesita examinarea tuturor perechilor (x,y,z) si a tuturor cailor x->y

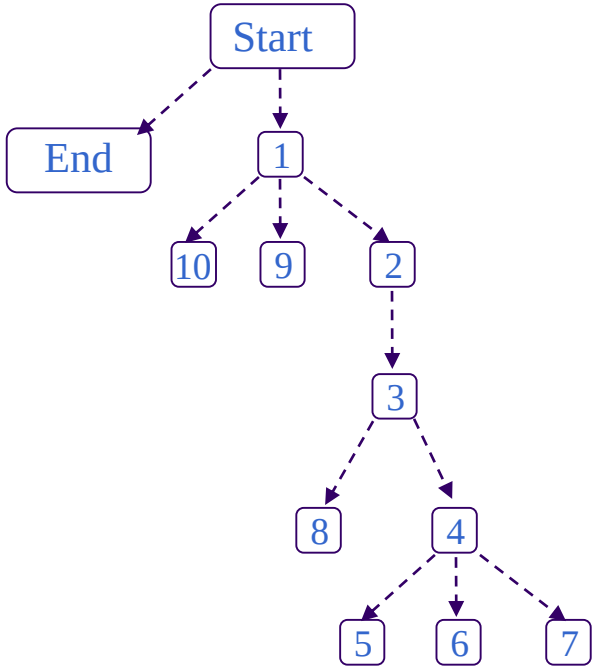
Algoritm mai eficient

- Foloseste notiunea de “Frontiera de dominare”
- Frontiera de dominare a blocului B: DF(B)
  - Setul de bb nedominate de B, care au cel putin un predecesor dominat de B.
  - Definitia se poate extinde la un set de noduri.
- Frontiera de dominare iterata a lui B, IDF(B)
  - Cel mai mic set de noduri care contine DF(B) si e punct fix relativ la aplicarea functiei DF.

Conceptul de DF



(a) CFG



(b) Arbore de dominare

Nota: Consideram o legatura implicita intre ‘start’ si ‘end’

DF(3) = {3, 10}

Definitia formala

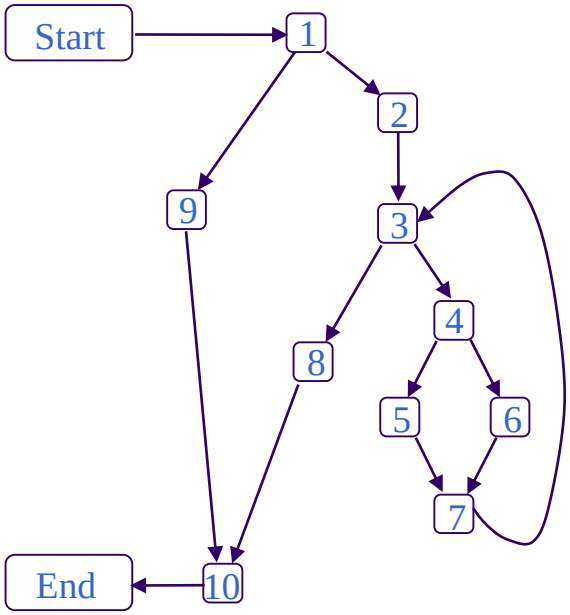
Frontiera de dominare DF (x) a unui nod x e setul de noduri y care au un predecesor z, cu proprietatea ca x dom z dar x ! sdom y.

Pentru un set de noduri S:

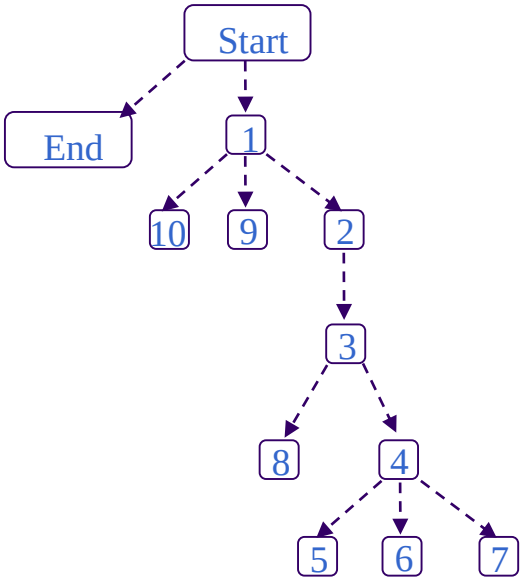
$$DF(S) = \cup (DF(x)), x \text{ in } S$$

Frontiera de dominare iterata *IDF(S)* pt un set de noduri S e multimea maximala din secventa:

$$IDF_1(S) = DF(S),$$
$$IDF_{i+1}(S) = DF(\cup (S, IDF_i(S))$$



(a)



(b)

DF(4) = {3}

IDF(4) = {3, 10, end}

IDF 1 (4) = DF{3}
IDF 2 (4) = DF{3 U *{3, 10}} = {3, 10, END}*
IDF 3 (4) = DF{3 U *{3, 10, END}} = {3, 10, END}*
In iteratia aceasta nu mai sunt schimbari, prin urmare
IDF(4) = {3, 10, END}

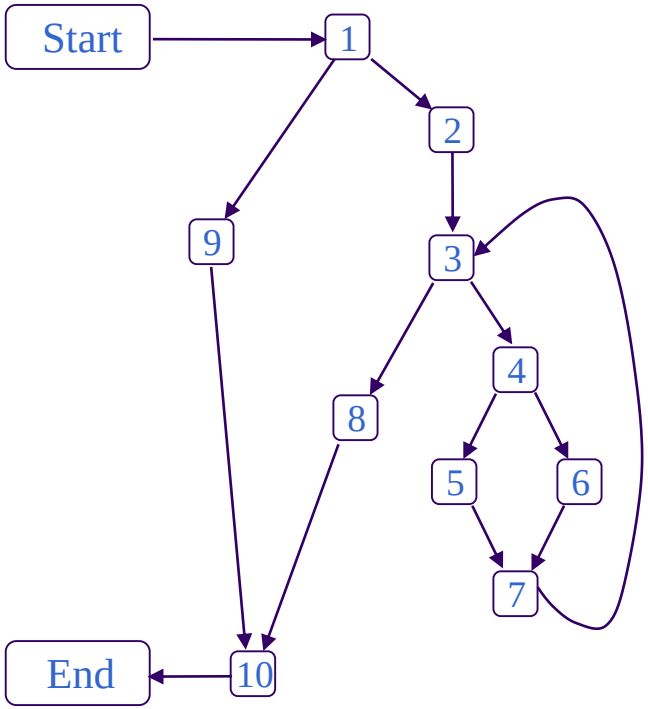
Cum se insereaza functii ϕ

1. Precalculeaza *DF(x)* pt. fiecare nod *x.*
2. Determina setul de noduri N_a. care contin definitia variabilei a
3. Calculeaza IDF(N_a) -> aici se insereaza fct. $\phi(a)$

Complexitatea calculului DF pentru un nod:

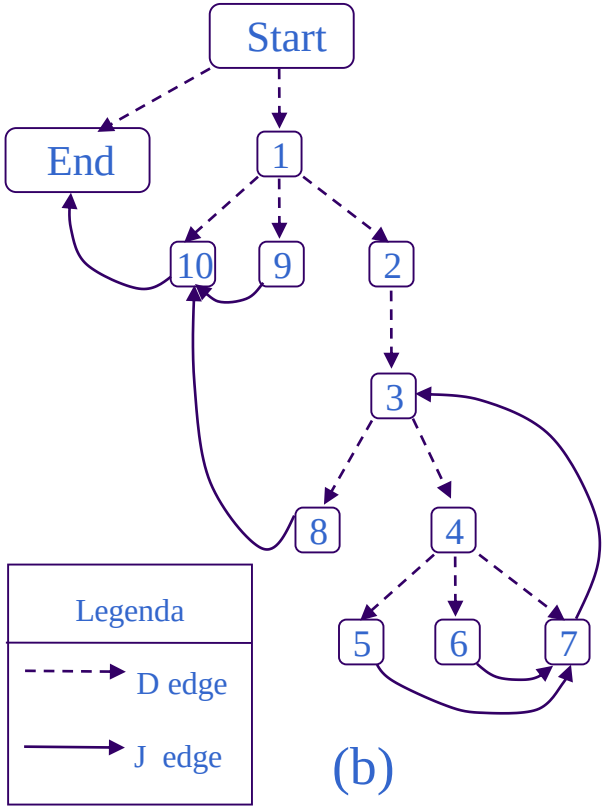
- implementare directa, O(N*N)
- Se poate si O(N)

Cum se calculeaza DF: Grafuri DJ



(a)

Graficul fluxului de control



(b)

Graful DJ

Level 0
Level 1
Level 2
Level 3
Level 4
Level 5

Calculul frontierei de dominare

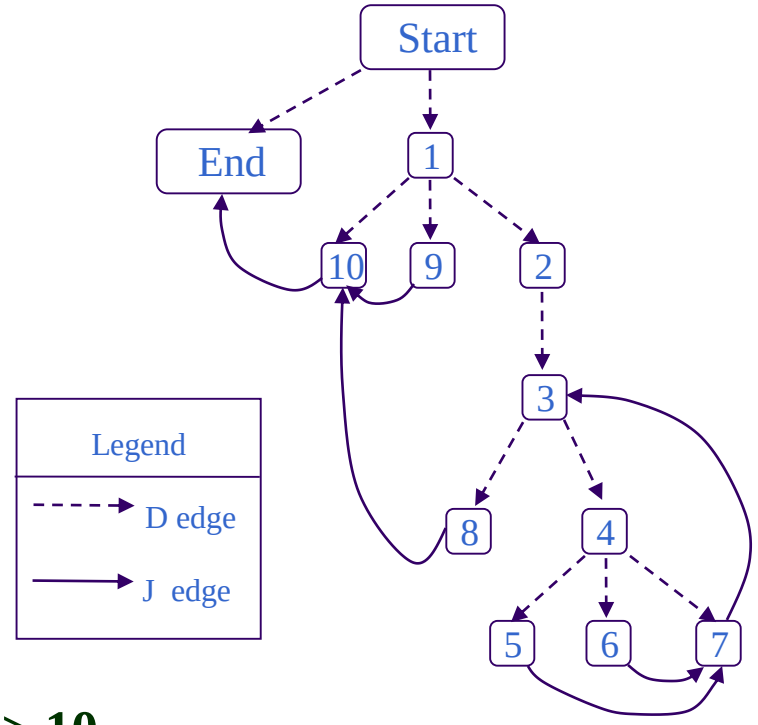
J-edge: o muchie x → y din CFG , x !sdom y

Formal:

DF(x) = Ø
For each y, x dom *y*
if*(y → z is J-edge)*
if*(z.level<= x.level)*
DF(x) = DF(x) U {*z*}

Complexitatea algoritmului e *0(*|*N*| + |*E*|*)*, cazul cel mai defavorabil

Exemplu



Note: legatura 8 -> 10
Level(10) < level(3). 10 este in DF(3) si IDF(3)

Eliminarea functiei ϕ

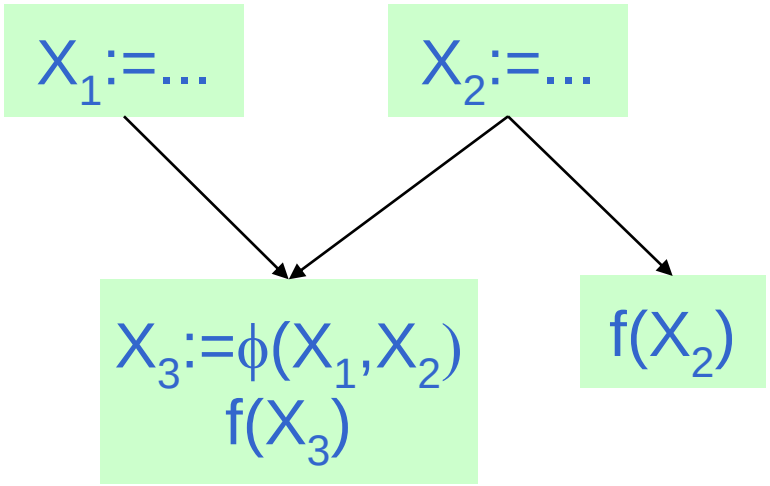
Cum implementam o functie ϕ care "stie" pe ce cale s-a ajuns la ea?

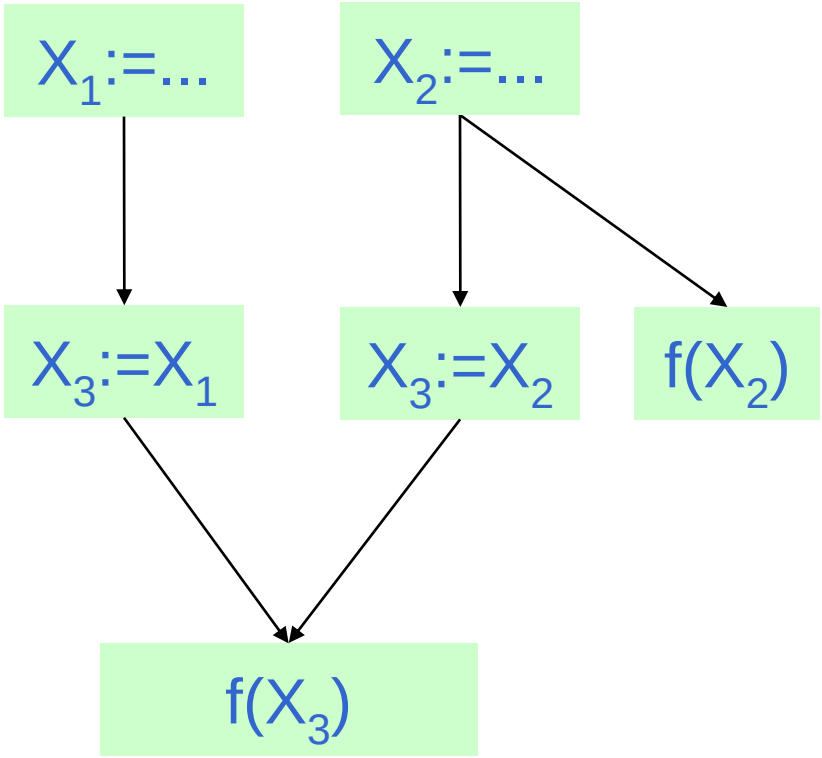
Rasp. 1: Nu o implementam! Functia ϕ e o conventie folosita doar pentru a conecta definiri cu folosiri in timpul optimizarilor, dar nu e implementata in practica.

(dupa optimizari ar putea sa nu mai fie posibil)

Rasp. 2: Ca sa 'scapam' de functiile ϕ, putem insera instructiuni "MOVE" pe toate caile de control.

Eliminarea functiei ϕ





Se poate folosi un singur registru pentru X_1,X_2,X_3?

Extragerea codului invariant

- O definire *d: t := x * y* este invarianta in bucla daca
  - x, y sunt constante, sau
  - toate definirile lui x si y se gasesc in afara buclei, sau
  - exista o singura definire a lui x (sau y) in bucla, si acea definire este invarianta.
- Conditii suficiente (conservatoare)

Extragerea codului invariant

- Codul invariant *d: t := x * y* poate fi mutat in pre-header daca
  - Definirile lui x si y se gasesc in afara buclei
  - Exista o singura definire a lui t in bucla
  - “d” domina toate iesirile din bucla in care t este in viata.
  - t nu este folosit la iesirea din pre-header
- Cum se scriu conditiile pentru SSA?
  - t nu are functii Φ in bucla

Extragerea codului invariant

```
 t=0
L1:
 i=i+1
 t=a*b
 m[i]=t
 if i<N goto L1
L2:
 x=t
```

```
 t=0
L1:
if i<N goto L2
 i=i+1
 t=a*b
 m[i]=t
 goto L1
L2:
 x=t
```

```
 t=0
L1:
 i=i+1
 t=a*b
 m[i]=t
 t=0
 q=t
 if i<N goto L1
L2:
 x=t
```

CPL
IR crunch

Detectia variabilelor de inductie

- Variabila de inductie IV – “index” in bucla
  - Cresc la fiecare iteratie cu un pas constant
- Variabila de inductie de baza:
  - `I = I + c`, c este un invariant in bucla
- Variabila de inductie derivata
  - `J = I*a + b`, `I` este variabila de inductie, a,b sunt invarianti
- De ce?
  - Strength reduction
  - Analiza de dependenta

Variabile de inductie

```
int *a;
int i;
for (i=0; i<100; i++)
  a[i] = 100 - 2*i;
```

```
 i=0
L1:
 if i>=100 goto L2
 t1 = 2*i
 t2 = 100 - t1
 t3 = 4*i
 t4 = &a + t3
 *t4 = t2
 i=i+1
 goto L1
L2:
```

Detectia variabilelor de inductie

- Detectia variabilelor de baza
  - O singura definire in bucla, de tip I = I + c, c invariant
  - I = I * 1 + c. Notatie : I = (I, 1, c)
- Detectia variabilelor derivate
  - O singura definire in bucla, J = I*a + b, I este IV de baza, a,b sunt invarianti → J=(I,a,b)
  - O singura definire in bucla, K = J*a + b, J=(I,p,q), I nu este definit intre J si K
    - → K=(I, a*p, a*q+b)

CPL
IR crunch

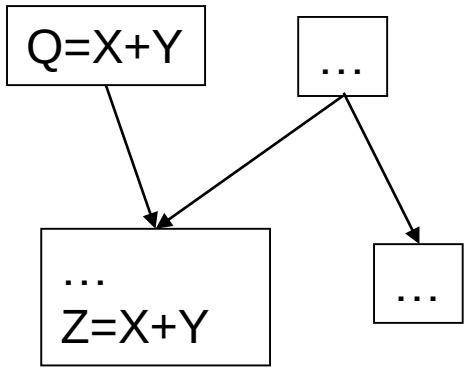
Eliminarea variabilelor de inductie

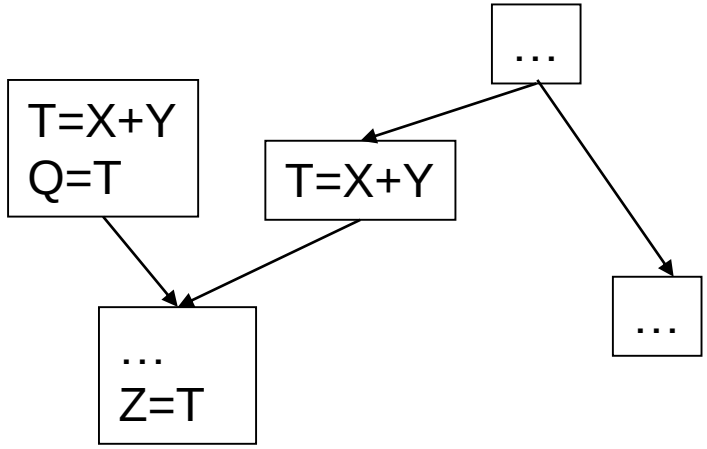
- Compunerea functiilor liniare e liniara
- O variabila tip J=(I,a,b)
  - In preheader: J = I * a + b (dupa definirea lui I)
  - In bucla: J = J + a
- Exercitiu - transformarea

Backup slides

Analize de flux complexe

- PRE – Eliminarea redundantei partiale
  - Include eliminarea subexpresiilor comune
  - Include scoaterea invariantilor in afara buclei
  - Implementare: Eliminarea muchiilor critice urmat de un set complex de analize de flux

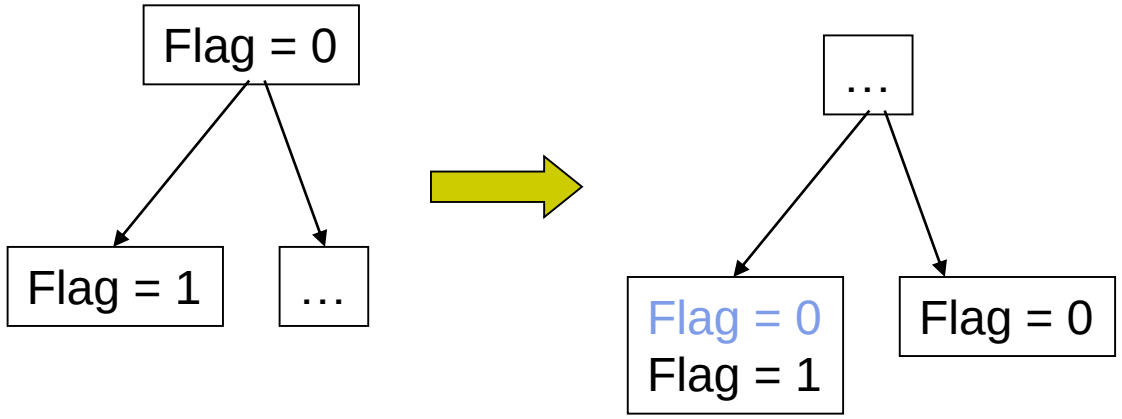




Analize de flux complexe

- “Partial dead code” → “True dead + True live”
- Determinarea punctului de inserare a instructiunilor

```
Flag = 0
If (..)
   {
   Flag = 1
   …
   }
Else { …}
```

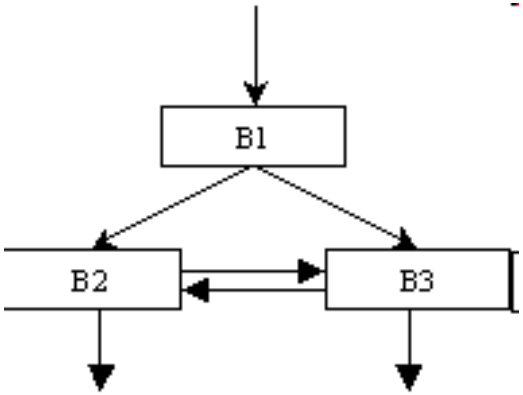


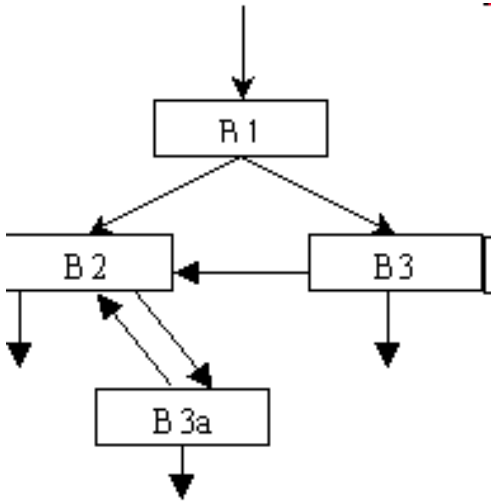
Analiza pe regiuni

- Alternativa la algoritmul iterativ
- Regiune – secventa de noduri dominate de un nod "header"
  - Daca m,n,h sunt noduri, n e in regiune, h e header, h dom n, h dom m, si exista cale m→n, atunci m e in regiune.
- Tipuri de regiuni
  - Aciclice (noduri = alte regiuni)
  - Bucle naturale (un arc inapoi)
  - Regiuni improprii

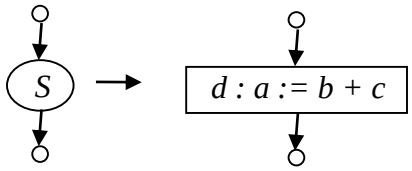
Node splitting

- Regiuni improprii – componente tare conexe ce nu sunt bucle naturale.
- Duplicarea nodurilor cu mai multi predecesori.
- Ce noduri duplicam?
- O euristica buna: “Making Graphs Reducible with Controlled Node Splitting”, Janssen & Corporaal



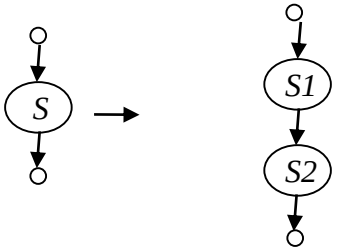


Analiza pe regiuni (RD)



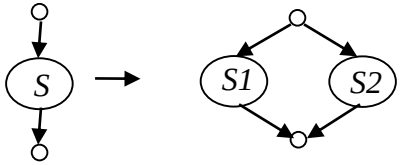
gen[S] = {d}

kill [S] = {orice def a:=...}



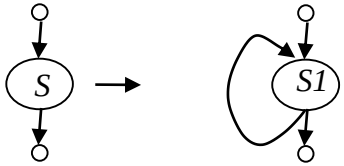
gen [S] = gen [S2] ∪ (gen [S1] - kill [S2])

kill [S] = kill [S1] ∪ kill [S2]



gen [S] = gen [S1] ∪ gen [S2]

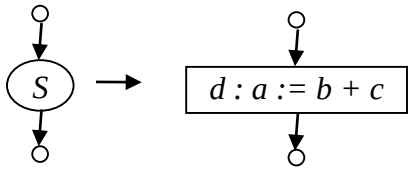
kill [S] = kill [S1] ∩ kill [S2]



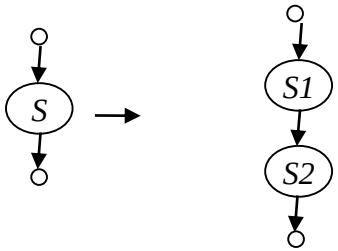
gen [S] = gen [S1]

kill [S] = kill [S1]

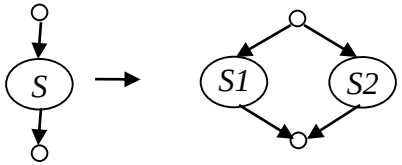
Analiza pe regiuni (RD)



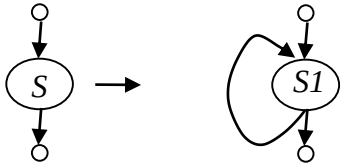
out [S] = gen [S] ∪ (in [S] - kill [S])



in [S1] = in [S]
in [S2] = out [S1]
out [S] = out [S2]



in [S1] = in [S]
in [S2] = in [S]
out [S] = out [S1] ∪ out [S2]



in [S1] = in [S] ∪ out [S1]
out [S]= out [S1]