Compilatoare

Analiza fluxului de date

Analiza fluxului de date (DFA)

- Analiza la nivelul unei proceduri
- “Executia” statica a procedurii la momentul compilarii.
- Rezultatul: informatii despre cum poate fi transformat codul – in mod legal.

DFA - utilitatea

S : A $\leftarrow$ 2 (def A)

S : B $\leftarrow$ 10 (def B)

⋮

S: C $\leftarrow$ A + B este C constantă (12) ?

for I = 1 to C

X[I] = Y[I] + D[I-1]

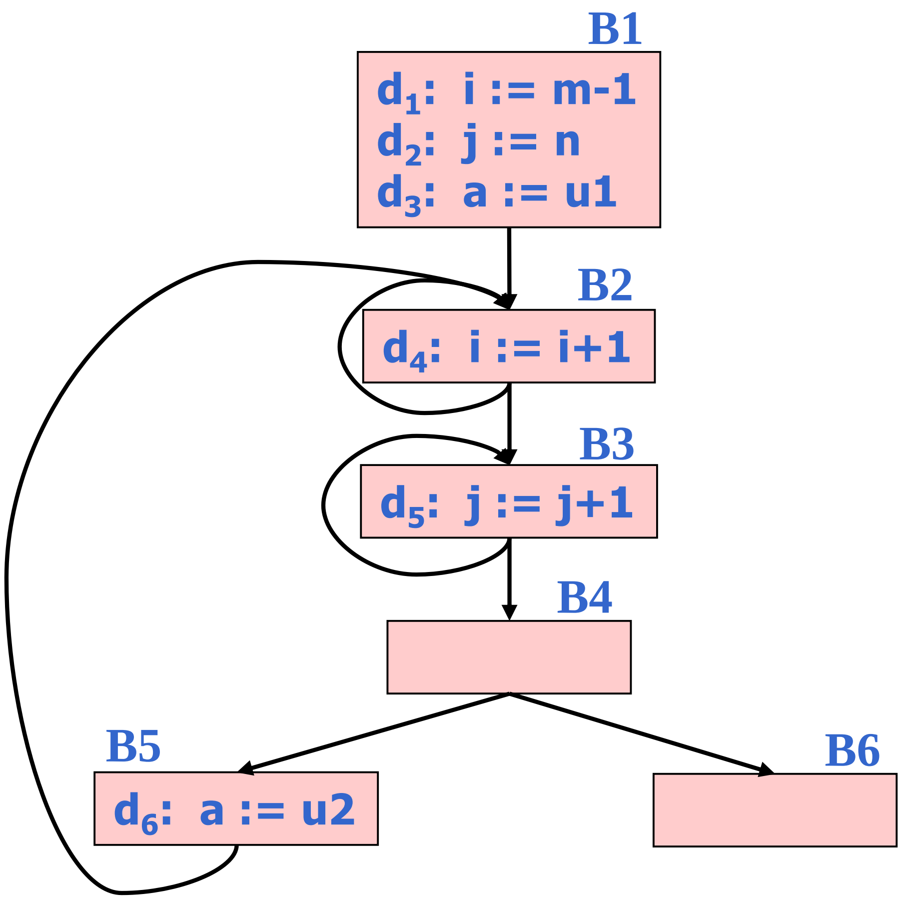
Exemplu: analiza def/use

S: V1 = V2 x V3

- S e o “definire” a lui V1 (def)
- S e o “folosire” a lui V2 sau V3 (use)

- Unde va fi folosit V1? (‘exposed uses’)
- Unde a fost definit V2? (‘reaching definitions’)

Puncte şi Căi

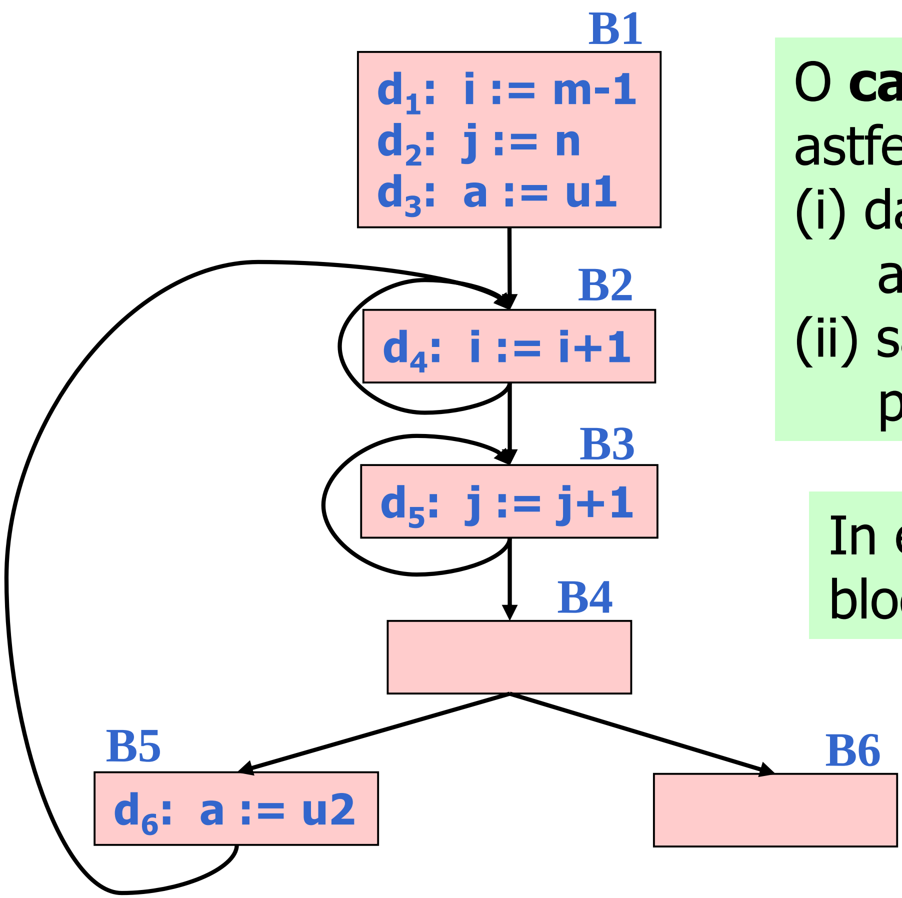


puncte într-un basic block:
- între instrucţiuni
- la începutul bb
- după ultima instrucţiune

Câte puncte au blocurile B1,B2, B3, B4 şi B5?

B1 are 4, B2, B3 şi B5 au câte 2, B4 are 1/2

Puncte şi Căi



O **cale** e o secvenţă de puncte p_1, p_2, …, p_n astfel încât:

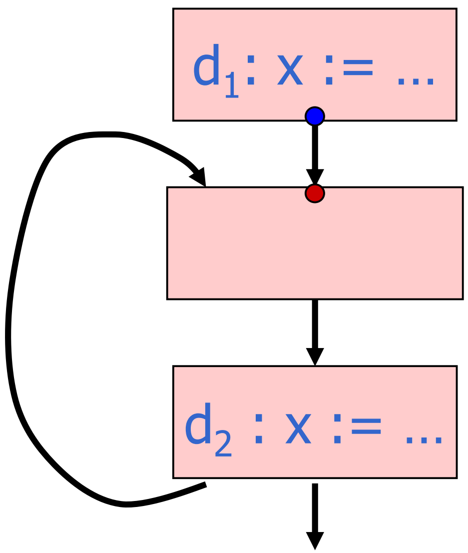
(i) dacă p_i apare imediat înainte de S, atunci p_{i+1} apare imediat după S.

(ii) sau p_i e sfârşitul unui basic block şi p_{i+1} e începutul unui bloc succesor

In exemplu, există o cale de la începutul blocului B5 la începutul blocului B6?

Da, B5-> B2,B3,B4, B6.

Reach şi Kill



Kill

o definire $\mathbf{d_1}$ a variabilei **v**
e "omorâtă" între $\mathbf{p_1}$ şi $\mathbf{p_2}$
dacă în <u>oricare cale</u> de la $\mathbf{p_1}$ la $\mathbf{p_2}$
există altă definire a lui **v**.

Reach

o definire **d** "ajunge" la un punct **p**
dacă $\exists$ o cale **d** $\rightarrow$ **p**, de-a lungul căreia variabila nu e redefinită

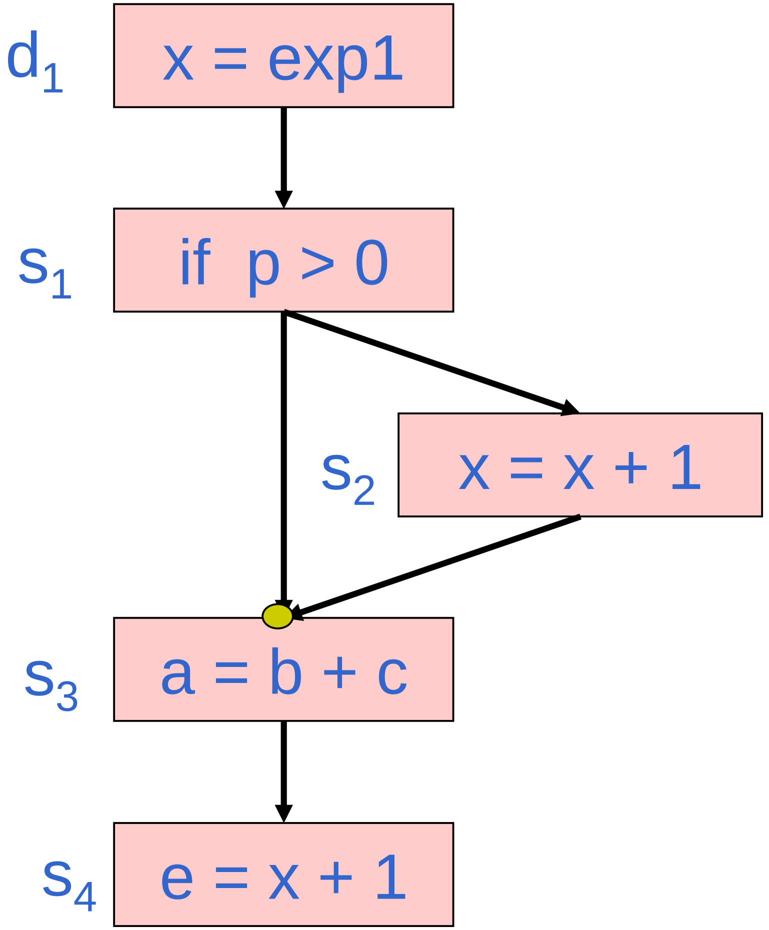
În exemplu, d_1 şi d_2 ajung la punctele 🔴 şi 🔵 ?

d_1, d_2 ajung la punctul 🔴
Doar d_1 ajunge la punctul 🔵

Exemplul 1

Poate d_1 ajunge la p_1?

d_1	x = exp1
s_1	if p > 0
s_2	x = x + 1
	← p_1
s_3	a = b + c
s_4	e = x + 1

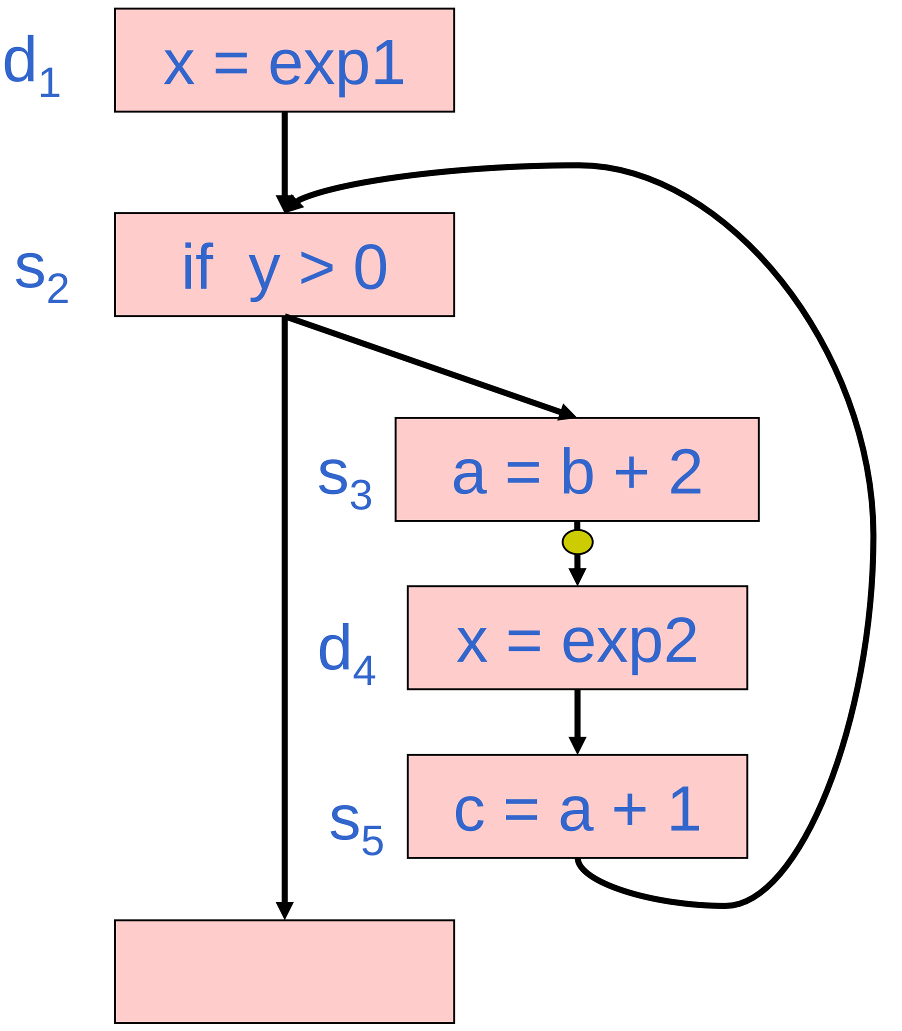


Exemplul 2

d_1 şi d_4 ajung la punctul p_3?

d_1	x = exp1
s_2	while y > 0 do
s_3	a = b + 2
d_4	x = exp2
s_5	c = a + 1
	end while

$\longleftarrow p_3$



Metoda iterativă de DFA

- Stabileşte un set de functii de flux (pentru fiecare instructiune / basic block)
- Stabileşte un set de ecuaţii de flux (între basic blocks)
- Stabileşte valorile iniţiale
- Rezolvă/aplică iterativ ecuaţiile de flux, până când se ajunge la un **punct fix**.

Metoda iterativă de DFA (Exemplu: Reaching definitions)

- Stabilirea ecuatiilor de flux pentru fiecare instructiune:

$$out[S] = gen[S] \vee \{ in[S] - kill[S] \}$$

informatie la sfârşitul unei instrucţiuni

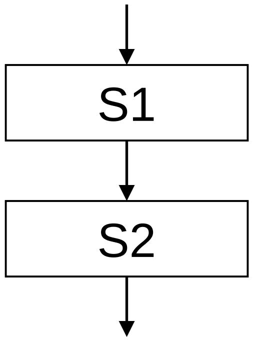
info. generată de *S*

intrarea *S* care nu este "omorâtă" de S

Ecuaţiile de flux depind de problema pe care vrem sa o rezolvăm

Metoda iterativă de DFA (Exemplu: Reaching definitions)

- Compunerea ecuatiilor de flux pentru un basic block



$$out\,[S_1] \;=\; gen\,[S_1] \quad V \quad \{\, in\,[S_1] - kill\,[S_1] \,\}$$

$$in[S_2] = out[S_1]$$

$$out\,[S_2] \;=\; gen\,[S_2] \quad V \quad \{\, in\,[S_2] - kill\,[S_2] \,\}$$

$$out\,[S_2] \;=\; gen\,[S_2] \quad V \quad \{\, \{gen\,[S_1] \quad V \quad \{\, in\,[S_1] - kill\,[S_1] \,\}\} \; - kill\,[S_2] \,\}$$

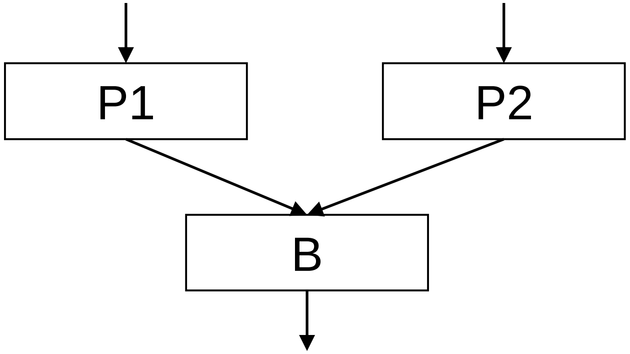
$$out\,[S_2] \;=\; \{\, gen\,[S_2]\ V\ \{gen\,[S_1] - kill[S_2]\,\}\ V\ \{\, in\,[S_1] - \{\, kill\,[S_1]\ V\ kill\,[S_2] \,\}\}$$

$$out\,[B] \;=\; gen\,[B] \quad V \quad \{\, in\,[B] - kill\,[B] \,\}$$

Daca se cunosc valorile analizei la inceputul blocului, se pot calcula pentru fiecare instructiune

Metoda iterativă de DFA (Exemplu: Reaching definitions)

- Calculul ecuatiilor de flux intre basic block-uri



in [B] = out[P_1] V out [P_2]

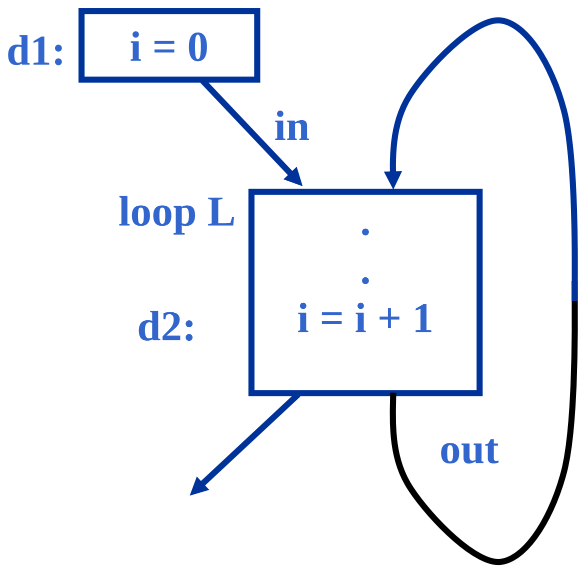
- Stabilirea conditiilor limita

out[ENTRY] = { }

Analiza gaseste o solutie pentru ecuatiile de mai sus

Exemplu

- Folosim RD (reaching def) ca exemplu:



Întrebare:

Care e setul de "reaching definitions" la ieşire din bucla L?

in (L) = {d1} $\cup$ out(L)
gen (L) = {d2}
kill (L) = {d1}

Solutia

Prima iteratie

out(L) = *gen* (L) $\cup$ (in (L) - kill (L))
= {d2} $\cup$ ({d1} - {d1})
= {d2}

A doua iteratie

out(L) = *gen* (L) $\cup$ (in (L) - kill (L))

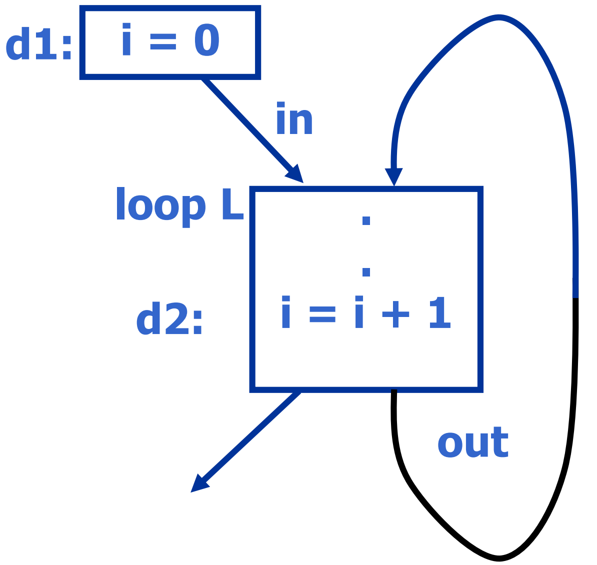
dar acum:

in (L) = {d1} $\cup$ out(L) = {d1} $\cup$ {d2}
= {d1, d2}

deci:

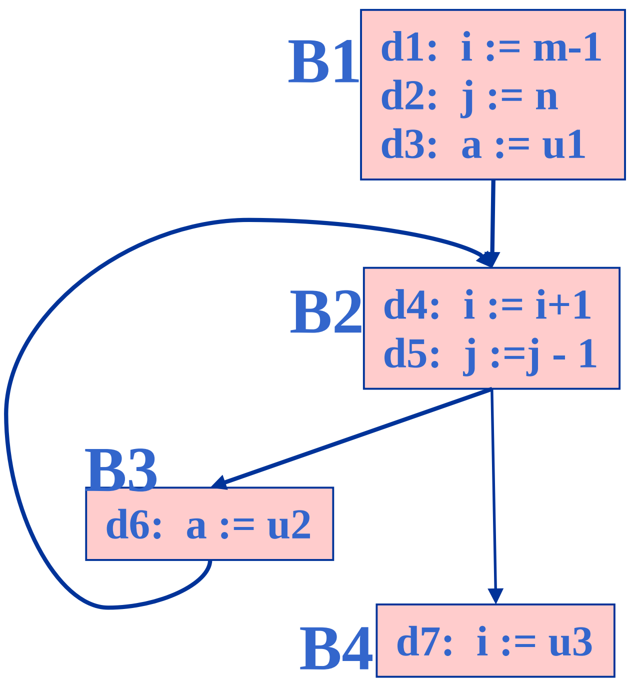
out (L) = {d2} $\cup$ ({d1, d2} - {d1})
= {d2} $\cup$ {d2}
= {d2}

Am ajuns la un punct fix!



in (L) = {d1} $\cup$ out(L)
gen (L) = {d2}
kill (L) = {d1}

Algoritm Iterativ pentru Reaching Definitions



Pas 1: Calculeaza gen si kill pentru fiecare basic block

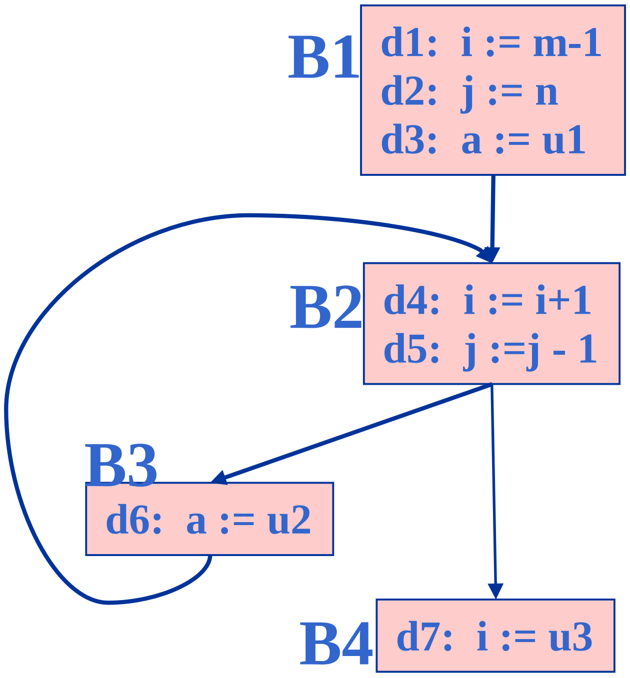
gen[B1] = {d1, d2, d3}
kill[B1] = {d4, d5, d6, d7}

gen[B2] = {d4, d5}
kill [B2] = {d1, d2, d7}

gen[B3] = {d6}
kill [B3] = {d3}

gen[B4] = {d7}
kill [B4] = {d1, d4}

Algoritm Iterativ pentru Reaching Definitions



Pas 2: Pentru fiecare basic block,
out[B] = gen[B]

Initializare:

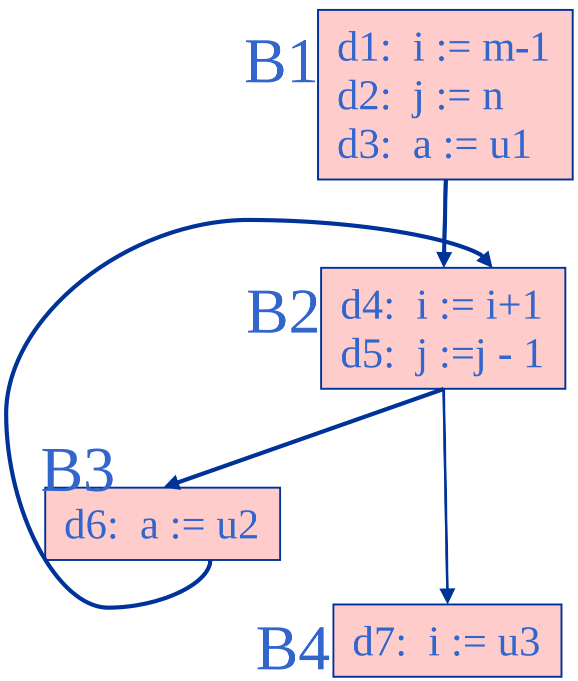
in[B1] = ∅
out[B1] = {d1, d2, d3}

in[B2] = ∅
out[B2] = {d4, d5}

in[B3] = ∅
out[B3] = {d6}

in[B4] = ∅
out[B4] = {d7}

Algoritm Iterativ pentru Reaching Definitions



while (nu am ajuns la punct fix):

in[B] = ∪ out[P] unde P este un predecesor al lui B

out[B] = gen[B] ∪ (in[B]-kill[B])

Initializare:

in[B1] = ∅
out[B1] = {d1, d2, d3}

in[B2] = ∅
out[B2] = {d4, d5}

in[B3] = ∅
out[B3] = {d6}

in[B4] = ∅
out[B4] = {d7}

Prima Iteratie:

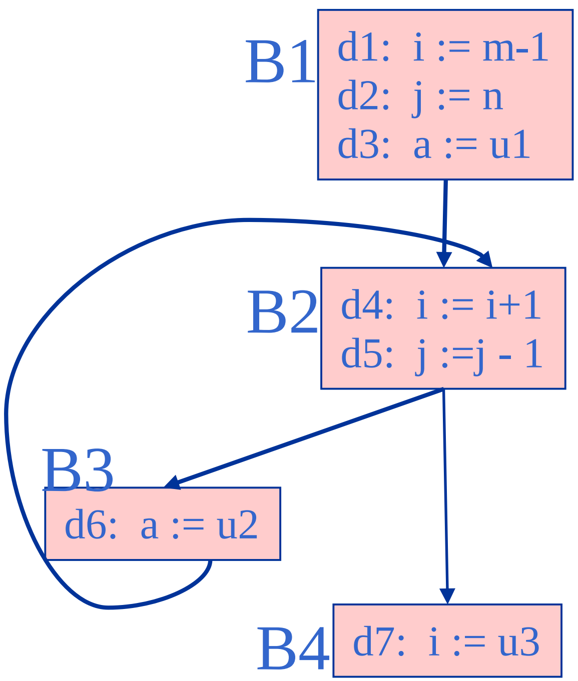
in[B1] = ∅
out[B1] = {d1, d2, d3}

in[B2] = {d1, d2, d3, d6}
out[B2] = {d3, d4, d5, d6}

in[B3] = {d3, d4, d5, d6}
out[B3] = {d4, d5, d6}

in[B4] = {d3, d4, d5, d6}
out[B4] = {d3, d5, d6, d7}

Algoritm Iterativ pentru Reaching Definitions



while (nu am ajuns la punct fix):

in[B] = ∪ out[P] unde P este un predecesor al lui B

out[B] = gen[B] ∪ (in[B]-kill[B])

Prima Iteratie:

in[B1] = ∅
out[B1] = {d1, d2, d3}

in[B2] = {d1, d2, d3, d6}
out[B2] = {d3, d4, d5, d6}

in[B3] = {d3, d4, d5, d6}
out[B3] = {d4, d5, d6}

in[B4] = {d3, d4, d5, d6}
out[B4] = {d3, d5, d6, d7}

A doua Iteratie:

in[B1] = ∅
out[B1] = {d1, d2, d3}

in[B2] = {d1, d2, d3, d4, d5, d6}
out[B2] = {d3, d4, d5, d6}

in[B3] = {d3, d4, d5, d6}
out[B3] = {d4, d5, d6}

in[B4] = {d3, d4, d5, d6}
out[B4] = {d3, d5, d6, d7}

Complexitate – Reaching definitions

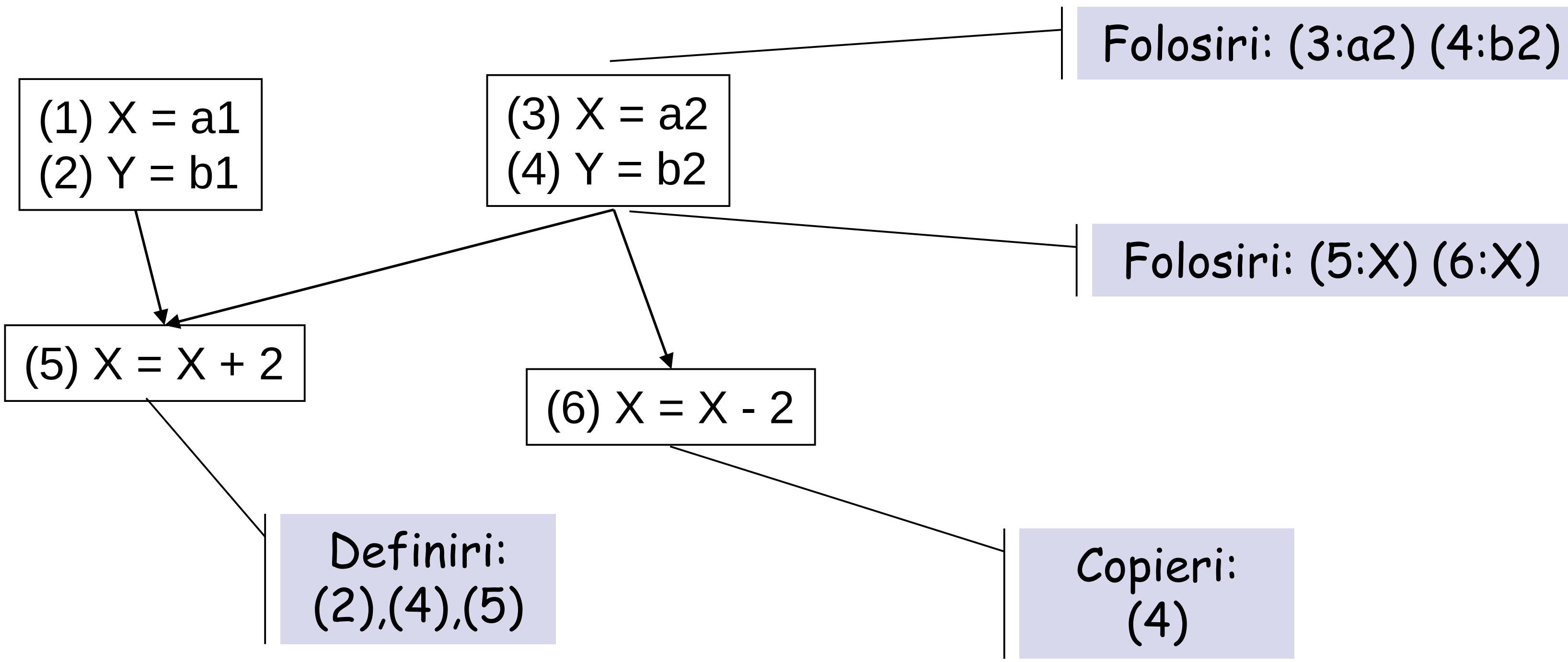
- Numar de iteratii limitat de dimensiunea CFG.
- Alegerea ordinii de parcurgere a nodurilor e importanta.
- Cum poate fi implementat eficient? (‘worklist’)

Cum rezolvam o problema de DFA

- Pasul 1: Care sunt valorile analizate? (cum ne reprezentam datele?)
- Pasul 2: Directia problemei? Daca o reducem la un BB, ne uitam “in jos” sau “in sus” pt. a afla raspunsul?
- Pasul 3: Ce se intampla cu informatia, pe un “if”? (meet vs. join)
- Pasul 4: Care e functia de flux a fiecarui bloc? (reprezentare: kill/def – cand e posibil)
- Pasul 5: Cum initializam informatia de flux? (hint: daca facem “OR” initializam cu 0, daca facem “AND” initializam cu 1)
- Pasul 6: Iteratia, pana ajungem la punct fix.

Analize de flux de date

- Definiri disponibile ("reaching definitions")
- Folosiri expuse ("exposed uses")
- Propagarea copierilor



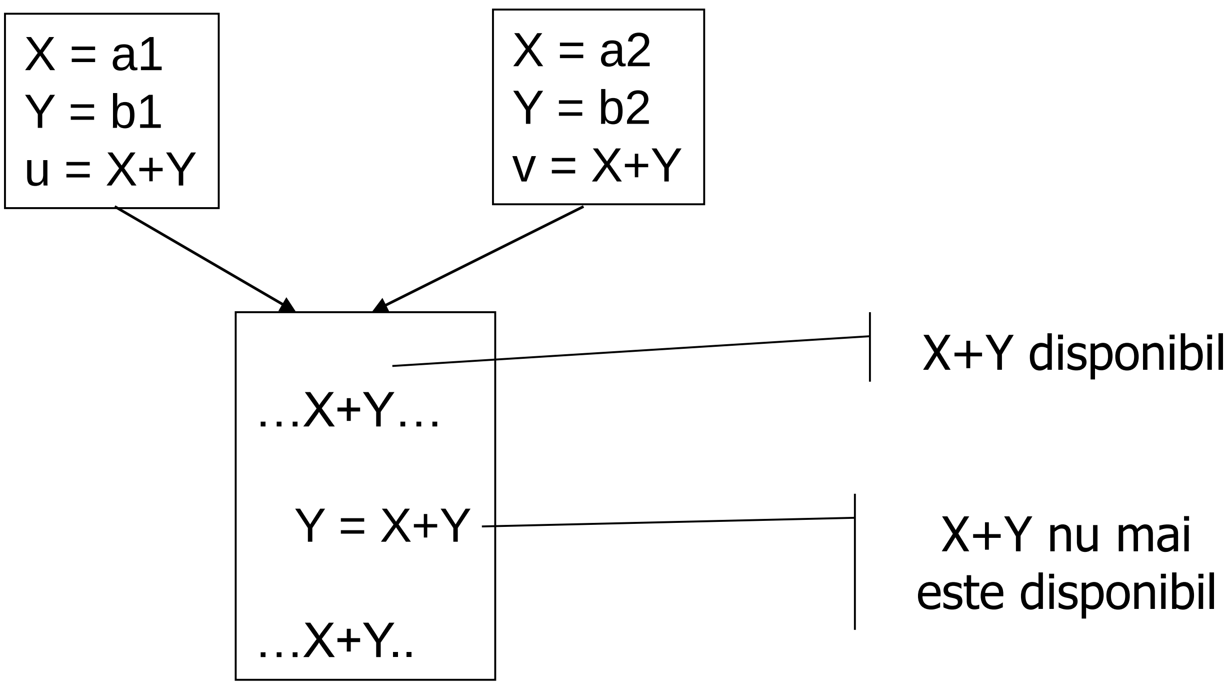
Alte analize de flux

- Exposed uses - reversul lui “reaching definitions”
  - Analiza inapoi
  - Cum se calculeaza functiile de flux?
  - Cum se combina informatiile din basic block-urile succesoare?

- Propagarea copierilor
  - Determina perechile de variabile care au intotdeauna aceeasi valoare la un punct al programului, in urma unei instructiuni de copiere.

Alte analize de flux

- Expresiile disponibile
  - Determina expresiile care au fost deja calculate in program.
  - Utilizare: eliminarea calculelor redundante

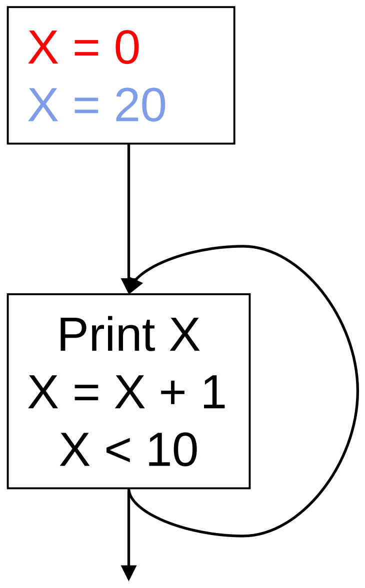


Alte analize de flux

- Propagarea constantelor
  - Determina perechile “variabila = valoare constanta”.
  - Doua valori speciale: “nedefinita” si “nu e constanta”
    - const $\cap$ not-a-const -> not-a-const
    - const $\cap$ undef -> const
    - constA $\cap$ constA -> constA
    - constA $\cap$ constB -> not-a-const

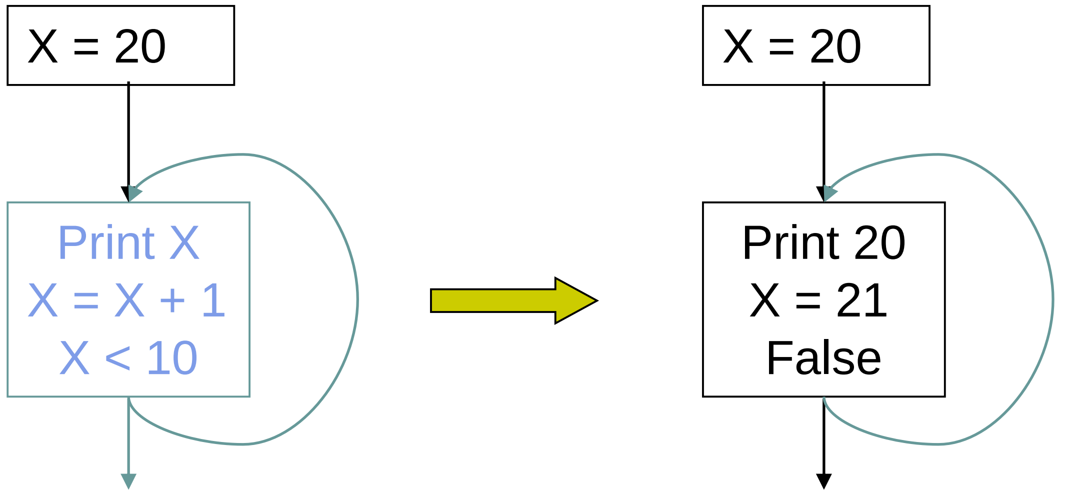
  - Cate iteratii sunt necesare in exemplu?
  - Se poate simplifica pentru X = 20 ?

- Generalizare: domenii (“Range analysis”)
  - Determina multimea valorilor unei variabile



Alte analize de flux

- Propagarea conditionala a constantelor
  - Marcheaza muchiile din graf “vizitate”
  - Aplica operatorul “meet” doar pe predecesorii “vizitati”



Alte analize de flux

- Analiza variabilelor în viaţă
  - O variabilă *V* este *live* la sfârşitul unui basic block *n*, dacă există o cale fara definiri de la *n* la o folosire a variabilei V din alt bloc *n'*.
  - “live variable analysis problem” - determină setul de variabile care sunt “live” (în viaţă) pt orice punct din bloc.
  - Utilizari: alocarea registrilor

Alte analize de flux

- Eliminarea codului “mort“
  - Instructiune cu efecte laterale (nu poate fi eliminata)
    - Util(In, V) = Use(V) or (Util(Out,V) and not Def(V))
  - Instructiune fara efecte laterale (A = f(V))
    - Util(In, V) = (Util(Out,A) and Use(V))
      or (Util(Out,V) and not Def(V))

Backup Slides

Formalismul matematic

- Este algoritmul iterativ corect?
- Algoritmul converge catre o solutie?
- Este solutia exacta?

(Gary Kildall, 1972)

Formalismul matematic

- Latice – structura algebrica.
- Elementele laticei - proprietăti abstracte ale variabilelor, expresiilor sau altor componente din program.
- Fiecărui punct din program i se asociază un element de latice care memorează proprietăţile urmărite de analiză, in acel punct.
- Functiile de flux modeleaza efectul pe care îl are fiecare componentă a programului asupra elementelor de latice.
- Analiza calculeaza valorile laticei care rezolva ecuatiile date de functiile de flux.

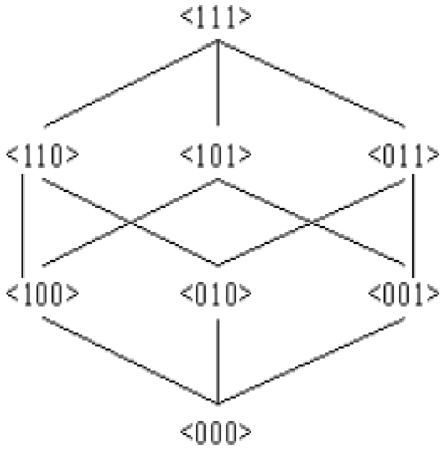
Formalismul matematic

- O latice L este formată dintr-o mulţime de valori si două operaţii pe care le vom nota ∩ ("meet") si U ("join") si care au următoarele proprietăţi:

1. Pentru orice x, y Є L exista z Є L si w Є L unici, astfel încât $x \cap y = z$ şi $x \cup y = w$ ***(închidere)***
2. Pentru orice x, y Є L, $x \cap y = y \cap x$ si $x \cup y = y \cup x$ ***(comutativitate)***
3. Pentru orice x, y, z Є L, $(x \cap y) \cap z = x \cap (y \cap z)$ şi $(x \cup y) \cup z = x \cup (y \cup z)$ ***(asociativitate)***
4. Pentru orice x, y Є L, $(x \cap y) \cup y = y$ si $(x \cup y) \cap x = x$ ***(absorbtia)***
5. Exista doua elemente unice ale lui L, pe care le numim min (notat ⊥) şi max (notat T), astfel încât ∀x Є L, x ∩ ⊥ = ⊥ si x U T = T ***(unicitatea existentei elementelor de minim si maxim)***.
6. Numeroase latici sunt şi ***distributive***, adică pentru orice x, y, z Є L, avem: (x ∩ y) U z = (x U z) ∩ (y U z) şi (x U y) ∩ z = (x ∩ z) U (y ∩ z).

Latici pt analiza de date

- In cazul analizei de date, cele mai multe dintre laticile folosite au ca elemente constituente vectori de biţi iar operaţiile de bază sunt reprezentate de operaţiile AND (meet) si OR (join) aplicate pe biţi.
- Elementul min,max – vectori de 0 / 1
- Pentru o problema particulara de analiza a fluxului de date, o functie de flux (f : L → L) modeleaza efectul unei parti de program asupra "datelor"



Semi-latice

- Algoritmul iterativ nu foloseste decat o singura operatie, ∩ (+ functiile de flux)

1. Pentru orice $x, y \in L$ exista $w \in L$ unic, astfel încât $x \cap y = w$
2. Pentru orice $x \in L$, $x \cap x = x$
3. Pentru orice $x, y \in L$, $x \cap y = y \cap x$
4. Pentru orice $x, y, z \in L$, $(x \cap y) \cap z = x \cap (y \cap z)$
5. Exista un element unic al lui L, max (notat T), astfel încât $\forall x \in L$, $x \cap T = T$
6. Optional, exista un element min (notat $\perp$) astfel incat $x \cap \perp = x$
7. Exista o relatie de ordine partiala, $x \subseteq y$ daca $x \cap y = x$

Algoritmul iterativ

- Directia inainte

```
out[entry] = init;
for each block B: out[B] = T;
while out changes
    for each block B
        in[B] = ∩ out[P], P pred B
        out[B] = f_B(in[B]);
```

- Directia inapoi

```
in[exit] = init;
for each block B: in[B] = T;
while in changes
    for each block B
        out[B] = ∩ in[S], S succ B
        in[B] = f_B(out[B]);
```

Functii monotone

- Operatiile U şi ∩ introduc o relaţie de ordine partiala pe elementele laticei
  - $x \subseteq y \Leftrightarrow x \cap y = x$. (+ op. duală)
  - proprietati = reflexivitate, antisimetrie, tranzitivitate
- *Inaltimea* unei latici – lungimea maxima a unui lant x min $\subset$ x $\subset$… $\subset$ y $\subset$ max
- O funcţie ce mapeaza laticea pe ea insasi (f : L $\rightarrow$ L) este **monotona** dacă pentru $\forall$ x,y ∈ L, $x \subseteq y \Rightarrow f(x) \subseteq f(y)$

Algoritmul iterativ

- Daca algoritmul iterativ converge, atunci rezultatul este corect - o solutie a ecuatiilor de flux.
- Daca toate functiile de flux sunt monotone si algoritmul converge, atunci solutia este maximala (Maximum Fixed Point)
- Daca functiile de flux sunt monotone si laticea are inaltime finita, algoritmii de analiza a fluxului de date se termina.
  - Range analysis – latice infinita

Corectitudine si precizie

- De ce e algoritmul corect?
  - Fiecare iteratie executa un pas pe o cale din CFG (conservator)
  - MFP – echivalent cu iterarea la infinit - acopera toate caile

- Cazul ideal vs. MOP vs. MFP
  - Ideal - toate caile posibile din program sunt executate, laticea contine proprietatile comune.

    $IN_B = \bigcap f_{Bn}(\ldots f_{B1}(IN_{entry})))$ B1…BN cale executata pana la B
  - Meet Over Paths – toate caile din CFG sunt executate

    $IN_B = \bigcap f_{Bn}(\ldots f_{B1}(IN_{entry})))$ B1…BN cale pana la B
  - Maximum Fixed Point - ce calculam noi

Functii distributive

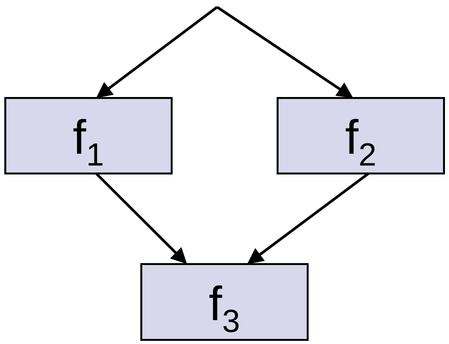
- $f(x \cap y) = f(x) \cap f(y)$
- Daca functiile de flux sunt distributive, atunci MOP si MFP calculeaza acelasi lucru
- Functiile de flux de tipul

  $OUT_B(IN_B) = GEN_B \cup (IN_B - KILL_B)$

  sunt distributive.

MFP vs. MOP

- MFP =MOP pt. functii de flux distributive
- Diferenta – de ex. propagarea constantelor

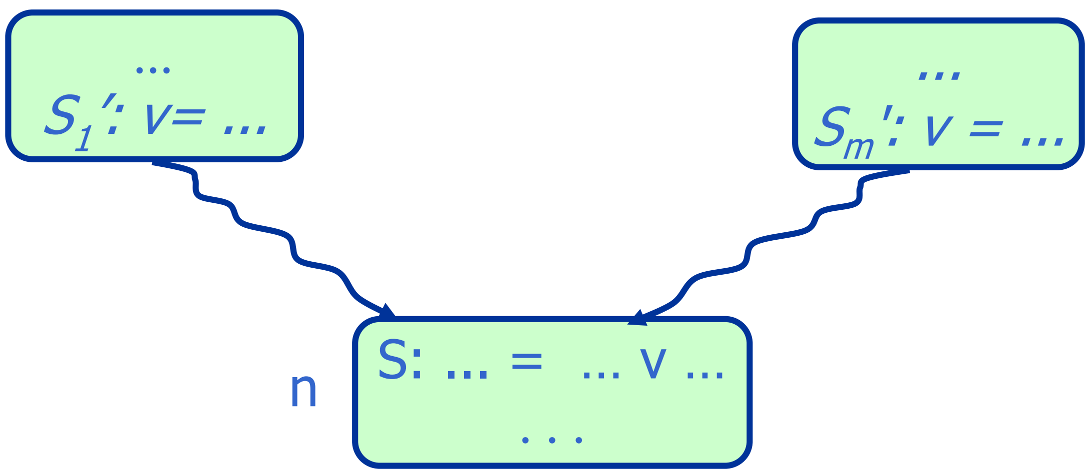


- MOP=f3(f1(entry)) ∩ f3(f2(entry))
- MFP=f3(f1(entry) ∩ f2(entry))

```
     bool a,b,x;
(1)  a = false;
(2)  b = false;
(3)  if (x)
(4)    {
(5)    a = true
(6)    }
     else
       {
(7)    b = true
(8)    }
(9)  z = a or b
(10)     if (z) ...
```

UD Chain

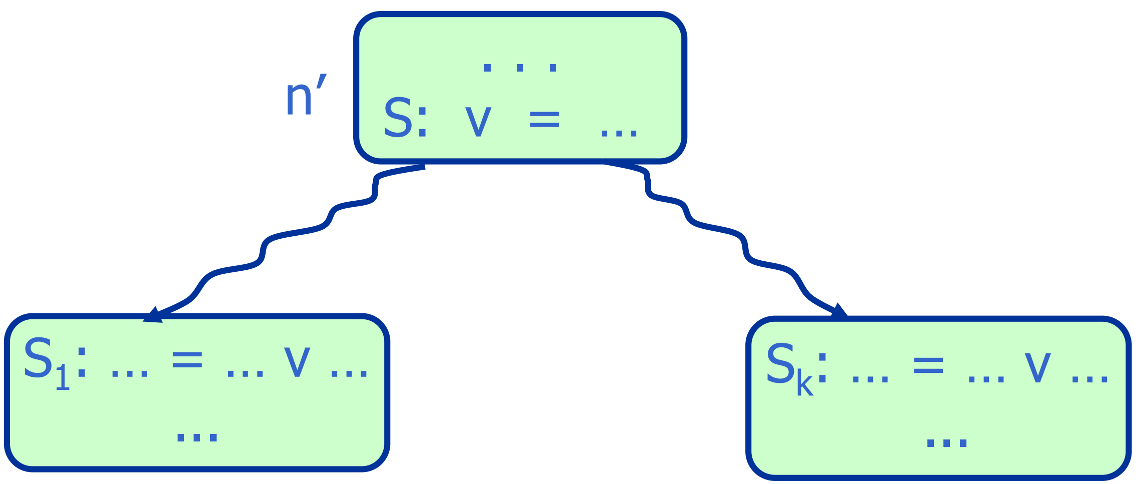
Un lant use-def(UD chain) e o lista a tuturor definirilor care pot ajunge la o folosire a unei variabile.



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

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")