Analýza toku dát

Ján Šturc

O čom to je ?

- Počas kompilácie usudzujeme o vlastnostiach a chovaní sa programu počas behu.
- Čo nás zaujíma
  - Vlastnosti, ktoré musia platiť vždy (invarianty) napr.:
    - V príkaze x:= y + z je y vždy rovné 1.
    - Pointer p vždy ukazuje do poľa a.
    - Príkaz S sa nikdy nevykoná.
  - Vlastnosti, ktoré platia často napr.:
    - Spravidla sa príkaz S_1 vykoná častejšie ako príkaz S_2.
  - Vlastnosti, ktoré sa musia aspoň raz splniť počas vykonávania programu (intermitenty) napr.:
    - Počas vykonávania programu premenná x niekedy nadobudne hodnotu a.

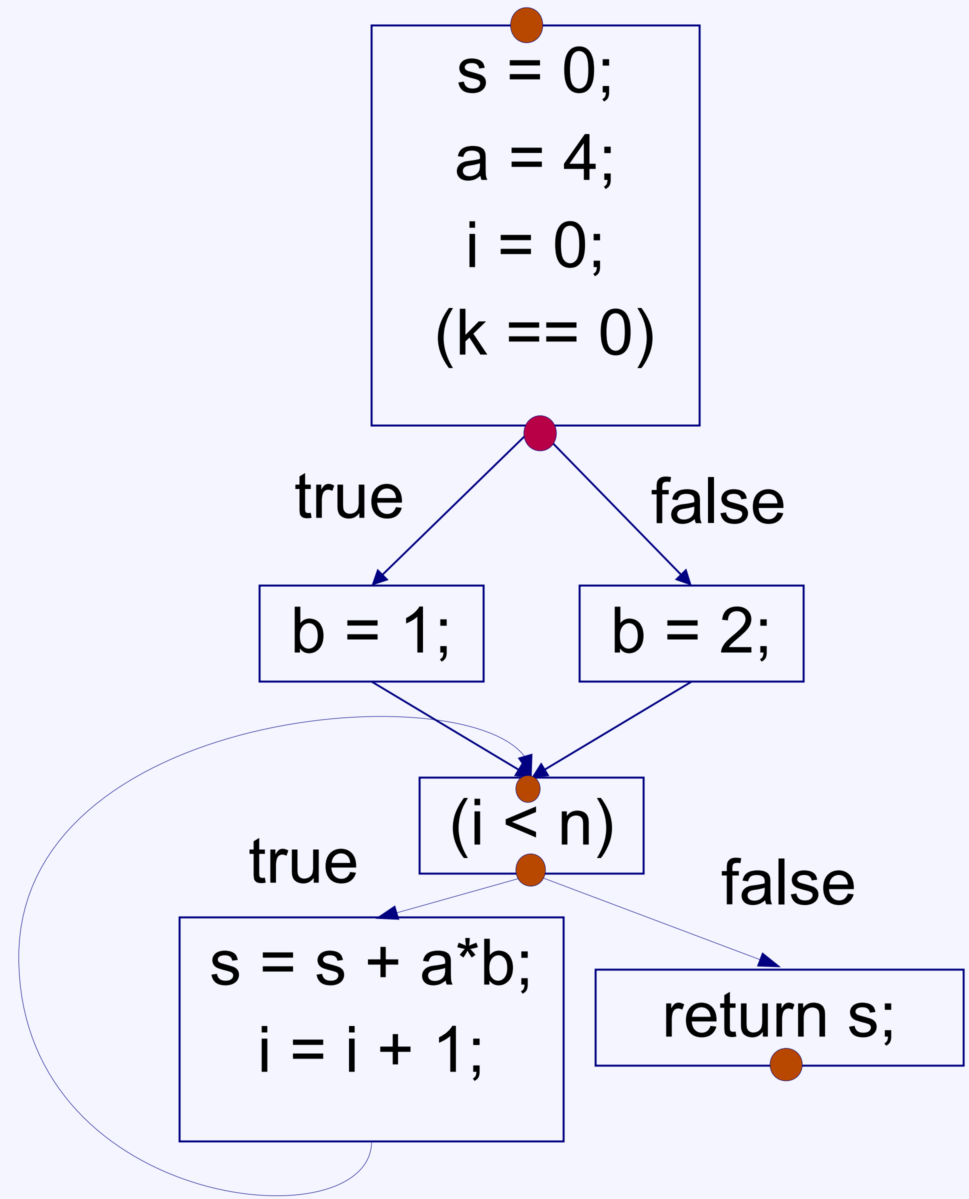
Graf toku riadenia (blokový diagram)

- Uzly reprezentujú základné bloky
  - Základný blok je postupnosť inštrukcií obsahujúca najviac jeden skok ako poslednú inštrukciu.
  - Všetky vstupy do základného bloku vedú cez jeho hlavičku (prvú inštrukciu)
  - Základný blok je (má byť) maximálny.
  - Dôsledok: Ak sa základný blok začne vykonávať vykoná sa celý
- Hrany reprezentujú vlastný tok riadenia (skoky a vetvenie)
  - Vzhľadom na definované inštrukcie je možné len binárne vetvenie.
  - Pri zovšeobecnených úlohách obmedzenie na vetvenie neplatí.

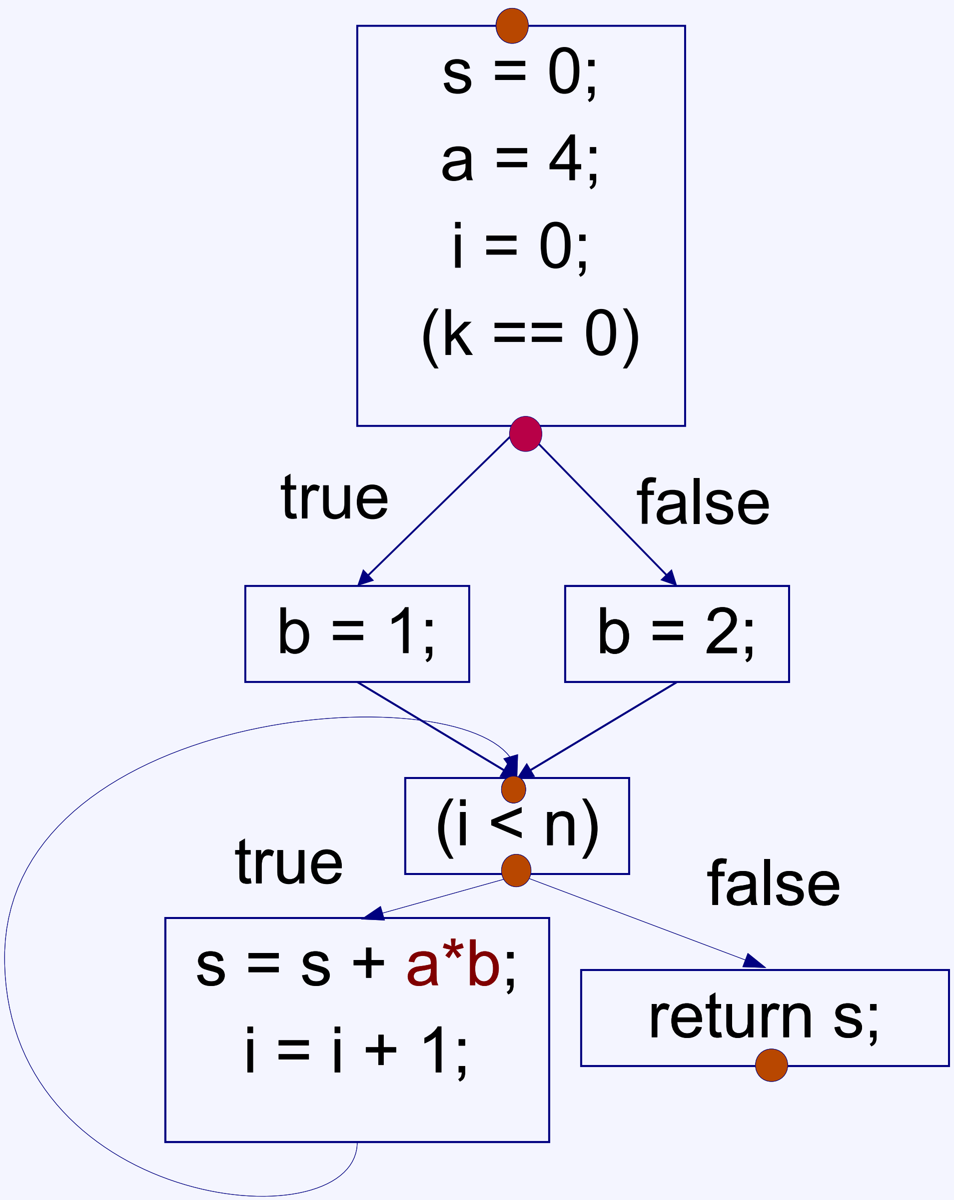
Príklad grafu toku riadenia

```
into add(n, k) {
    s = 0; a = 4; i = 0;
    if (k == 0) b = 1;
    else b = 2;
    while (i < n) {
        s = s + a*b;
        i = i + 1;
    }
    return s;
}
```

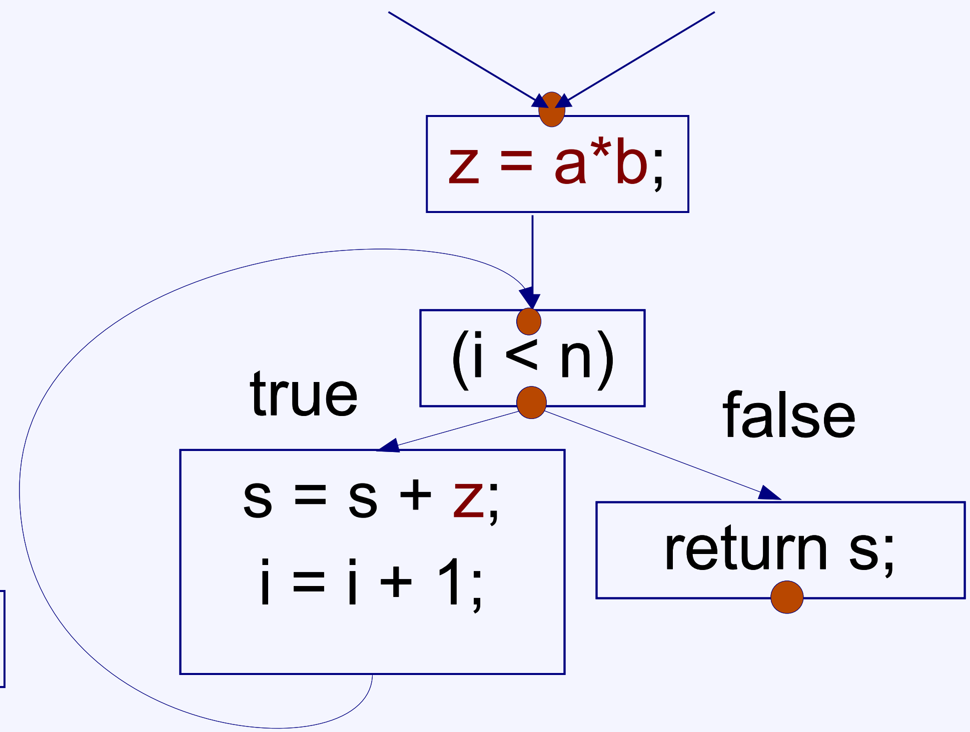
if (k == 0) ~~return~~ 4*n;
  else ~~return~~ 8*n;



Optimalizácia



a*b nezávisí od premennej cyklu. Môžeme vypočítať raz, mimo cyklu.



Zaujímavé body grafu toku riadenia

- Vstup do programu n_0
- Body vetvenia (split)
  - succ(n) = množina všetkých nasledníkov
  - Vzhľadom na definíciu inštrukcie je stupeň týchto bodov vždy dva
- Body spájania (merge)
  - pred(n) = množina všetkých predchodcov
  - Môžu byť ľubovolného stupňa.
- Výstup z programu
  - n_F
  - Eventuálne môžeme mať aj množinu finálnych uzlov

Problémy toku dát

- V čase kompilácie usudzujeme o hodnotách premenných a výrazov v čase behu.
- Vo všetkých zaujímavých bodoch chceme vedieť
  - Z ktorého príkazu priradenia pochádza hodnota premennej v tomto bode ?
  - Ktorá premenná obsahuje hodnoty, ktoré nemajú po tomto bode použitie? (mŕtve premenné)
  - Aké sú možné hodnoty (range) premenných v tomto bode ?

Hlavná myšlienka

- Reprezentovať požadovanú informáciu hodnotami z úplného zväzu $\mathcal{L} = \langle L, \sqcup, \sqcap, \top, \bot; \sqsubseteq \rangle$.
- Popísať transformácie pri vykonávaní programu zväzovými operáciami.
  - Pre každý blok definujeme prenosovú funkciu f_B: $L \rightarrow L$.
- Ukázať, že tieto transformácie sú monotónne.
- Simuláciou behu programu
  - Doprednou (forward), alebo
  - Reverznou (backward)
- Určiť pevný bod
  - Minimálny teoreticky sú možné aj iné pevné body.
- Pre väčšinu úloh je tento zväz priestor booleovských vektorov (hyperkocka) s operáciami a čiastočným usporiadaním po bitoch.

Prenosové funkcie

- Prenosové funkcie charakterizujú účinok bloku B na informáciu o toku dát
- Trieda prenosových funkcií F
  - Musí obsahovať identickú funkciu (efekt prázdneho príkazu)
  - Byť uzavretá vzhľadom na kompozíciu (zložený príkaz) $\forall(f,g\in F)(h = f(g(x))\in F)$
  - Všetky funkcie v F musia byť monotónne $\forall(f\in F)(f(x) \sqsubseteq x)$ v $\forall(f\in F)(x \sqsubseteq f(x))$
    - Poslednú podmienku možno oslabiť na existenciu pevného bodu.
- Často bývajú tieto funkcie distributívne
- Distributívnosť implikuje monotónnosť

  $x \sqsubseteq y$ znamená $x\sqcup y=y$. Teda $f(x\sqcup y)=f(y)$. Podľa distrubutívnosti $f(x\sqcup y)=f(x)\sqcup f(y)$. Teda $f(x)\sqcup f(y)=f(y)$, to znamená $f(x) \sqsubseteq f(y)$.

Dopredná analýza toku dát

- Pre každý uzol n a jemu príslušný blok B_n je definované
  - in_n informácia vchádzajúca do uzlu n.
  - out_n informácia vychádzajúca z uzlu n.
  - f_{B_n} prenosová funkcia bloku B_n.
- Riešenie musí splňovať rovnice: (rovnice toku dát)

$$in_{n_0} = I_0 \qquad \text{precondition}$$

$$\forall n \qquad out_n = f_{B_n}(in_n)$$

$$\forall n \neq n_0 \qquad in_n = \coprod_{m \in pred(n)} out_m$$

- Množinu rovníc iterujeme podľa (Tarského) vety o pevnom bode.
- V ďalšom uvedieme efektívnejšie metódy iterácie.

Reverzná analýza toku dát

- Pre každý uzol n a jemu príslušný blok B_n je definované
  - in_n informácia vstupe do uzlu n.
  - out_n informácia na vystupe z uzlu n.
  - rf_{B_n} revezná prenosová funkcia bloku B_n.
- Riešenie musí splňovať rovnice: (rovnice toku dát)

$$\forall n \qquad in_n = rf_{B_n}(out_n)$$

$$\forall n \neq n_f \qquad out_n = \coprod_{m \in succ(n)} in_m$$

$$out_{n_f} = O_f \qquad \text{postcondition}$$

- Rovnice toku dát sa dajú zovšeobecniť aj pre množinu terminálnych bodov
- Vlastne pre každý smer analýzy stačí horná resp. dolná polovica zväzu. Optimalizačné polozväzy.

Rovnice toku dát

- Úlohou kompilátoru je zostaviť rovnice toku dát pre jednotlivé úlohy analýzy toku dát
  - Reaching definitions forward
  - Available expressions forward
  - Live variables backward
  - Use-definition chain: ku každému použitiu premennej pridávame zoznam všetkých premenných, ktoré „reaches it“.
  - Definition-use chain: naopak ku každej definícii zoznam miest programu, kde je použitá.
- Iné atypické problémy
  - Sign analysis (znamienková analýza)
- Riešenie rovníc je delegované na samostatný program, od ktorého si optimalizátor prevezme výsledky.
  - Presnejšie kladie mu dotazy

Metódy riešenia rovníc toku dát

- Naívna iterácia – round robin (A. Kildall. 1971)

Initilize: **for all** n **do** {out_n:= ⊥; in_n:= ⊥;}

$in_{n_0} = I_0$; /* alebo $out_{n_f} = O_f$; */

Iterate: **repeat**

for all equations **do** compute equation;

until change occured;

- Tento algoritmus opakuje mnohé výpočty zbytočne
- Efektívnejšiu implementáciu dosiahneme, keď si počas práce algoritmu budeme udržovať pracovný zoznam (*worklist*) uzlov, ktorých sa zmeny týkajú.

Worklist algoritmus pre dopredné DFE

for each n **do** $out_n := f_n(\perp)$;

$in_{n_0} := I$; $out_{n_0} := f_{n_0}(I)$;

worklist := N - { n_0 };

while worklist ≠ Ø **do**

{ remove a node n from worklist;

$in_n = \sqcup_{m \in pred(n)} out_m$;

$out_n = f_{B_n}(in_n)$;

if out_n changed **then** worklist := worklist ∪ succ(n);

/* all successors of n must be added to the worklist */

}

- V programe je zamlčaná implementácia worklist. Používa sa zásobník, fronta (queue) a prioritná fronta.
- Štruktúry pre worklist musia byť implementáciou množiny !

Worklist algoritmus pre reverzné DFE

for each n **do** $in_n := f_n(\perp)$

for each $n \in N_{final}$ **do** { $out_n := O$; $in_n := rf_{B_n}(O)$; }

worklist := $N - N_{final}$;

while worklist ≠ Ø **do**

{ remove a node n from worklist;

$out_n = \coprod_{m \in succ(n)} in_m$;

$in_n = rf_{B_n}(out_n)$;

if in_n changed **then** worklist := worklist $\cup$ pred(n);

/* all predecessors of n must be added to the worklist */

}

- V tomto prípadade program zohľadňuje možnosť viacerých terminálnych uzlov.

Správnosť algoritmu

- Konštrukcia algoritmu zabezpečuje, že:
  - Pre každý uzol platí vzťah medzi vstupom a výstupom (dopredne alebo spätne).
  - Ak sa pri výpočte uzlu zmenia hodnoty, sa jeho následníci (predchodcovia pri reverznom spracovaní) dostanú znova do zoznamu spracovania worklist. Pri ich vybraní zo zonamu sa prepočítajú ich hodnoty.
- Znamená to, že program môže skončiť, iba ak sú splnené rovnice toku dát.
- Skončenie:
  - Každá rovnica pre uzol je monotónna. To zaručuje, že niekedy nastane prípad, že hodnota v uzle sa už pri ďalších výpočtoch nebude meniť. Podľa Tarského vety.
  - Každy uzol sa raz stabilizuje. Keď sa stabilizujú všetky uzly, výpočet skončí.

$T_1 - T_2$ redukcia grafu

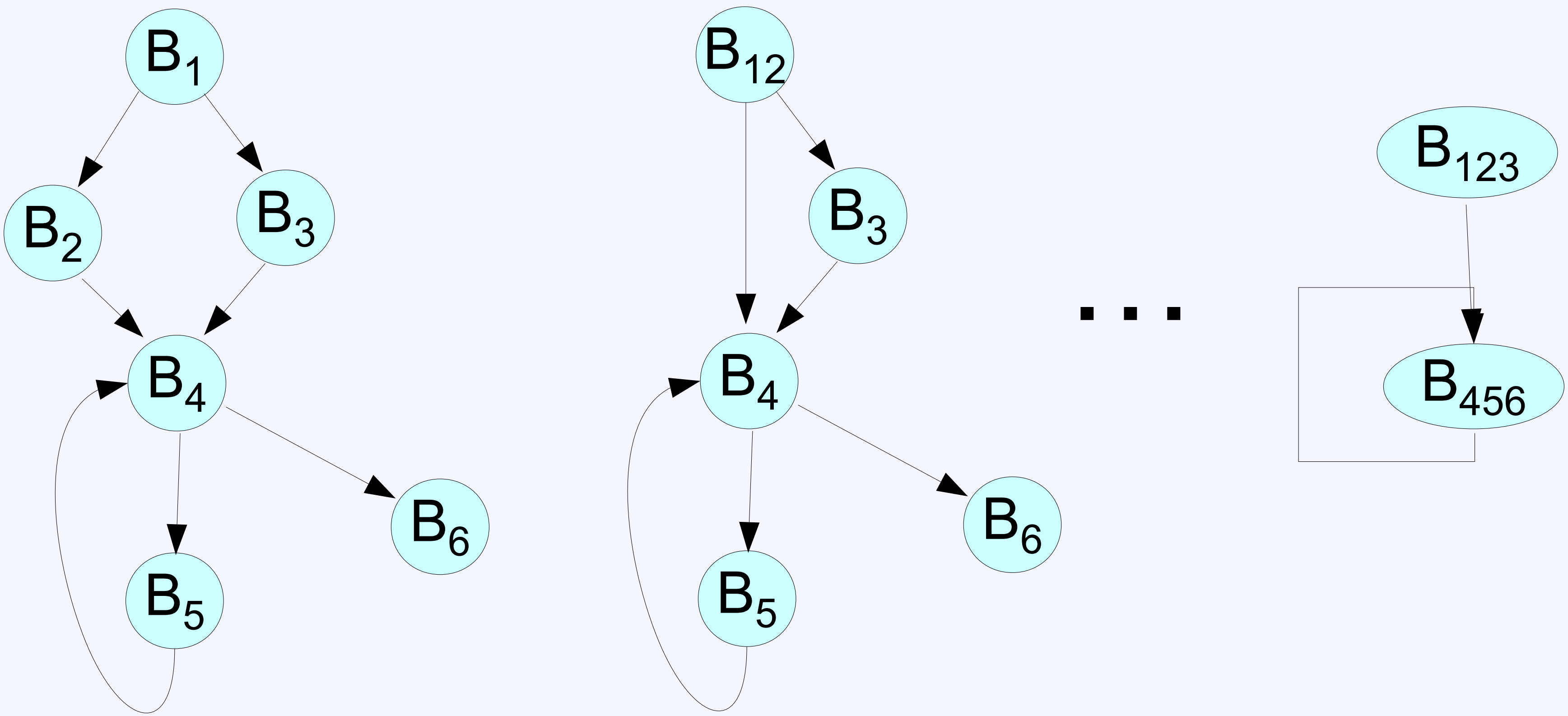
- Nech je daný orientovaný graf G=<N, E>, uvažujme transformácie (operácie):

T_1: Ak hrana $e \in E$ je slučka, potom ju odstráň.

T_2: Ak vrchol n má jediného predchodcu vrchol p. Potom zlúč n a p do jedného vrcholu (np). Hranu $p \rightarrow n$ zrušíme. Pričom succ(np) = succ(n) $\cup$ succ(p). Zjednotenie vychádzajúcich hrán. pred(np) = pred(p).

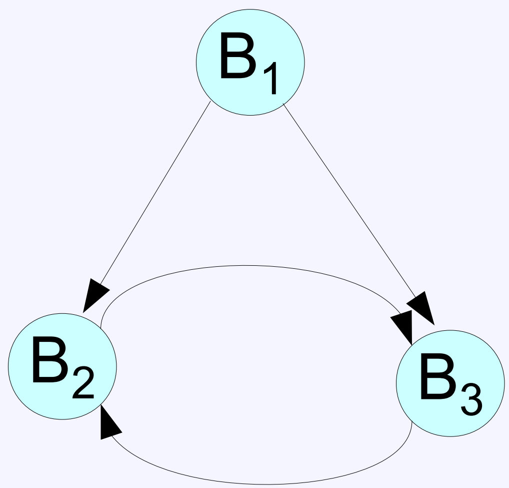
- Opakované použitie $T_1 - T_2$ transformácie.
- Poradie operácií: Keď je možné T_1 nerobíme T_2.
  - Ak je možných viac T_1 operácií (počiatočný graf môže mať slučky) na ich poradí nezáleží.
  - Na poradí T_2 operácií nezáleží.

Príklad

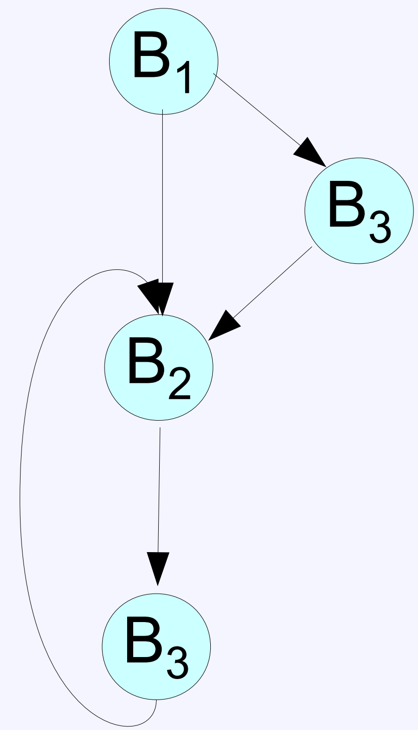


Dva možné konce $T_1 - T_2$ redukcie

- Jediný vrchol – redukovateľný graf
- Graf homomorfný skoku do cyklu – neredukovateľný graf



Je to vlastne jediné neštruktúrovateľné použitie skoku. Dá sa opraviť opakovaním jedného z blokov cyklu.



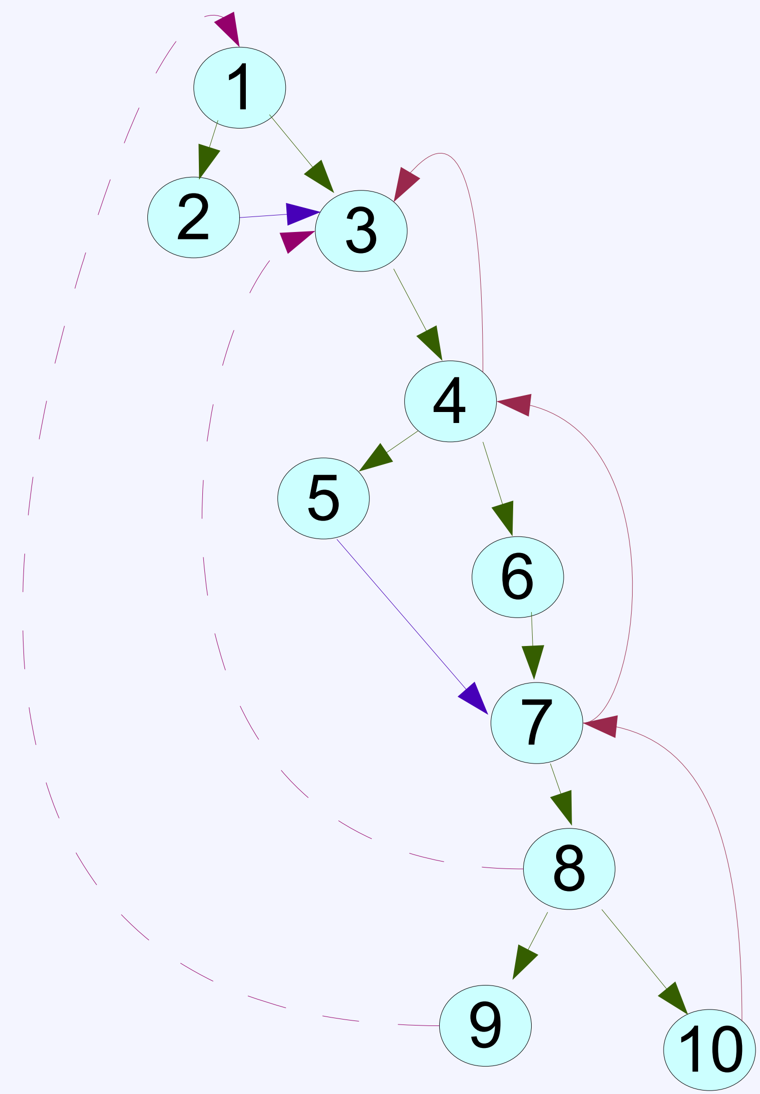
Je to aj odpoveď na otázku:
Čo stojí štruktúrované programovanie ?
Najhorší prípad B_1 je jeden príkaz a B_2 a B_3 sú rovnaké. 3/2 násobné zväčšenie programu.

Riadenie výpočtu redukciou grafu

- Pri každej transformácii T_2 spojíme dva uzly. To zodpovedá približne zloženiu dvoch blokov eventuálne nejaké premenné „vypadnú“.
- Pri transformácii T_1 „stabilizujeme cyklus“
- Pre redukovateľné grafy tak dostaneme efektívnejší algoritmus.
- Nevýhody tohto prístupu
  - Eliminácia závisí na na množine prenosových funkcií F.
  - Efektívnosť závisí na výbere poradia transformácií.
  - Nefunguje pre neredukovateľné grafy.
- Väčšina grafov toku riadenia je redukovateľná
- O reverzných grafoch (s obrátenou orientáciou hrán) to neplatí.

Depth-first kostra dept-first číslovanie grafu

```
procedure dfs(n);
{ mark(n);  //  visited
   for each successor s of n do
     if s not marked then
        { add edge n → s to T;
          dfs(s);                          }
   dfn[n]:= i;
   i:= i – 1;
}  // main program follows
{ i:= 0;
  for each node n of G do
     { unmark(n);  // unvisited
       i:= i + 1;                          }
  dfs(n_0);
}
```



Vlastnosti redukovateľných grafov

- Hrany grafu sú rozdelené do troch tried:
  1) Kostrové – dopredné (zelené)
  2) Spätné (bordové) vedú ku svojmu predchodcovi (nemusí byť ani vlastný, ani bezprostredný).
  3) Priečné (modré)
- Hrana n $\rightarrow$ p spätná práve vtedy, keď dfn(p) $\leq$ dfn(n).
- Hĺbka grafu G *depth(G)* je maximálny možný počet spätných hrán v acyklickej ceste.
- Hĺbka grafu v redukovateľnom grafe nikdy nie je väčšia, ako maximálny počet vnorených cyklov v tomto grafe.
- Počet iterácií pri výpočte pevného bodu redukovateľného grafu T_1 – T_2 transformáciami je *depth(G)+3.*

Úlohy analýzy toku dát

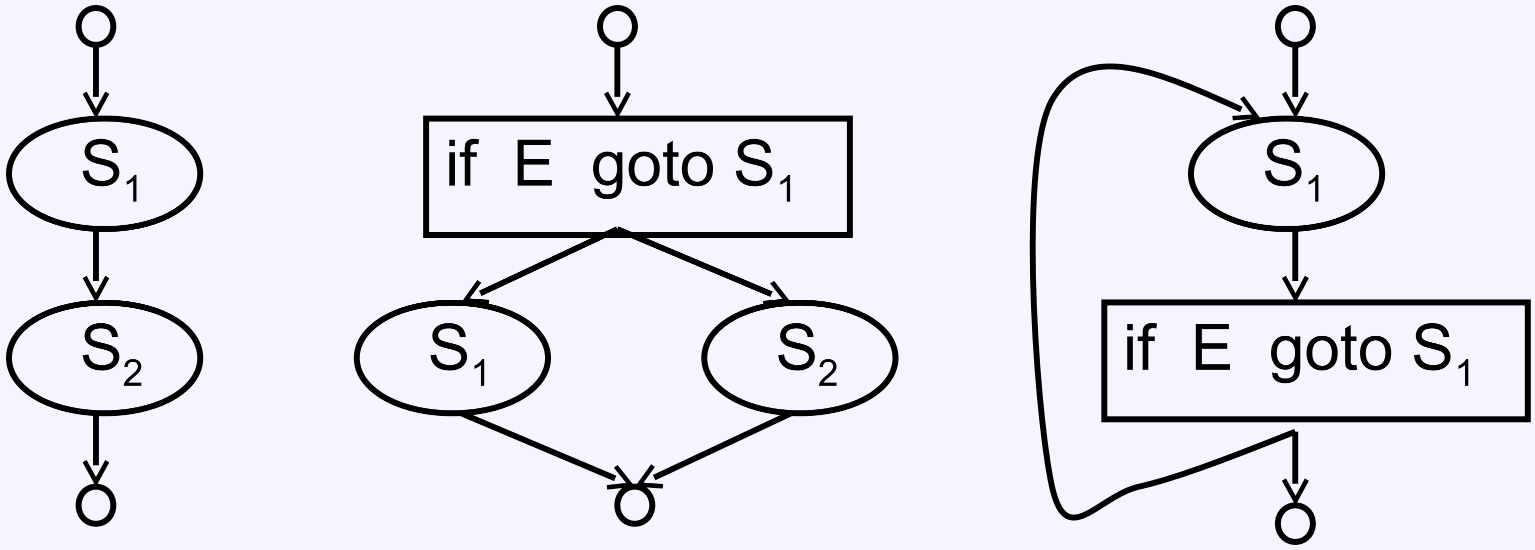
- Na špecifikáciu úlohy analýzy toku riadenia treba určiť:
  1. Doménu (obor definície)
  2. Optimalizačný polozväz
  3. Prenosové funkcie
  4. Inicializáciu
- Množina rovníc je už určená grafom toku riadenia a predošlými špecifikáciami.

Platnosť priradení

- Definícia premennej x je priraďovací príkaz x:= … .
- Priradenie môže byť zrejmé (unambigous)
  - Priraďovací príkaz x:= … .
  - Načítanie do premennej x: read(x), get(x), … .
- Skryté (ambigous) priradenia:
  - Priradenie cez smerník *y:= … (alebo priradenie prvku poľa).
  - Volanie procedúry, ktorej argumenty sú volané referenciou.
  - Volanie procedúry používajúcej nelokálne premenné (side efect).
- Definícia *d* premennej *x* z bodu *p* programu môže platiť (reaches) v bode *n* programu. Ak existuje cesta z *p* do *n*, po ktorej sa nevyskytuje žiadné zrejmé priradenie premennej x. Reaching definitions.
- Definícia *d* premennej *x* z bodu *p* programu musí platiť (is available) v bode *n* programu. Ak na žiadnej ceste z *p* do *n*, sa nevyskytuje žiadné (ani skryté) priradenie premennej x. Available expressions.

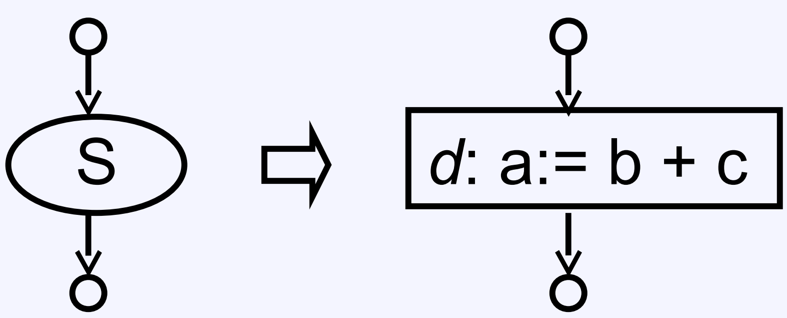
Syntaxou riadeným prekladom

$$
\begin{aligned}
S \rightarrow\ & \mathbf{id}\ \text{‘:=’}\ E \\
|\ & S_1\ \text{‘;’}\ S_2 \\
|\ & \mathbf{if}\ E\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
|\ & \mathbf{do}\ S\ \mathbf{while}\ E
\end{aligned}
$$



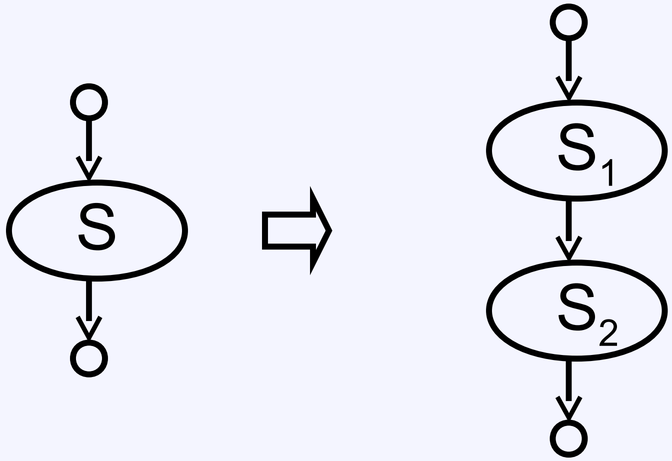
Rovnice toku dát

- V každom príkaze (bloku S):
  - Niečo vznikne (je nejaké priradenie) – gen[S]
  - Eventuálne nejaké priradenie prestane platiť – kill[S]
- Každý príkaz (blok) má atribúty:
  - Zdedený atribút in[S]
  - Syntetizované atribúty out[S], gen[S] a kill[S]
- Platí rovnica (prenosová funkcia) príkazu (bloku):
  out[S] = gen[S] ∪ (in[S] – kill[S])
  - gen[S] obsahuje len posledné priradenia premennej bloku.
  - kill[S] sa vzťahuje len na definície do bloku vchádzajúce.



gen[S] = {*d*}
kill[S] = D_a – {*d*}
out[S] = gen[S] ∪ (in[S] – kill[S])
Kde D_a je množina všetkých doterajších definícií premennej a.

Zložený a podmienený príkaz



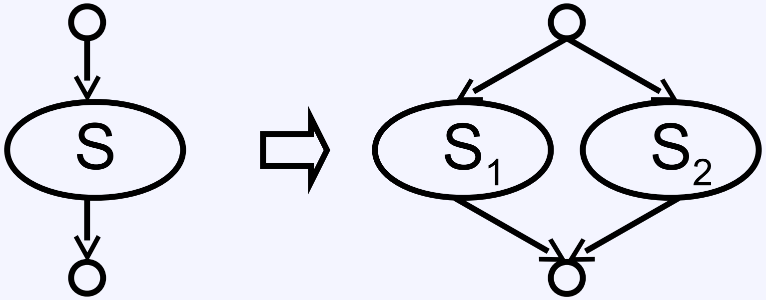
gen[S] = $gen[S_2] \cup (gen[S_1] - kill[S_2])$

kill[S] = $kill[S_2] \cup (kill[S_1] - gen[S_2])$

$in[S_1]$ = in[S]

$in[S_2] = out[S_1]$

out[S] = $out[S_2]$



gen[S] = $gen[S_1] \cup gen[S_2]$

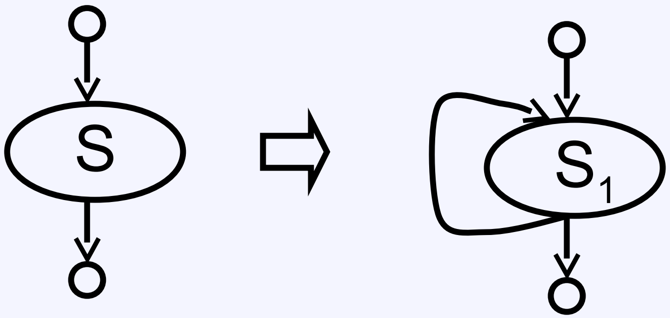
kill[S] = $kill[S_1] \cap kill[S_2]$

$in[S_1]$ = in[S]

$in[S_2]$ = in[S]

out[S] = $out[S_1] \cup out[S_2]$

Cyklus a jeho stabilizácia



$gen[S] = gen[S_1]$
$kill[S] = kill[S_1]$
$in[S_1] = in[S] \cup gen[S_1]$
$out[S] = out[S_1]$

$in[S_1] = in[S] \cup out[S_1]$
$out[S_1] = gen[S_1] \cup (in[S_1] - kill[S_1])$

$i = j \cup o$
$o = g \cup (i - k)$

Predpokladáme $o_0 = Ø$
$i_1 = j \cup o_0 = j$
$o_1 = g \cup (i_1 - k) = g \cup (j - k)$
$i_2 = j \cup o_1 = j \cup g \cup (j - k) = j \cup g$
$o_2 = g \cup (i_2 - k) = g \cup (j \cup g - k) = g \cup (j - k)$

Dátové štruktúry pre výpočet

- Potrebujeme reprezentovať množiny definícií.
  - Vhodnou reprezentáciou je powerset (bitový vektor)
  - Operácie $\cap$ a $\cup$ sú boolovské operácie po bitoch.
- Doména L je množina všetkých podmnožín množiny všetkých definícií v programe.
  - $\sqcup$ je zjednotenie $\cup$
  - $\sqcap$ je prienik $\cap$
  - $\sqsubseteq$ je množinová inklúzia $\subseteq$
  - $\perp$ je prázdna množina $\emptyset$
  - $\top$ je množina všetkých definícii D
- Uvedená štruktúra množina všetkých podmnožín voľakej množiny je boolovský (teda aj úplný) zväz.

Reaching definitions

- L = powerset všetkých definícií v programe (množina všetkých podmnožín množiny všetkých definícií v programe)
- $\sqcup = \cup$ (order is $\subseteq$)
- $\perp = \emptyset$
- $I_0 = in_{n_0} = \perp$
- F = množina všetkých funkcií tvaru $f(x) = gen \cup (x\text{-}kill)$
  - *kill* je množina všetkých definícií, ktoré sú v uzle zrušené
  - *gen* je množina definícií, ktoré uzol generuje.
- Úloha je dopredná.

Available expressions

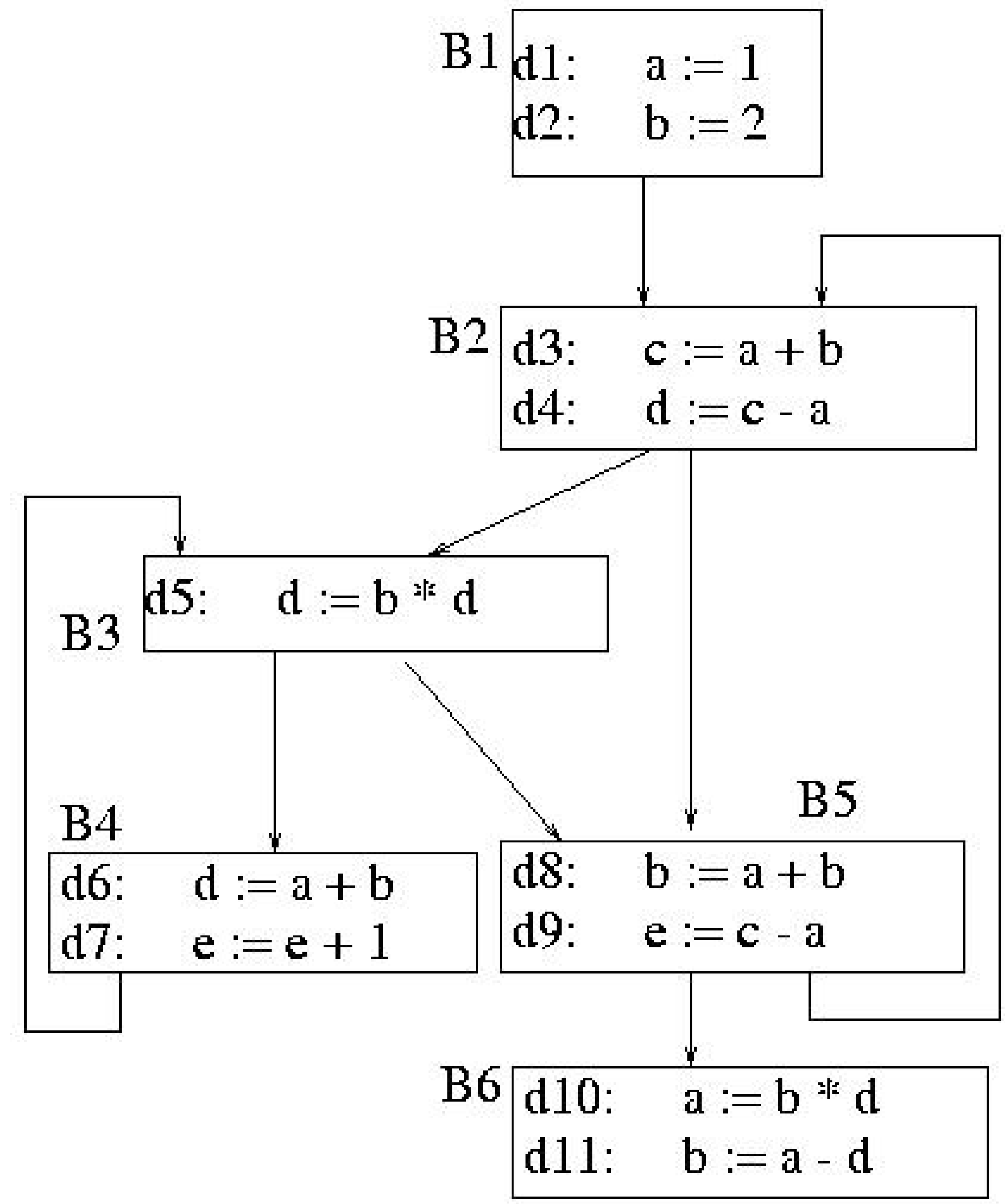
- L = powerset všetkých definícií v programe (množina všetkých podmnožín množiny všetkých definícií v programe)
- $\sqcup$ = $\cap$ (order is $\supseteq$)
- $\perp = \emptyset$
- $I_0 = in_{n_0} = \perp$
- F = množina všetkých funkcií tvaru $f(x) = gen \cup (x\text{-}kill)$
  - *kill* je množina všetkých definícií, ktoré sú v uzle zrušené
  - *gen* je množina definícií, ktoré uzol generuje.
- Úloha je dopredná.

Živé premenné – live variables

- L = powerset všetkých premenných v programe (množina všetkých podmnožín množiny všetkých premenných v programe)
- $\sqcup = \cup$ (order is $\subseteq$)
- $\perp = \emptyset$
- $I_0 = in_{n_0} = \perp$
- F = množina všetkých funkcií tvaru $f(x) = gen \cup (x\text{-}kill)$
  - *kill* je množina premenných, ktorým uzol priraďuje hodnotu skôr ako ich použije
  - *gen* je množina premenných, ktoré uzol používa skôr ako ich definuje
- Úloha je reverzná.

Príklad

Block	DEF	USE
B1	{a,b}	{ }
B2	{c,d}	{a,b}
B3	{ }	{b,d}
B4	{d}	{a,b,e}
B5	{e}	{a,b,c}
B6	{a}	{b,d}



DFA úlohy z dračej knihy 1

	Reaching Definitions	Live Variables	Available Expressions
Domain	Sets of definitions	Sets of variables	Sets of expressions
Direction	Forwards	Backwards	Forwards
Transfer function	$gen_B \cup (x - kill_B)$	$use_B \cup (x - def_B)$	$e\_gen_B \cup (x - e\_kill_B)$
Boundary	$\text{OUT}[\text{ENTRY}] = \emptyset$	$\text{IN}[\text{EXIT}] = \emptyset$	$\text{OUT}[\text{ENTRY}] = \emptyset$
Meet ($\wedge$)	$\cup$	$\cup$	$\cap$
Equations	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigwedge_{P,pred(B)} \text{OUT}[P]$	$\text{IN}[B] = f_B(\text{OUT}[B])$ $\text{OUT}[B] = \bigwedge_{S,succ(B)} \text{IN}[S]$	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigwedge_{P,pred(B)} \text{OUT}[P]$
Initialize	$\text{OUT}[B] = \emptyset$	$\text{IN}[B] = \emptyset$	$\text{OUT}[B] = U$

Pozn: Ullman všetky úlohy formuluje pre horný polozväz a hladá minimálny pevný bod.
Duálny problém dolný polozväz a maximálny pevný bod.
Úplný zväz potrebujeme len pre obojsmerné problémy.

DFA úlohy z dračej knihy 2

	(a) Anticipated Expressions	(b) Available Expressions
Domain	Sets of expressions	Sets of expressions
Direction	Backwards	Forwards
Transfer function	$f_B(x) =$ $e\_use_B \cup (x - e\_kill_B)$	$f_B(x) =$ $(anticipated[B].in \cup x) - e\_kill_B$
Boundary	$\text{IN}[\text{EXIT}] = \emptyset$	$\text{OUT}[\text{ENTRY}] = \emptyset$
Meet ($\wedge$)	$\cap$	$\cap$
Equations	$\text{IN}[B] = f_B(\text{OUT}[B])$ $\text{OUT}[B] = \bigwedge_{S,succ(B)} \text{IN}[S]$	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigwedge_{P,pred(B)} \text{OUT}[P]$
Initialization	$\text{IN}[B] = U$	$\text{OUT}[B] = U$

Anticipated expressions = very busy expressions

DFA úlohy z dračej knihy 3

	(c) Postponable Expressions	(d) Used Expressions
Domain	Sets of expressions	Sets of expressions
Direction	Forwards	Backwards
Transfer function	$f_B(x) =$ $(earliest[B] \cup x) - e\_use_B$	$f_B(x) =$ $(e\_use_B \cup x) - latest[B])$
Boundary	$\text{OUT}[\text{ENTRY}] = \emptyset$	$\text{IN}[\text{EXIT}] = \emptyset$
Meet ($\wedge$)	$\cap$	$\cup$
Equations	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigwedge_{P,pred(B)} \text{OUT}[P]$	$\text{IN}[B] = f_B(\text{OUT}[B])$ $\text{OUT}[B] = \bigwedge_{S,succ(B)} \text{IN}[S]$
Initialization	$\text{OUT}[B] = U$	$\text{IN}[B] = \emptyset$

earliest[B] = anticipated[B].in – available[B].in

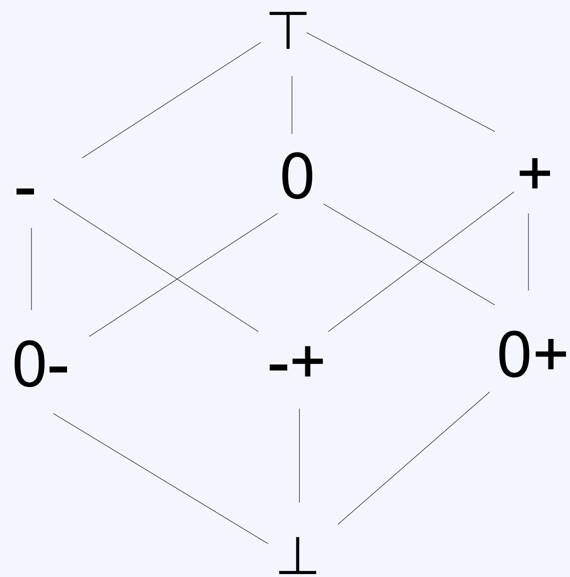
latest[B] = (earliest[B] ∪ postponable[B].in) ∩ $(e\_use_B \cup \neg(\bigcap_{S \in succ(B)}(earliest[S] \cup postponable[S].in)))$

Atypické DFA úlohy – sign analysis

- Zaujímame sa len o znamienká premenných a výrazov.

Výpočtový zväz

Dolný polozväz.
Dopredná úloha.
⊤ je chybová hodnota napr. delenie nulou.



Tabuľka násobenia

×	⊥	0-	-+	0+	-	0	+	⊤
⊥	⊥	⊥	⊥	⊥	⊥	0	⊥	⊤
0-	⊥	0+	⊥	0-	0+	0	0-	⊤
-+	⊥	⊥	-+	⊥	-+	0	-+	⊤
0+	⊥	0-	⊥	0+	0-	0	0+	⊤
-	⊥	0+	-+	0-	+	0	-	⊤
0	0	0	0	0	0	0	0	⊤
+	⊥	0-	-+	0+	-	0	+	⊤
⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤

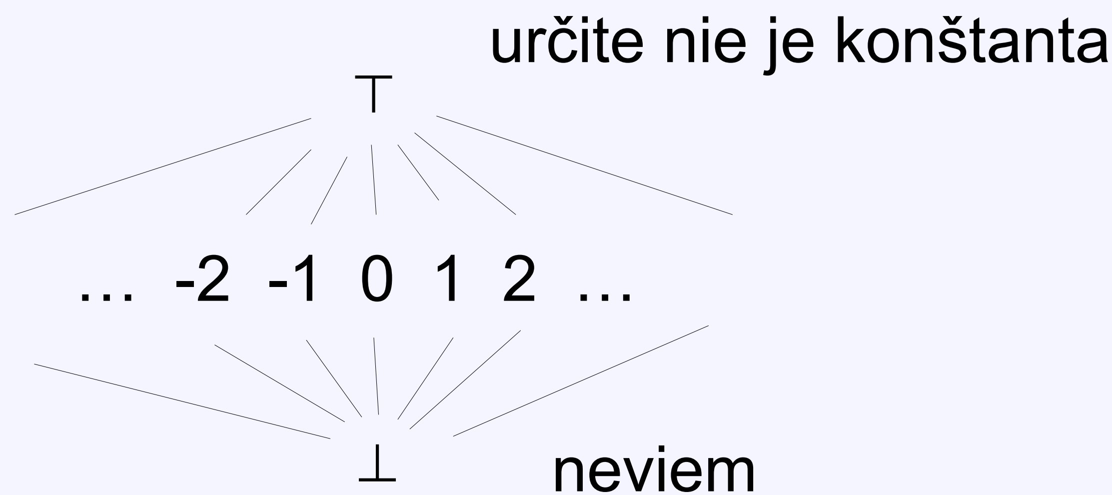
Modus operandi:
Výrazy v blokoch vyhodnocujeme podľa tabuliek operácii.
Výrazy, ktoré vznikli po rôznych cestách podľa zväzových operácií.

Prenosové funkcie:

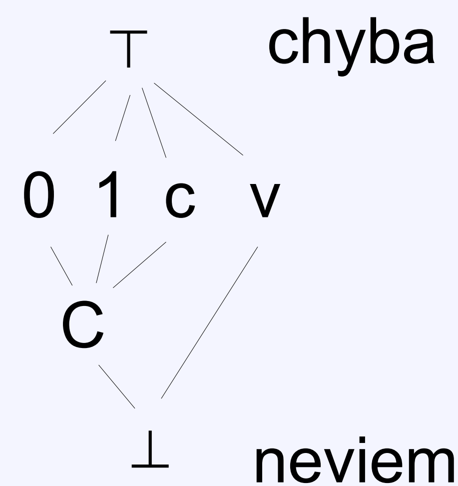
Na počiatku každá premenná má hodnotu ⊥.
Príkaz v:= c priradí premennej v jednu z hodnôt { -, 0, +}, podľa hodnoty konštanty c.
Priraďovací v:= a op b. Určí hodnotu premennej v z hodnôt operandov podľa tabuľky pre operáciu op.

Constant folding, constant propagation

- Používa sa:



- Skúsil by som:



Dá sa využiť, že:
$0 \times v = 0$
$a / a = 1$

Nevýhoda: Treba vzlášť sledovať aj rovnosť konštánt po cestách.