Script generated by TTT

Title: Seidl: Programmoptimierung (17.12.2012)

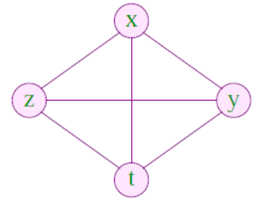
Date: Mon Dec 17 14:01:39 CET 2012

Duration: 91:15 min

Pages: 48

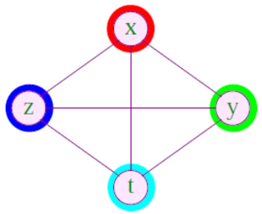
Special Case: Basic Blocks

	$\mathcal{L}$
	x, y, z
$A_1 = x + y;$	x, z
$M[A_1] = z;$	x
$x = x + 1;$	x
$z = M[A_1];$	x, z
$t = M[x];$	x, z, t
$A_2 = x + t;$	x, z, t
$M[A_2] = z;$	x, t
$y = M[x];$	y, t
$M[y] = t;$	



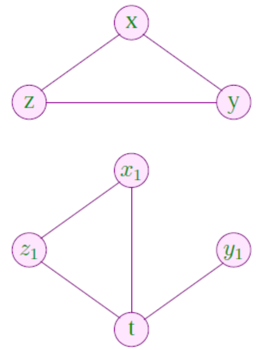
Special Case: Basic Blocks

	$\mathcal{L}$
	x, y, z
$A_1 = x + y;$	x, z
$M[A_1] = z;$	x
$x = x + 1;$	x
$z = M[A_1];$	x, z
$t = M[x];$	x, z, t
$A_2 = x + t;$	x, z, t
$M[A_2] = z;$	x, t
$y = M[x];$	y, t
$M[y] = t;$	



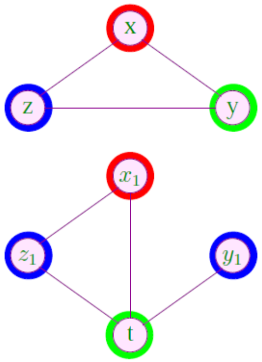
The live ranges of x and z can be split:

	$\mathcal{L}$
	x, y, z
$A_1 = x + y;$	x, z
$M[A_1] = z;$	x
$x_1 = x + 1;$	x_1
$z_1 = M[A_1];$	x_1, z_1
$t = M[x_1];$	x_1, z_1, t
$A_2 = x_1 + t;$	x_1, z_1, t
$M[A_2] = z_1;$	x_1, t
$y_1 = M[x_1];$	y_1, t
$M[y_1] = t;$	

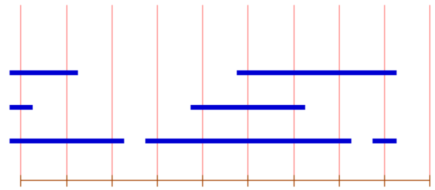


The live ranges of x and z can be split:

	$\mathcal{L}$
	x, y, z
$A_1 = x + y;$	x, z
$M[A_1] = z;$	x
$x_1 = x + 1;$	x_1
$z_1 = M[A_1];$	x_1, z_1
$t = M[x_1];$	x_1, z_1, t
$A_2 = x_1 + t;$	x_1, z_1, t
$M[A_2] = z_1;$	x_1, t
$y_1 = M[x_1];$	y_1, t
$M[y_1] = t;$	



Interference graphs for minimal live ranges on basic blocks are known as interval graphs:

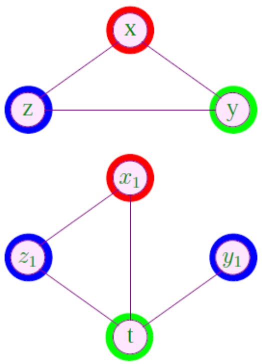


vertex = interval

edge = joint vertex

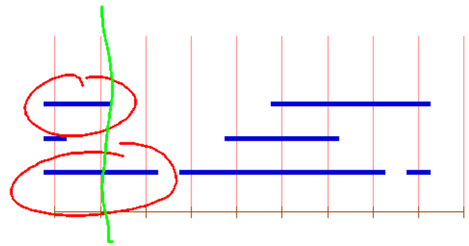
The live ranges of x and z can be split:

	$\mathcal{L}$
	x, y, z
$A_1 = x + y;$	x, z
$M[A_1] = z;$	x
$x_1 = x + 1;$	x_1
$z_1 = M[A_1];$	x_1, z_1
$t = M[x_1];$	x_1, z_1, t
$A_2 = x_1 + t;$	x_1, z_1, t
$M[A_2] = z_1;$	x_1, t
$y_1 = M[x_1];$	y_1, t
$M[y_1] = t;$	



603

Interference graphs for minimal live ranges on basic blocks are known as interval graphs:

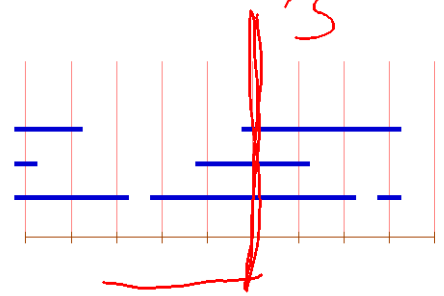


vertex = interval

edge = joint vertex

604

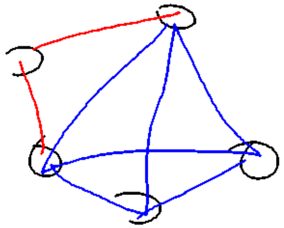
Interference graphs for minimal live ranges on basic blocks are known as interval graphs:



vertex $=$ interval

edge $=$ joint vertex

The covering number of a vertex is given by the number of incident intervals.



Theorem:

maximal covering number

$=$ size of the maximal clique

$=$ minimally necessary number of colors :-)

Graphs with this property (for every sub-graph) are called perfect ...

A minimal coloring can be found in polynomial time :-))

Idea:

- → Conceptually iterate over the vertices $0, \ldots, m-1$!
- → Maintain a list of currently free colors.
- → If an interval starts, allocate the next free color.
- → If an interval ends, free its color.

This results in the following algorithm:


```
free = [1, ..., k];
for (i = 0; i < m; i++) {
    init[i] = []; exit[i] = [];
}
forall (I = [u, v] ∈ Intervals) {
    init[u] = (I :: init[u]); exit[v] = (I :: exit[v]);
}
for (i = 0; i < m; i++) {
    forall (I ∈ init[i]) {
        color[I] = hd free; free = tl free;
    }
    forall (I ∈ exit[i]) free = color[I] :: free;
}
```

```
free = [1, ..., k];
for (i = 0; i < m; i++) {
    init[i] = []; exit[i] = [];
}
forall (I = [u, v] ∈ Intervals) {
    init[u] = (I :: init[u]); exit[v] = (I :: exit[v]);
}
for (i = 0; i < m; i++) {
    forall (I ∈ init[i]) {
        color[I] = hd free; free = tl free;
    }
    forall (I ∈ exit[i]) free = color[I] :: free;
}
```


Discussion:

→ For arbitrary programs, we thus may apply some heuristics for graph coloring ...

→ If the number of real register does not suffice, the remaining variables are spilled into a fixed area on the stack.

→ Generally, variables from inner loops are preferably held in registers.

→ For basic blocks we have succeeded to derive an optimal register allocation :-)

The number of required registers could even be determined before-hand !

→ This works only once live ranges have been split ...

Generalization: Static Single Assignment Form

We proceed in two phases:

Step 1:

Transform the program such that each program point v is reached by at most one definition of a variable x which is live at v.

Step 2:

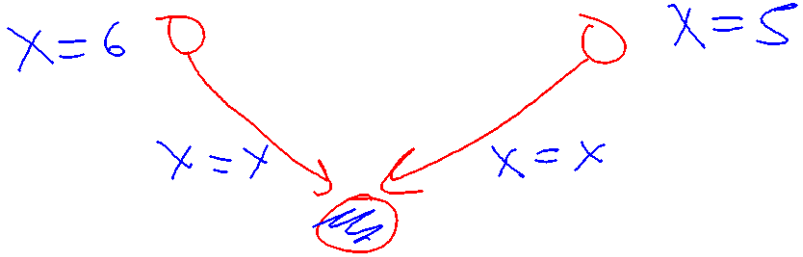
- Introduce a separate variant x_i for every occurrence of a definition of a variable x !
- Replace every use of x with the use of the reaching variant x_h ...

Implementing Step 1:

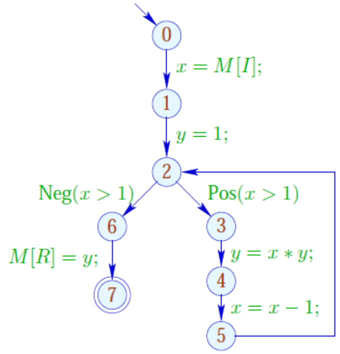
- Determine for every program point the set of reaching definitions.
- If the join point v is reached by more than one definition for the same variable x which is live at program point v, insert definitions $x = x;$ at the end of each incoming edge.

Implementing Step 1:

- Determine for every program point the set of reaching definitions.
- If the join point v is reached by more than one definition for the same variable x which is live at program point v, insert definitions $x = x;$ at the end of each incoming edge.



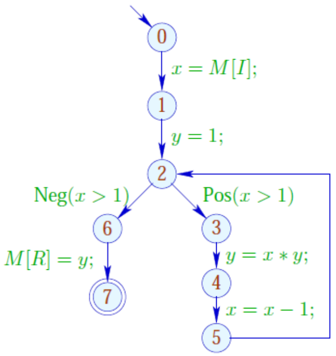
Example



Reaching Definitions

	$\mathcal{R}$
0	$\langle x, 0\rangle, \langle y, 0\rangle$
1	$\langle x, 1\rangle, \langle y, 0\rangle$
2	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
3	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
4	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 4\rangle$
5	$\langle x, 5\rangle, \langle y, 4\rangle$
6	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
7	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

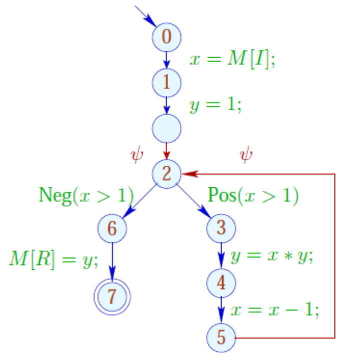
Example



Reaching Definitions

	$\mathcal{R}$
0	$\langle x, 0\rangle, \langle y, 0\rangle$
1	$\langle x, 1\rangle, \langle y, 0\rangle$
2	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
3	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
4	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 4\rangle$
5	$\langle x, 5\rangle, \langle y, 4\rangle$
6	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
7	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

Example



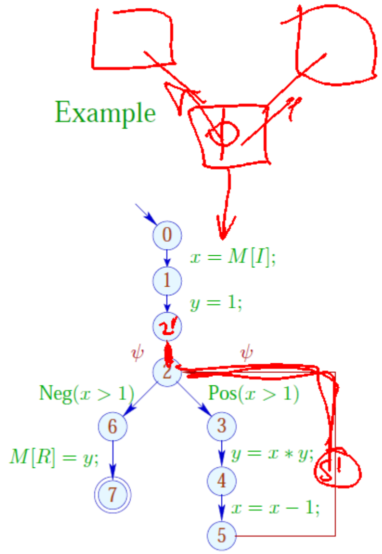
Reaching Definitions

	$\mathcal{R}$
0	$\langle x, 0\rangle, \langle y, 0\rangle$
1	$\langle x, 1\rangle, \langle y, 0\rangle$
2	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
3	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
4	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 4\rangle$
5	$\langle x, 5\rangle, \langle y, 4\rangle$
6	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
7	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

where $\psi \equiv x = x \mid y = y$

(x,2), (y,2)

Example



Reading Definitions

	$\mathcal{R}$
0	$\langle x, 0\rangle, \langle y, 0\rangle$
1	$\langle x, 1\rangle, \langle y, 0\rangle$
2	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
3	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
4	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 4\rangle$
5	$\langle x, 5\rangle, \langle y, 4\rangle$
6	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$
7	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

where $\psi \quad \equiv \quad x = x \mid y = y$

Reaching Definitions

The complete lattice $\mathbb{R}$ for this analysis is given by:

$$\mathbb{R} = 2^{Defs}$$

where

$$Defs = Vars \times Nodes \qquad Defs(x) = \{x\} \times Nodes$$

Then:

$$\begin{aligned}
[\![(_, x = r;, v)]\!]^\sharp R &= R \backslash Defs(x) \cup \{\langle x, v\rangle\} \\
[\![(_, x = x \mid x \in L, v)]\!]^\sharp R &= R \backslash \bigcup_{x \in L} Defs(x) \cup \{\langle x, v\rangle \mid x \in L\}
\end{aligned}$$

The ordering on $\mathbb{R}$ is given by subset inclusion $\subseteq$ where the value at program start is given by $R_0 = \{\langle x, start\rangle \mid x \in Vars\}$.

Reaching Definitions

The complete lattice $\mathbb{R}$ for this analysis is given by:

$$\mathbb{R} = 2^{Defs}$$

where

$$Defs = Vars \times Nodes \qquad Defs(x) = \{x\} \times Nodes$$

Then:

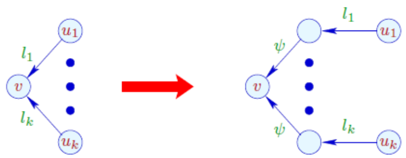
$$\begin{aligned}
[\![(_, x = r;, v)]\!]^\sharp R &= R \backslash Defs(x) \cup \{\langle x, v\rangle\} \\
[\![(_, x = x \mid x \in L, v)]\!]^\sharp R &= R \backslash \bigcup_{x \in L} Defs(x) \cup \{\langle x, v\rangle \mid x \in L\}
\end{aligned}$$

The ordering on $\mathbb{R}$ is given by subset inclusion $\subseteq$ where the value at program start is given by $R_0 = \{\langle x, start\rangle \mid x \in Vars\}$.

Assumption:

No join point is the endpoint of several definitions of the same variable.

The Transformation SSA, Step 1:



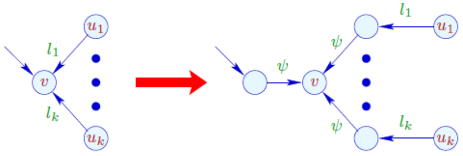
where $k \geq 2$.

The label ψ of the new in-going edges for v is given by:

$$\psi \;\equiv\; \{x = x \mid x \in \mathcal{L}[v], \#(\mathcal{R}[v] \cap Defs(x)) > 1\}$$

If the node v is the start point of the program, we add auxiliary edges whenever there are further ingoing edges into v:

The Transformation SSA, Step 1 (cont.):

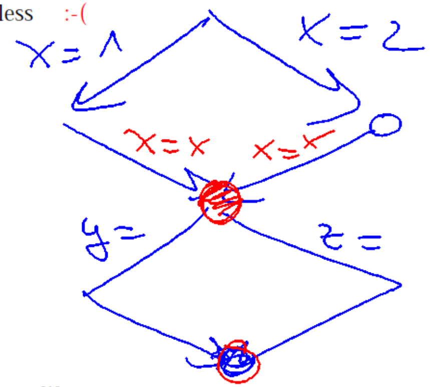


where $k \geq 1$ and ψ of the new in-going edges for v is given by:

$$\psi \equiv \{x = x \mid x \in \mathcal{L}[v], \#(\mathcal{R}[v] \cap \mathit{Defs}(x)) > 1\}$$

Discussion

- Program start is interpreted as (the end point of) a definition of every variable x :-)
- At some edges, parallel definitions ψ are introduced !
- Some of them may be useless :-(



Discussion

- Program start is interpreted as (the end point of) a definition of every variable x :-)
- At some edges, parallel definitions ψ are introduced !
- Some of them may be useless :-(

Improvement:

- We introduce assignments $x = x$ before v only if the sets of reaching definitions for x at incoming edges of v differ !
- This introduction is repeated until every v is reached by exactly one definition for each variable live at v.

Theorem

Assume that every program point in the controlflow graph is reachable from start and that every left-hand side of a definition is live. Then:

1. The algorithm for inserting definitions $x = x$ terminates after at most $n \cdot (m + 1)$ rounds were m is the number of program points with more than one in-going edges and n is the number of variables.
2. After termination, for every program point u, the set $\mathcal{R}[u]$ has exactly one definition for every variable x which is live at u.

Theorem

Assume that every program point in the controlflow graph is reachable from start and that every left-hand side of a definition is live. Then:

1. The algorithm for inserting definitions $x = x$ terminates after at most $n \cdot (m+1)$ rounds were m is the number of program points with more than one in-going edges and n is the number of variables.
2. After termination, for every program point u, the set $\mathcal{R}[u]$ has exactly one definition for every variable x which is live at u.

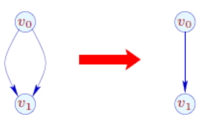
Discussion

The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

Discussion

The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

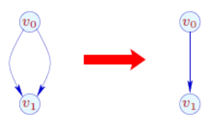
A well-structured cfg can be reduced to a single vertex or edge by:



Discussion

The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

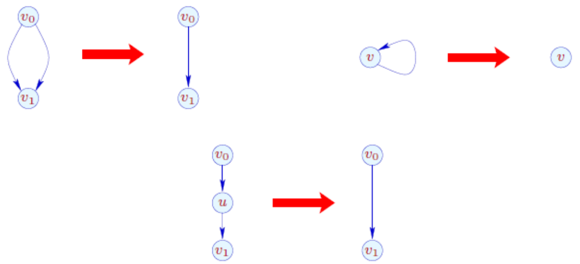
A well-structured cfg can be reduced to a single vertex or edge by:

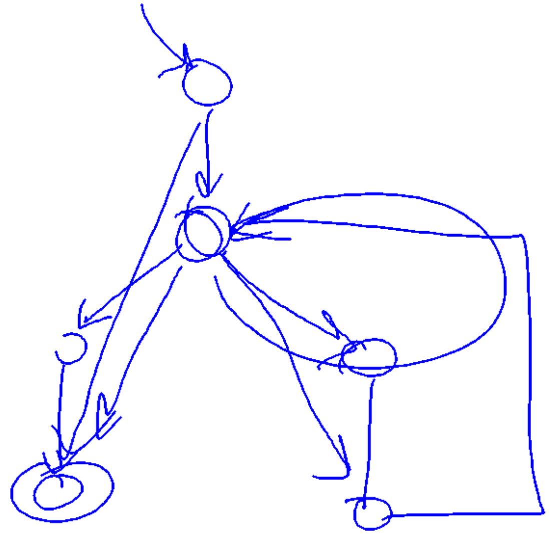


Discussion

The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

A well-structured cfg can be reduced to a single vertex or edge by:

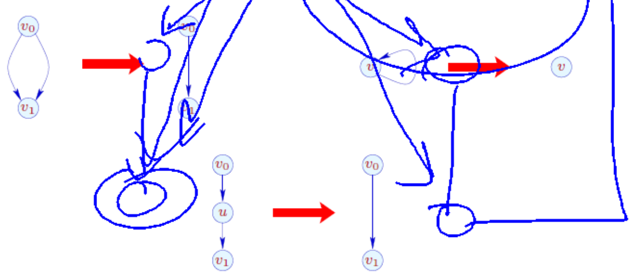




Discussion

The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

A well-structured cfg can be reduced to a single vertex or edge by:



Discussion (cont.)

- Reducible cfgs are not the exception — but the rule :-)
- In Java, reducibility is only violated by loops with breaks/continues.
- If the insertion of definitions does not terminate after k iterations, we may immediately terminate the procedure by inserting definitions $x = x$ before all nodes which are reached by more than one definition of x.

Assume now that every program point u is reached by exactly one definition for each variable which is live at u ...

The Transformation SSA, Step 2:

Each edge (u, lab, v) is replaced with $(u, \mathcal{T}_{v,\phi}[lab], v)$ where $\phi\, x = x_{u'}$ if $\langle x, u' \rangle \in \mathcal{R}[u]$ and:

$$\begin{array}{lll}
\mathcal{T}_{v,\phi}[\,;\,] & = & ; \\
\mathcal{T}_{v,\phi}[\mathsf{Neg}(e)] & = & \mathsf{Neg}(\phi(e)) \\
\mathcal{T}_{v,\phi}[\mathsf{Pos}(e)] & = & \mathsf{Pos}(\phi(e)) \\
\mathcal{T}_{v,\phi}[x = e] & = & x_v = \phi(e) \\
\mathcal{T}_{v,\phi}[x = M[e]] & = & x_v = M[\phi(e)] \\
\mathcal{T}_{v,\phi}[M[e_1] = e_2] & = & M[\phi(e_1)] = \phi(e_2)] \\
\mathcal{T}_{v,\phi}[\{x = x \mid x \in L\}] & = & \{x_v = \phi(x) \mid x \in L\}
\end{array}$$

Remark

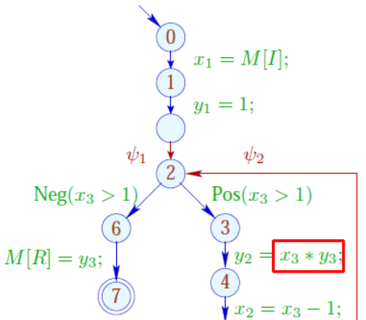
The multiple assignments:

$$pa = x_v^{(1)} = x_{v_1}^{(1)} \mid \ldots \mid x_v^{(k)} = x_{v_k}^{(k)}$$

in the last row are thought to be executed in parallel, i.e.,

$$[\![pa]\!]\,(\rho, \mu) = (\rho \oplus \{{x^{(i)}}_v \mapsto \rho({x^{(i)}}_{v_i}) \mid i = 1, \ldots, k\}, \mu)$$

Example



$$\begin{array}{lll}
\psi_1 & = & x_3 = x_1 \mid y_3 = y_1 \\
\psi_2 & = & x_3 = x_2 \mid y_3 = y_2
\end{array}$$

Theorem

Assume that every $\boxed{\text{program}}$ point is reachable from start and the program is in SSA form without assignments to dead variables.

Let λ denote the maximal number of simultaneously live variables and G the interference graph of the program variables. Then:

$$\lambda = \omega(G) = \chi(G)$$

where $\omega(G), \chi(G)$ are the maximal size of a clique in G and the minimal number of colors for G, respectively.

A minimal coloring of G, i.e., an optimal register allocation can be found in polynomial time.

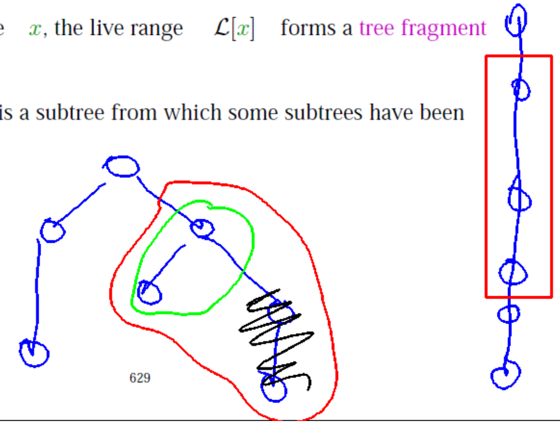
Discussion

- By the theorem, the number λ of required registers can be easily computed :-)
- Thus variables which are to be spilled to memory, can be determined ahead of the subsequent assignment of registers !
- Thus here, we may, e.g., insist on keeping iteration variables from inner loops.
- Clearly, always $\lambda \leq \omega(G) \leq \chi(G)$:-)

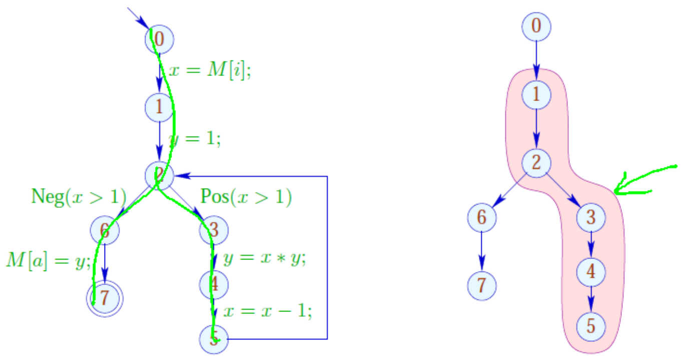
  Therefore, it suffices to color the interference graph with λ colors.
- Instead, we provide an algorithm which directly operates on the cfg ...

Observation

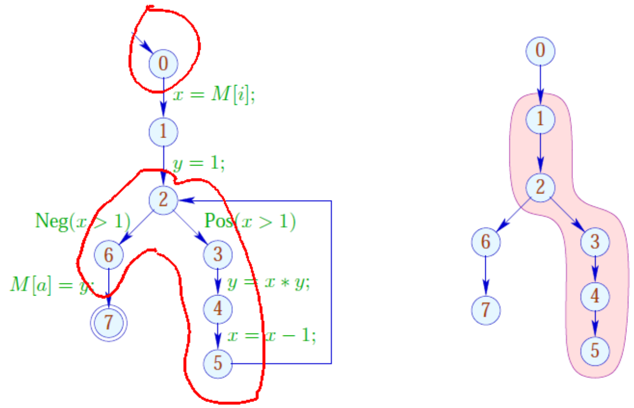
- Live ranges of variables in programs in SSA form behave similar to live ranges in basic blocks !
- Consider some dfs spanning tree T of the cfg with root start.
- For each variable x, the live range $\mathcal{L}[x]$ forms a tree fragment of T !
- A tree fragment is a subtree from which some subtrees have been removed ...



Example



Example



Discussion

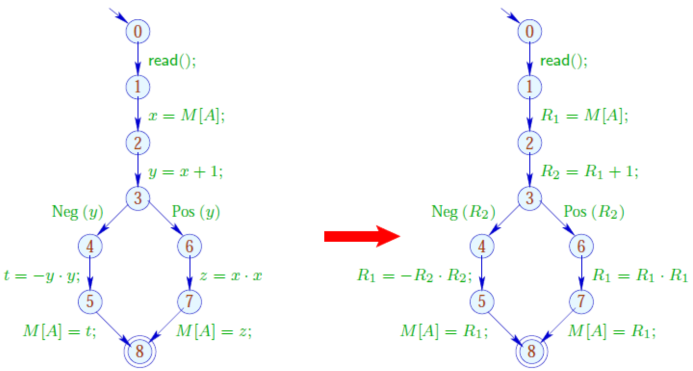
- Although the example program is not in SSA form, all live ranges still form tree fragments :-)
- The intersection of tree fragments is again a tree fragment !
- A set C of tree fragments forms a clique iff their intersection is non-empty !!!
- The greedy algorithm will find an optimal coloring ...

The Greedy Algorithm

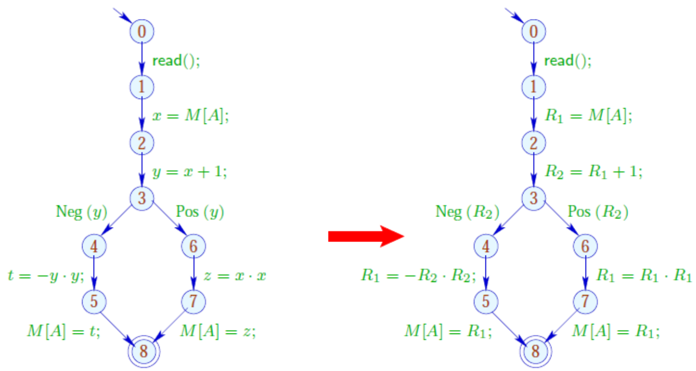
forall $(u \in \mathit{Nodes})$ $\mathit{visited}[u] = \mathbf{false}$;
forall $(x \in \mathcal{L}[\mathit{start}])$ $\Gamma(x) = \mathsf{extract}(\mathit{free})$;
$\mathsf{alloc}(\mathit{start})$;

void alloc (**Node** u) {
  $\mathit{visited}[u] = \mathbf{true}$;
  forall $((\mathit{lab}, v) \in \mathit{edges}[u])$
    if $(\neg \mathit{visited}[v])$ {
      forall $(x \in \mathcal{L}[u] \backslash \mathcal{L}[v])$ $\mathsf{insert}(\mathit{free}, \Gamma(x))$;
      forall $(x \in \mathcal{L}[v] \backslash \mathcal{L}[u])$ $\Gamma(x) = \mathsf{extract}(\mathit{free})$;
      $\mathsf{alloc}(v)$;
    }
}

Example

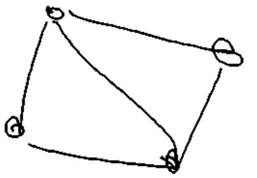


Example



635

Remark:



- Intersection graphs for tree fragments are also known as cordal graphs ...
- A cordal graph is an undirected graph where every cycle with more than three nodes contains a cord :-)
- Cordal graphs are another sub-class of perfect graphs :-))

- Cheap register allocation comes at a price:

  when transforming into SSA form, we have introduced parallel register-register moves :-(