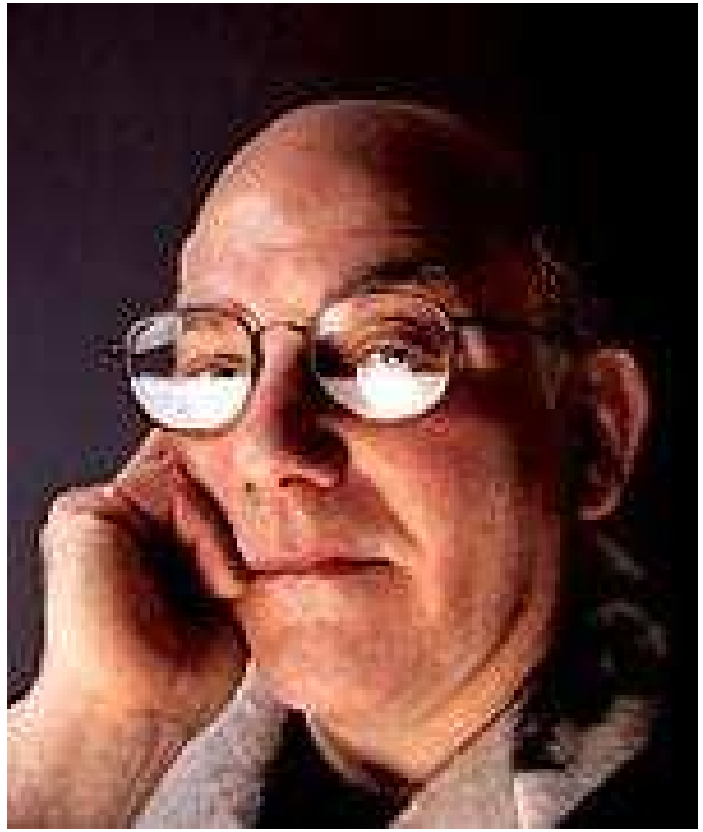
Sviatoslav Sergeevich Lavrov,
Russian Academy of Sciences (1962)



Gregory J. Chaitin, University of Maine (1981)

Abstract Problem:

Given: Undirected Graph (V, E) .

Wanted: Minimal coloring, i.e., mapping $c : V \to \mathbb{N}$ mit

(1) $c(u) \neq c(v)$ for $\{u, v\} \in E$;

(2) $\bigsqcup\{c(u) \mid u \in V\}$ minimal!

- In the example, 3 colors suffice :-) But:
- In general, the minimal coloring is not unique :-(
- It is NP-complete to determine whether there is a coloring with at most k colors :-((

$$\Longrightarrow$$

We must rely on heuristics or special cases :-)

Greedy Heuristics:

- Start somewhere with color 1;
- Next choose the smallest color which is different from the colors of all already colored neighbors;
- If a node is colored, color all neighbors which not yet have colors;
- Deal with one component after the other ...

... more concretely:

```
forall (v ∈ V) c[v] = 0;
forall (v ∈ V) color (v);

void color (v) {
    if (c[v] ≠ 0) return;
    neighbors = {u ∈ V | {u,v} ∈ E};
    c[v] = ⊓{k > 0 | ∀ u ∈ neighbors : k ≠ c(u)};
    forall (u ∈ neighbors)
        if (c(u) == 0) color (u);
}
```

The new color can be easily determined once the neighbors are sorted according to their colors :-)

Discussion:

→ Essentially, this is a Pre-order DFS :-)

→ In theory, the result may arbitrarily far from the optimum :-(

→ ... in practice, it may not be as bad :-)

→ ... Warning: differen variants have beenpatented !!!

Discussion:

→ Essentially, this is a Pre-order DFS :-)

→ In theory, the result may arbitrarily far from the optimum :-(

→ ... in practice, it may not be as bad :-)

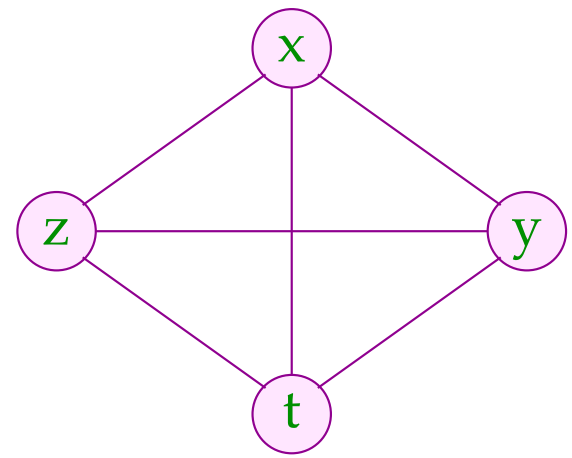
→ ... Warning: differen variants have beenpatented !!!

The algorithm works the better the smaller life ranges are ...

Idea: Life Range Splitting

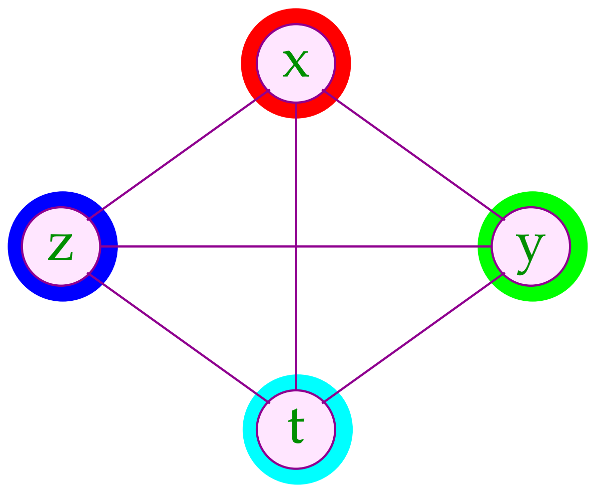
Special Case: Basic Blocks

	$\mathcal{L}$
	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
$M[y] = t;$	



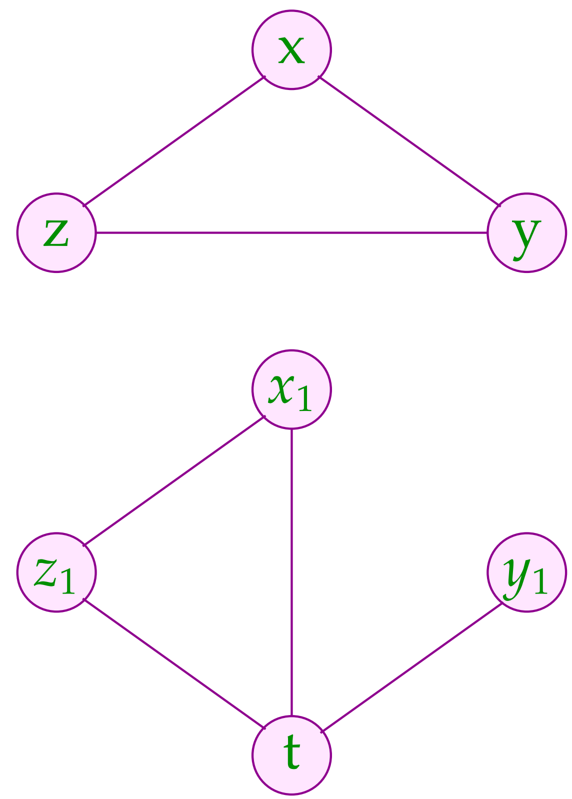
Special Case: Basic Blocks

	$\mathcal{L}$
	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
$M[y] = t;$	



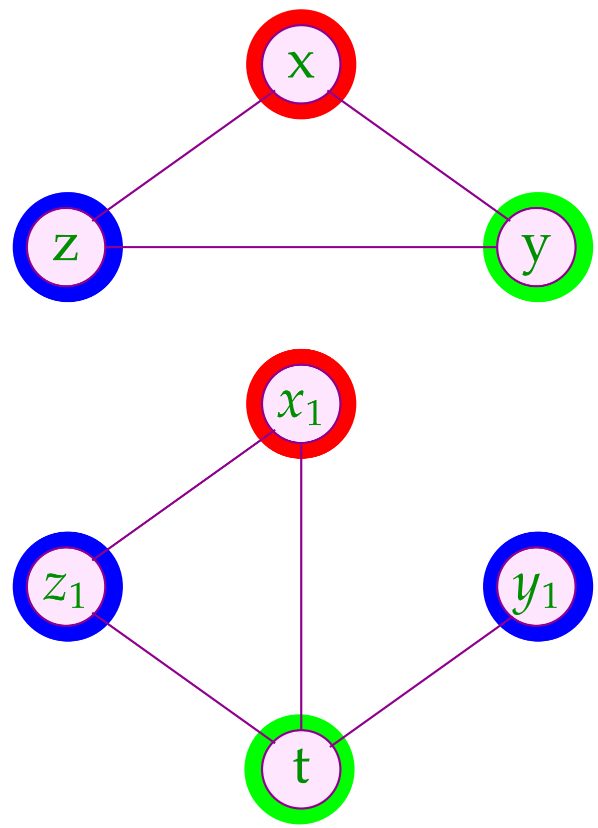
The live ranges of x and z can be split:

	$\mathcal{L}$
	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
$M[y_1] = t;$	

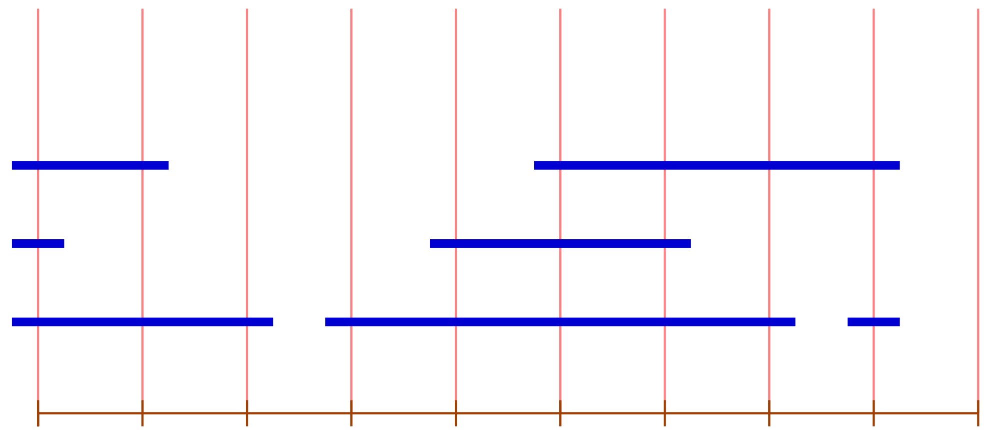


The live ranges of x and z can be split:

	$\mathcal{L}$
	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 covering number of a vertex is given by the number of incident intervals.

Theorem:

maximal covering number

$$= \text{size of the maximal clique}$$

$$= \text{maximally necessary number of colors} \quad \text{:-)}$$

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

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

Idea:

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

This results in the following algorithm:

$$
\begin{array}{l}
\mathsf{free} = [1, \ldots, k];\\
\mathsf{for}\ \ (i = 0; i < m; i{+}{+})\ \ \{\\
\qquad \mathsf{init}[i] = [\,];\ \ \mathsf{exit}[i] = [\,];\\
\}\\
\mathsf{forall}\ \ (I = [u, v] \in \mathsf{Intervals})\ \ \{\\
\qquad \mathsf{init}[u] = (I :: \mathsf{init}[u]);\ \ \mathsf{exit}[i] = (I :: \mathsf{exit}[v]);\\
\}\\
\qquad \mathsf{for}\ \ (i = 0; i < m; i{+}{+})\ \ \{\\
\qquad \mathsf{forall}\ \ (I \in \mathsf{init}[i])\ \ \{\\
\qquad\qquad \mathsf{color}[I] = \mathsf{hd}\ \mathsf{free};\ \ \mathsf{free} = \mathsf{tl}\ \mathsf{free};\\
\qquad \mathsf{forall}\ \ (I \in \mathsf{exit}[i])\ \ \mathsf{free} = \mathsf{color}[I] :: \mathsf{free};\\
\qquad \}\\
\}
\end{array}
$$

Discussion:

- → For basic blocks we have succeeded to derive an optimal register allocation :-)
- → The same problem for simple loops (circular arc graphs) is already NP-hard :-(
- → For arbitrary programs, we thus may apply some heuristics for graph coloring ...
- → which always works better the less live ranges overlap :-)
- → 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.

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.

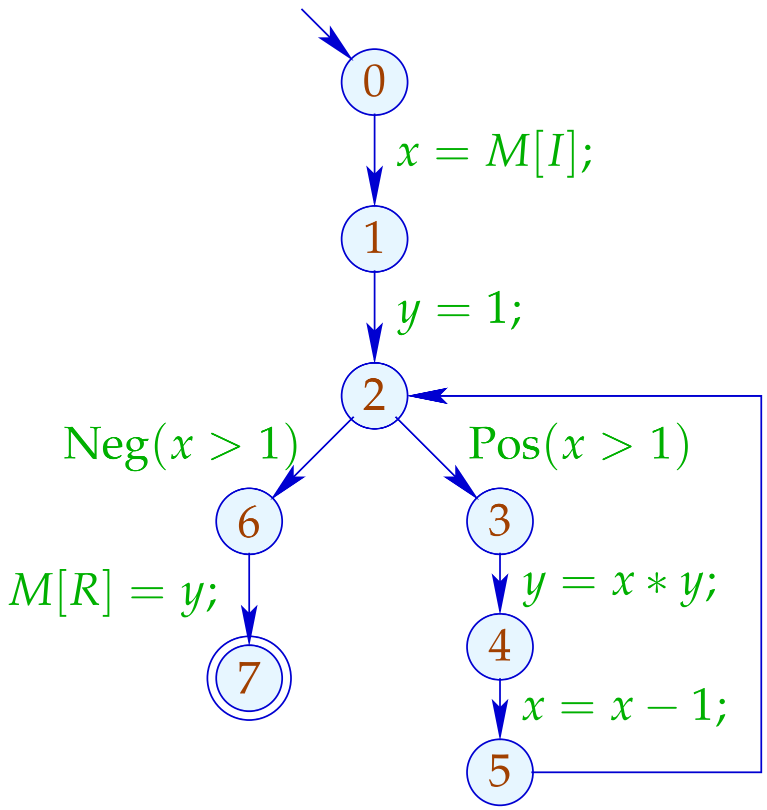
Step 2:

- Introduce a separate variant x_i for every ocurrence 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.

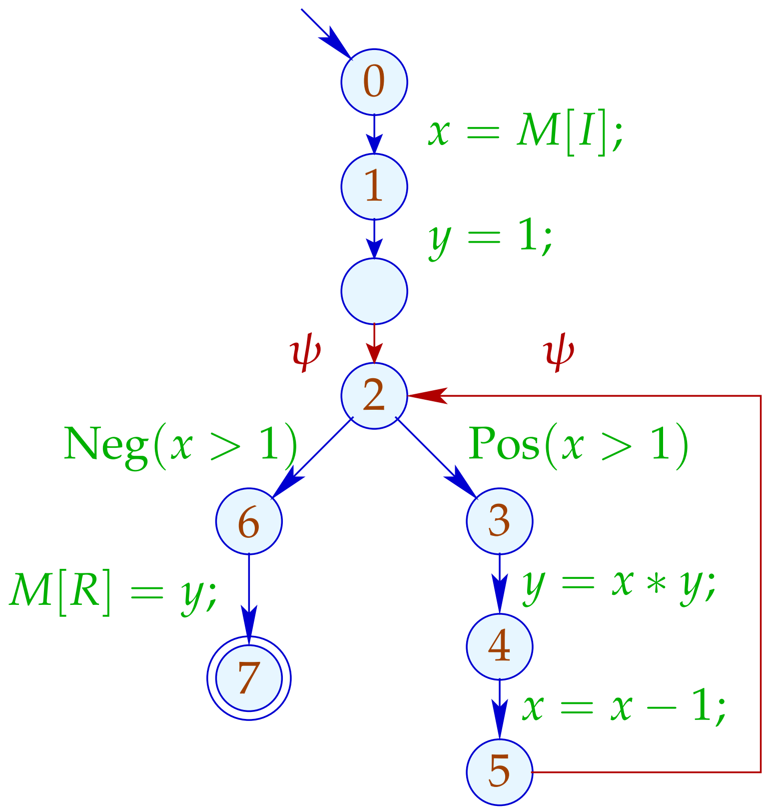
Example



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



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

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	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

Reaching Definitions

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

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

where

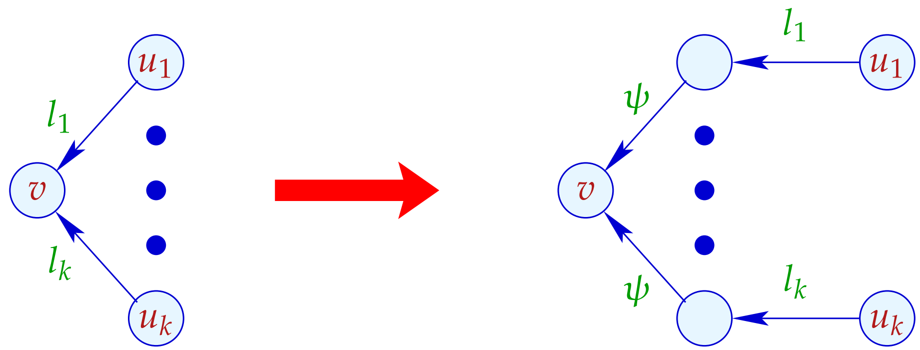
$$Defs = Vars \times Nodes \qquad Defs(x) = \{x\} \times Nodes$$

Then:

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

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

The Transformation SSA, Step 1:



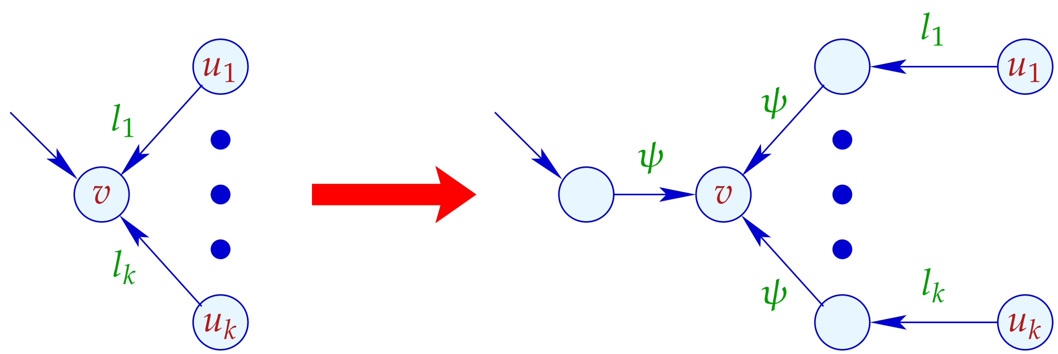
where $k \geq 2$.

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

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

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:

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

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.

Discussion

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

Discussion

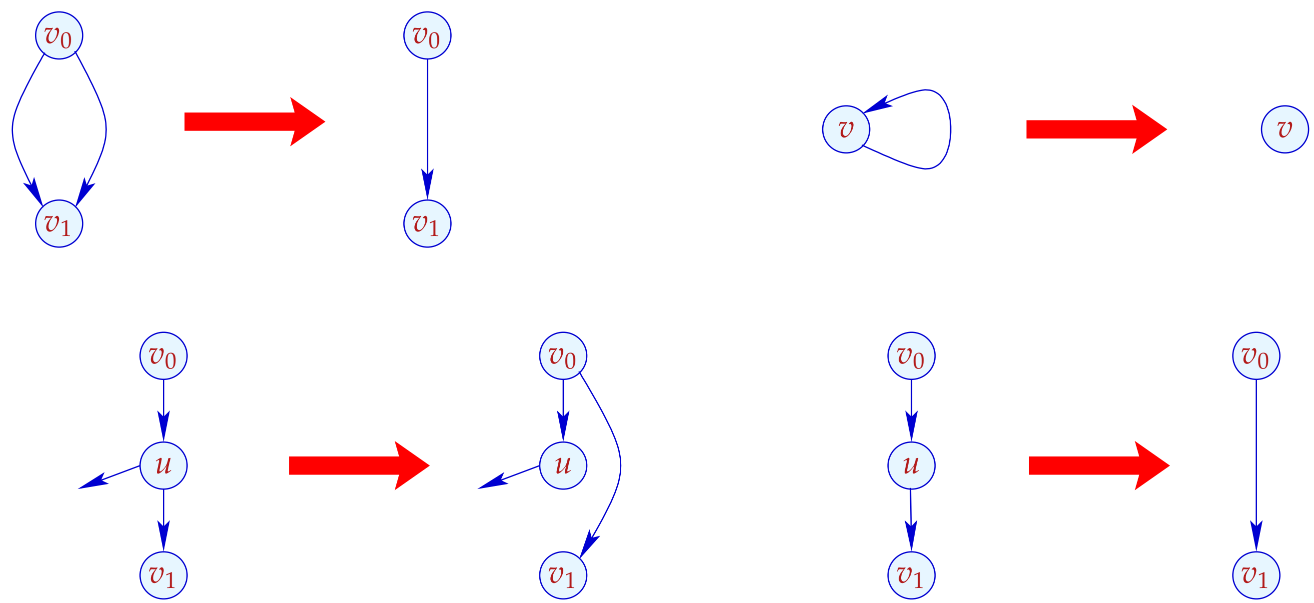
The efficiency crucially depends on the number of iterations. If the cfg is well-structured, it terminates already after one iteration !

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 (cont.)

- Reducible cfgs are not the exception — but the rule :-)
- In Java, reducibility is only violated by switches with omitted breaks.
- 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

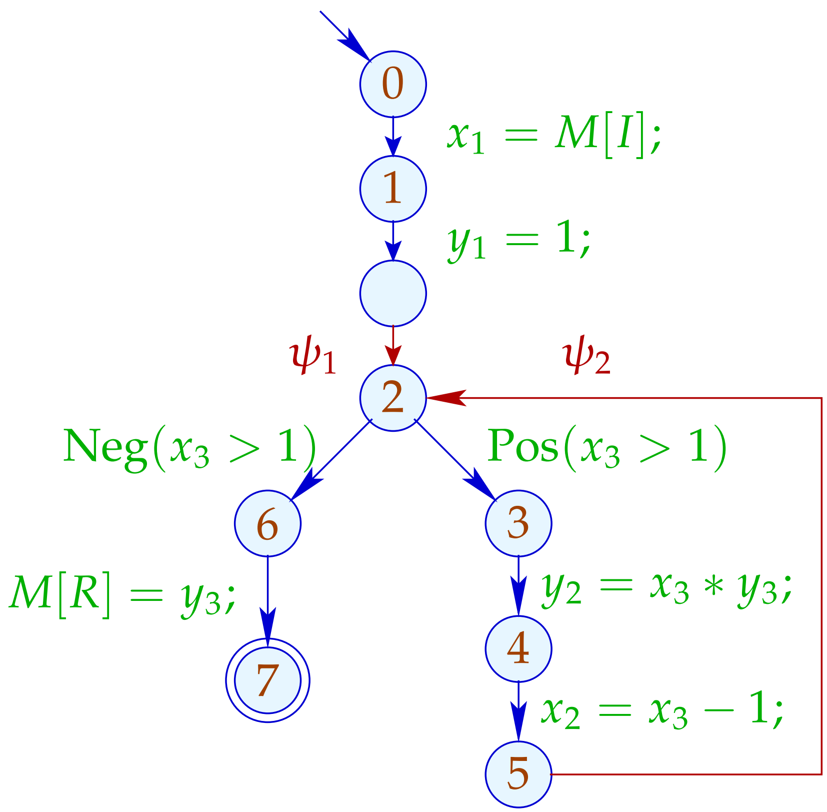
The multiple assignments:

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

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



$$\psi_1 = x_3 = x_1 \mid y_3 = y_1$$

$$\psi_2 = x_3 = x_2 \mid y_3 = y_2$$

Theorem

Assume that every 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.

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