(2) The Call-String Approach:

Idea:

→ Compute the set of all reachable call stacks!

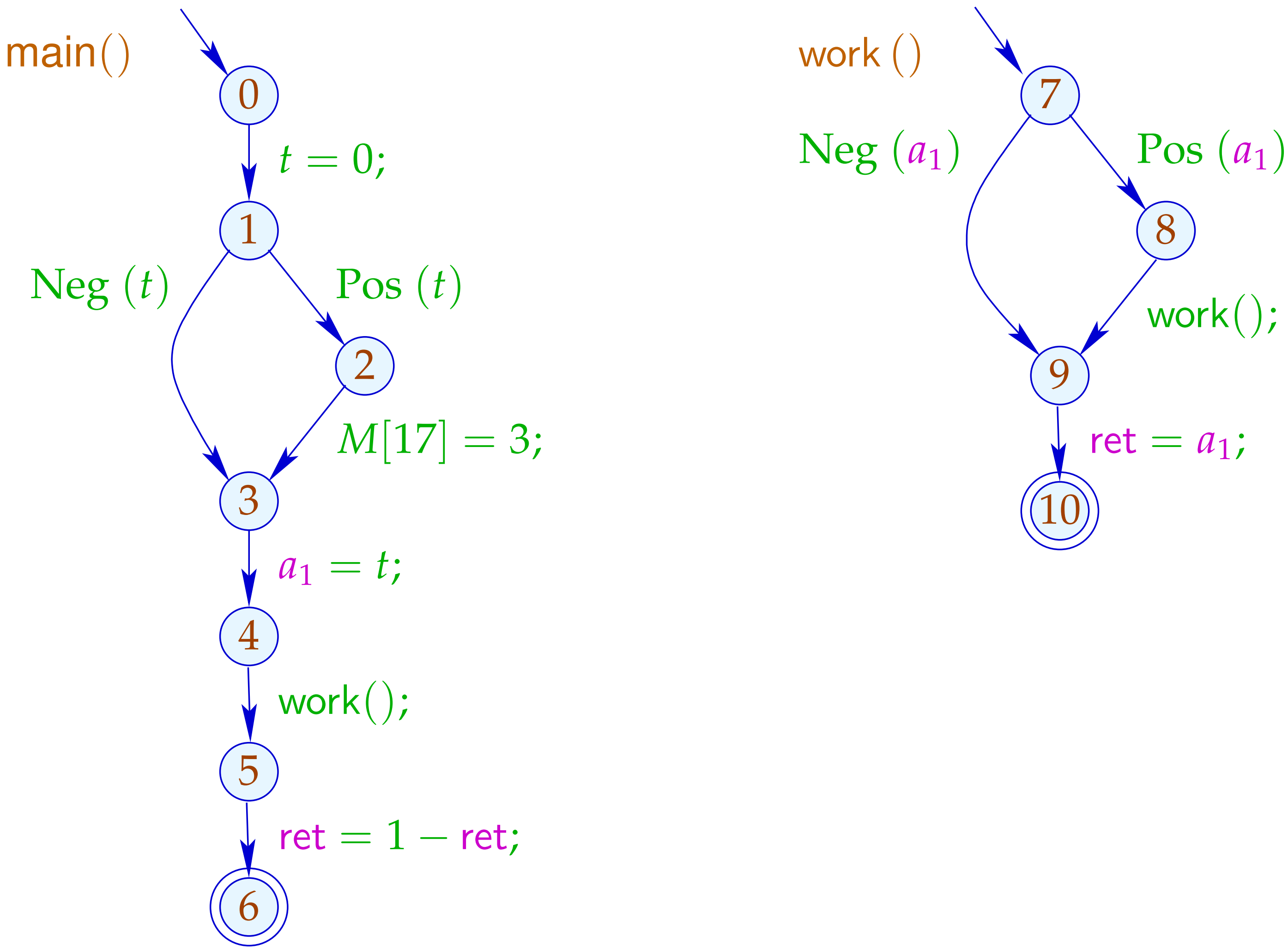
→ In general, this is infinite :-(

→ Only treat stacks up to a fixed depth d precisely! From longer stacks, we only keep the upper prefix of length d :-)

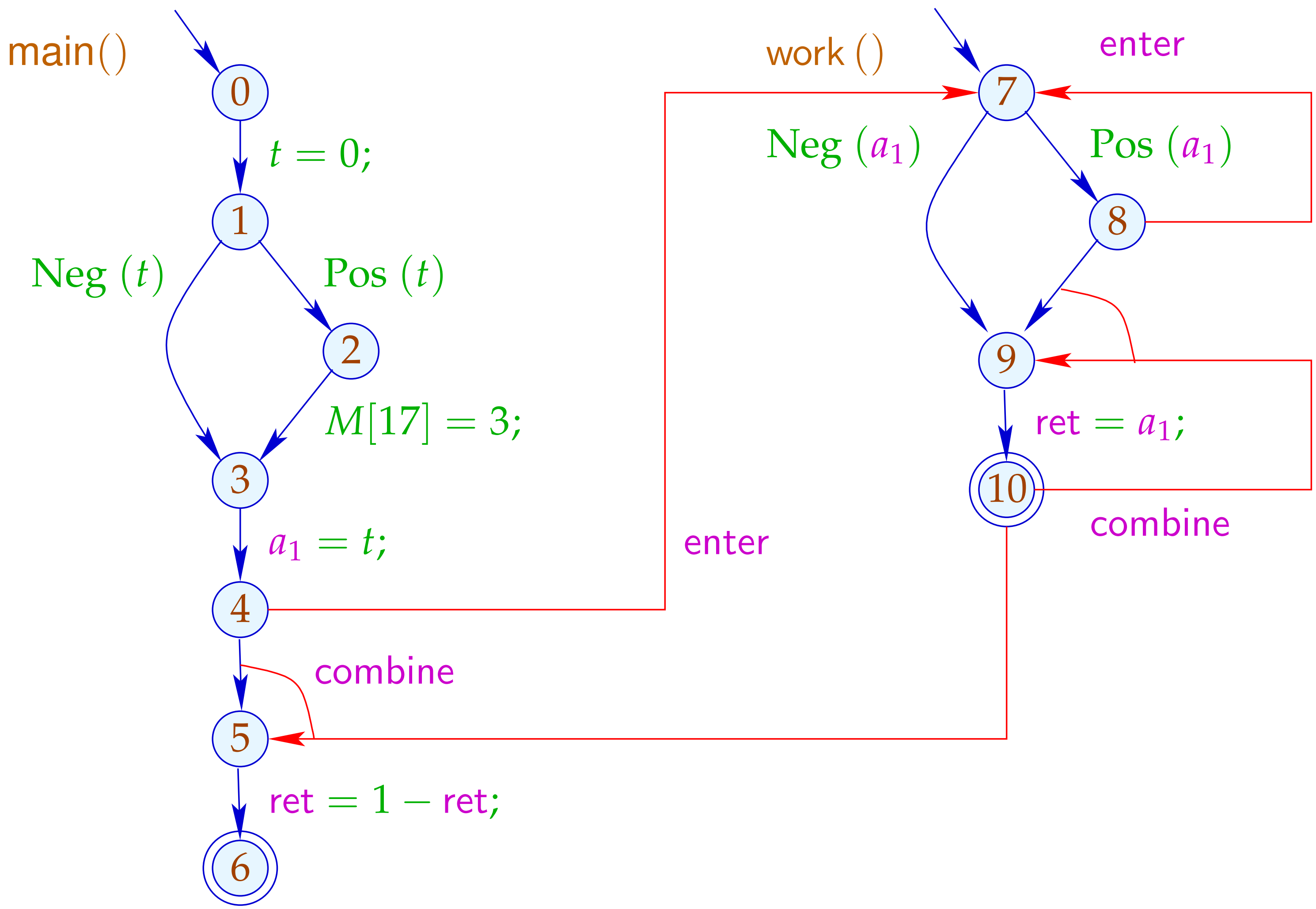
→ Important special case: $d = 0$.

$\Longrightarrow$ Just track the current stack frame ...

... in the Example:



... in the Example:



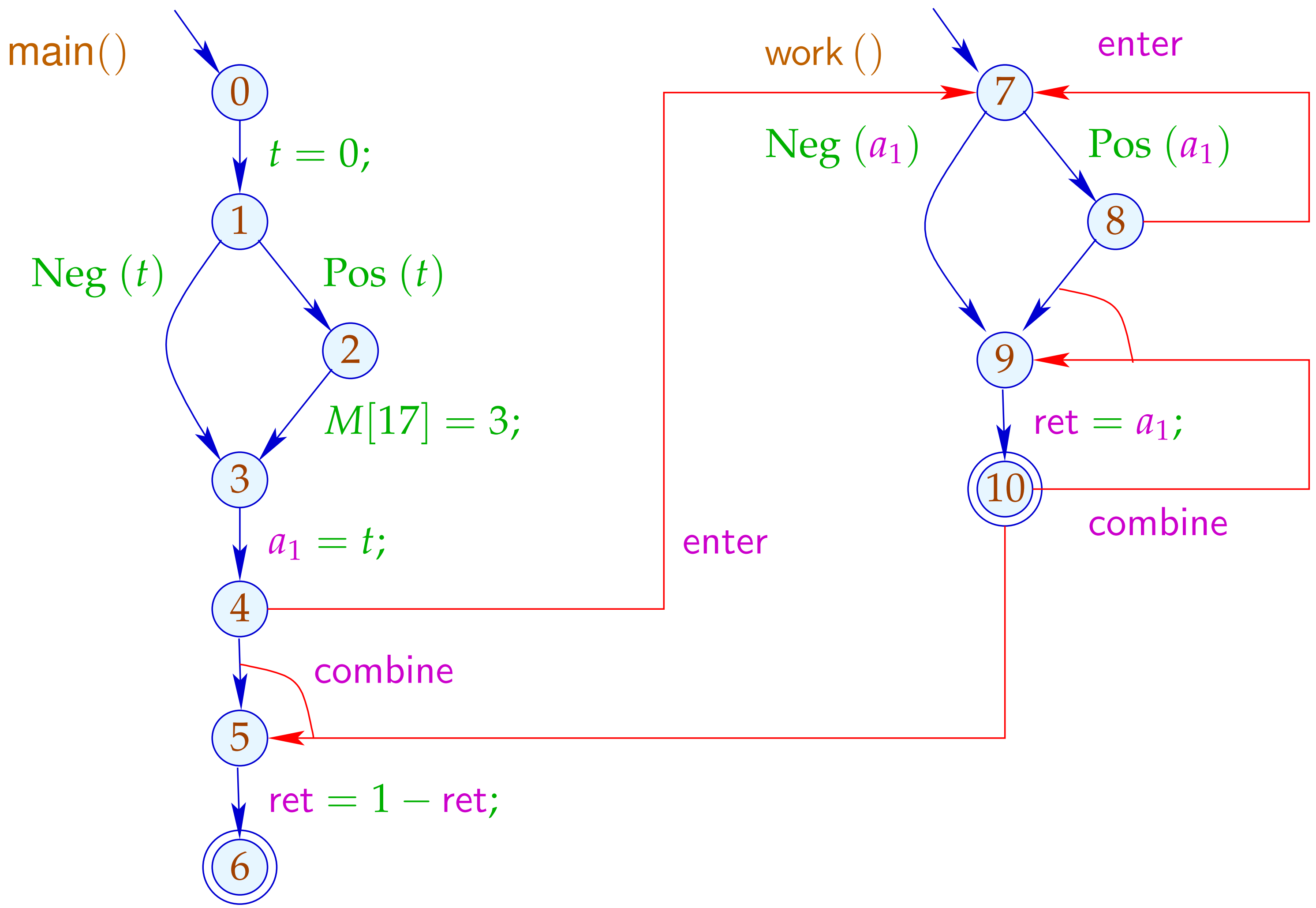
The conditions for $5, 7, 10$, e.g., are:

$$\begin{array}{lll}
\mathcal{R}[5] & \sqsupseteq & \textsf{combine}^\sharp\,(\mathcal{R}[4], \mathcal{R}[10]) \\
\mathcal{R}[7] & \sqsupseteq & \textsf{enter}^\sharp\,(\mathcal{R}[4]) \\
\mathcal{R}[7] & \sqsupseteq & \textsf{enter}^\sharp\,(\mathcal{R}[8]) \\
\mathcal{R}[9] & \sqsupseteq & \textsf{combine}^\sharp\,(\mathcal{R}[8], \mathcal{R}[10])
\end{array}$$

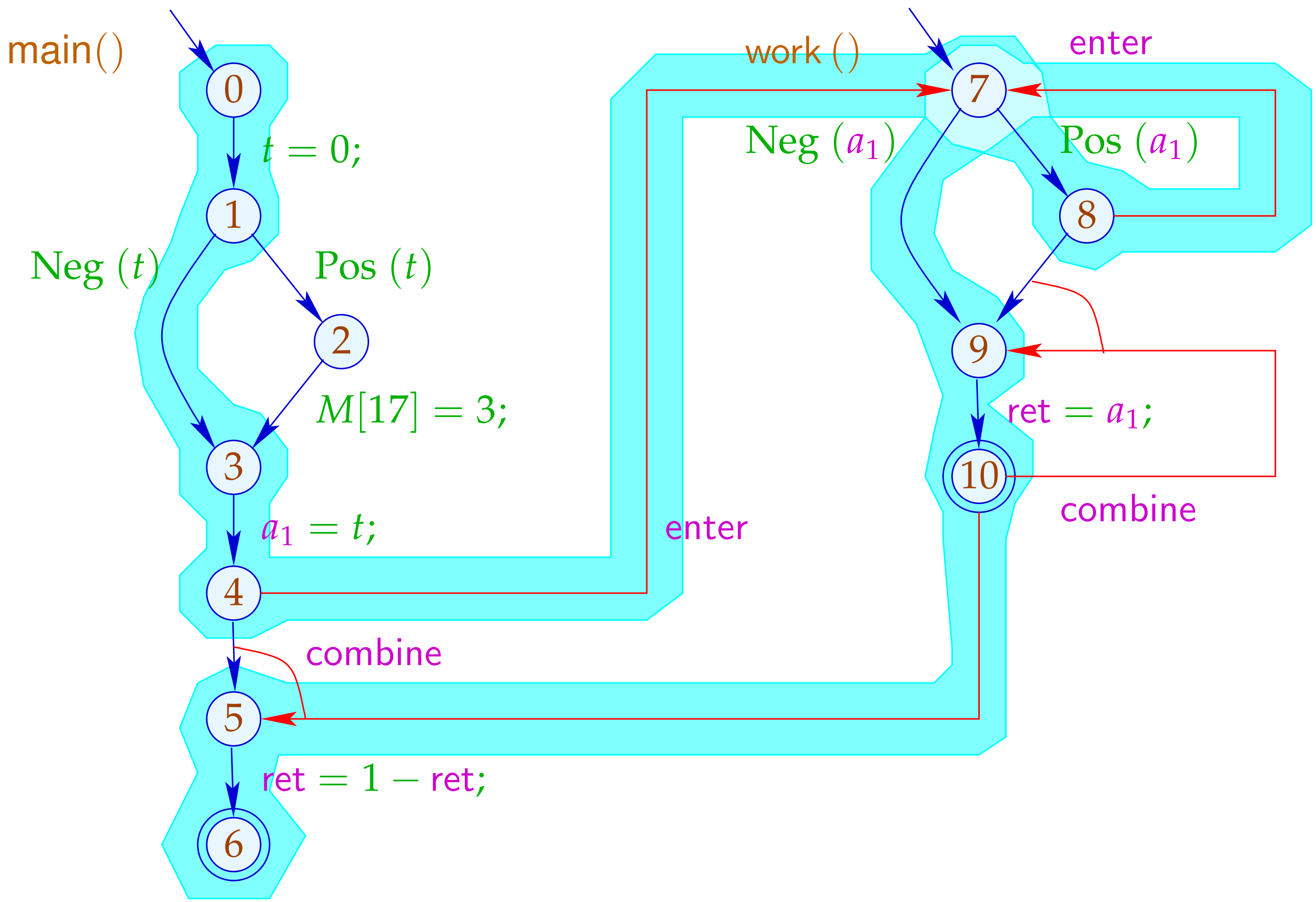
Warning:

The resulting super-graph contains obviously impossible paths ...

... in the Example this is:



... in the Example this is:



Note:

→ In the example, we find the same results: more paths render the results less precise.

In particular, we provide for each procedure the result just for one (possibly very boring) argument :-(

→ The analysis terminates — whenever $\mathbb{D}$ has no infinite strictly ascending chains :-)

→ The correctness is easily shown w.r.t. the operational semantics with call stacks.

→ For the correctness of the functional approach, the semantics with computation forests is better suited :-)

3 Exploiting Hardware Features

Question: How can we optimally use:

... Registers

... Pipelines

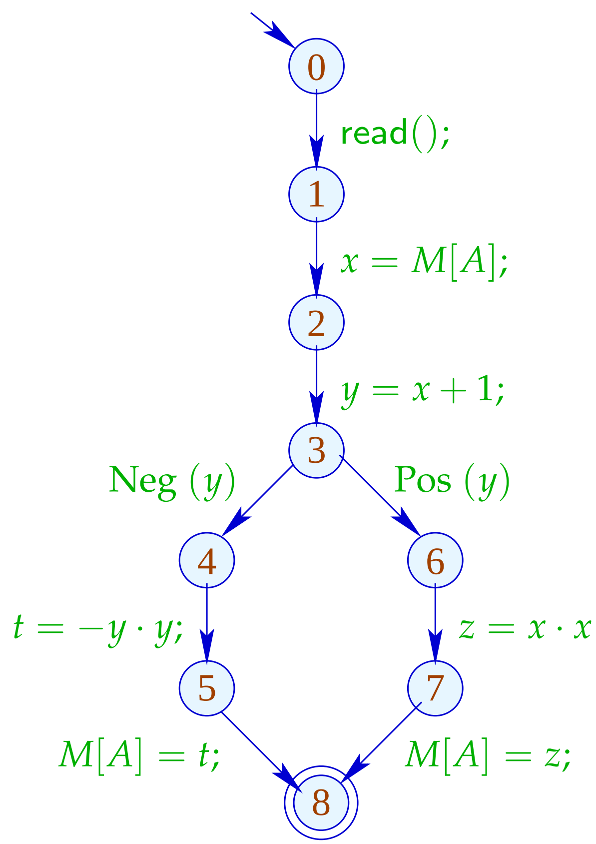
... Caches

... Processors ???

3.1 Registers

Example:

$$
\begin{array}{l}
\text{read}();\\
x = M[A];\\
y = x + 1;\\
\text{if } (y) \ \{\\
\qquad z = x \cdot x;\\
\qquad M[A] = z;\\
\} \text{ else } \{\\
\qquad t = -y \cdot y;\\
\qquad M[A] = t;\\
\}
\end{array}
$$



The program uses 5 variables ...

Problem:

What if the program uses more variables than there are registers
:-(

Idea:

Use one register for several variables :-)

In the example, e.g., one for x, t, z ...

$\mathsf{read}();$

$x = M[A];$

$y = x + 1;$

$\mathsf{if}\ \ (y)\ \ \{$

$\qquad z = x \cdot x;$

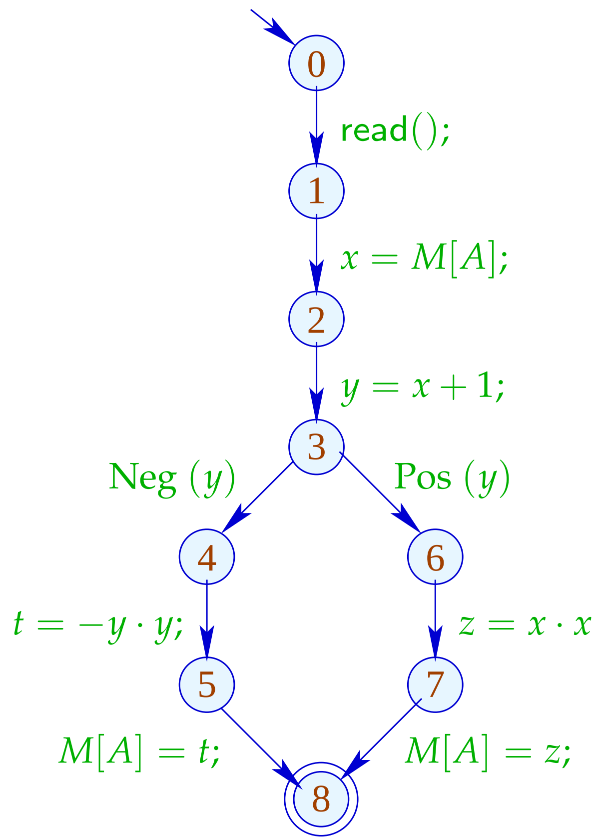
$\qquad M[A] = z;$

$\}\ \ \mathsf{else}\ \ \{$

$\qquad t = -y \cdot y;$

$\qquad M[A] = t;$

$\}$



read();

$R = M[A];$

$y = R + 1;$

if (y) {

  $R = R \cdot R;$

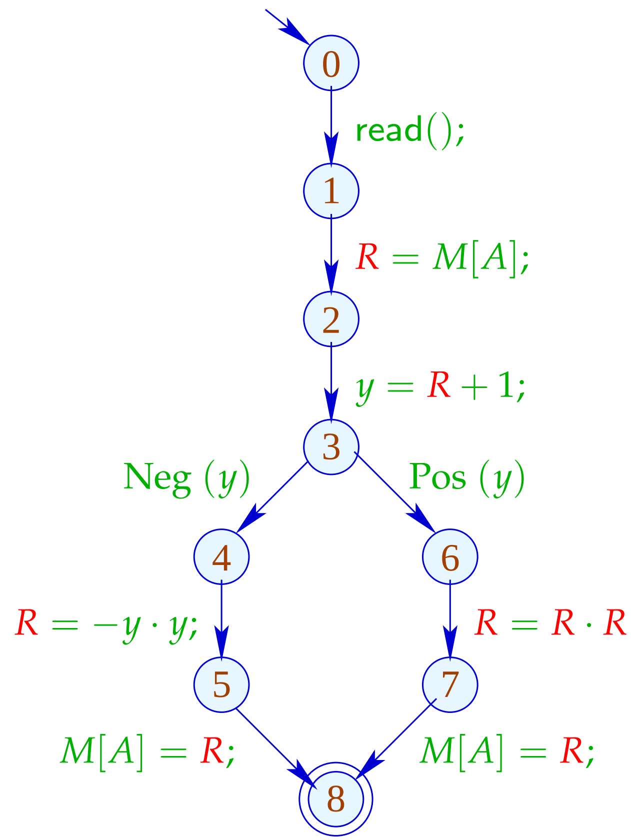
  $M[A] = R;$

} else {

  $R = -y \cdot y;$

  $M[A] = R;$

}



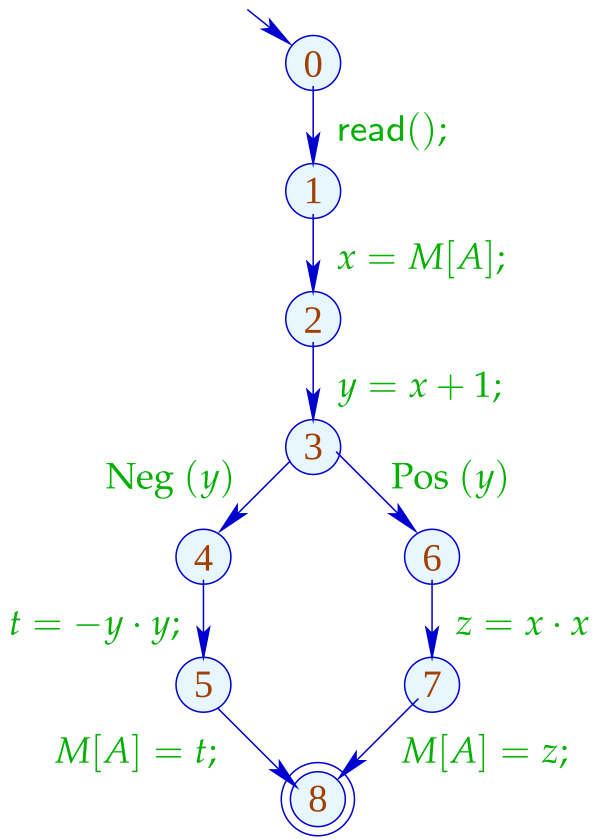
Warning:

This is only possible if the live ranges do not overlap :-)

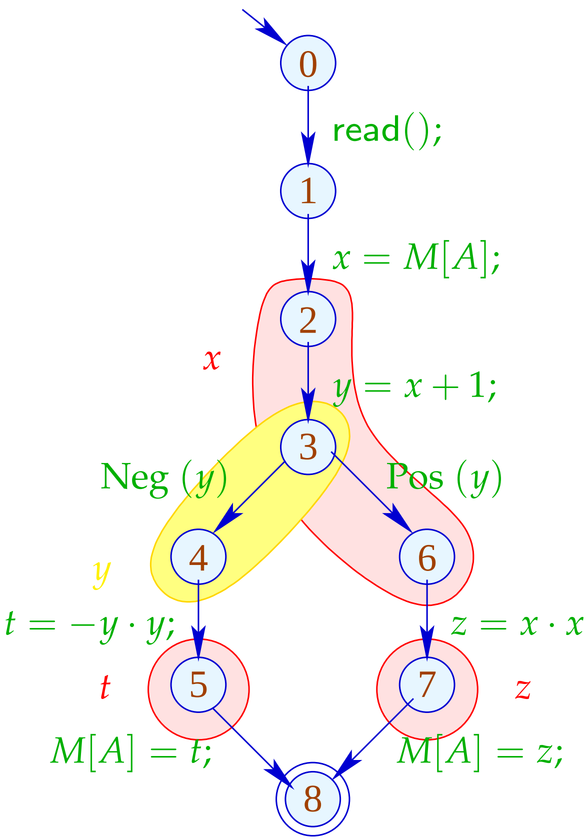
The (true) live range of x is defined by:

$$\mathcal{L}[x] = \{u \mid x \in \mathcal{L}[u]\}$$

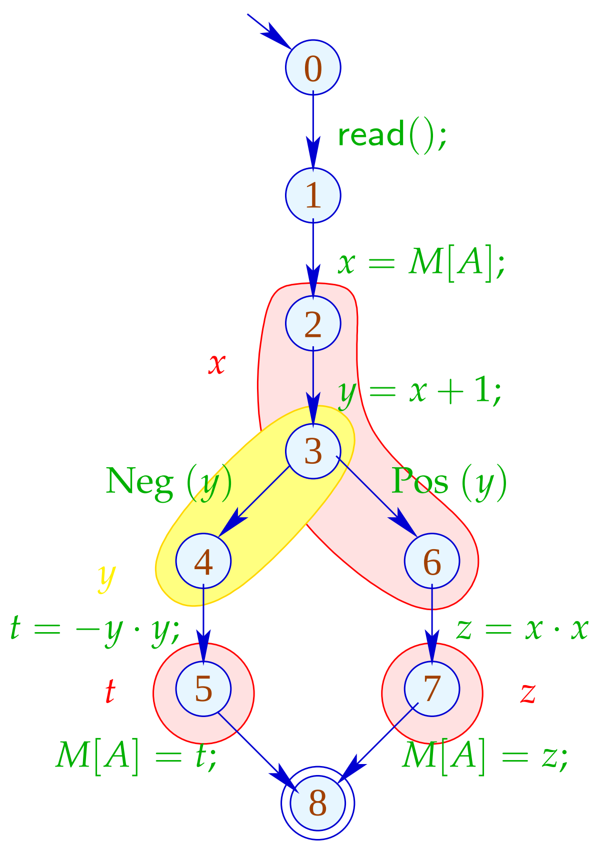
... in the Example:



	$\mathcal{L}$
8	$\emptyset$
7	$\{A, z\}$
6	$\{A, x\}$
5	$\{A, t\}$
4	$\{A, y\}$
3	$\{A, x, y\}$
2	$\{A, x\}$
1	$\{A\}$
0	$\emptyset$



	$\mathcal{L}$
8	$\emptyset$
7	$\{A, z\}$
6	$\{A, x\}$
5	$\{A, t\}$
4	$\{A, y\}$
3	$\{A, x, y\}$
2	$\{A, x\}$
1	$\{A\}$
0	$\emptyset$



Live Ranges:

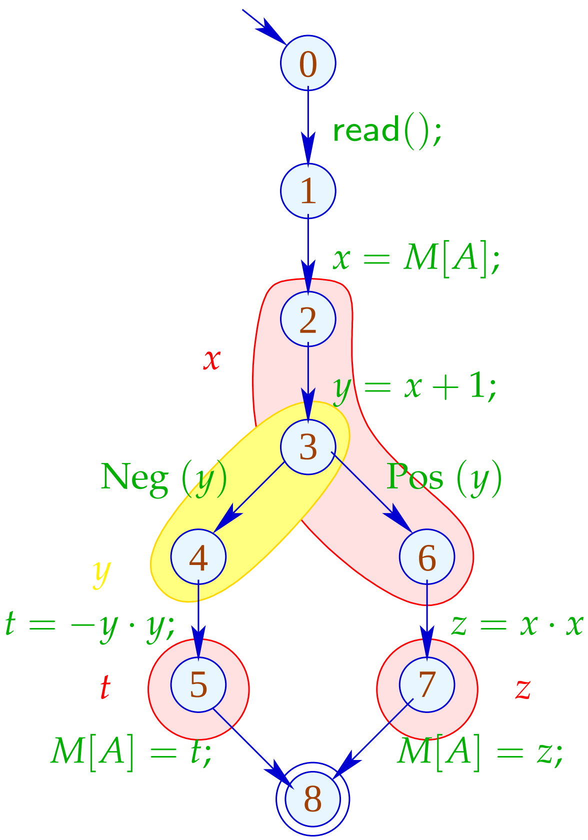
A	$\{1,\ldots,7\}$
x	$\{2,3,6\}$
y	$\{2,4\}$
t	$\{5\}$
z	$\{7\}$

In order to determine sets of compatible variables, we construct the Interference Graph $I = (Vars, E_I)$ where:

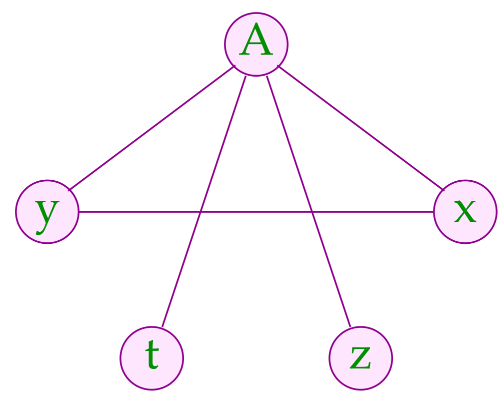
$$E_I = \{\{x, y\} \mid x \neq y, \mathcal{L}[x] \cap \mathcal{L}[y] \neq \emptyset\}$$

E_I has an edge for $x \neq y$ iff x, y are jointly live at some program point :-)

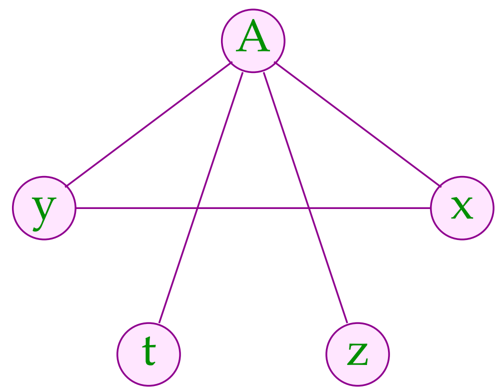
... in the Example:



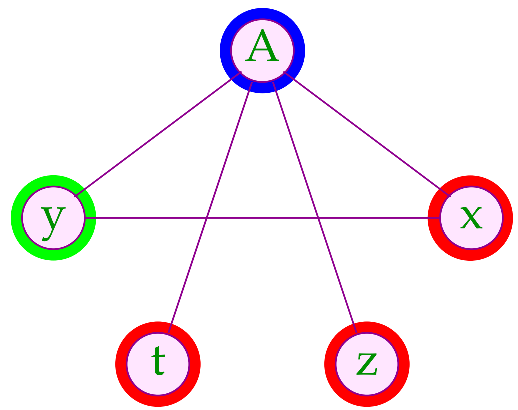
Interference Graph:



Variables which are not connected with an edge can be assigned to the same register :-)

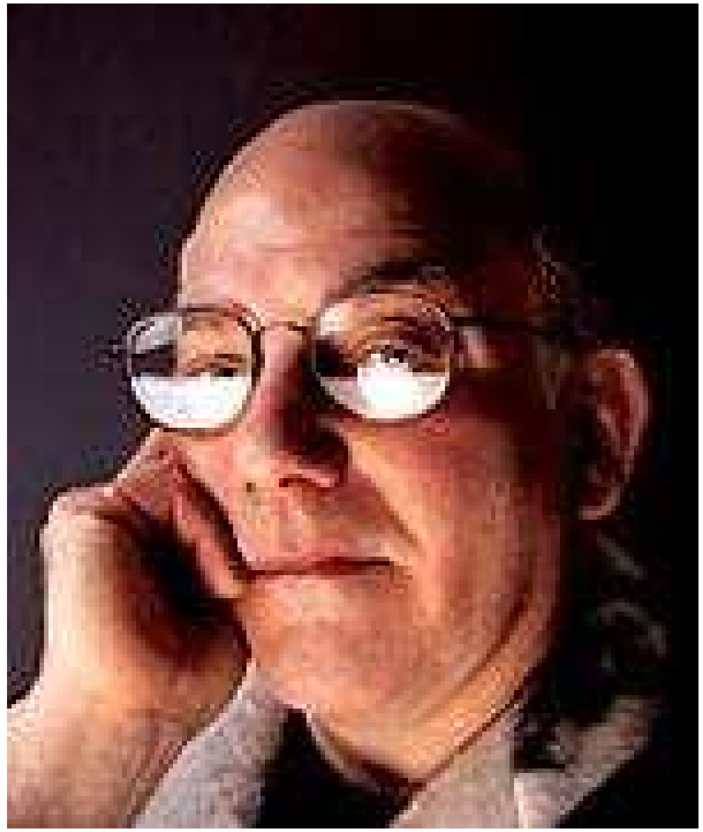


Variables which are not connected with an edge can be assigned to the same register :-)



Color $==$ Register

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:

$$\text{forall } (v \in V) \;\; c[v] = 0;$$
$$\text{forall } (v \in V) \;\; \text{color } (v);$$

```
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 been patented !!!

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 :-)

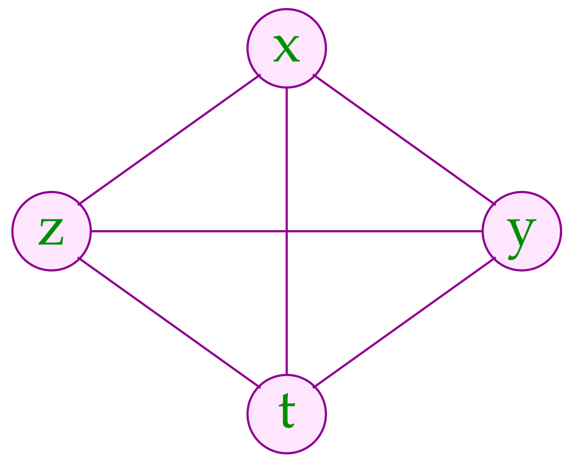
→ ... Warning: differen variants have beenpatented !!!

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

Idea: Life Range Splitting

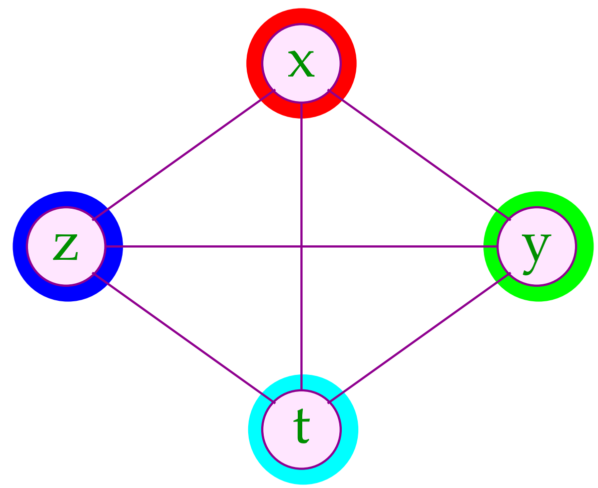
Special Case: Basic Blocks

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



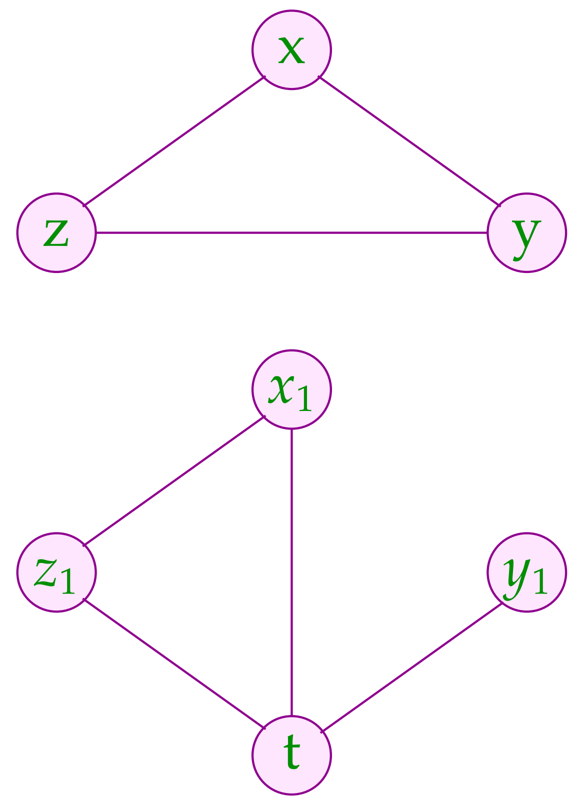
Special Case: Basic Blocks

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



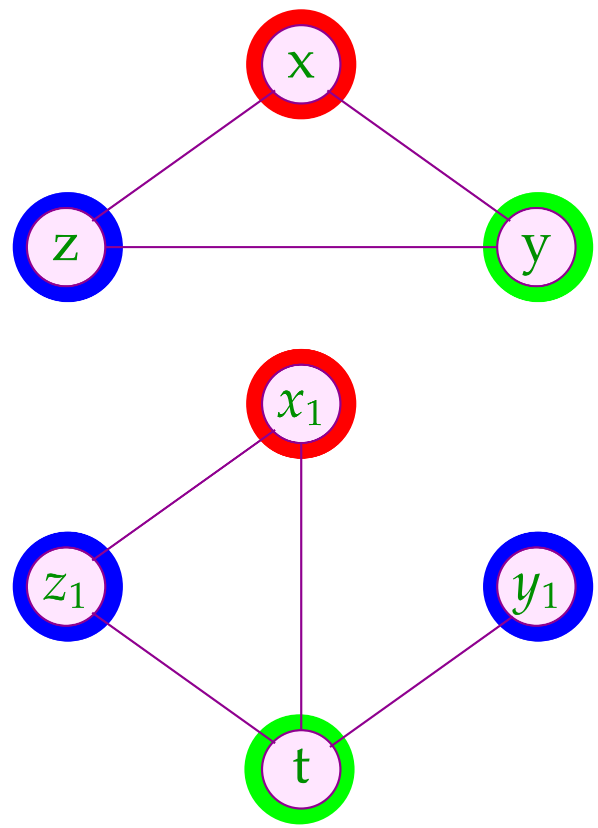
The live ranges of x and z can be split:

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

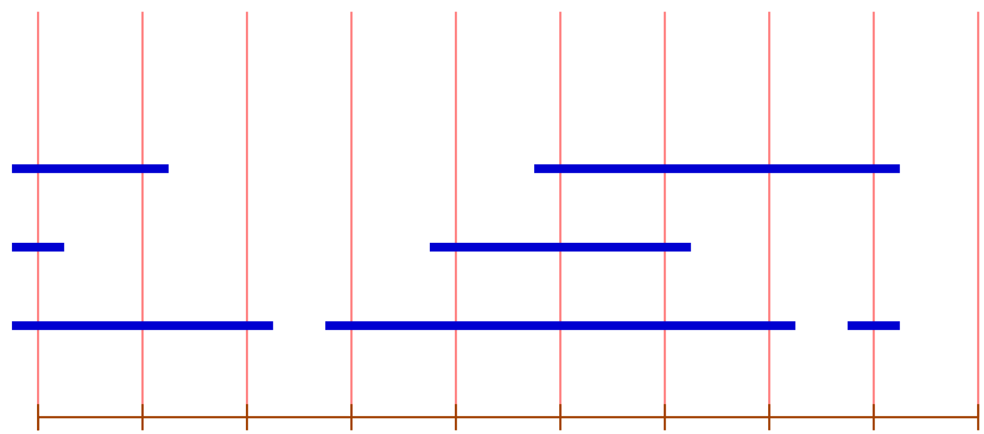


The live ranges of x and z can be split:

	$\mathcal{L}$
	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

$=$ 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:

- $\longrightarrow$ 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:

$\mathsf{free} = [1, \ldots, k];$

$\mathsf{for}\ \ (i = 0; i < m; i{+}{+})\ \ \{$

$\qquad \mathsf{init}[i] = [];\ \ \mathsf{exit}[i] = [];$

$\}$

$\mathsf{forall}\ \ (I = [u, v] \in \mathsf{Intervals})\ \ \{$

$\qquad \mathsf{init}[u] = (I :: \mathsf{init}[u]);\ \ \mathsf{exit}[i] = (I :: \mathsf{exit}[v]);$

$\}$

$\mathsf{for}\ \ (i = 0; i < m; i{+}{+})\ \ \{$

$\qquad \mathsf{forall}\ \ (I \in \mathsf{init}[i])\ \ \{$

$\qquad\qquad \mathsf{color}[I] = \mathsf{hd}\ \mathsf{free};\ \ \mathsf{free} = \mathsf{tl}\ \mathsf{free};$

$\qquad \}$

$\qquad \mathsf{forall}\ \ (I \in \mathsf{exit}[i])\ \ \mathsf{free} = \mathsf{color}[I] :: \mathsf{free};$

$\}$

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.