Script **generated by TTT**

Title: Seidl: Programmoptimierung (23.12.2015)

Date: Wed Dec 23 10:21:07 CET 2015

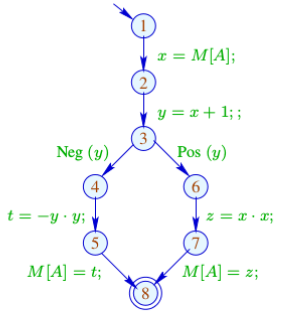
Duration: 89:13 min

Pages: 52

3.1 Registers

Example

$x = M[A];$
$y = x + 1;$
if (y) {
  $z = x \cdot x;$
  $M[A] = z;$
} else {
  $t = -y \cdot y;$
  $M[A] = t;$
}



The program uses 5 variables ...

Problem

What if the program uses more variables than there are registers.

Idea

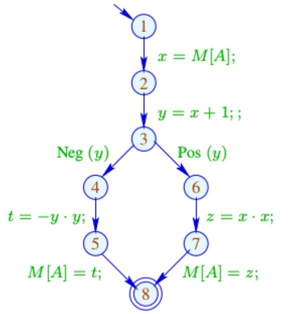
Use one register for several variables.

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

3.1 Registers

Example

$x = M[A];$
$y = x + 1;$
if (y) {
  $z = x \cdot x;$
  $M[A] = z;$
} else {
  $t = -y \cdot y;$
  $M[A] = t;$
}



582

The program uses 5 variables ...

Problem

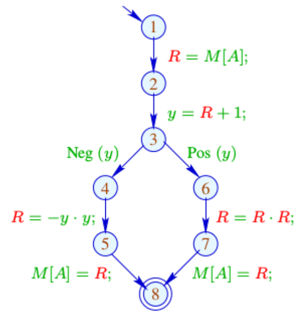
What if the program uses more variables than there are registers.

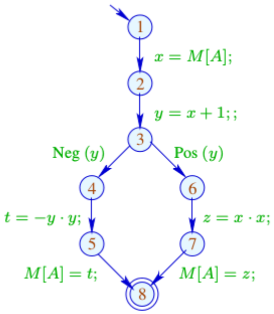
Idea

Use one register for several variables.

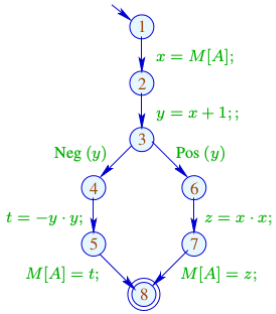
In the example, e.g., one for x, t, z ...

$R = M[A];$
$y = R + 1;$
if (y) {
    $R = R \cdot R;$
    $M[A] = R;$
} else {
    $R = -y \cdot y;$
    $M[A] = R;$
}

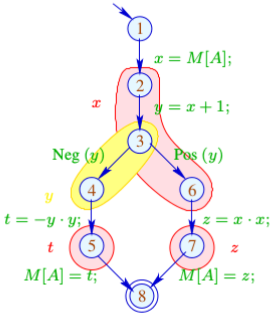




	$\mathcal{L}$
8	$\emptyset$
7	$\{A, z\}$
6	$\{A, x\}$
5	$\{A, t\}$
4	$\{A, y\}$
3	$\{A, x, y\}$
2	$\{A, x\}$
1	$\{A\}$
0	$\emptyset$



	$\mathcal{L}$
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:

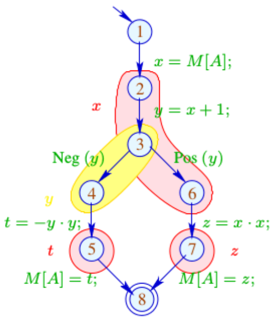
A	$\{0, \ldots, 7\}$
x	$\{2, 3, 6\}$
y	$\{2, 4\}$
t	$\{5\}$
z	$\{7\}$

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

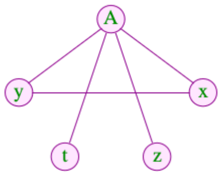
$$E_I = \{\{x, y\} \mid x \neq y, \mathcal{L}[x] \cap \mathcal{L}[y] \neq \emptyset\}$$

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

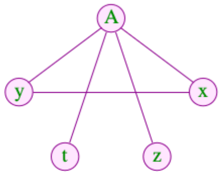
... in the Example:



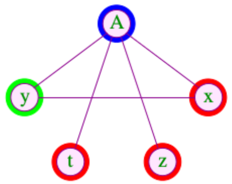
Interference Graph:



Variables which are not connected with an edge can be assigned to the same register.

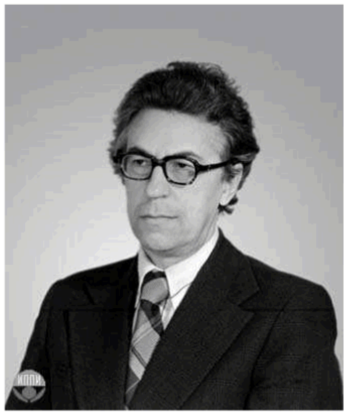


Variables which are not connected with an edge can be assigned to the same register.

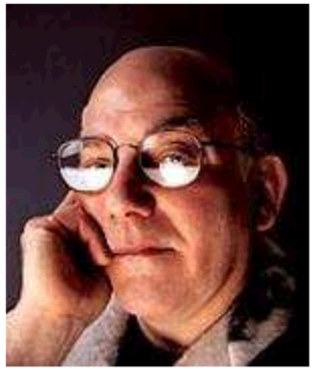


Color = Register

Sviatoslav Sergeevich Lavrov,
Russian Academy of Sciences (1962)



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

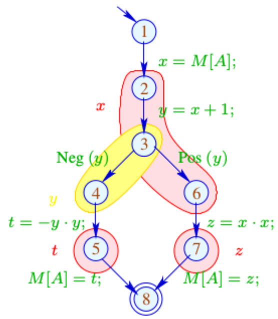
(1) $c(u) \neq c(v)$ for $\{u, v\} \in E$;

(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.



Live Ranges:

A	$\{0, \dots, 7\}$
x	$\{2, 3, 6\}$
y	$\{2, 4\}$
t	$\{5\}$
z	$\{7\}$

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.

→ ... Anecdote: different 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.

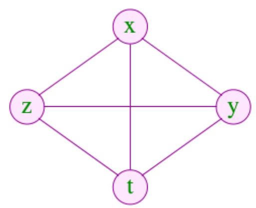
→ ... Anecdote: different variants have been patented !!!

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

Idea: Life Range Splitting

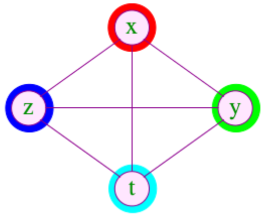
Special Case: Basic Blocks

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



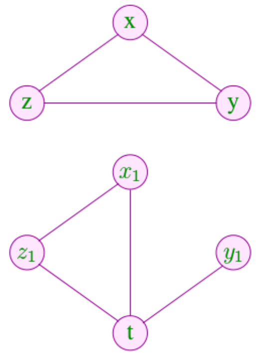
Special Case: Basic Blocks

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

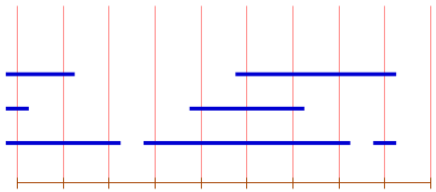


The live ranges of x and z can be split:

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



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

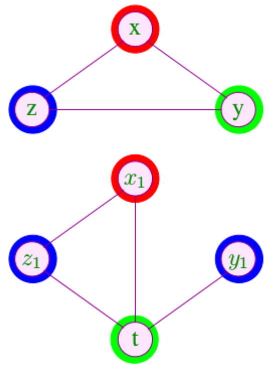


vertex $=$ interval

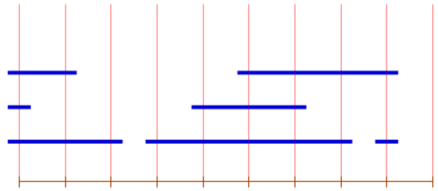
edge $=$ joint vertex

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
$M[y_1] = t;$	



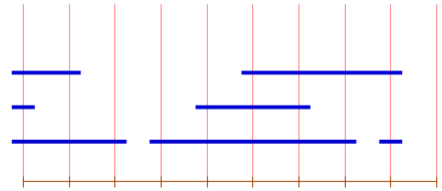
Interference graphs for minimal live ranges on basic blocks are known as interval graphs:



vertex $=$ interval

edge $=$ joint vertex

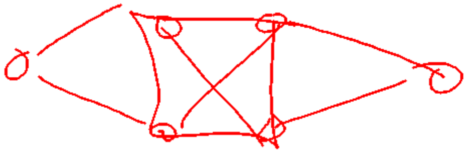
Interference graphs for minimal live ranges on basic blocks are known as interval graphs:



vertex $=$ interval

edge $=$ joint vertex

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



Theorem

maximal covering number

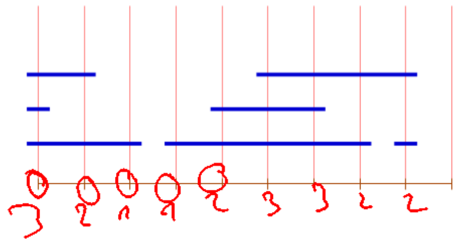
$=$ size of the maximal clique

$=$ 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.

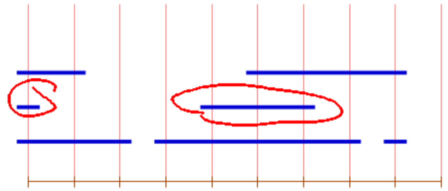
Interference graphs for minimal live ranges on basic blocks are known as interval graphs:



vertex $=$ interval

edge $=$ joint vertex

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.

→ Splitting of live ranges for full programs results programs in static single assignment form ...

Discussion

- Every live variable should be defined at most once ??
- Every live variable should have at most one definition ?
- All definitions of the same variable should have a common end point !!!

⟹ Static Single Assignment Form

Discussion

- Every live variable should be defined at most once ??
- Every live variable should have at most one definition ?
- All definitions of the same variable should have a common end point !!!

⟹ Static Single Assignment Form

610

How to arrive at SSA 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 ...

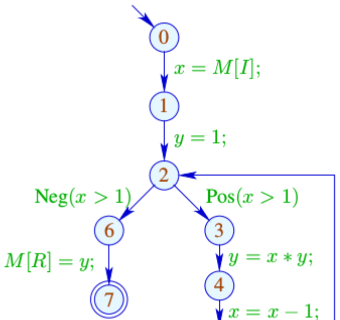
Implementing Step 1

- Determine for every program point the set of reaching definitions.
- **Assumption**

  All incoming edges of a join point v are labeled with the same parallel assignment $x = x \mid x \in L_v$ for some set L_v.

  Initially, $L_v = \emptyset$ for all v.
- 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 x into L_v, i.e., add definitions $x = x;$ at the end of each incoming edge of v.

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

Implementing Step 1

- Determine for every program point the set of reaching definitions.
- **Assumption**

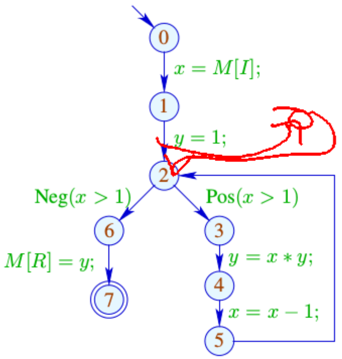
  All incoming edges of a join point v are labeled with the same parallel assignment $x = x \mid x \in L_v$ for some set L_v.

  Initially, $L_v = \emptyset$ for all v.
- 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 x into L_v, i.e., add definitions $x = x;$ at the end of each incoming edge of v.

612

Example

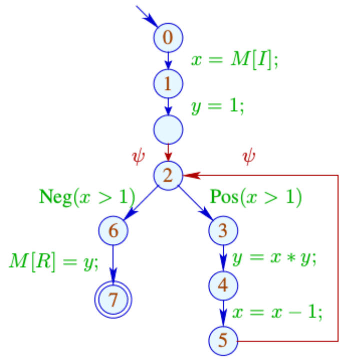
Reaching Definitions



	$\mathcal{R}$
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	$\langle x, 1\rangle, \langle x, 5\rangle, \langle y, 2\rangle, \langle y, 4\rangle$

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

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

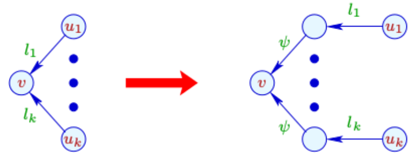
$$Defs = Vars \times Nodes \qquad Defs(x) = \{x\} \times Nodes$$

Then:

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

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



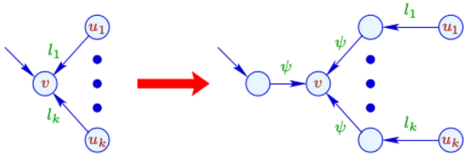
where $k \geq 2$.

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

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

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:

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

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.

618

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.