3.1 Registers

Example:

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

\[\text{Neg}(y) \quad \text{Pos}(y)\]

\[\text{read}();
\quad x = M[A];
\quad y = x + 1;
\quad \text{Neg}(y) \quad \text{Pos}(y)
\quad t = -y \cdot y;
\quad z = x \cdot x
\quad M[A] = t;
\quad M[A] = z;\]
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  \( x, t, z \) ...
read();
x = M[A];
y = x + 1;
if (y) {
    z = x \cdot x;
    M[A] = z;
} else {
    t = -y \cdot y;
    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:

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

... in the Example:
read();

x = M[A];

y = x + 1;

Neg(y)  Pos(y)

\[ t = -y \cdot y; \]

\[ z = x \cdot x \]

M[A] = t;

\[ M[A] = z; \]
read();

\( x = M[A]; \)

\( y = x + 1; \)

\( z = x \cdot x \)

\( t = -y \cdot y; \)

\( M[A] = t; \)

\( M[A] = z; \)

\( \mathcal{L} \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>7</td>
<td>{A, z}</td>
</tr>
<tr>
<td>6</td>
<td>{A, x}</td>
</tr>
<tr>
<td>5</td>
<td>{A, t}</td>
</tr>
<tr>
<td>4</td>
<td>{A, y}</td>
</tr>
<tr>
<td>3</td>
<td>{A, x, y}</td>
</tr>
<tr>
<td>2</td>
<td>{A, x}</td>
</tr>
<tr>
<td>1</td>
<td>{A}</td>
</tr>
<tr>
<td>0</td>
<td>{A}</td>
</tr>
</tbody>
</table>
read();

\[ x = M[A]; \]

\[ y = x + 1; \]

\[ z = x \cdot x \]

\[ t = -y \cdot y; \]

\[ M[A] = t; \]

\[ M[A] = z; \]

Live Ranges:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( {0, \ldots, 7} )</td>
</tr>
<tr>
<td>( x )</td>
<td>( {2, 3, 6} )</td>
</tr>
<tr>
<td>( y )</td>
<td>( {2, 4} )</td>
</tr>
<tr>
<td>( t )</td>
<td>( {5} )</td>
</tr>
<tr>
<td>( z )</td>
<td>( {7} )</td>
</tr>
</tbody>
</table>
In order to determine sets of compatible variables, we construct the Interference Graph \( I = ( \text{Vars}, E_I) \) 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:
read();

1

$x = M[A]$

2

$y = x + 1$

3

$y = \text{Neg}(y)$

$y = \text{Pos}(y)$

4

$t = -y \cdot y$

5

$t = M[A]$

6

$z = x \cdot x$

7

$z = M[A]$

8

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) \(\bigcup\{c(u) \mid u \in V\}\) 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 \(:-((\)

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

\begin{verbatim}
void \text{color} (v) {
  if \((c[v] \neq 0)\) return;
  neighbors = \{u \in V \mid \{u, v\} \in E\};
  c[v] = \bigcap\{k > 0 \mid \forall u \in \text{neighbors} : k \neq c(u)\};
  forall \((u \in \text{neighbors})\)
    if \((c(u) == 0)\) \text{color} (u);
}
\end{verbatim}

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

597
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

\[ A_1 = x + y; \]
\[ M[A_1] = z; \]
\[ x = x + 1; \]
\[ z = M[A_1]; \]
\[ t = M[x]; \]
\[ A_2 = x + t; \]
\[ M[A_2] = z; \]
\[ y = M[x]; \]
\[ M[y] = t; \]

\[ L \]
\[ x, y, z \]
\[ x, z \]
\[ x \]
\[ x, z \]
\[ x, z, t \]
\[ x, z, t \]
\[ x, t \]
\[ y, t \]
### Special Case: Basic Blocks

<table>
<thead>
<tr>
<th>Expression</th>
<th>( \mathcal{L} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1 = x + y; )</td>
<td>( x, y, z )</td>
</tr>
<tr>
<td>( M[A_1] = z; )</td>
<td>( x, z )</td>
</tr>
<tr>
<td>( x = x + 1; )</td>
<td>( x )</td>
</tr>
<tr>
<td>( z = M[A_1]; )</td>
<td>( x, z )</td>
</tr>
<tr>
<td>( t = M[x]; )</td>
<td>( x, z, t )</td>
</tr>
<tr>
<td>( A_2 = x + t; )</td>
<td>( x, z, t )</td>
</tr>
<tr>
<td>( M[A_2] = z; )</td>
<td>( x, t )</td>
</tr>
<tr>
<td>( y = M[x]; )</td>
<td>( y, t )</td>
</tr>
<tr>
<td>( M[y] = t; )</td>
<td></td>
</tr>
</tbody>
</table>

[Diagram of graph with nodes x, y, z, and t connected by edges]
The live ranges of $x$ and $z$ can be split:

<table>
<thead>
<tr>
<th>$A_1 = x + y$;</th>
<th>$M[A_1] = z$;</th>
<th>$x_1 = x + 1$;</th>
<th>$z_1 = M[A_1]$;</th>
<th>$t = M[x_1]$;</th>
<th>$A_2 = x_1 + t$;</th>
<th>$M[A_2] = z_1$;</th>
<th>$y_1 = M[x_1]$;</th>
<th>$M[y_1] = t$;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z$</td>
<td>$x, z$</td>
<td>$x_1$</td>
<td>$x_1, z_1$</td>
<td>$x_1, z_1, t$</td>
<td>$x_1, z_1, t$</td>
<td>$x_1, t$</td>
<td>$y_1, t$</td>
<td></td>
</tr>
</tbody>
</table>

\[ \begin{array}{c}
A_1 = x + y; \\
M[A_1] = z; \\
x_1 = x + 1; \\
z_1 = M[A_1]; \\
t = M[x_1]; \\
A_2 = x_1 + t; \\
M[A_2] = z_1; \\
y_1 = M[x_1]; \\
M[y_1] = t;
\end{array} \]
The live ranges of $x$ and $z$ can be split:

<table>
<thead>
<tr>
<th>Equation</th>
<th>$\mathcal{L}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = x + y$;</td>
<td>$x, y, z$</td>
</tr>
<tr>
<td>$M[A_1] = z$;</td>
<td>$x, z$</td>
</tr>
<tr>
<td>$x_1 = x + 1$;</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$z_1 = M[A_1]$;</td>
<td>$x_1, z_1$</td>
</tr>
<tr>
<td>$t = M[x_1]$;</td>
<td>$x_1, z_1, t$</td>
</tr>
<tr>
<td>$A_2 = x_1 + t$;</td>
<td>$x_1, z_1, t$</td>
</tr>
<tr>
<td>$M[A_2] = z_1$;</td>
<td>$x_1, t$</td>
</tr>
<tr>
<td>$y_1 = M[x_1]$;</td>
<td>$y_1, t$</td>
</tr>
<tr>
<td>$M[y_1] = t$;</td>
<td></td>
</tr>
</tbody>
</table>
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, \ldots, k];

for (i = 0; i < m; i++) {
    init[i] = []; exit[i] = [];
}

forall (I = [u, v] \in \text{Intervals}) {
    init[u] = (I :: init[u]); exit[v] = (I :: exit[v]);
}

for (i = 0; i < m; i++) {
    forall (I \in \text{init}[i]) {
        color[I] = \text{hd} \ free; free = \text{tl} \ free;
    }
    forall (I \in \text{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.
Example

\[ x = M[I]; \]
\[ y = 1; \]
\[ M[R] = y; \]

\[ M[R] = y; \]
\[ x = M[I]; \]
\[ y = 1; \]
\[ \text{Neg}(x > 1) \]
\[ \text{Pos}(x > 1) \]

\[ \text{Neg}(x > 1) \]
\[ \text{Pos}(x > 1) \]

\[ y = x \times y; \]
\[ x = x - 1; \]
Example

Reaching Definitions

\[ R \]

<table>
<thead>
<tr>
<th></th>
<th>( \langle x, 0 \rangle, \langle y, 0 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle y, 0 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 4 \rangle )</th>
<th>( \langle x, 5 \rangle, \langle y, 4 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</th>
<th>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( \langle x, 5 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 5 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 5 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 5 \rangle, \langle y, 4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td>( \langle x, 1 \rangle, \langle x, 5 \rangle, \langle y, 2 \rangle, \langle y, 4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where \( \psi \equiv x = x \mid y = y \)
Reaching Definitions

The complete lattice \( \mathcal{R} \) for this analysis is given by:

\[
\mathcal{R} = 2^{\text{Defs}}
\]

where

\[
\text{Defs} = \text{Vars} \times \text{Nodes} \quad \text{Defs}(x) = \{x\} \times \text{Nodes}
\]

Then:

\[
\begin{align*}
\llbracket (_{-}, x = r;, v) \rrbracket^\# \mathcal{R} & = \mathcal{R} \backslash \text{Defs}(x) \cup \{\langle x, v \rangle\} \\
\llbracket (_{-}, x = x \mid x \in L, v) \rrbracket^\# \mathcal{R} & = \mathcal{R} \backslash \bigcup_{x \in L} \text{Defs}(x) \cup \{\langle x, v \rangle \mid x \in L\}
\end{align*}
\]

The ordering on \( \mathcal{R} \) is given by subset inclusion \( \subseteq \) where the value at program start is given by \( \mathcal{R}_0 = \{\langle x, \text{start} \rangle \mid x \in \text{Vars}\} \).
Assumption:
No join point is the endpoint of several definitions of the same variable.

The Transformation SSA, Step 1:

\[
\begin{align*}
\psi &\equiv \{x = x \mid x \in \mathcal{L}[v], \#(\mathcal{R}[v] \cap \text{Defs}(x)) > 1\}
\end{align*}
\]

where \(k \geq 2\).
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 $\psi$ of the new in-going edges for $v$ is given by:

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

- Program start is interpreted as (the end point of) a definition of every variable \( x : - ) \)
- At some edges, parallel definitions \( \psi \) 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 $\psi$ 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 $u$ 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 \texttt{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:

![Diagram of graph transformations]
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 \((u, \text{lab}, v)\) is replaced with \((u, \mathcal{T}_{v,\phi}[\text{lab}], v)\) where
\[
\phi(x) = x' \quad \text{if} \quad \langle x, u' \rangle \in \mathcal{R}[u] \quad \text{and:}
\]
\[
\mathcal{T}_{v,\phi}[;] = ;
\]
\[
\mathcal{T}_{v,\phi}[\text{Neg}(e)] = \text{Neg}(\phi(e))
\]
\[
\mathcal{T}_{v,\phi}[\text{Pos}(e)] = \text{Pos}(\phi(e))
\]
\[
\mathcal{T}_{v,\phi}[x = e] = x_v = \phi(e)
\]
\[
\mathcal{T}_{v,\phi}[x = M[e]] = x_v = M[\phi(e)]
\]
\[
\mathcal{T}_{v,\phi}[M[e_1] = e_2] = M[\phi(e_1)] = \phi(e_2)
\]
\[
\mathcal{T}_{v,\phi}[[x = x \mid x \in L]] = \{x_v = \phi(x) \mid x \in L\}
\]
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.,

\[ \begin{bmatrix} [pa] \end{bmatrix} (\rho, \mu) = (\rho \oplus \{ x_{v}^{(i)} \mapsto \rho(x_{v_i}^{(i)}) \mid i = 1, \ldots, k \}, \mu) \]
Example

\[ x_1 = M[I]; \]
\[ y_1 = 1; \]
\[ \text{Neg}(x_3 > 1) \]
\[ M[R] = y_3; \]

\[ y_2 = x_3 \times y_3; \]
\[ x_2 = x_3 - 1; \]

\[ \psi_1 \]
\[ = x_3 = x_1 \mid y_3 = y_1 \]

\[ \psi_2 \]
\[ = x_3 = x_2 \mid y_3 = y_2 \]