... in the Example this is:

```
main()

1. t = 0;
2. M[17] = 3;
3. a1 = t;
4. combine
5. ret = 1 - ret;
6. enter
7. work()

main()

1. t = 0;
2. M[17] = 3;
3. a1 = t;
4. combine
5. ret = 1 - ret;
6. enter
7. work()
```

... in the Example this is:

```
main()

1. t = 0;
2. M[17] = 3;
3. a1 = t;
4. combine
5. ret = 1 - ret;
6. enter
7. work()

main()

1. t = 0;
2. M[17] = 3;
3. a1 = t;
4. combine
5. ret = 1 - ret;
6. enter
7. work()
```
... in the Example this is:

main()

1. \( t = 0; \)
2. Pos (t)
3. Neg (t)
4. \( M[17] = 3; \)
5. \( a_1 = t; \)
6. combine
7. enter
8. Pos \( a_1 \)
9. \( ret = a_1; \)
10. combine

... in the Example this is:

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

- \( \mathcal{R}[5] \subseteq \text{combine}(\mathcal{R}[4], \mathcal{R}[10]) \)
- \( \mathcal{R}[7] \subseteq \text{enter}(\mathcal{R}[4]) \)
- \( \mathcal{R}[7] \subseteq \text{enter}(\mathcal{R}[8]) \)
- \( \mathcal{R}[9] \subseteq \text{combine}(\mathcal{R}[8], \mathcal{R}[10]) \)

Warning:

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

... in the Example this is:

main()

1. \( t = 0; \)
2. Pos (t)
3. Neg (t)
4. \( M[17] = 3; \)
5. \( a_1 = t; \)
6. combine
7. enter
8. Pos \( a_1 \)
9. \( ret = a_1; \)
10. combine

... in the Example this is:
3 Exploiting Hardware Features

Question: How can we optimally use:

... Registers
... Pipelines
... Caches
... Processors ???

3.1 Registers

Example:

read();
x = M[A];
y = x + 1;
if (y) {
  z = x * x;
  M[A] = z;
}
else {
  t = -y * 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 ...

---

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

---

**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:
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. :D

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 \(c : V \rightarrow \mathbb{N}\) 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 \(\because\) But:
- In general, the minimal coloring is not unique \(\because\)
- It is NP-complete to determine whether there is a coloring with at most \(k\) colors \(\because\)

We must rely on heuristics or special cases \(\because\)

---

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:

for all \((v \in V)\) \(c[v] = 0\);
for all \((v \in V)\) color \((v)\);

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

The new color can be easily determined once the neighbors are sorted according to their colors \(\because\)
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 !!!

Special Case: Basic Blocks

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

Idea: Life Range Splitting
The live ranges of $x$ and $z$ can be split:

| $A_1 = x + y$; $M[A_1] = z$; $x_1 = x + 1$; $z_1 = M[x_1]$; $t = M[x_1]$; $y_1 = M[x_1]$; $M[y_1] = t$; | $x, y, z$; $x, z$; $x$; $x_1$; $x_1, z_1$; $x_1, z_1, t$; $y_1, t$; |

The live ranges of $x$ and $z$ can be split:

| $A_1 = x + y$; $M[A_1] = z$; $x_1 = x + 1$; $z_1 = M[x_1]$; $t = M[x_1]$; $y_1 = M[x_1]$; $M[y_1] = t$; | $x, y, z$; $x, z$; $x$; $x_1$; $x_1, z_1$; $x_1, z_1, t$; $y_1, t$; |

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

Interference graphs for minimal live ranges on basic blocks are known as interval graphs:
The covering number of a vertex is given by the number of incident intervals.

**Theorem:**

maximal covering number

\[=\text{size of the maximal clique}\]

\[=\text{minimally necessary number of colors}\]

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:
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 ...