3.1 Registers

Example

\[ x = M[A]; \]
\[ y = x + 1; \]
\[ \text{if} \quad \{ \]
\[ z = x \cdot x; \]
\[ M[A] = z; \]
\[ \} \quad \text{else} \quad \{ \]
\[ t = -y \cdot y; \]
\[ M[A] = t; \]
\[ \} \]

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 \( x, t, z \) ...
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 \( x, t, z \) ...

\[
R = M[A];
\]
\[
y = R + 1;
\]
\[
\text{if } \langle y \rangle \{ \]
\[
R = R \cdot R;
\]
\[
M[A] = R;
\}
\[
\text{else } \{ \]
\[
R = -y \cdot y;
\]
\[
M[A] = R;
\}
\]

\[
\begin{array}{c|c}
\text{\( L \)} & 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 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{\( L \)} & 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 \\
\end{array}
\]
In order to determine sets of compatible variables, we construct the Interference Graph \( I = (\text{Vars}, E_I) \) where:

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

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

... in the Example:

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.
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. $|\{c(u) \mid u \in V\}|$ 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.

Live Ranges:

- $A$ = \{0, …, 7\}
- $x$ = \{2, 3, 6\}
- $y$ = \{2, 4\}
- $z$ = \{5\}

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 \((v \in V)\) \(\text{color}(v)\);

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

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 !!!

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

Idea: Life Range Splittin

Special Case: Basic Blocks

| \(L\) | \(x, y, z\) |
| \(x, y\) | \(x, z\) |
| \(M[A_1] = z\) | \(x\) |
| \(x = x + 1\) | \(x\) |
| \(t = M[A_1]\) | \(x, z, t\) |
| \(A_2 = x + t\) | \(x, z, t\) |
| \(M[A_2] = z\) | \(x, t\) |
| \(y = M[A_1]\) | \(y, t\) |
| \(M[y] = t\) |
Special Case: Basic Blocks

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

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

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

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.
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, ..., 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:

```plaintext
free = [1, ..., k];
for (i = 0; i < m; i++) {
    init[i] = []; exit[i] = [];
}
forall (l = [u, v] ∈ intervals) {
    init[u] = (l :: init[u]); exit[v] = (l :: exit[v]);
}
for (i = 0; i < m; i++) {
    forall (l ∈ init[i]) {
        color[l] = hd free; free = tl free;
    }
    forall (l ∈ exit[i]) free = color[l] :: free;
}
```
free = [1, \ldots, k];
for (i = 0; i < m; i++) {
    init[i] = []; exit[i] = [];
}

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

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

\textbf{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 !!!

\textbf{Static Single Assignment Form}

\textbf{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 !!!

\textbf{Static Single Assignment Form}
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 \( x_A \)

---

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

$$\mathcal{R} = 2^{\text{Def}s}$$

where

$$\text{Def}s = \text{Vars} \times \text{Nodes} \quad \text{Def}s(x) = \{x\} \times \text{Nodes}$$

Then:

$$[\_., x = r; v] \mathcal{R} = \mathcal{R}\setminus\text{Def}s(x) \cup \{(x, v)\}$$

$$[\_., x = x | x \in L, v] \mathcal{R} = \bigcup_{x \in L} \text{Def}s(x) \cup \{(x, v) | x \in L\}$$

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

The label $\psi$ of the new in-going edges for $v$ is given by:

$$\psi \equiv \{x = x | x \in \mathcal{L}[v], \#(\mathcal{R}[v] \cap \text{Def}s(x)) > 1\}$$
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 $\psi$ of the new in-going edges for $v$ is given by:

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

---

**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 $v$ 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 control flow 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 where \( m \) 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 \( R[u] \) has exactly one definition for every variable \( x \) which is live at \( u \).