Syntax-directed approach to inserting φ nodes

if - then - else

Processing else then

copy ST \rightarrow ST' (at the end of preceding basic block)

then - part

basic block assignment stat
  right hand side (expression)
  left hand side variable V

V_{new} \rightarrow \text{in symbol table}

Current = new
new ++
in block B continue

insert ( update ) \( \phi \) node

\[ V = \phi( V_{\text{then}} , V_{\text{open}} ) \]

\[ \text{don't know version yet} \]

If there are multiple assignments to \( V \):

change "\( V_{\text{then}} \)" in \( \phi \) node to latest version
2) process else part

reset S.T. to ST'

set the "next" fields of all variables in S.T. to the value of the next field in the S.T. of the end of the then part

process stmts in else part

assignment stmt

   \[ \text{RHS} \leftarrow \text{LHS} \]

variable \( V \) is set

\[ V_{\text{new}} = \ldots \quad \text{in the symbol table} \]

current = new

new++
In B continue, insert or update \( \phi \) node.

There are two cases:

1) there exist a \( \phi \) node inserted as a consequence of processing the part

\[
v = \phi(v_{\text{then}}, v_{\text{open}})
\]

replace

\[
V_{\text{else}}
\]

2) there is no \( \phi \) node for \( v \)

insert \( \phi \) node

\[
v = \phi(v_{\text{open}}, V_{\text{else}})
\]

(If there are multiple assignments to \( v \) in the part –

4) update corresponding \( \phi \) node: latest \( V_{\text{else}} \)
After processing then-part, else part

in B continue:

\[ V = \emptyset (V_{\text{then}}, V_{\text{else}}) \]

\[ V = \emptyset (V_{\text{then}}, V_{\text{open}}) \]

\[ V = \emptyset (V_{\text{open}}, V_{\text{else}}) \]

Set \( V_{\text{open}} \) to version at the end of preceding basic block.

- Create new versions for LHS of \( \phi \) node.
2.5.3 While loops

\[ y = \ldots \]

while \((y > 0)\) {

\[ y = y - 1 \]

\[ y = y - 1 \]

Basic Idea: insert \( \phi \)-node into \( B_{\text{test}} \) because two paths merge (previous block \( y \) set in loop body)
Processing Btest - block that controls loop execution

- insert $\phi$ node for loop control variable

\[ \check{v}_{\text{new}} = \phi( v_{\text{init}}, v_{\text{open}} ) \]

before loop

- process block of loop body ...

- path up $\phi$ node in Btest replace $v_{\text{open}}$ with last version set in the loop
3.0 General approach to insert \( \phi \) nodes

The syntactically directed approach works well for

well-structured programs

... but it does not work for languages/programs

that have unrestricted control-flow statements

- goto stmts
- longjmp
- try-catch exception handling

How do we turn an arbitrary CFG into SSA form?

- Find place to insert \( \phi \) node
- Determine version to be used in \( \phi \) nodes and statements/expressions
3.1 Intro

Let us assume there is an initial (dedicated) assignment to each variable.

For \( v, x, y \), we assume assignments

\[
\begin{align*}
v_0 &= \\
x_0 &= \\
y_0 &= \\
\end{align*}
\]

in the START node.

0: "pseudo" assignment

2) at least one assignment on every path

3) if a start or \( \varnothing \) node reads version 0 of a variable: potential danger (uninitialized \( v \))
If block A dominates block B:

No φ node is needed in path from A to B.

A: b.b. with a definition

B: b.b. with a use

A path from A to B (no further assignment to x along this path)
\( A \text{ dom } B \) : A dominates B

\( A \gg B \) : weak domination
\((A \text{ weakly dominates } B)\)

\( B \gg B \)
\( A \gg A \)

All nodes \( X \) weakly dominate them selves

\( A \gg B \) : strong domination
\( A \gg B : A \uplus B \)
definition (in A) $\Rightarrow$ use (in B)

def $\not\Rightarrow$ use

$B_1 \not\rightarrow B$

At least two paths that reach $B$ with different definitions

(path 1, path 2)

($B_1, B_2$: last definitions along path to $B$)
def di dominates use uj

if di is in a block B that dominates b.b. X that contains the use uj.

There are multiple def that start a (d_1, ..., d_k) path to u, d_i / u
let’s look at this situation in more depth (limit to 3 definitions)

\[ \text{d}_1 \text{ dominates all nodes in this region} \]

\[ \text{d}_2 \text{ dom.} \]

\[ \text{d}_3 \text{ dom} \]

\[ \text{not dominated by d}_1 \]

\[ \text{not dominated by d}_2 \]

\[ \text{not dominated by d}_3 \]

\[ \text{dominance frontier} \]

\[ \text{U} \]
We are looking for nodes $x, x', \ldots$ with two properties:

(i) $x, x', \ldots$ are not dominated by definition $d_1, d_2, \ldots$

(ii) predecessor of $x, x', \ldots$ are dominated

These nodes form the "dominance frontier" $x, x'$ must be on a path $d_1 \rightarrow u \rightarrow d_2 \rightarrow u$

$x$ may be $u$