CS 473: COMPILER DESIGN
STATIC SINGLE ASSIGNMENT
Many dataflow analyses (liveness, reaching definitions, etc.) are about relating variable uses to variable definitions.

What if there was exactly one definition for every variable?

\[
\begin{align*}
  a &= x + y \\
  b &= a - 1 \\
  a &= y + b \\
  b &= x \times 4 \\
  a &= a + b
\end{align*}
\]
Static Single Assignment

- Many dataflow analyses (liveness, reaching definitions, etc.) are about relating variable uses to variable definitions
- What if there was exactly one definition for every variable?

\[
\begin{align*}
a &= x + y \\
b &= a - 1 \\
a &= y + b \\
b &= x \times 4 \\
a &= a + b
\end{align*}
\]

\[
\begin{align*}
a_1 &= x + y \\
b_1 &= a - 1 \\
a_2 &= y + b \\
b_2 &= x \times 4 \\
a_3 &= a + b
\end{align*}
\]
Many dataflow analyses (liveness, reaching definitions, etc.) are about relating variable uses to variable definitions.

What if there was exactly one definition for every variable?

\[
\begin{align*}
a &= x + y \\
b &= a - 1 \\
a &= y + b \\
b &= x \times 4 \\
a &= a + b
\end{align*}
\]

\[
\begin{align*}
a_1 &= x + y \\
b_1 &= a_1 - 1 \\
a_2 &= y + b_1 \\
b_2 &= x \times 4 \\
a_3 &= a_2 + b_2
\end{align*}
\]

We can tell by looking at a use exactly which def it refers to.

*Static Single Assignment* IR: one defining instruction for each variable
  - *Dynamically*, the def might run multiple times, with different values each time.
Static Single Assignment

• What if there was exactly one definition for every variable?

\[
\begin{align*}
    x &= 0 \\
    y &= x + 1 \\
    \text{test:} & \\
    \text{if } (x \geq 10) \text{ goto done} \\
    x &= x + 1 \\
    y &= x \times 4 \\
    \text{goto test}
\end{align*}
\]

\[
\begin{align*}
    x_1 &= 0 \\
    y &= x + 1 \\
    \text{test:} & \\
    \text{if } (x \geq 10) \text{ goto done} \\
    x_2 &= x + 1 \\
    y &= x \times 4 \\
    \text{goto test}
\end{align*}
\]

• Exercise: Which definition of \( x \) (\( x_1 \) or \( x_2 \)) does each use of \( x \) refer to?
Static Single Assignment

• What if there was exactly one definition for every variable?

\[
x = 0 \\
y = x + 1 \\
test: \\
\text{if (}x \geq 10\text{) goto done} \\
x = x + 1 \\
y = x * 4 \\
goto test
\]

\[
x_1 = 0 \\
y = x_1 + 1 \\
test: \\
\text{if (}x_1 \geq 10\text{) goto done} \\
x_2 = x_1 + 1 \\
y = x_2 * 4 \\
goto test
\]

• In general, this will be a problem at join points in the CFG, where different definitions come in along different paths.
Phi Functions

- Solution: “ϕ (phi) function”
  - Chooses among different versions of a variable based on the path by which control enters the phi node
  - Doesn’t exist in any source or target language; we’ll translate it into a bunch of moves later in the compiler

```plaintext
x = 0
y = x + 1
test:
if (x >= 10) goto done
x = x + 1
y = x * 4
goto test
```

```plaintext
x_1 = 0
y = x_1 + 1
test:
if (x_2 >= 10) goto done
x_2 = x_2 + 1
y = x_2 * 4
goto test
```
Phi Functions

• Solution: “ϕ (phi) function”
  – Chooses among different versions of a variable based on the path by which control enters the phi node
  – Doesn’t exist in any source or target language; we’ll translate it into a bunch of moves later in the compiler

\[ x = 0 \]
\[ y = x + 1 \]
\[ \text{test:} \]
\[ \text{if } (x \geq 10) \text{ goto done} \]
\[ x = x + 1 \]
\[ y = x \times 4 \]
\[ \text{goto test} \]

\[ x_1 = 0 \]
\[ y = x_1 + 1 \]
\[ \text{test:} \]
\[ x_3 = \phi(x_1, x_2) \]
\[ \text{if } (x_3 \geq 10) \text{ goto done} \]
\[ x_2 = x_3 + 1 \]
\[ y = x_2 \times 4 \]
\[ \text{goto test} \]
Phi Functions

• Static Single Assignment (SSA): every definition node defines a different variable
• We can tell where a use is defined just by looking at the variable name
• At join points, when multiple definitions might reach, we insert phi functions (ϕ) to pick the right value

\[
\begin{align*}
  x &= 0 \\
  y &= x + 1 \\
  \text{test:} \\
  \text{if (} x \geq 10 \text{) goto done} \\
  x &= x + 1 \\
  y &= x \times 4 \\
  \text{goto test}
\end{align*}
\]

\[
\begin{align*}
  x_1 &= 0 \\
  y &= x_1 + 1 \\
  \text{test:} \\
  x_3 &= \phi(x_1, x_2) \\
  \text{if (} x_3 \geq 10 \text{) goto done} \\
  x_2 &= x_3 + 1 \\
  y &= x_2 \times 4 \\
  \text{goto test}
\end{align*}
\]
Questions

Top
Converting to SSA

- Simple transformation: insert a phi function at every join point (node with at least two predecessors), then number every def and change every use to the def that reaches it.

- But every phi function is a new def, which means a new variable – that’s a lot of new variables! (which makes reg. alloc harder)

- We’d like to insert phi functions only where necessary – that is, where two or more different definitions of a variable reach the same point.

- We can figure this out using the dominator tree!
Dominance Frontier

Goal: find the first place(s) where each variable definition might overlap with other definitions of the same variable

Observation: there are no join points in the nodes dominated by a definition

2: x = a + b

dominated by 2
Observation: there are no join points in the nodes dominated by a definition

So the join points will be at the exits of the region dominated by the definition: the *dominance frontier*
Observation: there are no join points in the nodes dominated by a definition

So the join points will be at the exits of the region dominated by the definition: the dominance frontier
So the join points will be at the exits of the region dominated by the definition: the dominance frontier.

It’s enough to put phi functions at the dominance frontier of every definition in the program.
Converting to SSA

• We’d like to insert phi functions only where necessary – that is, where two or more different definitions of a variable reach the same point

• It’s enough to put phi functions at the dominance frontier of every definition in the program

• Algorithm: compute dominance frontiers for every definition (using the dominator tree), add phi functions to each node in a dominance frontier, then repeat (since each phi function is also a definition)

• Once we’ve inserted all the phi functions, we walk through the dominator tree, renaming each variable and all its uses (a to a₁, a₂, etc.)
Converting to SSA

• Step 1: compute the dominator tree and dominance frontiers
• Step 2: insert phi functions in the dominance frontiers, repeat as necessary
• Step 3: rename every definition, and update the uses with the new names – phi functions will have multiple versions, but everything else just has one

```
x = 0
y = x + 1
test:
if (x >= 10) goto done
x = x + 1
y = x * 4
goto test
```

```
x = 0
y = x + 1
test:
x = \phi(x)
if (x >= 10) goto done
x = x + 1
y = x * 4
goto test
```
Converting to SSA

• Step 1: compute the dominator tree and dominance frontiers
• Step 2: insert phi functions in the dominance frontiers, repeat as necessary
• Step 3: rename every definition, and update the uses with the new names – phi functions will have multiple versions, but everything else just has one

\[
\begin{align*}
x &= 0 \\
y &= x + 1 \\
test: \\
&\text{if } (x \geq 10) \text{ goto done} \\
x &= x + 1 \\
y &= x \times 4 \\
goto \text{test}
\end{align*}
\]

\[
\begin{align*}
x_1 &= 0 \\
y &= x + 1 \\
test: \\
&\text{if } (x \geq 10) \text{ goto done} \\
x_2 &= x + 1 \\
y &= x \times 4 \\
goto \text{test}
\end{align*}
\]
Converting to SSA

- Step 1: compute the dominator tree and dominance frontiers
- Step 2: insert phi functions in the dominance frontiers, repeat as necessary
- Step 3: rename every definition, and update the uses with the new names – phi functions will have multiple versions, but everything else just has one

```plaintext
x = 0
y = x + 1
test:
if (x >= 10) goto done
x = x + 1
y = x * 4
goto test
```

```plaintext
x_1 = 0
y = x_1 + 1
test:
x_3 = \phi(x)
if (x_3 >= 10) goto done
x_2 = x_3 + 1
y = x_2 * 4
goto test
```
Questions

Top
SSA Optimizations

• In SSA, every use has exactly one definition
  – And every definition dominates its uses!
• This makes lots of optimizations easy:

• Constant propagation: if a def is $x = c$, where $c$ is a constant, then delete it and replace all uses of $x$ with $c$
• Copy propagation: if a def is $x = y$, where $y$ is a variable, then delete it and replace all uses of $x$ with $y$

• Without SSA, we’d have to do reaching definition analysis, and only replace uses when we know the def is the only one that reaches
• In SSA, we’ve done the analysis in the IR!
Static Single Assignment: Summary

• Instead of computing relationships between uses and definitions for every optimization, we can encode it in the IR!

• One assignment statement per variable, so every use is immediately tied to its single definition

• Need to insert phi functions at join points, to resolve places where a use might refer to multiple definitions

• Makes analysis and optimization a lot easier, and is used in many compiler IRs (LLVM, gcc, etc.)
Thank you!

• It’s been a pleasure having you in class!

• Final project and remaining in-class exercises due at the end of the day today

• Please fill out course evaluations by Sunday if you haven’t already

• If you liked this class, or want to learn more about how to describe how a language *should* work, consider taking CS 476: Programming Language Design!