Static Single Assignment Form
Announcements

- HW3 is out
  - Standard homework: XTA
    - Submitty page is up
  - Alternative homework: Points-to over Pcode
    - No Submitty page
  - Please choose one and let me know by the end of week

- HW2 questions?
Announcements

Office hours

- Mondays 4-5pm on Webex
- Fridays 4-5pm on Webex
- Mondays and Thursdays 2pm - in SAGE 3713
Outline of Today’s Class

- SSA, introduction and motivation

- Constructing SSA
  - Step 1: Place $\phi$ functions
  - Step 2: Renaming variables
Static Single Assignment (SSA)

- SSA form transforms the 3-address code of a program so that each variable is defined exactly once (statically)
  - A sparse, efficient representation
  - Universally applied technique in compilers and static analysis
- Cytron et al. “Efficiently Computing Static Single Assignment Form and the Control Dependence Graph”, TOPLAS 1991
SSA

- **Multiple definitions for a given use**
  - E.g., def-use chains for `local_24`: (5,6) and (9,6)
  - Inconvenient
Without SSA

- Multiple definitions of a variable
  - Potentially expensive (O(n^2) def-use chains)

1. \( A = \ldots \)
2. \( A = \ldots \)
3. \( A = \ldots \)
4. \( A = \ldots \)
5. \( A = \ldots \)
6. \( A = \ldots \)
7. \( \ldots = \ldots A \)
8. \( \ldots = A \)
9. \( \ldots = A \)
10. \( \ldots = f(A) \)
11. \( \ldots = A \)
12. \( \ldots = A+5 \)
With SSA

- Single definition per variable
  - $O(n)$ # def-use chains

1. $A_1 = \ldots$
2. $A_2 = \ldots$
3. $A_3 = \ldots$
4. $A_4 = \ldots$
5. $A_5 = \ldots$
6. $A_6 = \ldots$
7. $\ldots = A_7$
8. $\ldots = A_7$
9. $\ldots = A_7$
10. $\ldots = f(A_7)$
11. $\ldots = A_7$
12. $\ldots = A_7+5$

$A_7 = \text{phi}(A_1, A_2, A_3, A_4, A_5, A_6)$
Def-Use Reasoning

- **a**: (1,3), (1,4), (1,6), (1,8), (1,9), (10,11)
- **b**: (2,3), (2,5), (2,6), (2,8), (8,2), (8,10), (8,3), (8,5), (8,6)
- **c**: (3,4), (3,9)
- **d**
- **e**

1. \( a = 1 \)
2. \( b = 2 \)
3. \( c = a + b \)
4. \( d = c - a \)
5. \( d = b + d \)
6. \( d = a + b \)
7. \( e = e + 1 \)
8. \( b = a + b \)
9. \( e = c - a \)
10. \( a = b \times d \)
11. \( b = a - d \)
Easy case

\[ A = \text{INPUT}; \quad A_1 = \text{INPUT}; \]
\[ Y = A + 10; \quad \text{becomes} \quad Y = A_1 + 10; \]
\[ A = A + Y; \quad A_2 = A_1 + Y; \]

The 2 definitions of \( A \) become definitions of \( A_1 \) and \( A_2 \) respectively.
If-then-else

if (B>0)
    A = 5;
else becomes else
    A = 10;

A1 = 5;
A2 = 10;

A3 = \phi(A1,A2)

Phi function introduces a (static) definition of A, A3. If control took True arm, then A3 is A1, otherwise it is A2.
Phi Functions

Loops

\[ A = 0; \]

while \((A \leq n)\)

\[ A = A + 1; \]

If control took forward edge, \(A2\) is \(A1\); otherwise, i.e., control took back edge, it is \(A3\).
Phi Functions

- But how do we know where to place those \(\phi\) functions?
Constructing SSA

- Step 1: Place \( \text{phi} \) functions
  - Step 1a: Compute \textit{dominance frontier} of each CFG node \( N, \text{DF}(N) \)
  - Step 1b: Place \( \phi \) function at dominance frontier
    - If a node \( N \) contains assignment to variable \( A \), put a \( \phi \) function for \( A \) at \( \text{DF}(N) \): \( A \ldots = \phi(\ldots) \)

- Step 2: Rename variables
  - \( A \) at \( N \) is indexed: e.g., \( A_k \)
  - \( A \) at \( \text{DF}(N) \) is indexed; \( \phi \) function arguments fill up: e.g., \( A_m = \phi(A_k, \ldots) \)
Dominators

- Let $M$ and $N$ be nodes in CFG
  - Recall that CFG has a single entry node $1$
  - $M$ dominates $N$ (written $M \geq N$) if every path from entry node $1$ to $N$ goes through $M$
  - $M$ strictly dominates $N$ if $M \geq N$ and $M \neq N$

- If $M > N$ and there is no $K$ such that $M > K$ and $K > N$ we say that $M$ immediately dominates $N$
Dominator Tree

- Dominator relation gives rise to the dominator tree
  - Nodes are the CFG nodes
  - There is an edge from M to N if M is the immediate dominator of N (Note: Not the CFG edges!)
Dominator Tree

- Known algorithms
- Other applications
- Tree:

1. \( a = 1 \)
2. \( b = 2 \)
3. \( c = a + b \)
4. \( d = c - a \)
5. \( d = b + d \)
6. \( d = a + b \)
7. \( e = e + 1 \)
8. \( b = a + b \)
9. \( e = c - a \)
10. \( a = b \cdot d \)
11. \( b = a - d \)
Where Do We Place $\phi$’s?

- A naïve approach: place a $\phi$ function for variable $A$ at every merge node
  - May end up with many unnecessary $\phi$’s!
- A better approach: place a $\phi$ function for $A$ only at those nodes that merge two (or more) distinct paths that define $A$
N is in the dominance frontier of M iff
- M dominates a predecessor of N
- M does not dominate N

Dominance Frontier of M
Dominance Frontier

- DF(B_2): \{2\}
- DF(B_3): \{b_2, y\}
- DF(B_4): \{b_2, y\}
- DF(B_5): \{b_2, y\}
- DF(B_6): \{2\}

1. a = 1
2. b = 2
3. c = a + b
4. d = c - a
5. d = b + d
6. d = a + b
7. e = e + 1
8. b = a + b
9. e = c - a
10. a = b * d
11. b = a - d
Constructing SSA

Step 1: Place \textit{phi} functions

- Step 1a: Compute dominance frontier of each CFG node $N$, $DF(N)$
- Step 1b: Place \textit{phi} function at dominance frontier
  - If a node $N$ contains assignment to variable $A$, put a \textit{phi} function for $A$ at $DF(N)$: $A_{\ldots} = \textit{phi}(\ldots)$

Step 2: Rename variables

- $A$ at $N$ is indexed: e.g., $A_k$
- $A$ at $DF(N)$ is indexed; \textit{phi} function arguments fill up: e.g., $A_m = \textit{phi}(A_k, \ldots)$
Place phi Functions

- Let $S$ be set of nodes that define variable $A$
- Place $A_{\ldots} = \phi(\ldots)$ at each node in $DF(S)$
  - But the phi node adds a definition of $A$

- Thus, we have to iterate
  - $DF_1 = DF(S)$
  - $DF_2 = DF(S \cup DF_1)$
  - ...
  - $DF_n = DF(S \cup DF_{n-1})$
Place phi Functions

- DF(B₂): \( \phi \)
- DF(B₃): \( \phi B₂ \)
- DF(B₄): \( \phi B₃ \)
- DF(B₅): \( \phi B₂ \)
- DF(B₆): \( \phi \phi \)

1. \( a = 1 \)
2. \( b = 2 \)
3. \( c = a + b \)
4. \( d = c - a \)
5. \( d = b + d \)
6. \( d = a + b \)
7. \( e = e + 1 \)
8. \( b = a + b \)
9. \( e = c - a \)
10. \( a = b \times d \)
11. \( b = a - d \)
Place phi Functions

- DF(B₃): {B₂}
- DF(B₄): {B₃}
- DF(B₅): {B₂}

\[
S(b) = \{B₂, B₅, B₆\} \cup \{B₂\}
\]

- DF(S(b)): \(DF(B₁) \cup DF(B₃) \cup DF(B₅) = \{B₂\}\)
- DF(S(d)): \(\{B₂, B₃\}\)
- DF(S(e)): \(\{B₂, B₃\}\)

1. \(a = 1\)
2. \(b = 2\)
3. \(c = a + b\)
4. \(d = c - a\)
5. \(d = b + d\)
6. \(d = a + b\)
7. \(e = e + 1\)
8. \(b = a + b\)
9. \(e = c - a\)
10. \(a = b * d\)
11. \(b = a - d\)
Place phi Functions

- DF(B₃): \{B₂\}
- DF(B₄): \{B₃\}
- DF(B₅): \{B₂\}

- DF(S(b)): \{B₂\}
- DF(S(d)): \{B₂, B₃\}
- DF(S(e)): \{B₂, B₃\}

1. a = 1
2. b = 2
3. c = a + b
4. d = c - a
5. d = b + d
6. d = a + b
7. e = e + 1
8. b = a + b
9. e = c - a
10. a = b * d
11. b = a - d
Another Round?

- DF(B₃): \{B₂\}
- DF(B₄): \{B₃\}
- DF(B₅): \{B₂\}

- DF⁺(S(b)): \{B₂\}
- DF⁺(S(d)): \{B₂, B₃\}
- DF⁺(S(e)): \{B₂, B₃\}

1. a = 1
2. b = 2
3. c = a + b
4. d = c - a
5. d = a + b + d
6. d = a + b
7. e = e + 1
8. b = a + b
9. e = c - a
10. a = b * d
11. b = a - d
Another Example

1. A = ... if X > Y
2. ...
3. ...
4. A = 1
5. A = 1

\[ A = \phi(\ldots) \]

6. \[ A = \phi(\ldots) \]

7. ...

\[ DF(S(A)) = DF(1, 4, 5) = \{6\} \]
\[ DF_1(S(A)) = \{6\} \]
\[ DF_2(S(A) \cup \{6\}) = \{6, 7\} \]
\[ DF^+(S(A)) = \{6, 7\} \]
Constructing SSA

- **Step 1: Place phi nodes**
  - Step 1a: Compute dominance frontier of each CFG node \( N, DF(N) \)
  - Step 1b: Place phi nodes at dominance frontier
    - If a node \( N \) contains assignment to variable \( A \), put a phi node for \( A \) at \( DF(N) \): \( A_{\ldots} = \text{phi}(\ldots) \)

- **Step 2: Rename variables**
  - \( A \) at \( N \) is indexed: e.g., \( A^{!k} \)
  - \( A \) at \( DF(N) \) is indexed; phi function arguments fill up: e.g., \( A^{!m} = \text{phi}(A^{!k},\ldots) \)
Rename Variables

- Depth-first traversal of dominator tree
  - Keep stack $\text{Stack}(A)$ with current def per variable
  - SEARCH($N$) visits dominator tree node $N$
    - If $N$ defines $A$, push new $#$ on top of $\text{Stack}(A)$
    - Recursive calls SEARCH(Child) use that $#$
    - Before SEARCH($N$) exits, pop $#$ off $\text{Stack}(A)$

- Amazingly, it works!
rename Variables

1. \( a = 1 \)
2. \( b = 2 \)
3. \( c = a + b \)
4. \( d = c - a \)
5. \( d = b + d \)
6. \( d = a + b \)
7. \( e = e + 1 \)
8. \( b = a + b \)
9. \( e = c - a \)
10. \( a = b * d \)
11. \( b = a - d \)
1. \(a1 = 1\)
2. \(b1 = 2\)
3. \(c = a + b\)
4. \(d = c - a\)
5. \(d = b + d\)
6. \(d = a + b\)
7. \(e = e + 1\)
8. \(b = a + b\)
9. \(e = c - a\)
10. \(a = b * d\)
11. \(b = a - d\)
**Rename Variables**

1. $a_1 = 1$
2. $b_1 = 2$

3. $b_2 = \phi(b_1,?)$
4. $d_1 = \phi(?,?)$
5. $e_1 = \phi(?,?)$

6. $d = a + b$
7. $e = e + 1$

8. $b = a + b$
9. $e = c - a$

10. $a = b * d$
11. $b = a - d$
1. a1 = 1
2. b1 = 2
3. c1 = a1 + b2
4. d2 = c1 - a1
5. d4 = b2 + d3
6. d = a + b
7. e = e + 1
8. b = a + b
9. e = c - a
10. a = b * d
11. b = a - d
Rename Variables

1. \( a_1 = 1 \)
2. \( b_1 = 2 \)

\( b_2 = \phi(b_1, ?) \)
\( d_1 = \phi(? , ?) \)
\( e_1 = \phi(? , ?) \)

3. \( c_1 = a_1 + b_2 \)
4. \( d_2 = c_1 - a_1 \)

5. \( d_4 = b_2 + d_3 \)
6. \( d_5 = a_1 + b_2 \)
7. \( e_3 = e_2 + 1 \)

8. \( b = a + b \)
9. \( e = c - a \)
10. \( a = b \cdot d \)
11. \( b = a - d \)
Rename Variables

1. a1 = 1
2. b1 = 2

b2 = phi(b1,b3)
d1 = phi(?,d3)
e1 = phi(?,e4)
3. c1 = a1+b2
4. d2 = c1-a1

5. d4 = b2+d3
6. d5 = a1+b2
7. e3 = e2+1
8. b3 = a1+b2
9. e4 = c1-a1
10. a = b*d
11. b = a-d
 Rename Variables

1. $a_1 = 1$
2. $b_1 = 2$

3. $c_1 = a_1 + b_2$
4. $d_2 = c_1 - a_1$

5. $d_4 = b_2 + d_3$
6. $d_5 = a_1 + b_2$
7. $e_3 = e_2 + 1$
8. $b_3 = a_1 + b_2$
9. $e_4 = c_1 - a_1$
10. $a_2 = b_3 * d_4$
11. $b_4 = a_2 - d_4$

$\phi$ denotes a function applied to its arguments.
Properties of SSA

- Every variable has a single assignment
- Assignment dominates (non-phi) uses
- Huge impact on compiler design