Assignment 1

SSA Construction/Destruction

Luca Della Toffola
luca.dellatoffola@inf.ethz.ch
SSA form **construction** and **destruction**
Assignment 1 – SSA Construction

3 steps

1. Compute
   a) Dominators
   b) Dominator Tree
   c) Dominance Frontier

2. Insertion of phi-nodes

3. Variables re-numbering
Assignment 1 – SSA Destruction

2 steps
1. Replace phi-nodes with variable copies
2. Phi-nodes removal
Outline

• Brief review of dominators and related topics
  – Algorithms for computation
• SSA Construction and Destruction
  – Where to look in the framework
• Resources
Dominator

\[ \text{DOM}(n) = \text{a node } y \text{ dominates } n \text{ if } y \text{ lies in every path from the root to } n. \]

A node \( n \) dominates itself
The root node dominates all the nodes
Strict Dominator

\[ \text{SDOM}(n) = \text{a node } y \text{ strictly dominates } n \text{ if } y \text{ dominates } n \text{ and } n \neq y. \]

\[ \text{DOM}(n) \text{ but } \text{without } n \text{ itself} \]
Immediate Dominator

**IDOM(n)** = for a node *n*, the node *y* which dominates *n* and is **closest** to it.

There is **only one** immediate dominator

The **closest strict dominator**
root

A

B

C

D

E

F

G

H

<table>
<thead>
<tr>
<th>n</th>
<th>IDOM(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
</tr>
<tr>
<td>B</td>
<td>{r}</td>
</tr>
<tr>
<td>C</td>
<td>{A}</td>
</tr>
<tr>
<td>D</td>
<td>{r}</td>
</tr>
<tr>
<td>E</td>
<td>{B}</td>
</tr>
<tr>
<td>F</td>
<td>{C}</td>
</tr>
<tr>
<td>G</td>
<td>{D}</td>
</tr>
<tr>
<td>H</td>
<td>{r}</td>
</tr>
</tbody>
</table>
Resources

- References in the assignment description
Iterated dominator algorithm

Basic strategy: solve iteratively the following

\[ \text{Dom}(n_0) = \{n_0\} \]

\[ \text{Dom}(n) = \left( \bigcap_{p \in \text{preds}(n)} \text{Dom}(p) \right) \cup \{n\} \]

Walk over the CFG, repeat until no changes
Iterated dominator algorithm

```python
for n in nodes:
    DOM(n) = {nodes}

while changes:
    for n in nodes:
        dom = intersect(DOM(n.preds))
        new_set = union(dom, n)
        if new_set != DOM(n):
            DOM(n) = new_set
```
Computing dominators

• **Iterated dominator algorithm**
  – Simple but inefficient
  – Needs for data-structures optimizations

• **Efficient algorithm** described in *Cooper et al.*
  – Still very simple
  – Not shown today
Dominator Tree

• Contains the nodes of the CFG with edges that reflect the dominance relation

• Each node \( n \) is a child of its \( \text{IDOM}(n) \)

• **Bottom-up** traversal results as \( \text{DOM}(n) \)
Dominator Tree
Dominator Tree

Why we need the dominator tree?

• *Cooper et al.* use the DT to calculate the DF

• *Cytron et al.* use the DT to rename variables
Dominance Frontier and SSA

For any assignment a phi-node needs to be inserted in the DF of the BB containing the assignment.

Phi-nodes are assignments too!
Dominance Frontier

$$DF(n) = \text{is the set of all nodes } y \text{ such that } n \text{ dominates a predecessor of } y \text{ but does not strictly dominate } y$$
Dominance Frontier

DF(A) = {} 

DF(B) = { D } 

DF(C) = { C D } 

DF(D) = {} 

DF(E) = { C } 

**n** dominates a predecessor of **y** & **n** not strictly dominates **y**
Dominance Frontier
Dominance Frontier
Dominance Frontier
Dominance Frontier

• Don’t ask directly for $DF(n)$

• Ask for those nodes $y$ where $n$ in $DF(y)$

• Simple algorithm based on three observations
Dominance Frontier

- Nodes in the DF are the joint points
- Joint points have multiple predecessors
Dominance Frontier

Predecessors of a joint point $j$ have $j$ in their DF unless they dominate $j$. 
Dominance Frontier

Dominators of predecessors of a joint point $j$ have $j$ in their DF unless they dominate $j$. 

![Diagram showing dominance relationships]
Dominance Frontier

```
for n in nodes:
    if | n.pred | >= 2:
        for p in n.pred:
            r = p
            while r != IDOM(n):
                DF(r) += n
            r = IDOM(r)
```

```
n = D
DF(B) = { D }  n = C
DF(E) = { C }
DF(C) = { D }  DF(C) = { C D }
DF(A) = { }    DF(A) = { }
```

```
DF(A) = { }  DF(B) = { D }
DF(B) = { }  DF(C) = { C D }
DF(C) = { }  DF(E) = { C }
DF(D) = { }  DF(D) = { }
```
A1 – SSA Form Construction

1. Build $\text{DOM}(BB)$, $\text{DT}(CFG)$, and $\text{DF}(BB)$
2. Insert phi-nodes at the $\text{DF}(BB)$ nodes
3. Renumber variables
A1 – SSA Form Destruction

1. Phi-nodes substitution using assignments
2. New assignments at the end of BB predecessor
x = 2;
if (y == 0) {
    x = 1;
} else {
    write(-1);
}
write(x);
Control Flow Graph

• Implemented in A0
  – One CFG per method
  – See classes cd.cfg.CFGBuilder, cd.ir.BasicBlock
Dominance Frontier

**CFG**

- BB1
  - DF = \{\}
- BB2
  - DF = \{ BB4 \}
- BB3
  - DF = \{ BB4 \}
- BB4
  - DF = \{\}

**Dominator Tree**

- BB1
  - BB2
  - BB3
  - BB4
Framework

Computation of $\text{IDOM}(BB)$ and $\text{DT}(\text{CFG})$ in $\text{cd.cfg.Dominator}$

<table>
<thead>
<tr>
<th>cd.ir.BasicBlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB dominatorTreeParent</td>
</tr>
<tr>
<td>List&lt;BB&gt; dominatorTreeChildren</td>
</tr>
<tr>
<td>Set&lt;BB&gt; dominanceFrontier</td>
</tr>
</tbody>
</table>
Insert phi-nodes

BB1
x = 2
COND (y==0)

true

BB2
x = 1

false

BB3
write(-1)

BB4
x = \phi(x, x)
write(x);
Renaming variables

BB1
x = 2
COND (y==0)

true

BB2
x = 1

false

BB3
write(-1)

BB4
x = phi(x, x)
write(x);
Renaming variables

BB1
\( x_1 = 2 \)
COND (y==0)

true

BB2
x = 1

false

BB3
write(-1)

BB4
x = phi(x, x)
write(x);
Renaming variables

BB1
\[ x_1 = 2 \]
\[ \text{COND (y==0)} \]

true

BB2
\[ x_2 = 1 \]

BB3
write(-1)

false

BB4
\[ x = \phi(x, x) \]
write(x);

write(C1)
Renaming variables

BB1
\[ x_1 = 2 \]
COND (y==0)

true

BB2
\[ x_2 = 1 \]

BB3
write(-1)

false

BB4
\[ x = \phi(x_2, x) \]
write(x);
Renaming variables

\[ x_1 = 2 \]
COND \( y == 0 \)

true

\[ x_2 = 1 \]
BB2

false

write(-1)
BB3

\[ x_3 = \phi(x_2, x) \]
write\( (x_3); \)
BB4
Renaming variables

BB1
\[ x_1 = 2 \]
COND (y==0)

BB2
\[ x_2 = 1 \]

BB3
write(-1)

BB4
\[ x_3 = \phi(x_2, x_1) \]
write(x_3);
Renaming variables

\[
\begin{align*}
BB1 & : x_1 = 2 \\
    & \text{COND (y==0)}
\end{align*}
\]

\[
\begin{align*}
true & \quad \rightarrow \quad false
\end{align*}
\]

\[
\begin{align*}
BB2 & : x_2 = 1 \\
BB3 & : \text{write(-1)}
\end{align*}
\]

\[
\begin{align*}
BB4 & : x_3 = \phi(x_2, x_1) \\
    & \text{write}(x_3);
\end{align*}
\]
Framework

- **VariableSymbol** has new field: **version**
- New **constructor** to create a **new version**
  - VariableSymbol(VariableSymbol v0sym, int version)
Framework

Phi nodes in class \texttt{cd.ir.Phi}

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{cd.ir.Phi} \\
\texttt{VariableSymbol v0sym} \\
\texttt{VariableSymbol lhs} \\
\texttt{List<Expr> rhs} \\
\hline
\end{tabular}
\end{center}

\begin{align*}
\text{write}(x) \\
\begin{split}
\text{x}_3 &= \text{phi}(x_1, x_2) \\
\end{split}
\end{align*}
Framework

**BasicBlock** contains all the **attached** phi-nodes

<table>
<thead>
<tr>
<th>cd.ir.BasicBlock</th>
<th>Map&lt;VS,Phi&gt; phis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Map key = Original Var. Symb. = $x_0$</td>
</tr>
</tbody>
</table>

Remember to add the **new symbol** to the method **locals**
Removing phi-nodes

BB1
\[ x_1 = 2 \]
\[ \text{COND (y==0)} \]

true

BB2
\[ x_2 = 1 \]

false

BB3
write(-1)

BB4
\[ x_3 = \text{phi}(x_2, x_1) \]
write(x_3);
Removing phi-nodes

BB1
\[ x_1 = 2 \]
COND (y==0)

true

BB2
\[ x_2 = 1 \]
\[ x_3 = x_2 \]

false

BB3
\[ x_3 = x_1 \]
write(-1)

BB4
\[ x_3 = \phi(x_2, x_1) \]
write(x_3);
Removing phi-nodes

BB1
$x_1 = 2$
COND (y==0)

BB2
$x_2 = 1$

BB4
$x_3 = \text{phi}(x_2, x_1)$
write(x_3);

true
false
Removing phi-nodes

Assignments $x_3 = x_1$ is redundant

$x_3 = x_1$

$\ldots$

$x_3 = x_2$

BB1
$x_1 = 2$
COND ($y == 0$)
$x_3 = x_1$

true

BB2
$x_2 = 1$
$x_3 = x_2$

false

BB4
$x_3 = \phi(x_2, x_1) \overline{write(x_3)}$
Removing phi-nodes

Solution: insert empty basic block
SSA Destruction

• Ensure that if a block in your CFG has *multiple predecessors*, each of those predecessors has *only one successor*  
  – Attention to *IfElse* and *WhileLoop*  
  – If required add an empty block

• *Already implemented* in the *A1 fragment*
Summary

• How to compute $\text{DOM}(n)$ and $\text{DF}(n)$
  – Algorithms shown in literature

• How to implement SSA form inside Javali

• Problems behind SSA destruction