Advanced Compiler Design – Assignment 1

SSA Construction & Destruction

Michael Fäs
michael.faes@inf.ethz.ch

(Original Slides by Luca Della Toffola)
SSA Pipeline

1. Build the control-flow graph of the program
2. Convert the program into **SSA form**
3. Apply optimizations
4. Convert the program back to **non-SSA form**
5. Generate code from the control-flow graph
Assignment 1 Tasks

1. Build the control-flow graph of the program
2. Convert the program into SSA form
3. Apply optimizations
4. Convert the program back to non-SSA form
5. Generate code from the control-flow graph

(Due date: Mar 18, 2015)
SSA Construction

1. Compute **Dominance Frontier**
   - Using Dominator Tree

2. Insert **ϕ-operations** where necessary
   - In join nodes with different incoming assignments
   - Use dominance frontier

3. Add **versions** to variables (renaming)
SSA Destruction

1. Replace $\phi$-operations with copy statements in predecessor blocks
Assignment 1

DOMINANCE RELATIONS REVIEW
Dominator

A (control flow graph) node \( y \) dominates \( n \) if \( y \) lies in every path from the root \( n_0 \) to \( n \).

\[
\text{DOM}(n) = \text{the set of all dominators of } n
\]

Note:
- Every node dominates itself
- The root node dominates all nodes
$n \quad \text{DOM}(n)$
Strict Dominator

A node $y$ strictly dominates $n$ if $y$ dominates $n$ and $n \neq y$.

$$ \text{SDOM}(n) = \text{the set of strict dominators of } n $$

Note:

- $\text{SDOM}(n) = \text{DOM}(n) \setminus \{n\}$
\begin{align*}
\begin{array}{c|c}
 n & \text{SDOM}(n) \\
\hline
 A & \{\text{root}\} \\
 B & \{\text{root}\} \\
 C & \{\text{root, A}\} \\
 D & \{\text{root}\} \\
 E & \{\text{root, B}\} \\
 F & \{\text{root, A, C}\} \\
 G & \{\text{root, D}\} \\
 H & \{\text{root}\}
\end{array}
\end{align*}
Immediate Dominator

Node $y$ immediately dominates $n$ if $y$ is the closest strict dominator of $n$ on any path from $n_0$ to $n$.

\[ \text{IDom}(n) = \text{the immediate dominator of } n \]

Note:
- Every node (except the root) has exactly one immediate dominator.
<table>
<thead>
<tr>
<th>n</th>
<th>IDOM(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{root}</td>
</tr>
<tr>
<td>B</td>
<td>{root}</td>
</tr>
<tr>
<td>C</td>
<td>{A}</td>
</tr>
<tr>
<td>D</td>
<td>{root}</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>{root}</td>
</tr>
</tbody>
</table>
Dominator Tree

• Contains all 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 in $\text{DOM}(n)$
CFG

Dominator Tree
DOMINATOR TREE CONSTRUCTION
Iterative Dominator Algorithm

Idea: Solve the following data-flow problem:

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

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

- For \( n \neq n_0 \), initialize \( \text{DOM}(n) \) with all nodes
- Update \( \text{DOM}(n) \) according to equations
- Walk over the CFG, repeat until no changes.
Dominator Algorithm: Example

\[
\text{DOM}(\text{root}) = \{ \text{root} \}
\]

for \( n \) in nodes \( \setminus \{ \text{root} \} \):
\[
\text{DOM}(n) = \text{nodes}
\]

while changed:
  for \( n \) in nodes:
    \( pre = \text{intersect}(\text{DOM}(\text{preds}(n))) \)
    \( new = \text{union}(pre, n) \)
    \( \text{DOM}(n) = new \)
Computing Dominators

• **Shown algorithm**
  – Simple but not very efficient
  – How to get $\text{IDom}(n)$ from $\text{Dom}(n)$?

• **Efficient algorithm** described in *A Simple, Fast Dominance Algorithm* by Cooper et al.
  – Same idea, still very simple
  – Not required for getting full marks
  – Also covers shown algorithm
Assignment 1 – SSA Construction

DOMINANCE FRONTIER
CONSTRUCTION
Dominance Frontier

\[ DF(n) = \text{all nodes } y \text{ such that } n \text{ dominates a predecessor of } y \text{ but does not strictly dominate } y. \]

Intuition:
The Dominance Frontier of \( n \) is where \( n \)'s dominance “ends”.
Dominance Frontier: Intuition
Dominance Frontier: Example

\[ DF(A) = \{\} \]

\[ DF(B) = \{D\} \]

\[ DF(C) = \{C, D\} \]

\[ DF(D) = \{\} \]

\[ DF(E) = \{C\} \]

*dominates a predecessor of* \( y \) *and* *not strictly dominates* \( y \)
Computing the DF

• Idea:
  – Given node $n$, do not compute $DF(n)$ directly
  – Instead, find nodes $y$ for which $n$ is in $DF(y)$

• Simple algorithm based on two observations
Computing the DF: Observation 1

Predecessors of a node $n$ have $n$ in their DF unless they dominate $n$. 
Computing the DF: Observation 2

Dominators of predecessors of a node $n$ have $n$ in their DF unless they dominate $n$. 

![Graph showing dominators and their relationships](image)
for \( n \) in nodes:
   for \( p \) in preds(\( n \)):
      \( r = p \)
      while \( r \neq \text{IDOM}(n) \):
         \( \text{DF}(r) += n \)
         \( r = \text{IDOM}(r) \)

\( n = D \)
\( n = C \)
\( \text{DF}(B) = \{D\} \)
\( \text{DF}(C) = \{D\} \)
\( \text{DF}(E) = \{C\} \)
\( \text{DF}(D) = \{\} \)

\( \text{DF}(A) = \{\} \)
\( \text{DF}(B) = \{D\} \)
\( \text{DF}(C) = \{C,D\} \)
\( \text{DF}(E) = \{C\} \)
\( \text{DF}(D) = \{\} \)
**ϕ-Function Insertion**

- ϕ-functions need to be inserted in all join nodes with **different incoming assignments**

  Assignments require a ϕ-function to be inserted in the **DF nodes of the block containing the assignment**

- ϕ-operations are assignments too!

![Diagram](image)
DF and $\phi$-Functions: Example

\[
x = \phi(x, x, x) \\
write(x)
\]
Variable Versioning

• Need to add versions to all variables
  – S.t. each version is assigned only once (statically)

• Make sure that always the “newest” version of a variable is used
  – Traverse the CFG, keep track of current (newest) and highest variable versions
    • Pro tip: Dominator tree...
  – Replace all uses of unversioned variables
    • Also in φ-functions!
Variable Versioning: Example

BB1
\( x_1 = 0 \)

BB2
\( x_2 = 1 \)

BB3
write(\( x_1 \))

BB4
\( x_3 = \phi(x_1, x_4) \)

BB5
\( x_4 = x_3 + 1 \)

BB6
\( x_5 = \phi(x_2, x_3) \)
write(\( x_5 \));
Assignment 1

SSA DESTRUCTION
SSA Destruction

• Replace $\phi$-operations with copy statements in predecessors
Removing $\phi$-Functions: Example 1

BB1
$x_1 = 0$

BB2
$x_2 = 1$
$x_3 = x_2$

BB3
write($x_1$)
$x_3 = x_1$

BB4
$x_3 = \phi(x_2, x_1)$
write($x_3$)
Removing $\phi$-Functions: Example 2

\[\begin{align*}
BB1 & \\
\quad x_1 & = 0 \\
\quad x_3 & = x_1 \\
BB2 & \\
\quad x_2 & = 1 \\
\quad x_3 & = x_2 \\
BB3 & \\
\quad x_3 & = \phi(x_2, x_1) \\
\quad \text{write}(x_3) & \\
\end{align*}\]

- Process may insert redundant assignments:
  \[x_3 = x_1 \quad \ldots \quad x_3 = x_2\]

- Solution: Insert empty blocks where necessary
Removing $\phi$-Functions: Example 2

- Process may insert **redundant** assignments:
  - $x_3 = x_1$
  - $x_3 = x_2$

- Solution: **Insert empty blocks** where necessary
  - **Already done** in CFG construction phase in A1 fragment
Assignment 1

PRACTICAL TIPS
(FRAMEWORK)
Framework: Dominator Tree

• Computation of $\text{IDOM}(n)$ and $\text{DF}(n)$ in cd.cfg.Dominator class

• Dominator tree edges directly in BasicBlock:

```plaintext
<table>
<thead>
<tr>
<th>cd.ir.BasicBlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>dominatorTreeParent: BB</td>
</tr>
<tr>
<td>dominatorTreeChildren: List&lt;BB&gt;</td>
</tr>
<tr>
<td>dominanceFrontier: Set&lt;BB&gt;</td>
</tr>
</tbody>
</table>
```

$\text{IDOM}(n)$
Framework: Variable Versions

- **VariableSymbol** has new field: version

- New constructor to create a new version:
  
  ```
  VariableSymbol(VariableSymbol v0sym, int version)
  ```
Framework: $\phi$-Operations

- $\phi$-operations represented by class `cd.ir.Phi`

```
write(x)
```

```
x_3 = \phi(x_1, x_2)
write(x_3)
```
Framework: $\phi$-Operations

- BasicBlock contains all attached $\phi$-ops

<table>
<thead>
<tr>
<th>cd.ir.BasicBlock</th>
<th>phis: Map&lt;VS,Phi&gt;</th>
</tr>
</thead>
</table>

- Map key is original variable (version 0)

- Add the new symbols to the method locals
Questions