CSC2/455 Software Analysis and Improvement
Dominator and SSA Form

Sreepathi Pai

URCS

February 5, 2020
Outline

Review

Dominator Analysis (DOM)

SSA Form

Postscript
Data flow analysis framework

- Live variable analysis
  - “Is there a read of this variable along any path?”
- Reaching Definitions
  - “Which definitions reach this use?”
- Available expressions
  - “Is this expression calculated previously and the result still usable?”
- Very Busy Expressions
  - “Are there expressions that can be precalculated?”
- Iterative data flow analysis
  - GEN, KILL, Transfer functions, Initialization
Outline

Review

Dominator Analysis (DOM)

SSA Form

Postscript
A node $n$ in the CFG dominates a node $m$ iff:
- $n$ is on all paths from entry to $m$
- by definition, a node $n$ always dominates itself

Dominators are a property of graphs
- i.e. has nothing to do with code in basic blocks
Example 1: Node with single predecessor
Example 1: Node with single predecessor (Answer)

ENTRY {ENTRY}

B1 {ENTRY, B1}

B2 {ENTRY, B1, B2}

EXIT {ENTRY, B1, B2, EXIT}
Example 2: Node with multiple predecessors

ENTRY

B1

B2 B3

B4

EXIT
Example 2: Node with multiple predecessors (Answer)
Example 3: Slightly more involved example
Example 3: Slightly more involved example (Answer)
Can we use data flow analysis to identify the dominators of a node?
Data flow analysis setup

- Domain of facts?
- GEN and KILL?
- Direction of analysis?
- Merge operator?
- Initialization?
Data flow analysis Equation

\[
\text{DOM}(n) = \{n\} \cup \bigcap_{m \in \text{pred}(n)} \text{DOM}(m)
\]

- **Initialization**
  - (for \(n \neq \text{ENTRY}\)): \(\text{DOM}(n) = N\) (where \(N\) is the set of all nodes)
  - (for \(n = \text{ENTRY}\)): \(\text{DOM}(n) = \text{ENTRY}\)
A node $m$ is post-dominated by a node $n$ iif:

- $n$ appears on every path from $m$ to EXIT.
- $n$ post-dominates itself, by definition.
Outline

Review

Dominator Analysis (DOM)

SSA Form

Postscript
Static Single Assignment (SSA) Form

- Intermediate Representation
  - Similar to 3 address code
- Each variable only written once
  - Static [in source] Single [once] assignment
- SSA form can be generated from 3 address code
  - Introduce $\phi$ functions
  - Rename variables
Example 1: Straight-line code

```
y = x + 1;
x = 2;
y = x + y + 2;
```

gets transformed to:

```
y_0 = x_0 + 1
x_1 = 2;
y_1 = x_1 + y_0 + 2;
```

From this example, when should we rename variables?
Example 2: Branches

\[
y = x + 1; \\
x = 2; \\
\text{if}(y > 3) \\
\quad y = 3; \\
\text{else} \\
\quad x = x \times 2; \\
y = x + y + 2;
\]

gets transformed to:
\[
y_0 = x_0 + 1 \\
x_1 = 2; \\
\text{if}(y_0 > 3) \\
\quad y_1 = 3; \\
\text{else} \\
\quad x_2 = x_1 \times 2; \\
y_2 = x_2 + y_1 + 2;
\]

Is this renaming correct?
Example 2: The CFG

ENTRY

\[ y_0 = x_0 + 1 \]
\[ x_1 = 2 \]

y_0 > 3

y_1 = 3
x_2 = x_1 * 2

y_2 = x_2 + y_1 + 2

EXIT
Example 2: Fix using $\phi$ functions

\[
\begin{align*}
  y_0 &= x_0 + 1 \\
  x_1 &= 2 \\
  y_0 &> 3 \\
  y_1 &= 3 \\
  x_2 &= x_1 \times 2 \\
  y_2 &= \phi(y_0, y_1) \\
  x_3 &= \phi(x_1, x_2) \\
  y_3 &= x_3 + y_2 + 2
\end{align*}
\]
Simple Algorithm for constructing SSA form: 1

- Insert $\phi$ functions
  - In which nodes of CFG?
  - For which variables?
- Rename variables
  - To what?
  - Helps to think of LHS (definition) renames and RHS (use) renames
Simple Algorithm for constructing SSA form: 2

- Insert $\phi$ functions
  - In join nodes, before all other code
  - For all variables defined or used in procedure
  - Each $\phi$ function has one argument per incoming edge
  - Use $y = \phi(y, y)$ form for variable $y$

- Rename variables
  - To what?
  - Helps to think of LHS (definition) renames and RHS (use) renames
Simple Algorithm for constructing SSA form: 3

ENTRY

\[
\begin{align*}
y &= x + 1 \\
x &= 2
\end{align*}
\]

EXIT

\[
\begin{align*}
y > 3 \\
y &= 3 \\
x &= x \times 2 \\
y &= \phi(y, y) \\
x &= \phi(x, x) \\
y &= x + y + 2
\end{align*}
\]
Simple Algorithm for constructing SSA form: Rename LHS

ENTRY

\[ y_0 = x + 1 \]
\[ x_1 = 2 \]

ENTRY

\[ y > 3 \]

ENTRY

\[ y_1 = 3 \]
\[ x_2 = x \times 2 \]

ENTRY

\[ y_2 = \phi(y, y) \]
\[ x_3 = \phi(x, x) \]
\[ y_3 = x + y + 2 \]

EXIT
Simple Algorithm for constructing SSA form: Rename RHS

- Note that in SSA form, only one definition reaches a use (except the uses in \( \phi \))
- The arguments to \( \phi \) are the definitions that reach it
Simple Algorithm for constructing SSA form: Rename RHS

ENTRY

\[ y_0 = x_0 + 1 \]
\[ x_1 = 2 \]

\[ y_0 > 3 \]

\[ y_1 = 3 \]
\[ x_2 = x_1 \times 2 \]
\[ y_2 = \phi(y_0, y_1) \]
\[ x_3 = \phi(x_1, x_2) \]
\[ y_3 = x_3 + y_2 + 2 \]

EXIT
Simple Algorithm for constructing SSA form: Renaming

- In actual compilers, renaming LHS and RHS can be done by simply calculating reaching definitions
  - Remember we had to track each definition there too (recall $y \neq 0$)
- This construction is called the *maximal SSA form*
  - Simple to construct
  - Wasteful, can introduce too many $\phi$ functions (not in our example)
Example: Redundant $\phi$ functions

Here, our method constructs a redundant $\phi$ function for $x_2$. 

ENTRY

\[ y_0 = x_0 + 1 \]
\[ x_1 = 2 \]

EXIT

\[ y_0 > 3 \]
\[ y_1 = 3 \]
\[ a = 3 \]

\[ y_2 = \phi(y_0, y_1) \]
\[ x_2 = \phi(x_1, x_1) \]

\[ y_3 = x_2 + y_2 + 2 \]
Example: Redundant $\phi$ functions (now with loops)

Here, $x_3$ is redundant, and its removal makes $x_2$ redundant.
Example: Non-redundant $\phi$ functions

ENTRY

$y_0 = x_0 + 1$
$x_1 = 2$

\[
y_1 = \phi(y_0, y_4) \quad y_1 > 3
\]

\[
\Rightarrow
\]

$\ldots$

$y_2 = 3$
$a = 3$

\[
y_3 = \phi(y_1, y_2) \quad y_4 = x_1 + y_3 + 2
\]

EXIT

This gets rid of the redundant $\phi$ functions.
Outline

Review

Dominator Analysis (DOM)

SSA Form

Postscript
References

- Chapter 9 of Cooper and Turczon
  - Section 9.2.1
  - Section 9.3