Advanced Compilers
CMPSCI 710
Spring 2003
D dominators, etc.
Emery Berger
University of Massachusetts, Amherst

Dominators, etc.
- Last time
  - Live variable analysis
    - backwards problem
  - Constant propagation
    - algorithms
    - def-use chains
- Today
  - SSA-form
  - dominators

Def-Use Chains: Problem

```
switch (j)
case x: i = 1; break;
case y: i = 2; break;
case z: i = 3; break;
switch (k)
case x: a = i; break;
case y: b = i; break;
case z: c = i; break;
```

- worst-case size of graph = $O(D*U) = O(N^2)$

SSA Form
- Static single assignment
  - each assignment to variable gets unique name
  - all uses reached by that assignment are renamed
  - exactly one def per use
  > sparse program representation:
  > use-def chain = (variable, [use_1, use_2, ...])

SSA Transformations
- New variable for each assignment, rename uses

```
v_i = 4
v_j = 5
v_6 = 6
v_{i+7}
```

- Easy for straight-line code, but what about control flow?

Φ-Functions
- At each join, add special assignment: "φ function":
  - operands indicate which assignments reach join
  - $j_{0h}$ operand = $j_{0h}$ predecessor
  - If control reaches join from $j_{0h}$ predecessor, then value of $φ(R,S,...)$ is value of $j_{0h}$ operand
**SSA Transformation, \( \Phi \) function**

\[
\begin{align*}
\text{if } & P \\
\text{then } & v \leftarrow 4 \\
\text{else } & v \leftarrow 6 \\
/* & \text{use } v */
\end{align*}
\]

**SSA Example II**

\[
\begin{align*}
v & \leftarrow 1 \\
\text{while } & (v < 10) \\
v & \leftarrow v + 1
\end{align*}
\]

**SSA Example III**

\[
\begin{align*}
\text{switch } & j \\
\text{case } & x: i \leftarrow 1; \text{ break;} \\
\text{case } & y: i \leftarrow 2; \text{ break;} \\
\text{case } & z: i \leftarrow 3; \text{ break;} \\
\text{switch } & k \\
\text{case } & x: a \leftarrow i; \text{ break;} \\
\text{case } & y: b \leftarrow i; \text{ break;} \\
\text{case } & z: c \leftarrow i; \text{ break;}
\end{align*}
\]

**Placing \( \Phi \) functions**

- Safe to put \( \phi \) functions for every variable at every join point
- But:
  - Inefficient – not necessarily sparse!
  - Loses information
- Goal: minimal \( \phi \) nodes, subject to need

**\( \Phi \) Function Requirement**

- Node \( Z \) needs \( \phi \) function for \( V \) if:
  - \( Z \) is **convergence point** for two paths originating at different nodes
  - Both originating nodes contain assignments to \( V \) or also need \( \phi \) functions for \( V \)

\[
\begin{align*}
v_1 & \leftarrow 1 \\
v_2 & \leftarrow 2
\end{align*}
\]

**Minimal Placement of \( \Phi \) functions**

- Naive computation of need is expensive:
  - Must examine all triples in graph
- Can be done in \( O(N) \) time
  - Relies on **dominance frontier** computation [Cytron et al., 1991]
  - Also can be used to compute control dependence graph
Control Dependence Graph

- Identifies conditions affecting statement execution
- Statement is control dependent on branch if:
  - one edge from branch definitely causes statement to execute
  - another edge can cause statement to be skipped

Control Dependence Example

Statement is control dependent on branch if:
- one edge from branch definitely causes statement to execute
- another edge can cause statement to be skipped

Dominators

- Before we do dominance frontiers, we need to discuss other dominance relationships
- \( x \) dominates \( y \) (\( x \) dom \( y \))
  - in CFG, all paths to \( y \) go through \( x \)
- \( \text{Dom}(v) \) = set of all vertices that dominate \( v \)
- Entry dominates every vertex
- Reflexive: \( a \) dom \( a \)
- Transitive: \( a \) dom \( b \), \( b \) dom \( c \) \( \Rightarrow \) \( a \) dom \( c \)
- Antisymmetric: \( a \) dom \( b \), \( b \) dom \( a \) \( \Rightarrow \) \( b = a \)
- Notice: in SSA form, a definition dominates its use

Finding Dominators

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

Algorithm:

1. \( \text{DOM}(\text{Entry}) = \{\text{Entry}\} \)
2. for \( v \in V - \{\text{Entry}\} \)
3. \( \text{DOM}(v) = V \)
4. repeat
5. \( \text{changed} = \text{false} \)
6. for \( n \in V - \{\text{Entry}\} \)
7. \( \text{olddom} = \text{DOM}(n) \)
8. \( \text{DOM}(n) = \{n\} \]
9. \( \bigcup \{\text{Pred}(n) \cup \text{DOM}(p)\} \)
10. if \( \text{DOM}(n) \neq \text{olddom} \)
11. \( \text{changed} = \text{true} \)
12. end repeat
13. if \( \text{changed} = \text{false} \)
14. break
15. end algorithm

Dominator Algorithm Example

Other Dominators

- Strict dominators
  - \( \text{Dom}(v) = \text{Dom}(v) - \{v\} \)
  - antisymmetric & transitive
- Immediate dominator
  - \( \text{Idom}(v) = \text{closest strict dominator of } v \)
  - \( d \text{ dom } v \) if \( d \text{ dom } v \) and \( x \in 2 \text{ Dom}(v) \), \( w \text{ dom } d \)
  - antisymmetric
  - \( \text{Idom} \) induces tree
**Dominator Example**

- **A** (Entry)
- **B**
- **C**
- **D**
- **E**
- **F**
- **G** (Exit)

**Dominator Tree**

- **A** (Entry)
- **B**
- **C**
- **D**
- **E**
- **F**
- **G** (Exit)

**Inverse Dominators**

- \( D^{-1}(v) \) = set of all vertices dominated by \( v \)
- reflexive, antisymmetric, transitive

**Inverse Dominator Example**

- \( n \text{Dom}(A): \{A\} \)
- \( n \text{Dom}(B): \{A,B\} \)
- \( n \text{Dom}(C): \{A,B,C\} \)
- \( n \text{Dom}(D): \{A,B,D\} \)
- \( n \text{Dom}(E): \{A,B,E\} \)
- \( n \text{Dom}(F): \{A,B,E,F\} \)
- \( n \text{Dom}(G): \{A,B,E,G\} \)

**Finally: Dominance Frontiers!**

- The **dominance frontier** \( D F(X) \) is set of all nodes \( Y \) such that:
  - \( X \) dominates a predecessor of \( Y \)
  - But \( X \) does not strictly dominate \( Y \)
- \( D F(X) = \{Y | (\exists P. P \in \text{PRED}(Y) \land X \text{ Dom } P) \land \neg \exists Q. X \text{ Dom! } Q \} \)

**Why Dominance Frontiers**

- Dominance frontier criterion:
  - If node \( x \) contains def of \( a \), then any node \( z \) in \( D F(x) \) needs \( a \) function for \( a \)
  - Intuition:
    - at least two non-intersecting paths converge to \( z \), and one path must contain node strictly dominated by \( x \)
Dominance Frontier Example

Node y is in dominance frontier of node x if:
x dominates predecessor of y
but does not strictly dominate y

DF(X) = \{Y | (P \in \text{Pred}(Y), X \text{ Dom } P) \land \neg X \text{ Dom } Y\}

Next Time
- Computing dominance frontiers
- Computing SSA form