263-2810: Advanced Compiler Design

2.4.2 Conditional statement

Thomas R. Gross

Computer Science Department
ETH Zurich, Switzerland
2.4.2 Conditional statement

- We saw that $\phi$ functions allows us to deal with a basic block that has two (or more) predecessors
  - Useful for block that follows the then-block and the else-block of a conditional statement
- How should we process each of these blocks?

- Recall:
  - Augmented symbol table contains for each variable
    - Current version (integer)
    - Next version (integer)
A simple example

- X =
- A =
- B =

BB0

BB1

BB2

BB3

... with versions ...
With versions

- Need a $\phi$ node in BB3 for B
  - Either value set in BB1 or value “passed through” BB2
**Variation**

- Need a $\phi$ node in BB3 for A
  - Either value set in BB2 or value “passed through” BB1
Processing blocks

- Consider a block BB with two successors
  - E.g., BB0
  - Starts two paths P1 and P2
  - Assume that P1, P2 contain a single basic block each (here BB1, BB2)

- Each successor block can be processed independently
  - Need (maybe) a \( \phi \) node at the point where the two paths merge
  - Recall: we deal with structured programs

- If only one path creates a new version: must make sure that the version in use in BB (i.e., the parent) appears as an argument to \( \phi \) function.
  - In case the path *without* the new version is taken
No assignment in BB1

- Need a $\phi$ node in BB3 for A
  - Either value set in BB2 or value “passed through” BB1
Processing blocks

- Consider a block BB with two successors
  - E.g., BB0
  - Starts two paths P1 and P2
  - Assume that P1, P2 contain a single basic block each (here BB1, BB2)

- Each successor block can be processed independently
  - Need (maybe) a $\phi$ node at the point where the two paths merge
  - Recall: we deal with structured programs

- If both paths create a new version: must make sure that versions are different
  - Both appear as an argument to $\phi$ function.
Two assignments

- Need a \( \phi \) node in BB3 for \( X \)
Conversion to SSA for conditional stmt

- **Combine two requirements**
  - Use old version if there is no assignment
  - Create different versions in both branches

- **Idea:**
  
  Assume that parent block (BB0) is processed (in SSA form, augmented symbol table)

1. Copy (store) current state of symbol table
2. Process then-block (BB1)
3. Copy (store) current state of symbol table
4. Restore state copied after processing parent, process else-block (BB2)
5. Insert $\phi$ node(s) into successor block (BB3)
A more detailed look

- Given a well-structured program. $B_{\text{parent}}$ is a basic block with two successors ($B_{\text{then}}$ and $B_{\text{else}}$).

- $B_{\text{then}}$ and $B_{\text{else}}$ roots of (well-structured) parts that merge in $B_{\text{continue}}$
  - $B_{\text{then}}$ xor $B_{\text{else}}$ may be empty resp. does not exist

- Assume that $B_{\text{parent}}$ is turned into SSA form
  - As described in 2.4.1 – Straight line code
Process then-part

- **Copy symbol table** \((ST_{\text{parent}})\)
  - Captures state at the start of processing \(B_{\text{then}}\)
  - Set \(B = B_{\text{then}}\)

- **Process B:**
  - Process RHS of each stmt/expression as in 2.4.1
  - Process variable \(D\) on LHS of each stmt as in 2.4.1 and
    - \(\text{update}_\text{then}(D, B_{\text{continue}})\)
  - If successor to \(B = B_{\text{continue}}\) then stop, else process successor \(B = \text{successor} (B)\)
update_then( Variable V, Block B)

- **If there is no $\phi$ node for V in B:**
  - Insert a $\phi$ node $V_{unknown} = \phi (V_{current}, V_{open})$
  - $V_{current}$ can be found in symbol table

- **If there is already a $\phi$ node**
  - Update $V_{unknown} = \phi (V_{old}, V_{open})$ to $V_{unknown} = \phi (V_{current}, V_{open})$
  - There is more than one assignment to V in the then-part

- **First assignment to V in then-part creates $\phi$ node in B**
- **continue**, subsequent assignments update

- $V_{unknown}$ – version number determined later
- $V_{open}$ – version number set after processing else part
Process else-part

- **Copy symbol table (ST\text{then})**
  - Captures state after processing processing $B_{\text{then}}$

- **Restore “current” fields of symbol table from ST\text{parent}**
  - Same state as at start of $B_{\text{then}}$
  - Keep “next” field to get new version numbers
  - Set $B = B_{\text{else}}$

- **Process B:**
  - Process RHS of each stmt/expression as in 2.4.1
  - Process variable $D$ on LHS of each stmt as in 2.4.1 and 
    update\_else$(D, B_{\text{continue}})$
  - If successor to $B = B_{\text{continue}}$ then stop, else process successor $B = \text{successor (B)}$
update_else( Variable V, Block B)

- **If there is no φ node for V in B:**
  - Insert a φ node \( V_{unknown} = \phi (V_{open}, V_{current}) \)
  - \( V_{current} \) can be found in symbol table

- **If there is already a φ node**
  - Update \( V_{unknown} = \phi (V_{whatever}, V_{old}) \) to \( V_{unknown} = \phi (V_{whatever}, V_{current}) \)
  - There is more than one assignment to V in the else-part or processing then-part inserted a φ node

- **First assignment to V in else-part creates φ node in B** continue if there isn’t one from then part, subsequent assignments update

- \( V_{unknown} \) – version number determined later

- \( V_{open} \) – version number set after processing else part
Finish – cleanup $B_{\text{continue}}$

- Now we have to complete the $\phi$ nodes in $B_{\text{continue}}$ and to update the symbol table.

- Three possible scenarios (for each variable with assignments in then or else part):
  - $V_{\text{unknown}} = \phi (V_{\text{then}}, V_{\text{else}})$
    - $V$ set in then-part as well as else-part
    - Node reflects last assignments in each part
    - Get next version from (current) ST
      - State at the end of else-part
      - This version is the “unknown” version
    - Update current & next version fields
- \( V_{\text{unknown}} = \phi \left( V_{\text{open}}, V_{\text{else}} \right) \)
  - \( V \) set only in else-part
  - Version in \( B_{\text{parent}} \) is used in case of path through then-part
    - Get this version from \( ST_{\text{parent}} \)
    - This is the “open” version
  - Get next version from (current) \( ST \)
    - State at the end of else-part
    - This version is the “unknown” version
  - Update current & next version fields
\[ V_{\text{unknown}} = \phi (V_{\text{then}}, V_{\text{open}}) \]

- \( V \) set only in then-part
- Version in \( B_{\text{parent}} \) is used in case of path through else-part
  - Get this version from \( ST_{\text{parent}} \) or from \( ST_{\text{else}} \)
  - This is the “open” version
- Get next version from from \( ST_{\text{then}} \)
  - State at the end of then-part
  - This version is the “unknown” version
- Update current & next version fields
2.4.3 Loops

- Consider only while loops
  ```
  while ( cond ) {
    body
  }
  ```

- Other constructs can be transformed to while loops

- $\phi$ nodes are points where different paths merge
While loops

φ nodes are points where different paths merge
- **Basic idea:** insert $\phi$ node(s) into $B_{test}$

- **Process loop body (along the lines of conditional statement)**
  - Start with $B = B_{body}$
  - For each statement/expression, transform RHS
  - Process each variable $D$ on LHS of each stmt as in 2.4.1 and $\text{update\_while}(D, B_{test})$
  - If successor to $B = B_{test}$ then stop, else process successor $B = \text{successor}(B)$
update\_while( Variable V, Block B)

- If there is no $\phi$ node for V in B:
  - Insert a $\phi$ node $V\_{new} = \phi (V\_{init}, V\_{current})$
  - $V\_{current}$ can be found in symbol table
- If there is already a $\phi$ node
  - Update $V\_{whatever} = \phi (V\_{init}, V\_{old})$ to $V\_{whatever} = \phi (V\_{init}, V\_{current})$
  - There is more than one assignment to V in the loop
- First assignment to V in loop creates $\phi$ node in $B_{test}$, subsequent assignments update
- $V\_{new}$ – new version
While loop example

\( X_1 = 10 \)

\( X_1 \rightarrow \phi(X_1, X_2) \)

if \((X_1 > 0)\)

\( X_2 = X_1 + 1 \)

\( B_{\text{init}} \)

\( B_{\text{test}} \)

\( B_{\text{body}} \)

\( B_{\text{continue}} \)
update_while( Variable V, Block B)

- If there is no \( \phi \) node for V in B:
  - Insert a \( \phi \) node \( V_{new} = \phi(V_{init}, V_{current}) \)
  - \( V_{current} \) can be found in symbol table

- If there is already a \( \phi \) node
  - Update \( V_{whatever} = \phi(V_{init}, V_{old}) \) to \( V_{whatever} = \phi(V_{init}, V_{current}) \)
  - There is more than one assignment to V in the loop

- First assignment to V in loop creates \( \phi \) node in \( B_{test} \), subsequent assignments update

- \( V_{new} \) – new version
  - Must re-process loop
- Read $B_{body}$ once to determine set of variables $V$ defined in loop
- Insert $\phi$ nodes for all variables $V$ in $V$
  \[ V_{new} = \phi (V_{init}, V_{open}) \]
- Update $V_{open}$ as loop body is processed
- Process loop body
While loop example

\[ X_1 = 10 \]

\[ X_2 = \phi(X_1, X_2) \]

if \((X_2 > 0)\)

\[ X_3 = X_2 + 1 \]