Compiler

Static Analysis:

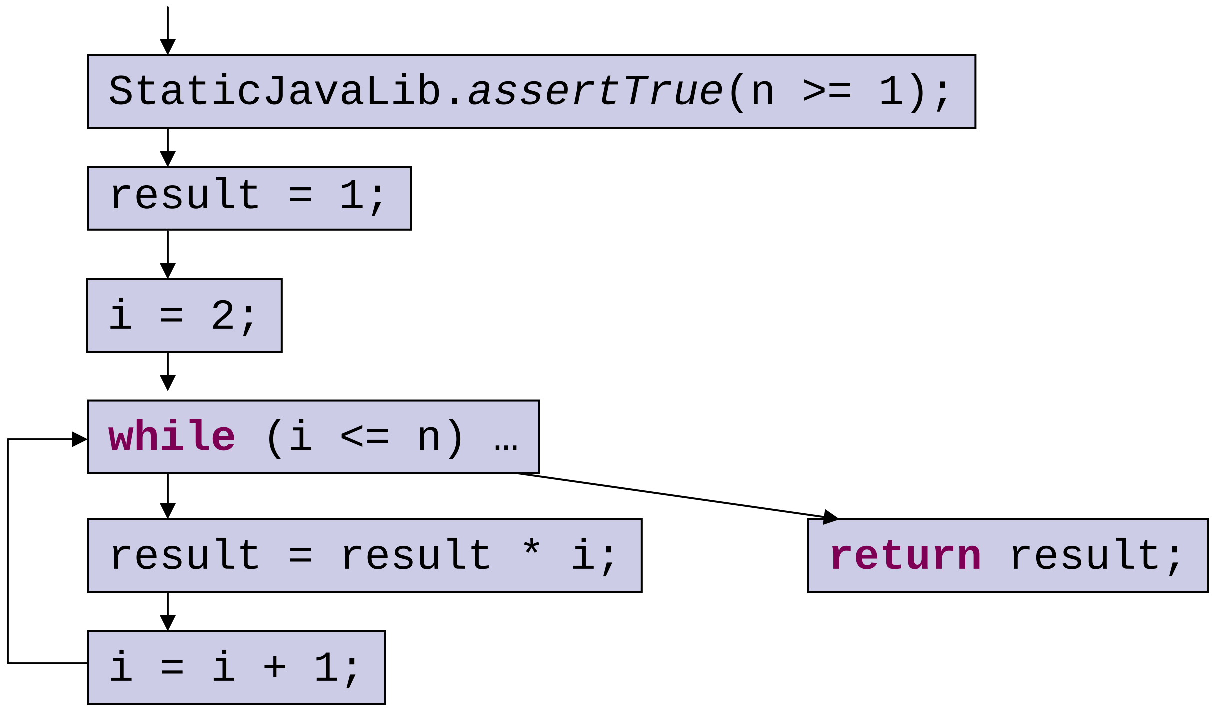
Control Flow Graph (CFG)

Control Flow Graph – Motivation

- We want a better program representation in addition to AST for describing program flow in a method body
  - initial statement
  - last statement
  - given a statement *s*, we want to be able get
    - next statements (successors) of *s*
    - previous statements (predecessors) of *s*

CFG Example

```
static int factorial(int n) {
  int result;
  int i;
```




```
}
```

Control Flow Graph – Definition

- A *CFG* is a 4-tuple (B, E, b_{init}, b_{last})
  - B: is a set of basic blocks,
    - a basic block is a straight-line piece of code without any jumps (e.g., a statement or a sequence of statements)
    - i.e., granularity of basic block is not fixed - it need not be maximal
    - in SJ/ESJ, we consider statements as basic blocks
  - E: B x B, is a relation on B
    - $(b_1, b_2) \in E$ means that the b_2 is a next (successor) basic block of b_1
  - b_{init} and b_{last} are the unique initial basic block and the last basic block of the *CFG*
    - if there is no unique basic block, we can create a "virtual" one
    - in SJ/ESJ, we always create a virtual last basic block

CFG Example

```
static int factorial(int n) {
  int result;
  int i;
  [StaticJavaLib.assertTrue(n >= 1);]1
  [result = 1;]2
  [i = 2;]3
  [while (i <= n) {
    [result = result * i;]5
    [i = i + 1;]6
  }]4
  [return result;]7
}
```

CFG for factorial

- $B = \{ 1, 2, 3, 4, 5, 6, 7 \}$
- $E = \{ (1, 2), (2, 3), (3, 4), (4, 5), (4, 7), (5, 6), (6, 4), (7, b_{last}) \}$
- $b_{init} = 1$
- b_{last} (virtual)

CFG API

sjc.analysis

Class CFG

```
java.lang.Object
  └sjc.analysis.CFG
```

```
public class CFG
extends java.lang.Object
```

This class is used to represent a Control Flow Graph (CFG) of a `MethodDeclaration`.

Author:
Robby

Field Summary

org.eclipse.jdt.core.dom.Statement	**end** Holds the end Statement of the CFG.
org.eclipse.jdt.core.dom.MethodDeclaration	**md** Holds the MethodDeclaration of this CFG.
java.util.Map<org.eclipse.jdt.core.dom.Statement,java.util.Set<org.eclipse.jdt.core.dom.Statement>>	**preds** Holds the mappings of Statements to their predecessors.
org.eclipse.jdt.core.dom.Statement	**start** Holds the start Statement of the CFG.
java.util.Map<org.eclipse.jdt.core.dom.Statement,java.util.Set<org.eclipse.jdt.core.dom.Statement>>	**succs** Holds the mappings of Statements to their successors.

Building CFG — Auxiliary Functions

- Suppose we have two functions *first* and *last* on a sequence of statements
  - *first*([]) = $\perp$ (undefined)
  - *last*([]) = $\perp$ (undefined)
  - *first*([s_1, s_2, s_3, …, s_N]) = s_1
  - *last*([s_1, s_2, s_3, …, s_N]) = s_N
  - *first*([*s*]) = *s*
  - *last*([*s*]) = *s*
- Notes
  - Convention: *S* denotes a sequence of statements, and *s* denotes an individual statement
  - *first* and *last* are called undefined (↑) if the given sequence is empty, otherwise they are defined (↓) as above

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [if (b2) {
       [return;]4
     } else {
       [while (b3) {
          [b3 = !b3;]6
        }]5
     }]3
  }]2
}
```

Building CFG –
Init and Last Statements

- For a method *m* whose body is a sequence of statements *S*

$$b_{init} = \begin{cases} first(S), \text{ if } first(S)\downarrow \\ b_{last}, \text{ otherwise} \end{cases}$$

Building CFG – Init and Last Statements (Impl.)

```
// we model E as a function that maps statement to set of
// statements instead of as relations
Map<Statement, Set<Statement>> E = new HashMap<Statement,Set<Statement>>();
Statement b_init; Statement b_last;
public boolean visit(MethodDeclaration node) {
  List l = node.getBody().statements();
  b_last = node.getBody(); // use method's body as virtual last
  if (l.size() == 0) {
    b_init = b_last; return false; // empty body
  }
  int i = 0; // we need to find the first actual SJ statement
  while (l.size() != i && l.get(i) instanceof VariableDeclarationStatement)
  { i++; }
  if (l.size() == i) {
    b_init = b_last; // no SJ statements in body
  } else {
    b_init = (Statement) l.get(i); // first SJ statement
    // add edge from last statement in method body to virtual b_last
    addEdge(last(l), b_last);
  }
  return true; }
```

Running Example

```
static void running(boolean b1,
        boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Method Declaration

- $E = \{\ \}$ (initially)
  - add $(2, b_{last})$
- $b_{init} = 1$

Building CFG – Sequence of Statement

- For $S = [\, s_1 \ldots s_N \,]$, $(s_i, s_{i+1}) \in E$
  - where $0 < i < N$ and s_i is not an if-statement or a return statement

```
public boolean visit(Block node) {
  List l = node.statements();
  int size = l.size();
  for (int i = 0; i < size - 1; i++) {
    Statement s = (Statement) l.get(i);
    if (s instanceof VariableDeclarationStatement) {
      continue;
    }
    // temporarily add edge for if-statement
    // addEdge doesn't really add if s is a return statement
    //   and its successor is not equal to last
    addEdge(s, (Statement) l.get(i + 1));
  }
  return true; }
```

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Method Body

- $E = \{ (2, b_{last}) \}$
  - add (1, 2)

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Assignment

- $E = \{ (1, 2), (2, b_{last}) \}$
  - ignored (already done when processing Block)

Building CFG – While Statement

- For s = while (e) { S_1 }
  - if $first(S_1)\downarrow$
    - $(s, first(S_1)) \in E$,
    - if $last(S_1)$ is not a return statement, $(last(S_1), s) \in E$
  - otherwise, $(s, s) \in E$

```
public boolean visit(WhileStatement node) {
  List l = ((Block) node.getBody()).statements();
  if (l.isEmpty()) {
    addEdge(node, node);
  } else {
    addEdge(node, first(l));
    addEdge(last(l), node);
  }
  return true;
}
```

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

While Statement

- $E = \{\, (1, 2), (2, b_{last}) \,\}$
  - add (2, 3)
  - add (4, 2)

Running Example

```
static void running(boolean b1,
        boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

While Body

- $E = \{ (1, 2), (2, b_{last}),$
  $(2, 3), (4, 2) \}$
  - add (3, 4)

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Assignment

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 2) \}$
  - ignored

Building CFG – If-Statement

- For [..., s = if (e) { S_1 } else { S_2 }, S_3],
  - if $first(S_1)\downarrow$
    - $(s, first(S_1)) \in E$,
    - if $last(S_1)$ not ret. stmt.
      - $(last(S_1), s') \in E$
  - otherwise
    - $(s, s') \in E$
  - if $first(S_2)\downarrow$
    - $(s, first(S_2)) \in E$,
    - if $last(S_2)$ not ret. stmt.
      - $(last(S_2), s') \in E$
  - otherwise
    - $(s, s') \in E$
- where

$$s' = \begin{cases} first(S_3), \text{ if } first(S_3)\downarrow \\ \text{next of } s \text{ from its parent, otherwise} \end{cases}$$

Building CFG – If-Statement (Impl.)

```java
public boolean visit(IfStatement node) {
  Set<Statement> set = getStatements(E, node);
  assert set.size() == 1;
  Statement next = set.iterator().next();
  E.remove(node); // remove temporary edge for if-statement
  List thenList = ((Block) node.getThenStatement()).statements();
  if (thenList.isEmpty()) { addEdge(node, next); }
  else {
    addEdge(node, first(thenList));
    addEdge(last(thenList), node);
  }
  List elseList = ((Block) node.getElseStatement()).statements();
  if (elseList.isEmpty()) { addEdge(node, next); }
  else {
    addEdge(node, first(elseList));
    addEdge(last(elseList), node);
  }
  return true;
}
```

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

If Statement

- E = { (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 2) }
  - next is 2, *from* (4, 2)
  - remove (4, 2)
  - add (4, 5); *then*
  - add (4, 6), (6, 2); *else*

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]¹
  [while (b1) {
    [b2 = !b2;]³
    [if (b2) {
       [return;]⁵
     } else {
       [while (b3) {
          [b3 = !b3;]⁷
        }]⁶
     }]⁴
  }]²
}
```

If-Then Body

- $E = \{ (1, 2), (2, b_{last}),$
  $(2, 3), (3, 4),$
  $(4, 5), (4, 6),$
  $(6, 2) \}$
  - nothing to be added (there is only one stmt)

Building CFG – Return Statement

- For s = return (*e*)? ;
  - $(s, b_{last}) \in E$

```
public boolean visit(ReturnStatement node) {
  addEdge(node, b_last);
  return false;
}
```

Running Example

```
static void running(boolean b1,
        boolean b2, boolean b3) {
  [b1 = true;]¹
  [while (b1) {
    [b2 = !b2;]³
    [if (b2) {
      [return;]⁵
     } else {
      [while (b3) {
         [b3 = !b3;]⁷
       }]⁶
     }]⁴
  }]²
}
```

Return Statement

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 5), (4, 6), (6, 2) \}$
  - add $(5, b_{last})$

Running Example

```
static void running(boolean b1,
        boolean b2, boolean b3) {
  [b1 = true;]¹
  [while (b1) {
    [b2 = !b2;]³
    [if (b2) {
       [return;]⁵
     } else {
       [while (b3) {
          [b3 = !b3;]⁷
        }]⁶
     }]⁴
  }]²
}
```

If-Else Body

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 5), (4, 6), (5, b_{last}), (6, 2) \}$
  - nothing to be added (there is only one stmt)

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]¹
  [while (b1) {
    [b2 = !b2;]³
    [if (b2) {
       [return;]⁵
     } else {
       [while (b3) {
          [b3 = !b3;]⁷
        }]⁶
     }]⁴
  }]²
}
```

While Statement

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 5), (4, 6), (5, b_{last}), (6, 2) \}$
  - add (6, 7)
  - add (7, 6)

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]¹
  [while (b1) {
    [b2 = !b2;]³
    [if (b2) {
       [return;]⁵
     } else {
       [while (b3) {
          [b3 = !b3;]⁷
        }]⁶
     }]⁴
  }]²
}
```

While Body

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 5), (4, 6), (5, b_{last}), (6, 2), (6, 7), (7, 6) \}$
  - nothing to be added (there is only one stmt)

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Assignment

- $E = \{ (1, 2), (2, b_{last}),$
  $(2, 3), (3, 4),$
  $(4, 5), (4, 6),$
  $(5, b_{last}), (6, 2),$
  $(6, 7), (7, 6) \}$
  - ignored

Running Example

```
static void running(boolean b1,
       boolean b2, boolean b3) {
  [b1 = true;]1
  [while (b1) {
    [b2 = !b2;]3
    [if (b2) {
       [return;]5
     } else {
       [while (b3) {
          [b3 = !b3;]7
        }]6
     }]4
  }]2
}
```

Done!

- $E = \{ (1, 2), (2, b_{last}), (2, 3), (3, 4), (4, 5), (4, 6), (5, b_{last}), (6, 2), (6, 7), (7, 6) \}$

For You To Do

- Try building the *CFG* for the Factorial example
- Try building the *CFG* for the following methods

```
int foo(int x) {
  [return x;]1
  [x = x + 1;]2
  [x = x - 1;]3
}
```

```
void bar(boolean b) {
  [if (b) { [return;]2 }
   else { [return;]3 }]1
  [return;]4
}
```

```
void bazzzzz() {
  [if (true) {
    [return;]2
   } else {
    [return;]3
   }]1
}
```

Statements Reachability

To compute the reachable statements from a particular statement *s*:

$$reachable(s) = \{\ s\ \} \cup E^{+}(s)$$

$$unreachable(b_{init}) = B \ / \ reachable(b_{init})$$

Note that for a relation R, its transitive closure (+) $R^{+} = \bigcup_{i \in N} R^{i}$

- i.e., $R^{+}(e) = R(e) \cup (\bigcup_{e' \in R(e)} R(e')) \cup \ldots$

Statements Reachability (Impl.)

```
public void endVisit(MethodDeclaration node) {
  Set<Statement> reachableSet = new HashSet<Statement>();
  reachable(reachableSet, b_init);
  …
}

protected void reachable(Set<Statement> set, Statement s) {
  if (set.contains(s)) { return; }
  set.add(s);
  for (Statement s2 : getStatements(E, s)) {
    reachable(set, s2);
  }
}
```

Building CFG — Cleaning Up

- We want to restrict E so it only relates reachable statements

$$E' = E \,/\, \{ (b_1, b_2) \mid b_1 \text{ or } b_2 \in \mathit{unreachable}(b_{init}) \}$$

```
int foo(int x) {
  [return x;]1
  [x = x + 1;]2
  [x = x - 1;]3
}
```

$$E = \{ (1, b_{last}), (2, 3), (3, b_{last}) \}$$

$$E' = \{ (1, b_{last}) \}$$

CFG succs/preds

For a *CFG* $(B, E, b_{init}, b_{last})$

- $succs(b) = \{ b' \mid (b, b') \in E' \}$
  - a function that given a basic block *b*, it returns *b*'s set of next basic blocks
- $preds(b) = \{ b' \mid (b', b) \in E' \}$
  - a function that given a basic block *b*, it returns *b*'s set of previous basic blocks

succs/preds Example

CFG for factorial

- $B = \{ 1, 2, 3, 4, 5, 6, 7 \}$
- $E' = \{ (1, 2), (2, 3),$
  $(3, 4), (4, 5), (4, 7),$
  $(5, 6), (6, 4),$
  $(7, b_{last}) \}$
- $b_{init} = 1$
- b_{last} (virtual)

- successors
  - $succs(1) = \{ 2 \}$
  - $succs(2) = \{ 3 \}$
  - $succs(3) = \{ 4 \}$
  - $succs(4) = \{ 5, 7 \}$
  - $succs(5) = \{ 6 \}$
  - $succs(6) = \{ 4 \}$
  - $succs(7) = \{ b_{last} \}$
- predecessors
  - $preds(1) = \{ \ \}$
  - $preds(2) = \{ 1 \}$
  - $preds(3) = \{ 2 \}$
  - $preds(4) = \{ 3, 6 \}$
  - $preds(5) = \{ 4 \}$
  - $preds(6) = \{ 5 \}$
  - $preds(7) = \{ 4 \}$

$succs^+$/$preds^+$ Example

- successors
  - $succs(1) = \{ 2 \}$
  - $succs(2) = \{ 3 \}$
  - $succs(3) = \{ 4 \}$
  - $succs(4) = \{ 5, 7 \}$
  - $succs(5) = \{ 6 \}$
  - $succs(6) = \{ 4 \}$
  - $succs(7) = \{ b_{last} \}$
- predecessors
  - $preds(1) = \{ \}$
  - $preds(2) = \{ 1 \}$
  - $preds(3) = \{ 2 \}$
  - $preds(4) = \{ 3, 6 \}$
  - $preds(5) = \{ 4 \}$
  - $preds(6) = \{ 5 \}$
  - $preds(7) = \{ 4 \}$
- successors (transitive closure)
  - $succs^+(1) = \{ 2, 3, 4, 5, 6, 7 \}$
  - $succs^+(2) = \{ 3, 4, 5, 6, 7 \}$
  - $succs^+(3) = \{ 4, 5, 6, 7 \}$
  - $succs^+(4) = \{ 4, 5, 6, 7 \}$
  - $succs^+(5) = \{ 4, 5, 6, 7 \}$
  - $succs^+(6) = \{ 4, 5, 6, 7 \}$
  - $succs^+(7) = \{ b_{last} \}$
- predecessors (transitive closure)
  - $preds^+(1) = \{ \}$
  - $preds^+(2) = \{ 1 \}$
  - $preds^+(3) = \{ 1, 2 \}$
  - $preds^+(4) = \{ 1, 2, 3, 4, 5, 6 \}$
  - $preds^+(5) = \{ 1, 2, 3, 4, 5, 6 \}$
  - $preds^+(6) = \{ 1, 2, 3, 4, 5, 6 \}$
  - $preds^+(7) = \{ 1, 2, 3, 4, 5, 6, 7 \}$

succs/preds Implementation

```
public void endVisit(MethodDeclaration node) {
  Set<Statement> reachableSet = new HashSet<Statement>();
  computeSuccsPreds(reachableSet, b_init);
}

protected void computeSuccsPreds(Set<Statement> set, Statement s) {
  if (set.contains(s)) { return; }
  set.add(s);
  for (Statement succS : getStatements(E, s)) {
    getStatements(succs, s).add(succS);
    getStatements(preds, succS).add(s);
    computeSuccsPreds(set, succS);
  }
}
```

For You To Do

- Give the *CFG*, and its *succs*, *preds*, *succs*$^+$, and *preds*$^+$ for

```
int foo(int x) {
  [return x;]¹
  [x = x + 1;]²
  [x = x + 1;]³
}
```

```
void bar(boolean b) {
  [if (b) { [return;]² }
   else { [return;]³ }]¹
  [return;]⁴
}
```

```
void bazzz() {
  [if (true) {
    [return;]²
   } else {
    [return;]³
   }]¹
}
```

- Is foo.2 $\in$ *succs*$^+$(foo.1)?
- Is bar.4 $\in$ *succs*$^+$(bar.1)?
- Is bazzz.3 $\in$ *succs*$^+$(bazzz.1)?

Assessment

Control Flow Graph

- This is (relatively) trivial to calculate for SJ
- It is easy to determine very approximate control flow info for Java
- It is hard to determine accurate info for Java and higher-order languages (e.g., where functions can be passed as arguments, exceptions)
- Often times control flow information is