Declarative Intraprocedural Flow Analysis of Java Source Code

Emma Nilsson-Nyman*   Torbjörn Ekman**
Görel Hedin*   Eva Magnusson*

* Dept of Computer Science
  Lund University

** Programming Tools Group
  University of Oxford
Goals

- Source code analysis is useful
  - Refactorings
  - Errors
  - Optimizations
- Simplify its implementation
- Intraprocedural control flow and data flow
  - Normally: Three address code
  - This paper: Declarative AST computations
Tool: Compiler

- Java 1.4 Compiler built using JastAdd

10,000 LOC
Tool: Compiler

- Java 1.4 Compiler built using JastAdd
- Attribute grammars - declarative AST computations
  - Synthesized and inherited attributes
Tool: Compiler

- Java 1.4 Compiler built using JastAdd
- Attribute grammars - declarative AST computations
  - Synthesized and inherited attributes
- Modern extensions
  - Reference attributes
    - Super-imposed graphs
  - Collection attributes
  - Circular attributes
  - Non-terminal attributes

10 000 LOC
Intraprocedural Control Flow

- Traditional control flow graphs (CFG)
  - Three address code
  - Basic blocks
  - Successors, predecessors
  - Entry and exit blocks
Intraprocedural Control Flow

- Our control flow graphs
  - Basic blocks -> Statements

AST:

```
MethodDecl
  ▼
  Block
  ▼
  WhileStmt
  ▼
  Stmt
```
Intraprocedural Control Flow

- Our control flow graphs
  - Basic blocks -> Statements
  - Entry and exit nodes
Intraprocedural Control Flow

- Our control flow graphs
  - Basic blocks -> Statements
  - Entry and exit nodes
  - Reference attributes
    - Successors

AST:

MethodDecl

Entry

Block

Exit

WhileStmt

Stmt
Intraprocedural Control Flow

- Our control flow graphs
  - Basic blocks -> Statements
  - Entry and exit nodes
  - Reference attributes
    - Successors
    - Predecessors

AST:

```
MethodDecl

Entry
Block
Exit

WhileStmt
Stmt
```
Intraprocedural Control Flow

Defining successors:

```java
syn Set Stmt.succ();
eq Stmt.succ() = following();
eq WhileStmt.succ() = following().union(getStmt());
...
```
Intraprocedural Control Flow

- Defining predecessors:

```java
coll Set Stmt.pred()
    [empty()] with add;
Stmt contributes this to
Stmt.pred() for each
succ();
```

AST:
Intraprocedural Control Flow

- Approx 240 LOC for Java exceptions
- Control Flow API:

```python
Stmt
Stmt entry()
Stmt exit()
Set succ()
Set pred()
```
Intraprocedural Data Flow

Data Flow (DF)

Live variables

```c
a = 0;
b = 0;
a = b + 5;
b = a + b;
...```

Compiler — CFG — DF — Application
Intraprocedural Data Flow

- Live variables

```
a = 0;
b = 0;
a = b + 5;
b = a + b;
...
```
Intraprocedural Data Flow

- Live variables

\[
\begin{align*}
\text{a} &= 0; \\
\text{b} &= 0; \\
\text{a} &= \text{b} + 5; \\
\text{b} &= \text{a} + \text{b}; \\
\text{...} &
\end{align*}
\]
Intraprocedural Data Flow

- Live variables

\[
\begin{align*}
\text{a} &= 0; \\
\text{b} &= 0; \\
\text{a} &= \text{b} + 5; \\
\text{b} &= \text{a} + \text{b}; \\
\ldots
\end{align*}
\]

Definition:
\[
in(s) = \text{use}(s) \cup (\text{out}(s) \setminus \text{def}(s))
\]

\[
\text{out}(s) = \bigcup_{x \in \text{succ}(s)} \text{in}(x)
\]
Intraprocedural Data Flow

- Live variables

```
a = 0;

b = 0;

a = b + 5;

b = a + b;
```

- `in = ∅, out = ∅`

- `in = ∅, out = {b}`

- `in = {b}, out = {a, b}`

- `in = {a, b}, out = {...}`
Intraprocedural Data Flow

- Live variables

\[
\begin{align*}
\text{a} &= 0; \\
\text{b} &= 0; \\
\text{a} &= \text{b} + 5; \\
\text{b} &= \text{a} + \text{b}; \\
\ldots
\end{align*}
\]

\[
\begin{align*}
in &= \emptyset, \quad out = \emptyset & \quad def &= \{a\}, \quad use = \emptyset \\
in &= \emptyset, \quad out = \{b\} & \quad def &= \{b\}, \quad use = \emptyset \\
in &= \{b\}, \quad out = \{a,b\} & \quad def &= \{a\}, \quad use = \{b\} \\
in &= \{a,b\}, \quad out = \ldots & \quad def &= \{b\}, \quad use = \{a,b\}
\end{align*}
\]

- Definition:

\[
in(s) = use(s) \cup (out(s) \setminus def(s))
\]
Intraprocedural Data Flow

Live variables

\[ a = 0; \]
\[ b = 0; \]
\[ a = b + 5; \]
\[ b = a + b; \]

\[ \text{in} = \emptyset, \text{out} = \emptyset \quad \text{def} = \{a\}, \text{use} = \emptyset \]
\[ \text{in} = \emptyset, \text{out} = \{b\} \quad \text{def} = \{b\}, \text{use} = \emptyset \]
\[ \text{in} = \{b\}, \text{out} = \{a,b\} \quad \text{def} = \{a\}, \text{use} = \{b\} \]
\[ \text{in} = \{a,b\}, \text{out} = \{\ldots\} \quad \text{def} = \{b\}, \text{use} = \{a,b\} \]

Definition:

\[
\begin{align*}
\text{in}(s) & = \text{use}(s) \cup (\text{out}(s) \setminus \text{def}(s)) \\
\text{out}(s) & = \bigcup_{x \in \text{succ}(s)} \text{in}(x)
\end{align*}
\]
Intraprocedural Data Flow

\[ \text{in}(s) = \text{use}(s) \cup (\text{out}(s) \setminus \text{def}(s)) \]

\textbf{syn} Set Stmt.in() \textbf{circular} [empty()];

\textbf{eq} Stmt.in() = \text{use()}.\text{union}(\text{out()}.\text{compl}(\text{def}()));
\[
\text{in}(s) = \text{use}(s) \cup (\text{out}(s) \setminus \text{def}(s))
\]

\text{syn} \text{ Set Stmt.in()} \text{ circular [empty()];}
\text{eq} \text{ Stmt.in()} = \text{use().union(out().compl(def()))};

\[
\text{out}(s) = \bigcup_{x \in \text{succ}(s)} \text{in}(x)
\]

\text{coll} \text{ Set Stmt.out()} \text{ circular [empty()]} \text{ with add;}
\text{Stmt contributes in()} \text{ to Stmt.out()}
\text{for each pred();}
Intraprocedural Data Flow

- Data Flow API:

  **Stmt**
  - Set in()
  - Set out()
  - Set def()
  - Set use()
Application: Dead Assignment

Dead assignment

Dead! →
\[
\begin{align*}
  &a = 0; \\
  &b = 0; \\
  &a = b + 5; \\
  &b = a + b; \\
  &\ldots
\end{align*}
\]
Dead Assignment (DA)

\[
a = 0; \\
b = 0; \\
a = b + 5; \\
b = a + b;
\]

Dead?
true
false
false
...
Application: Dead Assignment

Dead?

\[
\begin{align*}
\text{true} & \quad \text{def} = \{a\} \quad \text{out} = \emptyset \\
\text{false} & \quad \text{def} = \{b\} \quad \text{out} = \{b\} \\
\text{false} & \quad \text{def} = \{a\} \quad \text{out} = \{a,b\} \\
\ldots & \quad \text{def} = \{b\} \quad \text{out} = \{\ldots\} \\
\end{align*}
\]

\textbf{not} def(s) \subseteq out(s)
Application: Dead Assignment

\[ a = 0; \]
\[ b = 0; \]
\[ a = b + 5; \]
\[ b = a + b; \]

Dead?

true \hspace{1cm} \text{def} = \{a\} \hspace{1cm} \text{out} = \emptyset

false \hspace{1cm} \text{def} = \{b\} \hspace{1cm} \text{out} = \{b\}

false \hspace{1cm} \text{def} = \{a\} \hspace{1cm} \text{out} = \{a,b\}

... \hspace{1cm} \text{def} = \{b\} \hspace{1cm} \text{out} = \{...\}

\textbf{not} \hspace{0.5cm} \text{def}(s) \subseteq \text{out}(s)

\textbf{syn} \hspace{1cm} \textbf{boolean} \hspace{0.5cm} \text{Stmt.isDead}();

\textbf{eq} \hspace{0.5cm} \text{Stmt.isDead()} = \text{!def()}.subset(\text{out}());
Application: Dead Assignment

Dead Assignment API:

```java
Stmt
  boolean isDead()
```
Practical Experience

**Compilation Time (s)**

<table>
<thead>
<tr>
<th></th>
<th>kLOC</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>junit</td>
<td>5.2</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>antlr</td>
<td>39.0</td>
<td>8.4</td>
<td></td>
</tr>
<tr>
<td>bloat</td>
<td>42.0</td>
<td>8.7</td>
<td></td>
</tr>
<tr>
<td>jigsaw</td>
<td>101.0</td>
<td>19.9</td>
<td></td>
</tr>
</tbody>
</table>
Practical Experience

<table>
<thead>
<tr>
<th></th>
<th>kLOC</th>
<th>Compilation Time (s)</th>
<th>Declarative (s)</th>
<th>Imperative (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>junit</td>
<td>5.2</td>
<td>3.3</td>
<td>+ 0.6</td>
<td>+ 0.4</td>
</tr>
<tr>
<td>antlr</td>
<td>39.0</td>
<td>8.4</td>
<td>+ 2.8</td>
<td>+ 2.5</td>
</tr>
<tr>
<td>bloat</td>
<td>42.0</td>
<td>8.7</td>
<td>+ 2.2</td>
<td>+ 2.0</td>
</tr>
<tr>
<td>jigsaw</td>
<td>101.0</td>
<td>19.9</td>
<td>+ 3.8</td>
<td>+ 4.8</td>
</tr>
</tbody>
</table>
## Practical Experience

<table>
<thead>
<tr>
<th></th>
<th>Compilation Time (s)</th>
<th>Imperative (s)</th>
<th>Declarative (s)</th>
<th>Dead Assigns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kLOC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>junit</td>
<td>5.2</td>
<td>3.3</td>
<td>+ 0.6</td>
<td>+ 0.4</td>
</tr>
<tr>
<td>antlr</td>
<td>39.0</td>
<td>8.4</td>
<td>+ 2.8</td>
<td>+ 2.5</td>
</tr>
<tr>
<td>bloat</td>
<td>42.0</td>
<td>8.7</td>
<td>+ 2.2</td>
<td>+ 2.0</td>
</tr>
<tr>
<td>jigsaw</td>
<td>101.0</td>
<td>19.9</td>
<td>+ 3.8</td>
<td>+ 4.8</td>
</tr>
</tbody>
</table>
Practical Experience

![Graph showing compilation times for different languages and tools. The x-axis represents kLOC (thousands of lines of code), and the y-axis represents seconds. The graph includes data points for `antlr`, `j unit`, `bloat`, and `jigsaw`, with lines indicating compilation, declarative, and imperative approaches.](image-url)
Conclusions

- Concise - close to mathematical definitions
- Composable modules
- Source code instead of three address code
- Extensible
- Reasonable execution times
Future Work

- Additional Analyses
  - Interprocedural control flow and data flow
    - Preliminary work: metrics (Chidamber & Kemerer), call graphs
  - SSA graphs
- Additional Applications
  - Bug detection
  - Refactorings
    - On going work . . .
The End

Thanks for listening!
Questions?
Exceptions

- Propagate down (**inh**)
  - enclosingTryStmt()
  - hasEnclosingFinally()
- Propagate up (**syn**)
  - containsReturn()
Intraprocedural Data Flow

def(s)

coll Set Stmt.def() [empty()] with add;
VarAccess contributes decl()
  when isDest() && decl().isLocalVariable()
to Stmt.def() for enclosingStmt();

use(s)

coll Set Stmt.use() [empty()] with add;
VarAccess contributes decl()
  when isSource() && decl().isLocalVariable()
to Stmt.use() for enclosingStmt();