Hierarchical Dynamic Slicing

Tao Wang    Abhik Roychoudhury
National University of Singapore

Preamble

Software Debugging
- Time consuming
- Traditional Source-Level Debugger
  - The key problem is automation

Slicing for Debugging
- Slicing criterion
  - (Input, Variable, Line#)
- (Dynamic) Slice
  - A fragment of the program which directly or indirectly affect the behavior of the slicing criterion
  - Compute closure of control/data dep.
  - Treat the slice as bug-report.

Why Dynamic?
- Naturally corresponds to debugging via testing.
- Test program against test-cases.
- Find "offending" test-cases
  - Output "unexpected".
- Compute dynamic slice to debug the behavior in the offending test cases.

Dynamic Slicing
1. void setRunningVersion(boolean runningVersion){
2.   if( runningVersion ) {
3.     savedValue = value;
4.   } else{
5.     "savedValue = ";
6.   }
7.   this.runningVersion = runningVersion;
8.   System.out.println(savedValue);
9. }
Dynamic Slice

```java
void setRunningVersion(boolean runningVersion)
1. if (runningVersion) {
2.    savedValue = value;
3. } else{
4.    savedValue = "";
5.    this.runningVersion = runningVersion;
6.    System.out.println(savedValue);
}
```

Slicing Criterion

**So ... ?**

- Huge overheads
  - Backwards slicing requires trace storage.
  - Past work – Jslice tool for Java
    - Online trace compression & traversal
    - [http://jslice.sourceforge.net](http://jslice.sourceforge.net)
- Dynamic Slice is still too large ...
  - ... for human comprehension
  - In this paper

---

**An example**

```java
public static void main(String[] args) {
    init( db );
    operate( db );
    output ( db )
    return;
}
```

**SPECJVM DB program**

- `init()`: Initialize database
- `operate()`: Operate on database
- `output()`: Output database

**Divide exec. into “phases”**

```java
public static void main(String[] args) {
    init( db );
    operate( db );
    output ( db );
    return;
}
```

**Intra-phase control and data dependencies are suppressed.**

**Inter-phase dep. form input-output relationships.**

---

**Report inter-phase dep.**

- `init()`: Initialize database
- `operate()`: Operate on database
- `output()`: Output database

**Programmer helps zoom into**

One phase by inspecting the phase outputs

- (may/may not involve re-executing program)

---

**Re-execute phase 1 and observe db**
**Parallel dependence chains**

\[
\begin{align*}
\text{main()} & \rightarrow y \\
\text{f1()} & \rightarrow x_1 \\
\text{f2()} & \rightarrow x_2 \\
\text{f3()} & \rightarrow x_3 \\
\end{align*}
\]

\[
x_1 = \text{f1}(); \\
x_2 = \text{f2}(); \\
x_3 = \text{f3}(); \\
y = x_1 + x_2 + x_3;
\]

print \(y \rightarrow \) Criterion

**Programmer intervention ...**

... may not be needed, for example

\[
\begin{align*}
\text{main()} & \\
\text{func1()} & \\
\text{func2()} & \\
\text{func3()} & \\
\end{align*}
\]

All dependencies inside this phase, zoom into this one

**Key issues**

- Compute "phases" of an exec. trace
  - Diff. from phase detection in PL community
- Augment dynamic slicing algorithm
  - Mark inter-phase dependencies
  - Compute only reachable nodes from selected inter-phase dependency.
- Programmer intervention
  - Select the first suspicious inter-phase dep.
  - Comprehension guides computation.

**Phase detection**

- Divide an exec. trace at boundaries of
  - Loops
  - Method calls
  - Loop iterations
  - ...
  - and recursively again at these control structure boundaries.

**Differs from work in PL**

- Phase detection to guide program opt.
  - Divide a phase into fixed-length intervals.
  - Compute metric (e.g. Basic block vectors) for each interval.
  - Cluster adjacent intervals with similar metric.
- No notion of control flow context.
**Subject Programs**

<table>
<thead>
<tr>
<th>Subject Pgm.</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NanoXML</td>
<td>XML parser for Java</td>
<td>7646 LOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 classes</td>
</tr>
<tr>
<td>JTopas</td>
<td>Java library for parsing text</td>
<td>5400 LOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50 classes</td>
</tr>
<tr>
<td>Apache JMeter</td>
<td>Performance testing tool</td>
<td>43400 LOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>389 classes</td>
</tr>
</tbody>
</table>

SIR subjects --- for each used 3 buggy versions with seeded faults.

**Slicing Infrastructure - JSlice**
- Dynamic Slicing tool for Java programs
  - The first one
  - Open source, bytecode level slicing tool
    - [http://jslice.sourceforge.net](http://jslice.sourceforge.net)
  - Distributed with the Kaffe VM
  - 50+ registered users.
    - released 8 months ago
    - Adopted for teaching in few universities

**Slicing Infrastructure JSlice**
- GUI (a Eclipse plug-in)
- Execute the program
- Select
- Kaffe JVM
- Instrument
- Compact Bytecode Trace
- Reverse Translate
- Dynamic Slicing
  - (Stack simulation)
- Class Files
- Set of bytecodes
- Criterion = (inp, var, line)

**Statement Inst. Examined (Log scale)**

**Human Interventions**

---

ISSTA '07, London (UK)
Fault Localization

Failing Run → Successful Run

Compare Executions

Difference

= Bug report? (hierarchically construct and explore)

Slicing?

Experiments

Summary

Hierarchically Explore Bug-report

- Dynamic vs. Static
  - Thin Slicing – explore static slices

- On-line vs. post-mortem
  - DDgraph – explore dynamic dependence graph hierarchically.

- Inter-phase vs. intra-phase
  - Work on parallel pgm. debugging - present intra-phase dependencies

Summary

Hierarchically construct and explore the dynamic slice for understanding.

- Comprehension/computation interleaved.
- Structures programmer feedback
  - Only examine inter-phase dependencies
  - Not just view a programmer chosen phase.
- Detailed expt. on SIR benchmarks
- Efficacy on human subjects
  - To be distributed in future releases of Jslice.
- http://jslice.sourceforge.net/