Discriminating Influences among Instructions in a Dynamic Slice

Vijay Krishna Palepu    James A. Jones
University of California, Irvine, USA


UC Irvine
Spider Lab
protected void doBuildTemplate(RunData data, Context context) {
    throws Exception {
        if (TurbineUtils.HasPassedParameter("duration1", data)) {
            context.put("duration1", TurbineUtils.GetPassedParameter("duration1", data))
        } else {
            context.put("duration1","1 week");
        }
        if (TurbineUtils.HasPassedParameter("duration2", data)) {
            context.put("duration2", TurbineUtils.GetPassedParameter("duration2", data))
        } else {
            context.put("duration2","1 second");
        }
    }
}
Program Execution
Problem.

- “... for most real programs, the dynamic slices are too large for humans to inspect and comprehend.” – Wang and Roychoudhury, ISSTA 2007.

- “Breakpoints allowed developers to search for paths to a statement. But setting breakpoints was impractical when searching for many statements . . . .” – LaToza and Myers, ICSE 2010
Problem.

- Where, in the program’s execution, should we focus our attention?
Discrimination of Influences

Program Execution Time

Program Execution Trace

Program Execution Trace after Dynamic Slicing

Breakpoint Instruction

Slicing Criterion

This Work: Dynamic Slice after Discriminating Influences

Approach.

- Discriminate runtime instructions using degrees of influence on the slicing criterion.
Dynamic Relevance
Measure of Degree of Influence.

Slicing Criterion

Execution Time
Broader the scope of influence, lesser the dynamic relevance and vice versa.
Dynamic Relevance
Measure of Degree of Influence.

\[ \text{DynRel} (a, z) = \frac{|S_{bak}(z) \cap S_{fwd}(a)|}{|S_{fwd}(a)|} \]
Dynamic Relevance
Measure of Degree of Influence.

\[ \text{DynRel}(a, z) = \frac{|S_{\text{bak}}(z) \cap S_{\text{fwd}}(a)|}{|S_{\text{fwd}}(a)|} \]

- \( S_{\text{bak}}(z) \): backward slice from \( z \);
- \( S_{\text{fwd}}(a) \): forward slice from \( a \);
- \( S_{\text{bak}}(z) \cap S_{\text{fwd}}(a) \): runtime instructions common to \( S_{\text{bak}}(z) \) and \( S_{\text{fwd}}(a) \);
Dynamic Relevance
Measure of Degree of Influence.

\[
\text{DynRel} (a, z) = \frac{|S_{\text{bak}}(z) \cap S_{\text{fwd}}(a)|}{|S_{\text{fwd}}(a)|}
\]

- \( S_{\text{bak}}(z) \) : backward slice from \((z)\);
- \( S_{\text{fwd}}(a) \) : forward slice from \((a)\);

Execution Time
Dynamic Relevance
Measure of Degree of Influence.

\[ \text{DynRel}(a, z) = \frac{|S_{\text{bak}}(z) \cap S_{\text{fwd}}(a)|}{|S_{\text{fwd}}(a)|} \]

- \( S_{\text{bak}}(z) \): backward slice from \((z)\);
- \( S_{\text{fwd}}(a) \): forward slice from \((a)\);
- \( S_{\text{bak}}(z) \cap S_{\text{fwd}}(a) \): runtime instructions common to \(S_{\text{bak}}(z)\) & \(S_{\text{fwd}}(a)\);
Dynamic Relevance
Measure of Degree of Influence.

\[ \text{DynRel}(c, z) = \frac{6}{6} = 1.00 \]

\[ \text{DynRel}(a, z) = \frac{6}{9} = 0.66 \]
Implementation: Looking Glass
Program Execution Analyzer

Java Instrumenter & Execution Profiler

Program Execution Trace

Execution Trace Analyzer & Slicer

Backward and Forward Dynamic Slices
Emerging Results

- Task: Root Cause Analysis for an anomalous output.
- Subject: NanoXML (>7,000 LOCs), XML Parser, Java
- 20 test executions
Emerging Results

- Execution Trace: 100%
- Dynamic Slice: 40%
- Dynamic Relevance: 4%

1/10\textsuperscript{th} of the Dynamic slice in size.
Next Steps.

- Continuation of the empirical investigation.
- Development of novel software engineering analyses using discriminated influences.
- Envisioning variations of Dynamic Relevance.
Discriminating Influences among Instructions in a Dynamic Slice

Vijay Krishna Palepu  James A. Jones
University of California, Irvine, USA

Program Execution Time

Program Execution Trace

Program Execution Trace after Dynamic Slicing

This Work: Dynamic Slice after Discriminating Influences