Testing OO Programs - 2

- Regression testing for Java
  - What is *selective regression testing*?
    - What does *safe* mean in this context?
    - How find *safe* subset of *regression tests* to execute?
  - Program representations
    - Java interclass graph
    - Dealing with unanalyzed libraries
  - Experimental validation

Regression Testing

- Keep a set of test cases, used to test program after substantial change
  - *Test case* - program input and expected output
  - *Test suite* - set of test cases
  - *Adequacy* is assessed by coverage metrics (usually branches or statements covered)
- P' a modified version of P, T test suite, info about testing P with T are available during regression testing of P'
  - *Regression test selection* problem - What to retest from T?
  - *Test suite augmentation* problem - What new tests are needed?

Selective Regression Testing

- **Goal - minimize number of tests needed using information about changed code**
- **Inputs:**
  - **P, T, Coverage of T in P --> *Coverage matrix***
    - **Records edges covered by test t in T on P**
  - **P, P', *Dangerous* (affected) *entities***
    - **Given program construct e in P where P(j) is P executed with input j. e is *dangerous,* if for each input j causing P to cover e, P(j) and P'(j) may differ in behavior**

Selective Regression Testing

- **Rothermel and Harrold intraprocedural algorithm**
  - **Algorithm for comparing control flow graphs before and after code changes to find tests that exercise changed code**
  - **CFG edges were considered potential dangerous entities**
    - **Parallel traversal of CFG(P) and CFG(P'); when targets of like-labeled edges differed, then edge was called *dangerous***
    - **Use coverage matrix to find tests that will traverse the dangerous edges, to comprise T'.**
  - **Interprocedural version of the algorithm exists also based on code comparisons in the order of static execution path traversal of call sites**

G. Rothermel, M.J. Harrold, "A Safe, Efficient Regression Test Selection Technique, ACM TOSEM, vol 6, no 2, April 1997

Example of Previous Algm

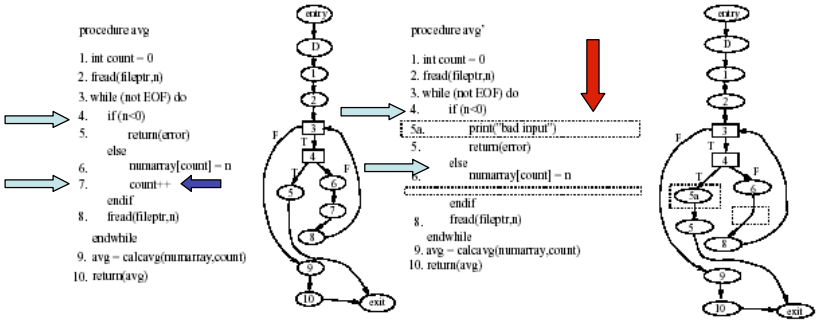

Figure 1: Example program, avg and its control-flow graph (left); modified version, avg' and its control-flow graph (right).

"Regression Test Selection for Java Software",M.J. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S.A. Spoon, S. Gujarathi, OOPSLA'01

Approach for Java

- **Use same 3-phase approach**
  - **Construct graph representation**
  - **Find dangerous edges (through comparison)**
  - **Based on coverage matrix (observations), select tests to cover dangerous edges**
- **Program considered to be in 2 parts**
  - **Analyzed part and external (e.g., unchanged library) part of program**
    - **External codes are used by program but not modified**

Assumptions

- Reflection not used on any component of analyzed part (includes Java.lang.Class)
  - Why? Too hard to analyze change effects
- Unanalyzed code has no knowledge of analyzed code
  - That is, no call-backs can occur, but user may extend library classes
- Test runs are deterministic and repeatable
  - Guarantees coverage info independent of run
  - Guarantees same results from a test that does not execute change-affected parts of the code.
  - Requires repeatability of multi-threaded codes

Java Interclass Graph (JIG)

- New representation for Java programs
  - Intuitively, a set of CFGs connected by call/return edges adjusted for polymorphic calls and with explicit embedded control-flow due to exceptions
    - Has variable and object type info
    - Analyzed methods
    - Calls from analyzed methods to other analyzed methods or unanalyzed methods
    - Calls from unanalyzed methods to analyzed methods
    - Exception handling info

Information in JIG

- **Encode primitive types in variable names so type changes will result in changes at use sites**
- **Encode class hierarchy in globally qualified names of classes and methods**
- **Call sites become call node plus return nodes with a connecting *path edge***
  - **If calls an analyzed method, then have a call edge from the call to the method entry (polymorphic --by CHA-- calls are attached to all possible called methods)**
- **Unanalyzed methods are represented by collapsed CFG with path edge from entry to exit**

Example

"Regression Test Selection for Java Software",M.J. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S.A. Spoon, S. Gujarathi, OOPSLA'01

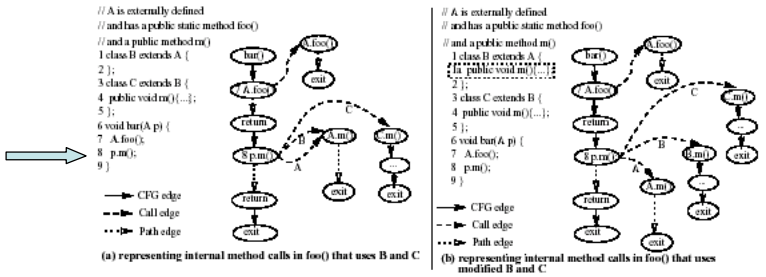

Figure 3: Example of internal method call representation.

New method B.m() inserted. Call at stmt * has changed target. so corresponding edge is dangerous.

More Details

- **Possible interactions through calls to analyzed code from the unanalyzed code**
  - **Assume happens only through polymorphism an overriding of unanalyzed method by analyzed method**
    - ***Class entry node* is connected to all public methods of a class A which can be executed on an object of type A**
      - **To catch changes in analyzed code that can affect such methods which are called from unanalyzed code**
      - **Need class entry node for any class that overrides a unanalyzed class or inherits such an overriding function**
    - ***Default node* is child of class entry node, to represent unanalyzed methods that can be executed on objects of this type; needed to handle additions or deletions of overriding method definitions**
- **Use a previously developed try-catch-finally representation of control flow due to exception handling**

A, unanalyzed

B, analyzed

C

Example

"Regression Test Selection for Java Software",M.J. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S.A. Spoon, S. Gujarathi, OOPSLA'01

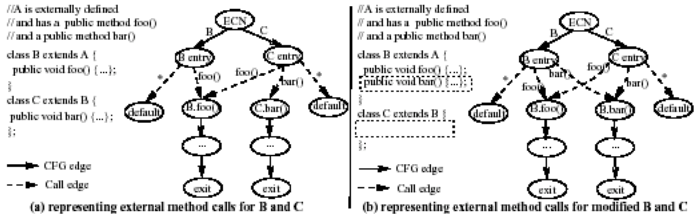

Figure 5: Example of external method call representation.

Deleted C.bar() and added B.bar().
What if deleted C.bar() and added nothing?
Now C inherits A.bar() represented by *default*.

Putting things together

- Analysis algorithm traverses JIGs in parallel and identifies *dangerous edges*
  - Node equivalence is *diff* for statements or label *diff* for non-statements
  - Algorithm is essentially same as in TOSEM article, with additions to handle new graph constructs
- Instrumentation using JVMPI of both edges and method targets of calls
  - Need to map observed edges to edges in JIG, including calls from unanalyzed methods and exception raising paths
    - If e is call from an unanalyzed method with analyzed class receiver then, Either the target is an analyzed class entry node, OR the class default node, in which case the call is mapped to all new instance creation statements for the class

Test Suite Reduction

- Four Java programs, each with several versions with associated test suites
  - Siena - internet-based event notification system
  - Jedit - text editor
  - Jmeter - desktop application for load testing
  - RegExp - GNU library for parsing regular expressions
- Measured reduction in size of regression suite needed; inconclusive results

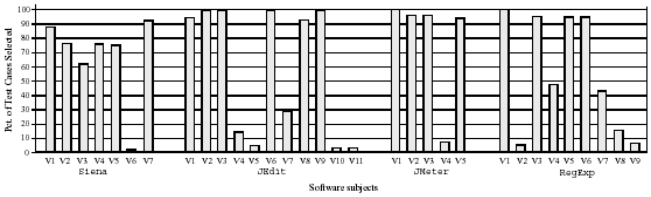

Figure 9: Regression test selection results for our software subjects.