On the use of constraints in dynamic slicing for program debugging

Franz Wotawa

Presented by Dinesh Rajan
February 21, 2012
How to enhance dynamic slicing for efficient bug detection and debugging?
Dynamic Slicing

● Dynamic slicing
  ○ small computation costs
  ○ automated bug detection
  ○ automated debugging

● But,
  ○ Too many bug candidates, false positives!
  ○ Limits applicability
Improving Dynamic Slicing

- Eliminate all bug candidates
  - except for the correct root cause (bug)

- How?
  - Assign a test case with inputs and expected outputs
  - For the failing output variable value
    - compute dynamic slice
  - Assume all statements in slice are correct
  - Eliminate statements that cannot be root cause
How to eliminate statements in slice

- Eliminate statements as follows:
  - Deduce values of variables \textit{in slice} based on outputs that satisfy test case
  - \textbf{One statement will hold a contradiction}
    - because of assumption of all statements in slice being correct
Formally...

● Debugging problem:

Given program P, test case T: (P, T) is a debugging problem iff T is a failing case.

● Formal Solution:

○ Compute Execution Trace Graph
  ■ Data dependencies
  ■ Control dependencies
  ■ Similar to PDG

○ Apply dynamic slicing algorithm

Similar to class lectures on PDG, Dynamic Slicing
Formal Elimination of Statements

- Constraint representation checking of Execution Trace
- **Three-step technique:**
  - Eliminate test elements
  - Static Single Assignment conversion
  - Constraint conversion
Solution Architecture

- Debugger()

1. Compute ETG
   - Control and data dependency
2. Apply dynamic slicing
3. Do constraint representation (eliminate bug candidates)
4. Validate bug candidate and its effects