Software Debugging – (1)

Abhik Roychoudhury
National University of Singapore
abhik@comp.nus.edu.sg

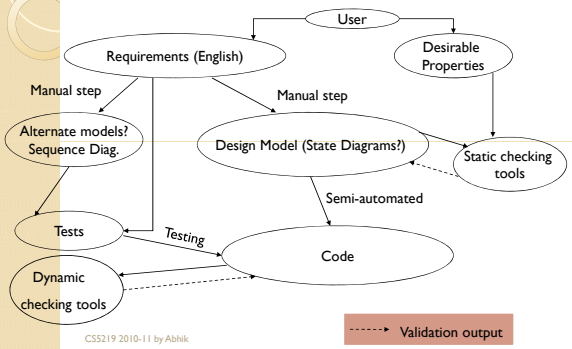
Programming

Creativity + Precision

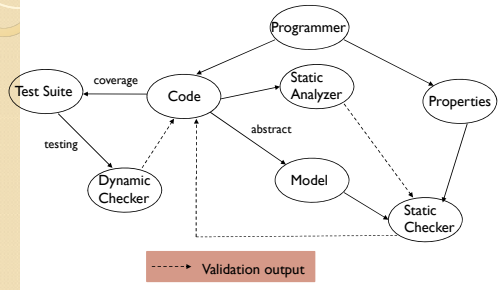
Software construction

- From a design model
  - In safety-critical domains – automotive, avionics.
  - D0 178C – software in airborne systems.
- Or, hand-constructed
  - Usual practice – audio, video and other domains.
  - UML models only for guidance.

Model-driven engineering



No model may be available.



The art of debugging

"A **software bug** (or just "bug") is an error, flaw, mistake, … in a computer program that prevents it from behaving as intended (e.g., producing an **incorrect result**). … Reports detailing bugs in a program are commonly known as **bug reports**, fault reports, … change requests, and so forth."
--- Wikipedia

More on the art…

"Even today, debugging remains very much of an art. Much of the computer science community has largely ignored the debugging problem….. over 50 percent of the problems resulted from the time and space chasm between symptom and root cause or inadequate debugging tools." (Hailpern & Santhanam, IBM Sys Jnl, 41(1), 2002)

-> Need methods and tools to trace back to the root cause of bug from the manifested error

-> What about the current tools?

Tools?

You were right: There's a needle in this haystack…

We should automatically produce the bug report via analysis of program and/or execution trace

Bug report is a small fragment of the program.

Organization

- Dynamic checking of programs
  - Dynamic slicing
  - Hierarchical slicing
  - Fault Localization

What is dynamic checking?

- Check program executions, not source code.
- How to generate program executions?
  - Testing (coverage based)
  - Testing (specification based)
- How to check program executions
  - Data and control dependencies (slicing)
  - By comparing against other program executions (fault localization).

SW Debugging: Social aspects

Software-controlled devices are ubiquitous ---
automotive control, avionics control and consumer electronics
Many of these software are safety-critical
⇒ should be validated extensively.

SW Debugging: Economics

- How often do bugs appear ?
- How many of them are critical?
- How much money does a company gain by using sophisticated debugging tools?
- Could it be avoided simply by sparing one more programmer?

SW Debugging: Economics

- SW project with 5 million LOC (note: Windows Vista is 50 million LOC !!)
  - **Assume linear scaling up of errors**
    - **Actually could be more errors --- we make more mistakes as the SW grows long and arduous.**
  - **1 hr to fix each major error**
    - **Actually much more**
  - **$40K salary per year**

$13 * \frac{5000000}{1000}$ = 65,000 bugs

$\frac{65,000}{44}$ **weeks = 1477 weeks =** $\frac{1477}{50} \approx$ **30 years = $1.2 M**

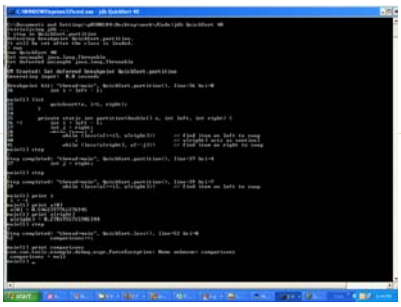
SW Debugging: tools

"Even today, debugging remains very much of an art. Much of the computer science community has largely ignored the debugging problem….. over 50 percent of the problems resulted from the time and space chasm between symptom and root cause or inadequate debugging tools." (Hailpern & Santhanam, IBM Sys Jnl, 41(1), 2002)

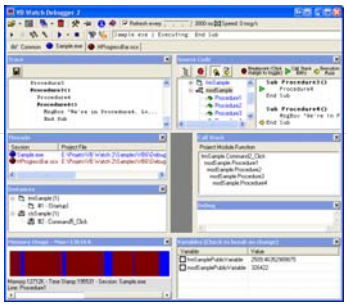
-> Need methods and tools to trace back to the root cause of bug from the manifested error

-> What about the current tools?

jdb on windows XP



VB watch debugger



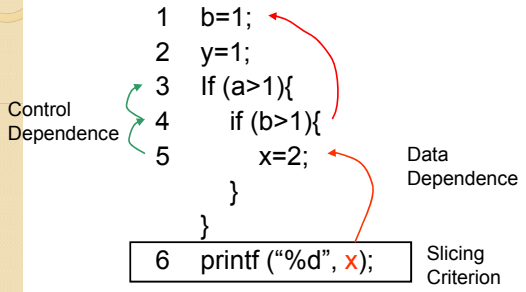
So, what did we see?

- Command line tool for Java
  - User can set breakpoints, and
  - Replay an execution, and
  - Watch it at the breakpoints.
- Lack of GUI is not the issue here.
  - Can easily collect and visualize more program info.
- Lack of automation is the problem!
  - Need automated trace analysis.

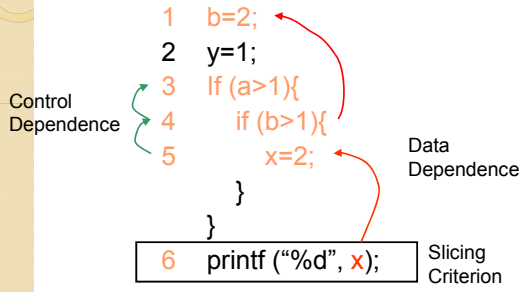
Program Slicing

```
1   b=1;
2   y=1;
3   If (a>1){
4       if (b>1){
5           x=2;
        }
    }
6   printf ("%d", x);
```

Program Slicing

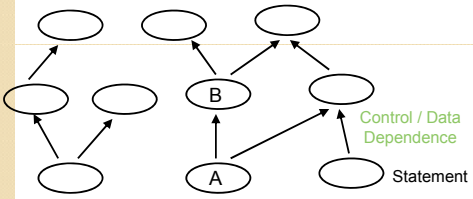


Program Slicing

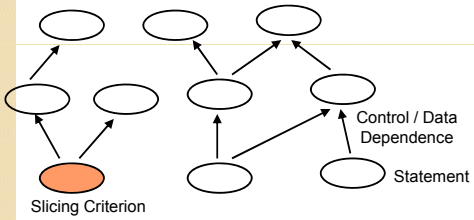


Program Dependence Graph

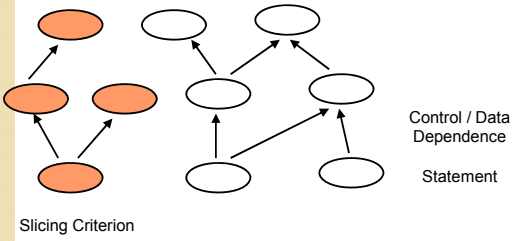
Constructed for the program. Node can be statements or instructions. Edges denote control and data dependencies.



Program Slicing



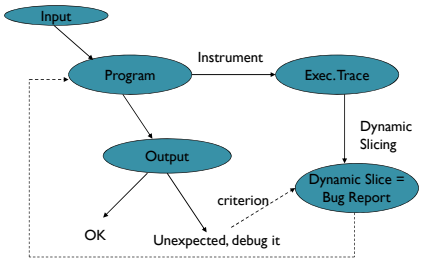
Program Slicing



Static vs Dynamic Slicing

- Static Slicing
  - source code
  - statement
  - static dependence
- Dynamic Slicing (useful for debugging)
  - a particular execution
  - statement instance
  - dynamic dependence

Dynamic Slicing for Debugging



Dynamic Slice

- Set slicing criterion
  - (Variable v at first instance of line 70)
  - The value of variable v at first instance of line 70 is unexpected.
- Dynamic slice
  - Closure of
    - Data dependencies &
    - Control dependencies
  - from the slicing criterion.

Dynamic data dependencies

An edge from a variable usage to the latest definition of the variable.

→ Do we consider this data dependence edge ?

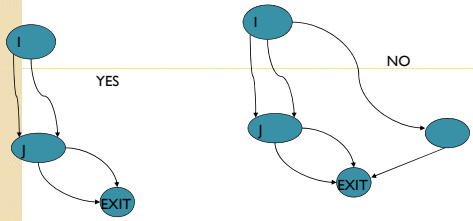
→ Remember that the slicing is for an input, so the addresses are resolved. In the trace, we have the memory addresses instead of the names A[i], A[j].

→ We thus define data dependences corresponding to memory locations rather than variable names.

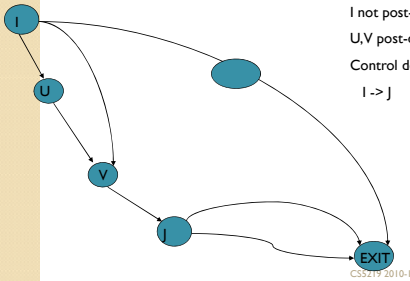
Static Control dependencies

Post-dominated: I,J – nodes in Control Flow Graph

I is post-dominated by J iff all paths from I to EXIT pass through J



Static control dependencies



I not post-dom by J

U,V post-dom by J

Control dependence

I -> J

Dynamic control dependencies

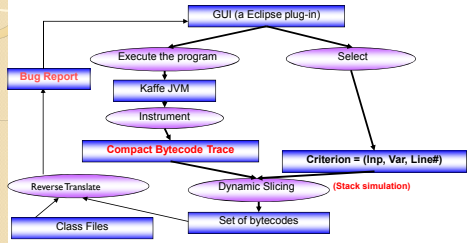
- X is dynamically control dependent on Y if
  - Y occurs before X in the execution trace
  - X's stmt. is statically control dependent on Y's stmt.
  - No statement Z between Y and X is such that X's stmt. is statically control dependent on Z's stmt.
- Captures the intuition:
  - What is the nearest conditional branch statement that allows X to be executed, in the execution trace under consideration.

Dynamic Slice

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

Slicing Criterion

Jslice: a dynamic slicing tool



http://jslice.sourceforge.net

Issues for such a slicing tool

- Online trace compression – beyond conventional string compression.
  - Full trace is never stored.
- Program dependence analysis on compressed trace – no decompression.
- Analysis at low-level (byte-code) to support third-party software.
  - Managing stack architecture.

Organization

- Dynamic checking of programs
  - Dynamic slicing
  - Hierarchical slicing
  - Fault Localization

Problem with dynamic slicing

- Huge overheads
  - Backwards slicing requires trace storage.
  - Jslice tool for Java
    - Online trace compression & traversal
    - http://jslice.sourceforge.net
- Dynamic Slice is still too large …
  - … for human comprehension
  - Now

An example

```
1   public static void main(String[] args) {
        …….
2.      init( db );
3       operate( db );
4.      output ( db )
5.      return;
}
```

SPECJVM DB program

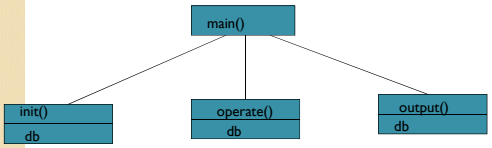
```
init( .. db) {
    db= ..
    ….
}
```

```
operate (… db ) {
      db =..
        …
}
```

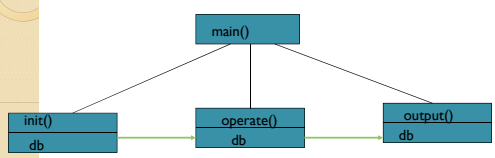
```
output (db) {
      ……
      print(db...);
}
```

Divide trace into phases

```
1   public static void main(String[] args) {
      .......
2.    init( db );
3     operate( db );
4.    output ( db );
5.    return;  }
```



Report inter-phase dependencies



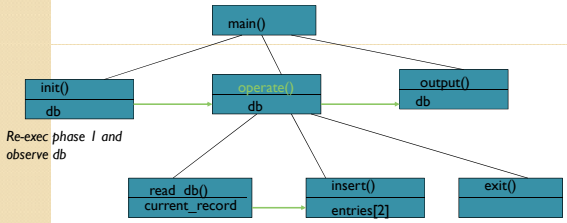
Intra-phase control and data dependencies are suppressed.

Inter-phase dep. form input-output relationships.

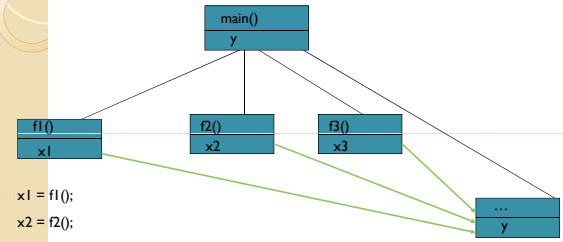
Programmer zooms into …

… one phase by inspecting the phase outputs

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



Parallel Dependence Chains



x1 = f1();

x2 = f2();

x3 = f3();

y = x1 + x2 + x3;

print y — Criterion

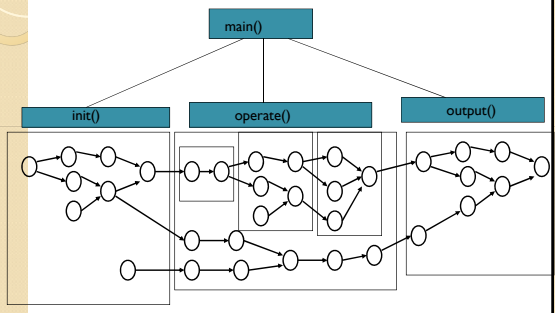
Hierarchical dynamic slicing

- Compute "phases" of an exec. trace
  - Control structure boundaries
- 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.

Programmer Intervention



So far

- Program Slicing
  - Static Slicing
  - Dynamic Slicing
    - One of the oldest debugging methods around.
  - Hierarchical Dynamic Slicing
    - Tackling the large dynamic slice in real-life programs.
- Now …
  - Relevant Slicing (why?)
    - While dynamic slice is large, it may still leave out some statements which are useful for explaining a given observable error.

Relevant Slicing

```
1 b=10;
2 x=1;
3 If (a>1){
4     if (b>1){
5         x=2;
      }
  }
6 printf ("%d", x);
```


Relevant Slicing

```
1 b=1;
2 x=1;
3 If (a>1){
4     if (b>1){
5         x=2;
      }
  }
6 printf ("%d", x);
```


Relevant Slicing

Source of Failure

Dynamic Slice

Execution is omitted

input: a=2

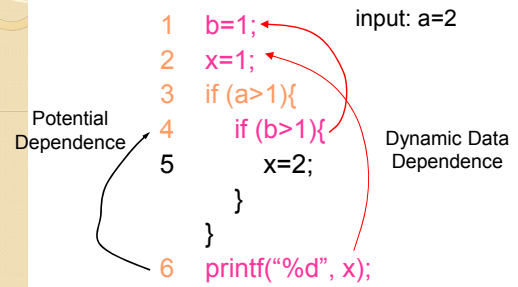
```
1 b=1;
2 x=1;
3 if (a>1){
4     if (b>1){
5         x=2;
      }
  }
6 printf ("%d", x);
```


Potential Dependence

input: a=2

```
1   b=1;
2   x=1;
3   If (a>1){
4       if (b>1){
5           x=2;
        }
    }
6   printf ("%d", x);
```

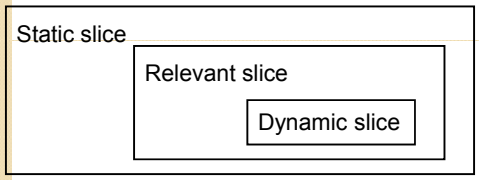
Relevant Slice



Program Slice

Static	Dynamic	Relevant	
1		1	1 b=1; input: a=2
2	2	2	2 x=1;
3			3 if (a>1){
4		4	4 if (b>1){
5			5 x=2;
			}
			}
6	6	6	6 printf ("%d", x);

Program Slicing



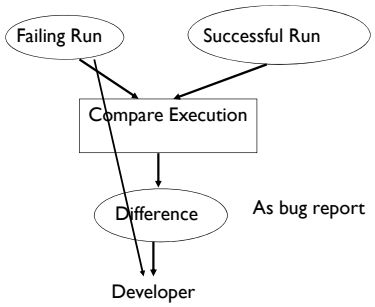
Organization

- Dynamic checking of programs
  - Dynamic slicing
  - Hierarchical slicing
  - Fault Localization

More on debugging

- Dynamic slicing analyzes the problematic execution trace.
  - Problematic: output is unexpected
  - OK: output is as expected.
- Alternatively:
  - We could compare a given problematic trace with an OK trace to localize the source of error.

Fault Localization: overview



Comparing executions

```
1 .  m=...
2.    if (m >= 0) {
3.        ...
4.        lastm = m;
5.    }
6.    …..
```

should be

if ((m >= 0) && (lastm!=m))

55

Comparing executions

```
1 .  m=...
2.    if (m >= 0) {
3.        ...
4.        lastm = m;
5.    }
6.    …..
```

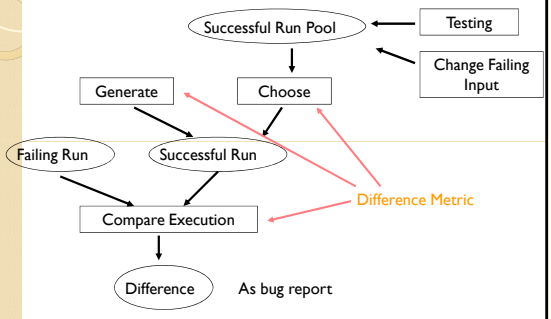
Failing run

```
1 .  m=...
2.    if (m >= 0) {
3.        ...
4.        lastm = m;
5.    }
6.    …..
```

Successful run

56

Fault localization



57

Example program

Program

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

 58

Comparing executions

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

Execution run π

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

Execution run π1

59

Set of statements

- S = Set of statements executed in π
  - {1,3,5,6,7,10}
- S1 = Set of statements executed in π1
  - {1,3,4,5,6,9,10}
- If π is faulty and π1 is OK
  - Bug report = S – S1 = {4,7}
- Choice of the execution run to compare with is very important.
  - We will see a method in the next lecture to take care of this problem!

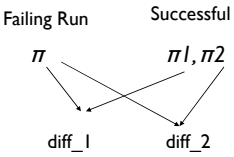
60

Do not take "statement sets"

```
while(…){          Trace 1      Trace 2
  if (c1){           if (c1)      if (c1)
    S1;}               S1           S2
  else{              S3           S3
    S2;              if (c1)      if (c1)
  }                    S2           S1
  S3;                S3           S3
}
```

Stmts(Trace1)–Stmts(Trace2)=ϕ

Another difference metric



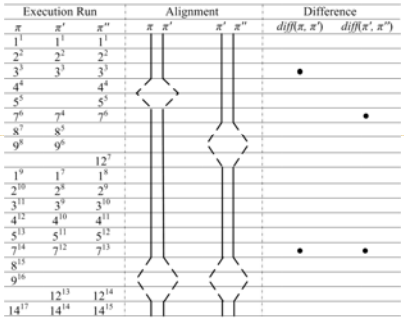
- Number of Branches
- Location of Branches

Difference b/w traces shown

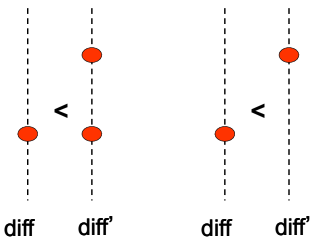
```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```


Trace alignment and differences



Comparison of differences



For more …

- Dynamic Slicing & Relevant Slicing
  - http://www.comp.nus.edu.sg/~abhik/pdf/toplas07.pdf
  - http://www.comp.nus.edu.sg/~abhik/pdf/JSlice-TR.pdf (for more details)
- Hierarchical Dynamic Slicing
  - http://www.comp.nus.edu.sg/~abhik/pdf/issta07.pdf
- Software Fault Localization
  - http://www.comp.nus.edu.sg/~abhik/pdf/cc06.pdf