Software Fault Diagnosis

Peter Zoeteweij,
Rui Abreu, Arjan J.C. van Gemund,

Thanks:
Jurryt Pietersma, Alexander Feldman, Rob Golsteijn
Automated Diagnosis

- Purpose: identify the root causes of system failures
- Originated in the AI area
  - expert systems
  - model-based diagnosis
- Primarily applied to hardware (circuits, mechanical devices), e.g.,
  - Line stuck at 0/1, valve stuck,
  - Sensors, amps not working,
  - Leakage, ...
Software

- Automated diagnosis ⇝ automated debugging
  - E.g., applications in recovery don’t require the level of detail needed for debugging
- Background: embedded systems
  - Functionality shift HW → SW
  - 25% annual growth rate
  - Nearly constant fault density
  - Decreasingly dependable systems
Dependability approaches

How to reverse the trend?

• Design / development time: decrease SW fault density + LOC
  – Formal methods, SW arch, code generation, testing, …

• Run-time: deal with imperfection
  – Fault detection, isolation, recovery (FDIR)
Software fault diagnosis

Contributes to dependability in two ways

• At (design / ) development time:
  – shortens the test – diagnose – repair cycle
  – More bugs solved → more reliable products
  – (or shorted time to market)

• At run-time:
  – Can serve as the basis for (automated) recovery
  – Requires recovery-oriented design
Outline

• Part I
  – Diagnosis principles
  – Model-Based Diagnosis
  – Spectrum-Based Fault Localization
  – Hands-on

• Part II
  – Existing systems
  – Current research
  – Case study
  – Further applications
  – Other approaches

This is what the tutorial really is about.
Spectrum-based fault localization

• Black-box technique: no modeling required
• Inherently inaccurate
• Appears to work well in practice
• Lends itself well to integration with existing testing schemes
• Low CPU and memory overhead
Integration with testing

Test suite

- t1
- t2
- t3
- t4
- t5
Integration with testing

System components are ranked according to likelihood of causing the detected errors

Status

- t1
- t2
- t3
- t4
- t5

✓
✗
**Terminology**

- **fault**
  - behavior ≠ expected behavior
  - *(segmentation fault)*
- **error**
  - system state that may cause a failure
  - *(index out of bounds)*
- **failure**
  - the cause of an error in the system
  - *(bug: array index un-initialized)*
For our purposes, the distinction between errors and failures is less relevant: failures are errors that affect the user; i.e. that are externally observable.

This depends on
• specification
• what can be observed
Example: rational bubble sort

```c
void RationalSort( int n, int *num, int *den )
{
    int i,j;

    for ( i=n-1; i>=0; i-- ) {
        assert( den[i] != 0 );
        for ( j=0; j<i; j++ ) {
            if ( RationalGT( num[j], den[j],
                             num[j+1], den[j+1] ) ) {
                swap( &num[j], &num[j+1] );
            } /* swap( &den[j], &den[j+1] ); */
        }
    }
}
```

**Fault:** forgot to swap denominators
**Error:** sequence is not a permutation of input sequence
**Failure:** output is not a sorted version of the input
Example: rational bubble sort

• Failure example:

\[
\begin{array}{c|c|c|c}
3 & 4 & 1 \\
1, 3, 4 \\
\end{array}
\rightarrow
\begin{array}{c|c|c|c}
4 & 3 & 1 \\
1, 3, 4 \\
\end{array}
\rightarrow
\begin{array}{c|c|c|c}
4 & 1 & 3 \\
1, 3, 4 \\
\end{array}
\rightarrow
\begin{array}{c|c|c|c}
1 & 4 & 3 \\
1, 3, 4 \\
\end{array}
\]
Example: rational bubble sort

- Faults do not automatically lead to errors: RationalSort works fine if the input array is already sorted, or if all denominators are equal:
Example: rational bubble sort

- Errors do not lead automatically to failures:

  \[
  \frac{4}{1}, \frac{2}{2}, \frac{0}{1}
  \]

  \[
  \frac{2}{1}, \frac{4}{2}, \frac{0}{1}
  \]

  \[
  \frac{2}{1}, \frac{0}{2}, \frac{4}{1}
  \]

  \[
  \frac{0}{1}, \frac{2}{2}, \frac{4}{1}
  \]

- Numerators are swapped
- Denominators are not swapped
- However, the end result is –by mere chance– correct
Fault Diagnosis

Identify component(s) that are root cause of failure

\[ y = f(x, h) \]

\( x, y \): observation vectors
\( f \): system function, \( f_i \): component functions
\( h \): system health state vector, \( h_i \): component health vars

Diagnose failure: solve inverse problem \( h = f^{-1}(x, y) \)
Diagnosis: \( h_2 = \) fault state, or \( h_4 \) and \( h_5 = \) fault state
Example Fault Diagnoses

- stuck-at-x, stuck valve, motor (HW)
- sensors, amps not working (HW, SW)
- wires disconnected, crossed (HW, SW)
- wrong function, leakage (HW, SW)
- excessive component delay (HW, SW)
- component intermittent out-of-spec (HW, SW)
- ...

TU Delft
suppose we only have nominal model (M) of f:
- can only test for system failure, not locate component failure

\[ y' = M(x) \]

model M of nominal behavior
suppose we also have models \((M_i)\) of all \(f_i\):
- can infer location(s) of failure (Model-Based Diagnosis)

\[
y = f(x, h)
\]

\[
y' = M(x, h')
\]

search for \(h' = h\) such that \(y'\) consistent with \(y\)
Modeling Information (3)

- suppose we only have trace on involvement of $f_i$:
  - can infer location(s) of failure (Spectrum-Based Diagnosis)

\[
\begin{align*}
\text{Pass/Fail} & = \text{y} = f(x,h) \\
\text{trace} & = \text{y'} = M(x) 
\end{align*}
\]
Models in diagnosis

- Define expected behavior
- Need not be explicit
- May help reason about faults

\[ y_1 = y_2 = x \]
Reasoning about faults
Reasoning about faults
Reasoning about faults
Reasoning about faults

Invalid explanation!
Reasoning about faults

![Diagram of a circuit with inputs and outputs]

<table>
<thead>
<tr>
<th>valid</th>
<th>(h₁,h₂,h₃)</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,0,1)</td>
<td></td>
<td>(1,1,1)</td>
</tr>
<tr>
<td>(0,0,1)</td>
<td></td>
<td>(0,1,1)</td>
</tr>
<tr>
<td>(1,0,0)</td>
<td></td>
<td>(1,1,0)</td>
</tr>
<tr>
<td>(0,1,0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,0,0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reasoning about faults

\[ P(\text{fail}) = 0.01 \]

<table>
<thead>
<tr>
<th>valid</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (1,0,1) ) 0.009801</td>
<td>( (1,1,1) )</td>
</tr>
<tr>
<td>( (0,0,1) ) 0.000099</td>
<td>( (0,1,1) )</td>
</tr>
<tr>
<td>( (1,0,0) ) 0.000099</td>
<td>( (1,1,0) )</td>
</tr>
<tr>
<td>( (0,1,0) ) 0.000099</td>
<td>( (1,1,0) )</td>
</tr>
<tr>
<td>( (0,0,0) ) 0.000001</td>
<td>( (1,1,0) )</td>
</tr>
</tbody>
</table>
Model-based diagnosis

model component $i$:
$h_i \Rightarrow (y_i = \neg x_i)$

model system:
$h_1 \Rightarrow (y_1 = \neg x_1)$
$h_2 \Rightarrow (y_2 = \neg x_2)$
$h_3 \Rightarrow (y_3 = \neg x_3)$
$y_1 = z$
$z = x_2$
$z = x_3$
Model-based diagnosis

• For a given input, the model specifies a function

\[ f : \text{health states} \times \text{input} \rightarrow \text{output} \]

• Diagnosis entails calculating the inverse:

\[ f^{-1} : \text{input} \times \text{output} \rightarrow \text{health states} \]

• \( f^{-1} \) can be computed by truth maintenance system
Improving MBD accuracy

(1,0,1)  
(0,0,1)  
(1,0,0)  
(0,1,0)  
(0,1,0)  
(0,0,0)
Improving MBD accuracy

Add a second observation

\begin{align*}
\text{multiple} \\
(1,0,1) \\
(0,0,1) & (0,0,1) \\
(1,0,0) & (1,0,0) \\
(0,1,0) & (0,1,0) \\
(0,0,0) & (0,0,0)
\end{align*}
Improving MBD accuracy

Add more probes

\[
\begin{align*}
\text{multiple} & \quad z=1 \\
(1,0,1) & \\
(0,0,1) & \quad (0,0,1) \\
(1,0,0) & \quad (1,0,0) \\
(0,1,0) & \quad (0,1,0) \quad (0,1,0) \\
(0,0,0) & \quad (0,0,0) \quad (0,0,0)
\end{align*}
\]
Model strength

- Different models capture different failure modes
- This is called the model “strength”
- “Stronger” inverter model:

\[
\begin{align*}
h_i = ok & \implies (y_i = \neg x_i) \\
h_i = stuck at 1 & \implies (y_i = 1) \\
h_i = stuck at 0 & \implies (y_i = 0) \\
h_i = bypass & \implies (y_i = x_i)
\end{align*}
\]
Model strength

model component $i$:

$h_i \Rightarrow (y_i = \neg x_i)$

$\neg h_i \Rightarrow \neg y_i$

“stuck-at-zero”
Improving MBD accuracy

\[
\begin{array}{cccc}
\text{weak} & \text{strong} \\
(1,0,1) & (1,0,1) \\
(0,0,1) & (0,0,1) \\
(1,0,0) & \\
(0,1,0) & \\
(0,0,0) & \\
\end{array}
\]
Weak model

\[ x \xrightarrow{c_1} y_1 \xrightarrow{c_2} y_2 \xrightarrow{c_3} y_1 = y_2 = x \]

- Expected behavior is specified
- Functionality of \( c_1, c_2, \) and \( c_3 \) is unknown
- \( c_1 \) and \( c_2 \) determine \( y_1 \)
- \( c_1 \) and \( c_3 \) determine \( y_2 \)
Weak model

- $c_1$ is involved both in correct and incorrect behavior
- $c_3$ is only involved in correct behavior
- $c_2$ is the only component that is exclusively involved in the computation of an incorrect result
Spectrum-based fault localization

• Identify the components / parts whose activities coincide with the occurrence of failures
• Software can be seen as an executable model, that tells us which parts of a system are involved in a computation
Program spectra

• Execution profiles that indicate, or count which parts of a software system are used in a particular test case
• Introduced in [Reps97] for diagnosing Y2K problems
• Many different forms exist [Harrold98]:
  – Spectra of program locations
  – Spectra of branches / paths
  – Spectra of data dependencies
  – Spectra of method call sub-sequences
Block / function hit spectra

Function hit spectrum

| $x_1$ | $x_2$ | ... | $x_i$ | ... | $x_n$ |

1: function $i$ called
0: function $i$ not called

Block hit spectrum

1: block $i$ executed
0: block $i$ not executed

Block:

- C statement (compound stmt)
- cases of a switch statement
Fault diagnosis

1. Spectra for $m$ test cases

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>...</th>
<th>$x_{1n}$</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>...</td>
<td>$x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>...</td>
<td>$x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>
Fault diagnosis

1. Spectra for $m$ test cases

|    | $x_{11}$ | $x_{12}$ | … | … | $x_{1n}$ |  | $e_1$ |
|----|----------|----------|---|---|---------|  |       |
| $x_{21}$ | $x_{22}$ | … | … | … | $x_{2n}$ |  | $e_2$ |
| … | … | … | … | … | … |  | … |
| $x_{m1}$ | $x_{m2}$ | … | … | … | $x_{mn}$ |  | $e_m$ |

Row $i$: the blocks that are executed in case $i$
Fault diagnosis

1. Spectra for $m$ test cases

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>…</th>
<th>$x_{1n}$</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>…</td>
<td>$x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>…</td>
<td>$x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>

Column $j$: the test cases in which block $j$ was executed
## Fault diagnosis

1. Spectra for $m$ test cases
2. Error detection per test case

### Table

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>…</th>
<th>$x_{1n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>…</td>
<td>$x_{2n}$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>…</td>
<td>$x_{mn}$</td>
</tr>
</tbody>
</table>

![](image.png)

$e_i = 1$ : error in the $i$-th test  
$e_i = 0$ : no error in the $i$-th test
Fault diagnosis

Compare every column vector with the error vector.

$$\begin{array}{ccc}
x_{11} & x_{12} & \cdots \cdots \cdots \\
x_{21} & x_{22} & \cdots \\
\cdots & \cdots & \cdots \\
x_{m1} & x_{m2} & \cdots \\
\end{array}$$

$$\begin{array}{c}
x_{1n} \\
x_{2n} \\
\cdots \\
x_{mn} \\
\end{array}$$

$$\begin{array}{c}
e_1 \\
e_2 \\
\cdots \\
e_m \\
\end{array}$$

similarity $s_j$
Fault diagnosis

Jaccard similarity coefficient:

block $j$

| 1 | 0 | 1 | 0 | 1 |

error vector

| 0 | 1 | 1 | 0 | 1 |

$s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}$
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}} \]

<table>
<thead>
<tr>
<th>block ( j )</th>
<th>error vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{2}{2 + a_{10} + a_{01}} \]

<table>
<thead>
<tr>
<th>block ( j )</th>
<th>error vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Fault diagnosis

Jaccard similarity coefficient:

\[
\begin{align*}
    s_j &= \frac{2}{2 + 1 + a_{01}} \\
    &\text{block } j \\
    &\begin{array}{c}
        1 \\
        0 \\
        1 \\
        0 \\
        1 \\
    \end{array} \\
    &\text{error vector} \\
    &\begin{array}{c}
        0 \\
        1 \\
        1 \\
        0 \\
        1 \\
    \end{array}
\end{align*}
\]
Fault diagnosis

Jaccard similarity coefficient:

\[
s_j = \frac{2}{2 + 1 + 1}
\]
Fault diagnosis

For every block: similarity with the error “block”

<table>
<thead>
<tr>
<th></th>
<th>$n$ blocks</th>
<th>error vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>...</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>...</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_2$</td>
<td>...</td>
</tr>
</tbody>
</table>

The block with the highest $s_i$ most likely contains the fault.
Fault diagnosis

\[ s = \frac{a_{11}}{a_{11} + a_{10} + a_{01}} \]

<table>
<thead>
<tr>
<th>part</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>test 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>test 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>test 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>test 5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \frac{2}{3} \quad \frac{1}{2} \quad \frac{1}{4} \quad \frac{3}{4} \quad \frac{1}{4} \quad \frac{1}{3} \quad \frac{2}{3} \)
Example: rational bubble sort

```c
void RationalSort( int n, int *num, int *den )
{
    int i,j;                                           /* block 1 */
    for ( i=n-1; i>=0; i-- ) {
        assert( den[i] != 0 );                          /* block 2 */
        for ( j=0; j<i; j++ ) {
            if ( RationalGT( num[j], den[j],             /* block 3 */
                    num[j+1], den[j+1] ) ) {
                swap( &num[j], &num[j+1] );               /* block 4 */
                /* swap( &den[j], &den[j+1] );  */
            }
        }
    }
}
```

**Fault:** forgot to swap denominators  
**Error:** sequence is not a permutation of input sequence  
**Failure:** output is not a sorted version of the input
Example (2)

<table>
<thead>
<tr>
<th>input</th>
<th>error</th>
<th>output</th>
<th>failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$  $\langle \rangle$</td>
<td>no</td>
<td>$\langle \rangle$</td>
<td>no</td>
</tr>
<tr>
<td>$I_2$  $\langle \frac{1}{4} \rangle$</td>
<td>no</td>
<td>$\langle \frac{1}{4} \rangle$</td>
<td>no</td>
</tr>
<tr>
<td>$I_3$  $\langle \frac{2}{1}, \frac{1}{1} \rangle$</td>
<td>no</td>
<td>$\langle \frac{1}{1}, \frac{2}{1} \rangle$</td>
<td>no</td>
</tr>
<tr>
<td>$I_4$  $\langle \frac{4}{1}, \frac{2}{2}, \frac{0}{1} \rangle$</td>
<td>yes</td>
<td>$\langle \frac{0}{1}, \frac{2}{2}, \frac{4}{1} \rangle$</td>
<td>no</td>
</tr>
<tr>
<td>$I_5$  $\langle \frac{3}{1}, \frac{2}{2}, \frac{4}{3}, \frac{1}{4} \rangle$</td>
<td>yes</td>
<td>$\langle \frac{1}{1}, \frac{2}{2}, \frac{4}{3}, \frac{3}{4} \rangle$</td>
<td>yes</td>
</tr>
<tr>
<td>$I_6$  $\langle \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1} \rangle$</td>
<td>no</td>
<td>$\langle \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1} \rangle$</td>
<td>no</td>
</tr>
</tbody>
</table>
Example (3)

<table>
<thead>
<tr>
<th>input</th>
<th>block</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$I_1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$I_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$I_4$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$I_5$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$I_6$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$s_j$</td>
<td>1/6</td>
<td>1/5</td>
</tr>
</tbody>
</table>

Block 4 has highest similarity coefficient -> most likely suspect
Hands-on

- Requirements:
  - Computer
  - Linux / Cygwin with gcc and gcov
  - getting_started.tar.gz
Build the tools

$ gunzip getting_started.tar.gz
$ tar xvf getting_started.gz

$ gcc gcov2spectrum.c -o gcov2spectrum
$ gcc diagnosis.c -o diagnosis
Instrument executable

$ gcc -fprofile-arcs -ftest-coverage rsort.c -o rsort
Run rsort

$ rsort 4 1 2 2 0 1

rsort

rsort.gcno

0/1 2/2 4/1

rsort.gcda
Run gcov

$ gcov rsort.c

rsort

rsort.gcno

rsort.gcda

gcov

rsort.c .gcov
Run gcov

$ gcov2spectrum < rsort.c.gcov
Diagnosis

• Perform these steps for all testcases
• After each run, remove rsort.gcda (it is incremental)
• Save the spectra in a file
runall script

...

$ ./rsort 2 1 1 1
$ gcov rsort.c
$ rm rsort.gcda
$ ./gcov2spectrum < rsort.c.gcov >> spectra.txt

$ ./rsort 4 1 2 2 0 1
$ gcov rsort.c
$ rm rsort.gcda
$ ./gcov2spectrum < rsort.c.gcov >> spectra.txt

...
Output

sorted:
sorted: 1/4
sorted: 1/1 1/2
sorted: 0/1 2/2 4/1
sorted: 1/1 2/2 4/3 3/4
sorted: 1/4 1/3 1/2 1/1
**spectra.txt**

Add a column of error information (… by hand)

<table>
<thead>
<tr>
<th>...</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Analysis

$ diagnosis 6 101 spectra.txt

22 (0.33) 23 (0.33) 24 (0.33) 11 (0.25) 6 (0.25)
20 (0.25) 7 (0.25) 54 (0.20) 45 (0.20) 19 (0.20)
53 (0.17) 56 (0.17) 52 (0.17) 51 (0.17) 33 (0.17)
32 (0.17) 16 (0.17) 35 (0.17) 18 (0.17) 39 (0.17)
44 (0.17) 15 (0.17) 57 (0.17)
Benchmark set

- Siemens test set [Hutchins94]:
  - 7 C programs, 20 – 124 blocks, 173 – 565 lines.
  - 7 – 32 faulty versions per program:
    132 faults in total
  - Up to 1000 – 5000 test cases per program:
    full code coverage

- Failure:
  Output of a faulty version differs from the output of the reference program.

- Google print_tokens.tar.gz
Model-based vs. Spectrum-based

Model-based
- Model used primarily for reasoning
- All generated explanations are valid
- Most likely diagnosis need not be actual cause
- Well suited for hardware

Spectrum-based
- Model used primarily for error detection
- Ranking contains invalid explanations
- Invalid explanations may rank high
- Well suited for software
Outline

• Part I
  – Diagnosis principles
  – Model-Based Diagnosis
  – Spectrum-Based Fault Localization
  – Hands-on

• Part II
  – Existing systems
  – Current research
  – Case study
  – Further applications
  – Other approaches
Existing applications

- Pinpoint: large on-line transaction processing systems (search engines, web mail) [Chen02]
- AMPLE: Java software (method call sequences) [Dallmeier05]
- Tarantula: visualizing test information to aid manual debugging [Jones02]
Pinpoint

- Spectra: usage profiles of servers, disks, applications
- Error detection:
  - exceptions that propagate to the J2EE layer
  - HTTP errors
AMPLE

• Spectra: hit spectra of *method call subsequences* received or issued by an object
• Similarities are interpreted as *weights*, which are accumulated per object and per class
• Diagnosis is at class level: classes with the highest weights are most suspect
• Parameters:
  – Subsequence length
  – Incoming / outgoing calls
Existing applications

• Pinpoint: large on-line transaction processing systems (search engines, web mail) [Chen02]
• AMPLE: Java software (method call sequences) [Dallmeier05]
• Tarantula: visualizing test information to aid manual debugging [Jones02]
Similarity Coefficients

- Jaccard (PinPoint)

\[ s_j = \frac{a_{11}}{a_{11} + a_{01} + a_{10}} \]

- Tarantula

\[ s_j = \frac{a_{11}}{a_{11} + a_{01}} + \frac{a_{10}}{a_{10} + a_{00}} \]

- AMPLE

\[ s_j = \left| \frac{a_{11}}{a_{01} + a_{11}} - \frac{a_{10}}{a_{00} + a_{10}} \right| \]

- Ochiai (molecular biology)

\[ s_j = \frac{a_{11}}{\sqrt{(a_{11} + a_{01}) \cdot (a_{11} + a_{10})}} \]
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}} \]
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}} \]
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{2}{2 + a_{10} + a_{01}} \]
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{2}{2 + 1 + a_{01}} \]
Fault diagnosis

Jaccard similarity coefficient:

\[ s_j = \frac{2}{2 + 1 + 1} \]
Diagnostic quality

- Percentage of blocks that need **not** be inspected:
Conclusion

• Under the specific conditions of our experiment, Ochiai coefficient outperforms 8 other coefficients.

• Why?

• To what extent does this depend on the conditions of our experiment?
  – Quality of the passed / failed information
  – Numers of runs
  – Artificial bugs in Siemens set
Ochiai outperforms Tarantula

\[
\frac{a_{11}/(a_{11}+a_{01})}{a_{11}/(a_{11}+a_{01}) + a_{10}/(a_{10}+a_{00})}
\]

\[
1 / \left( 1 + \frac{a_{10}}{a_{10}+a_{00}} \frac{a_{11}+a_{01}}{a_{11}} \right)
\]

\[
1 / (1 + c \frac{a_{10}}{a_{11}}), \quad \text{with } c = \frac{a_{11}+a_{01}}{a_{10}+a_{00}} = \frac{NF}{NP}
\]

\(a_{11}>0\)
Ochiai outperforms Tarantula

Tarantula

\[ 1 / (1 + c \frac{a_{10}}{a_{11}}) \]

Ochiai

\[ \frac{a_{11}}{\sqrt{((a_{11} + a_{01})(a_{11} + a_{10}))}} \]
Ochiai outperforms Tarantula

Tarantula

\[ \frac{1}{1 + c \frac{a_{10}}{a_{11}}} \]

Only presence in passed runs lowers the similarity

Ochiai

\[ \frac{a_{11}}{\sqrt{(a_{11} + a_{01})(a_{11} + a_{10})}} \]

Absence in failed runs also lowers the similarity
Ochiai outperforms Jaccard

Jaccard

\[
\frac{a_{11}}{a_{11} + a_{01} + a_{10}}
\]

Ochiai

\[
\frac{a_{11}}{\sqrt{((a_{11} + a_{01})(a_{11} + a_{10}))}}
\]
Ochiai outperforms Jaccard

\[
\frac{a_{11}}{\sqrt{(a_{11}+a_{01})(a_{11}+a_{10})}}
\]

square

\[
\frac{a_{11}^2}{((a_{11}+a_{01})(a_{11}+a_{10}))}
\]

rewrite denominator

\[
\frac{a_{11}^2}{a_{11}^2+a_{11}a_{10}+a_{11}a_{01}+a_{01}a_{10}}
\]

eliminate \(a_{11}\)

\[
\frac{a_{11}}{a_{11}+a_{10}+a_{01}+ a_{01}a_{10}/a_{11}}
\]

None of these steps modifies the ranking!
Ochiai outperforms Jaccard

**Jaccard**
\[
\frac{a_{11}}{a_{11} + a_{01} + a_{10}}
\]

**Ochiai**
\[
\frac{a_{11}}{a_{11} + a_{10} + a_{01} + \frac{a_{01}a_{10}}{a_{11}}}
\]

Differences are amplified
Quality of the passed / failed info

• Failure detection is a crude error detection mechanism.
• \( q_e = \frac{a_{11}}{(a_{11} + a_{10})} \)
• In the Siemens Set, \( q_e \) ranges from 1.4% on average for schedule2 to 20.3% on average for tot_info.
• Can be increased by excluding a run that contributes to \( a_{10} \)
• Can be decreased by excluding a run that contributes to \( a_{11} \)
Quality of the passed / failed info

Small fraction of fault activations detected is enough
Number of runs

• On average, for the Siemens set:
  – Adding more failed tests is safe
  – 6 failed tests are enough
  – The number of passed tests has no influence

• However:
  – For individual runs the effect of adding passed tests differs
  – It stabilizes around 20 passed tests
Influence of \#runs
Influence of #runs

- On average, for our benchmark:
  - Adding failed runs is safe
  - 6 failed runs is enough
  - The number of passed runs has no influence

- However
  - For individual runs, the effect of more passed runs differs
  - It stabilizes around 20
Dependence on Siemens set faults

- Investigate industrial relevance in TRADER project: improve the user-perceived reliability of high-volume consumer electronics devices
- Test case: television platform from NXP
- Partners:
  - Universities of Delft, Twente, Leiden,
  - Embedded Systems Institute, Design Technology Institute, IMEC Leuven
  - NXP, TASS
Embedded systems

- Low overhead
- Little infrastructure needed
- Consumer electronics
  - No time for exhaustive debugging
  - Helps to identify responsible teams / developers

- Diagnosis can drive a recovery mechanism, e.g., rebooting suspect processes
Case study – platform

- Control software of an analog TV
- Decoding RC input, displays the on-screen menu, teletext, optimizes parameters for audio / video processing based on signal analysis, etc.
- 450 K lines of C code
- 2 MB of RAM + 2 MB in development version
- CPU: MIPS running a small multi-tasking OS
- Work is organized in 315 logical threads
- UART connection to a PC
Case study

1. Load problem:
Diagnosis

• 150 hit spectra of 315 functions, corresponding to the logical threads (one per second):
  60 sec. TV, 30 sec. TXT, 60 sec. TV
• Marked the last 60 spectra as failed
• 2nd in ranking of 315 functions
Case study

2. Teletext lock-up:
   - Existing problem in another product line
   - Copied to our platform, triggered by a remote control key sequence
   - Inconsistency in two state variables, for which only specific combinations are allowed
Lock-up problem

• Fault: case study lock-up
  – In text mode, the sequence introduces a state inconsistency

• Error detection:
  – Check on the two variables involved in the inconsistency (Hasan Sozer)

• Collecting spectra:
  – Instrumentation using Front
  – Small Koala component for caching / transmitting spectra
  – Transaction: time between two key presses

• Diagnosis:
  – block that introduces the inconsistency
Bool mgkey__rkeyntf_OnUp (KeySource source, KeySystem system, KeyCommand command)
{
    hook_log (20345);
    if ((1) && Enabled) {
        Bool translated=0;
        hook_log (20346);
        hook_EndTransaction ();
        ...
        if ( !translated) {
            hook_log (20349);
            Translate (source, system, &command);
        }
    }
    if (command >= 1000 && command <= 1009) {
        hook_log (20350);
        seq[0] = seq[1];
        seq[1] = seq[2];
        seq[2] = seq[3];
        seq[3] = command - 1000;
        if ( !triggered) {
            hook_log (20351);
            if (seq[0] == 1 && seq[1] == 2) {
                hook_log (20353);
                triggered = 1;
                switch (seq[3]) {
                    case 1:
                        hook_log (20354);
                        tmode = 6;
                        break;
                    case 2:
                        hook_log (20355);
                        ...
                }
            }
        }
    }
}

start a new spectrum

log use of the block in the current spectrum

inconsistency

Remember block 20354
## Experiments

$n > 60,000$

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>...</th>
<th>$x_{1n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>...</td>
<td>$x_{2n}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>...</td>
<td>$x_{mn}$</td>
</tr>
</tbody>
</table>

| $e_1$ | $e_2$ | ... | $e_m$ |

6 - 26 trans.

13,451 – 13,796 blocks were executed in the scenarios.
Scenario 1

23 key presses:

P+  P-  Vol+  Vol-
Txt
751 100 121 100 Txt  Txt  Txt  751

- 23 spectra of 65535 bytes
- 23 error detection reports: 22 pass, 1 fail
Diagnosis

Block numbers sorted on decreasing similarity to the error vector:

20353 (1/1)  **20354** (1/1)  58890 (1/4)  3134 (1/5)  3664 (1/6)  3135 (1/6)  58889 (1/7)  59839 (1/8)  29569 (1/9)  1256 (1/9)  15755 (1/10)  20351 (1/10)  15781 (1/11)  15777 (1/11)  15778 (1/11)  15779 (1/11)  15782 (1/11)  15823 (1/11)  20432 (1/11)  15727 (1/11) ...

- Block 20354 is right at the top of the diagnosis!
- ... but it shares the first position with block 20353.
Scenario 2

26 key presses:

P+  P-  Vol+  Vol-
Txt

121 751 100 121 100 121 100 121 751 100 121 100 121 100 121 751

The sequence 121 7 exonerates block 20353
Diagnosis for scenario 2

- Block 20354 is diagnosed correctly
- Its Jaccard similarity to the error vector is twice as large as that of any other block
Error detection

Log an error on each transition from a consistent state
To an inconsistent state:

```c
void hook_EndTransaction()
{
  ...
  if (IS_TXT_OFF(tmode) == IS_DISPLAY_TXT(CurrentDisplayProfile) )
  {
    if ( gv_error_prev == 0 )
    {
      hook_log( PROFILE_SIZE * 2 - 1 );
    }
    gv_error_prev = 1;
  }
  else
  {
    gv_error_prev = 0;
  }
  ...
}
```

Sample code
Used for testing purposes only!
Error detection

Definition of error is important

- **Inconsistent state**: all transactions after the first inconsistency are considered to demonstrate an error. This obscures the actual cause.

- **Change to inconsistent state**: also obscures the cause if the scenario includes TV mode to Txt transitions after the first inconsistency.
Conclusion

- Spectrum-based fault diagnosis can help to improve the efficiency of the debugging phase
- It can easily be integrated with testing
- It is a feasible technique in the area of embedded systems
- Proof of concept tested on industrial code (TV software) and representative errors
Experiments

• Off-loading spectra via UART for off-line analysis
• Half-word encoded spectra
• Critical sections
• Idle time after 24 transactions (tests)
• TV ran really slow (zapping takes seconds), but stable
Resource constraints

- Limited memory
- Limited CPU time
- Concurrency
Memory constraints

- It is not possible to store the spectra for all tests on the embedded device
- E.g., block level instrumentation of the TV software yields spectra of over 60,000 flags
- Using a byte per flag, we could store 24 spectra
- Offloading a spectrum via the UART took several seconds
- Fortunately, we don’t need to store all spectra
Update counters at run-time

\[
    s = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}
\]

<table>
<thead>
<tr>
<th>Current sp.</th>
<th>passed</th>
<th>failed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>a_{00}</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>a_{10}</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>a_{01}</td>
<td></td>
<td>++</td>
</tr>
<tr>
<td>a_{11}</td>
<td></td>
<td>++</td>
</tr>
</tbody>
</table>
Perform diagnosis anytime

1. Calculate coefficients:

\[
s = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}
\]

<table>
<thead>
<tr>
<th></th>
<th>1/11</th>
<th>1/7</th>
<th>1/5</th>
<th>1/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{00}</td>
<td>...</td>
<td>3</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>a_{10}</td>
<td>...</td>
<td>10</td>
<td>11</td>
<td>...</td>
</tr>
<tr>
<td>a_{01}</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>a_{11}</td>
<td>...</td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>

2. Sort
# Self diagnosing system

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>test 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>test 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>test 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>test 5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a₀₀</th>
<th>a₀₁</th>
<th>a₁₀</th>
<th>a₁₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a₀₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a₁₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a₁₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Jaccard*
Self diagnosing system

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**test 1**

- test 2
- test 3
- test 4
- test 5

| a_{00} | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| a_{01} | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a_{10} | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| a_{11} | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Jaccard

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
Self diagnosing system

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>test 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| a_{00} | 1  | 0  | 0  | 0  | 0  | 1  | 1  |      |
| a_{01} | 1  | 1  | 1  | 0  | 1  | 0  | 0  |      |
| a_{10} | 0  | 1  | 1  | 1  | 1  | 0  | 0  |      |
| a_{11} | 0  | 0  | 0  | 1  | 0  | 1  | 1  |      |

Jaccard 0 0 0 $\frac{1}{2}$ 0 1 1 1
# Self diagnosing system

A table showing the results of various tests:

<table>
<thead>
<tr>
<th>Tests</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>test 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>test 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>test 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Four vectors are described:

<table>
<thead>
<tr>
<th>Vectors</th>
<th>a₀₀</th>
<th>a₀₁</th>
<th>a₁₀</th>
<th>a₁₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀₀</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a₀₁</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a₁₀</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a₁₁</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The Jaccard coefficient table:

<table>
<thead>
<tr>
<th>Jaccard</th>
<th>½</th>
<th>⅓</th>
<th>⅓</th>
<th>⅔</th>
<th>0</th>
<th>½</th>
<th>½</th>
</tr>
</thead>
</table>
## Self diagnosing system

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>test 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>test 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>test 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>test 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a_00</th>
<th>a_01</th>
<th>a_10</th>
<th>a_11</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>test 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>test 3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>test 4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>test 5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

| Jaccard | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{3}$ | $\frac{2}{3}$ | 0 | $\frac{1}{2}$ | $\frac{1}{2}$ |
## Self diagnosing system

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test 5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(a_{00})</th>
<th>(a_{01})</th>
<th>(a_{10})</th>
<th>(a_{11})</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

| Jaccard | \(\frac{2}{3}\) | \(\frac{1}{2}\) | \(\frac{1}{4}\) | \(\frac{3}{4}\) | \(\frac{1}{4}\) | \(\frac{1}{3}\) | \(\frac{2}{3}\) |
Self diagnosing system

We obtain the same diagnosis without storing any data related to the individual test cases.
CPU time constraints

- Many embedded systems have real-time constraints (e.g., update display area during vertical blank)
- Recording a spectrum involves setting a bit for every block / function / etc. executed: unavoidable, but affordable
- Processing the recorded spectra must be done on a low-priority thread
Spectrum cache

Tests / transactions should allow sufficient idle time to prevent overflow of the spectrum cache.
Concurrency

- Hit-spectra occupy a bit per block/function/etc.
- Bit level access is not atomic: two threads modifying different bits in the same word lead to incorrect results
- Options:
  - Critical section per update (time)
  - Use a word per block/function/etc. (space)
  - Record spectra per thread (time + space)
## Trade-offs

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small cache</td>
<td>Large cache</td>
</tr>
<tr>
<td>Idle time in tests</td>
<td>Fast tests</td>
</tr>
<tr>
<td>critical sections</td>
<td>atomic updates</td>
</tr>
<tr>
<td>bit encoded spectra</td>
<td>word encoded spectra</td>
</tr>
</tbody>
</table>
Further applications

• Systems that warn of possible errors within themselves [Reps97]
  – Obtain spectra for nominal behavior in a warming-up period
  – Generate a warning if previously unseen behavior is detected

• Recovery: reset those processes whose behavior appears to correlate with potentially dangerous situations
  – Form of software rejuvenation [Huang95]
  – Requires recovery-oriented design
Related work

Delta Debugging [Zeller]:

- **Search** for the smallest difference (*delta*) in the initial state (input) of a passed run and a failed run that causes the failure of interest
  - Maintain dependencies between variables to guarantee valid states
- Interactively advance both runs in the debugger, and repeat the search on the current state
- Stop when the failure of interest occurs
- This results in a sequence of failure-inducing states that helps to locate the fault
Related work

- Nearest-neighbor [Renieris, Reiss]
- Compare spectra for
  - Single failed run
  - Most similar passed run

- Can be seen as SFL on a subset of all spectra
- Many variants for selecting the subset are possible
- Initial experiments did not promise great benefits

- [Jones05]: Tarantula outperforms NN and DD
Related work

DD + dynamic slicing [Gupta et al]:

• Backward (dynamic) slice: all statements that influence the value of a variable at a point in the execution.

• Forward: all statements that are affected by a variable at a point in the execution.

• Intersect:
  – Forward slice of the minimal failure inducing input difference
  – Backward slice of the variables where the failure occurs
Related work

- Reported results are impressive
- Slicing is expensive

min. failure inducing input difference

faulty output
Related work

Model-based debugging

```c
void f( int x )
{
    int z, y1, y2;

    1. z = x+1;
    2. y1 = z*2;
    3. y2 = z+2;
    4. printf( "y1=%d, y2=%2\n", y1, y2 );
}
```
Related work

Model-based debugging

```c
void f( int x )
{
    int z, y1, y2;

    z = x+1;
    y1 = z*2;
    y2 = z+2;
    printf( "y1=%d, y2=%2d\n", y1, y2 );
}
```

\[ h_1 \Rightarrow z_{OK} \]
\[ h_2 \Rightarrow (z_{OK} \Rightarrow y_{1OK}) \]
\[ h_3 \Rightarrow (z_{OK} \Rightarrow y_{2OK}) \]

\( h_i \): statement \( i \) contributes to the intended behavior of the program
Related work

Model-based debugging

• Wotawa02: model-based debugging using dependency-based models is equivalent to slicing
• Survey in Mayer & Stumptner DX-07
Conclusion

• More than any other software fault diagnosis method, SFL is cheap, practicable, and appears to work.

• How valuable is it in real development scenarios?

• We will let you know!