What’s abnormal?

• Suppose we determine common properties of all *passing* runs.

• Now we examine a run which *fails* the test.

• Any difference in properties *correlates with* failure – and is likely to hint at failure causes
Detecting Anomalies

Properties

Differences correlate with failure

Properties
Properties

Data properties that hold in all runs:

- “At f(), x is odd”
- “0 ≤ x ≤ 10 during the run”

Code properties that hold in all runs:

- “f() is always executed”
- “After open(), we eventually have close()”
Techniques

<table>
<thead>
<tr>
<th>Dynamic Invariants</th>
<th>Value Ranges</th>
<th>Sampled Values</th>
</tr>
</thead>
</table>

## Techniques

<table>
<thead>
<tr>
<th>Dynamic Invariants</th>
<th>Value Ranges</th>
<th>Sampled Values</th>
</tr>
</thead>
</table>


Dynamic Invariants

Invariant
At $f()$, $x$ is odd

Property
At $f()$, $x = 2$
Daikon

- Determines *invariants* from program runs
- Written by Michael Ernst et al. (1998–)
- C++, Java, Lisp, and other languages
- analyzed up to 13,000 lines of code
public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}

• Run with 100 randomly generated arrays of length 7–13
Daikon

Postcondition
\[ b[] = \text{orig}(b[]) \]
return \( \text{sum}(b) \)

get trace

filter invariants

report results

Invariant

Trace
Getting the Trace

• Records all variable values at all function entries and exits

• Uses VALGRIND to create the trace
Filtering Invariants

• Daikon has a library of *invariant patterns* over variables and constants

• Only matching patterns are preserved
Method Specifications

**using primitive data**

<table>
<thead>
<tr>
<th>$x = 6$</th>
<th>$x \in {2, 5, -30}$</th>
<th>$x &lt; y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = 5x + 10$</td>
<td>$z = 4x + 12y + 3$</td>
<td>$z = fn(x, y)$</td>
</tr>
</tbody>
</table>

**using composite data**

<table>
<thead>
<tr>
<th>A subseq B</th>
<th>$x \in A$</th>
<th>sorted$(A)$</th>
</tr>
</thead>
</table>

checked at method entry + exit
**Object Invariants**

<table>
<thead>
<tr>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string.content[string.length] = ‘\0’</code></td>
</tr>
<tr>
<td><code>node.left.value ≤ node.right.value</code></td>
</tr>
<tr>
<td><code>this.next.last = this</code></td>
</tr>
</tbody>
</table>

Checked at entry + exit of public methods
Matching Invariants

public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}
### Matching Invariants

<table>
<thead>
<tr>
<th>==</th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>✘</td>
<td></td>
<td></td>
<td>✘</td>
<td></td>
<td>✘</td>
</tr>
<tr>
<td>size(b[])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum(b[])</td>
<td>✘</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig(n)</td>
<td>✘</td>
<td></td>
<td></td>
<td></td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>✘</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✘</td>
</tr>
</tbody>
</table>

**Pattern**

```
A == B
```

**Variables**

```
s  size(b[])  sum(b[])  orig(n)  n
return ...
```
# Matching Invariants

Matching Invariants

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>A == B</td>
<td>s size(b[]) sum(b[]) orig(n) ret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>==</th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>n</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>size(b[])</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>sum(b[])</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>orig(n)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ret</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

run 2
### Matching Invariants

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>n</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>size(b[])</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td>✘</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum(b[])</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig(n)</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td></td>
<td></td>
<td>✘</td>
</tr>
</tbody>
</table>

**Pattern**

```
A == B
```

**Variables**

```
s size(b[]) sum(b[]) n orig(n) return ...
```
Matching Invariants

<table>
<thead>
<tr>
<th>==</th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>×</td>
<td>×</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>n</td>
<td>✘</td>
<td>×</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>size(b[])</td>
<td>✘</td>
<td>×</td>
<td>✘</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>sum(b[])</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>orig(n)</td>
<td>✘</td>
<td>✘</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>ret</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

- $s == \text{sum}(b[])$
- $s == \text{ret}$
- $n == \text{size}(b[])$
- $\text{ret} == \text{sum}(b[])$
Matching Invariants

```java
public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}
```

- `s == sum(b[])`
- `s == ret`
- `n == size(b[])`
- `ret == sum(b[])`
Enhancing Relevance

- Handle polymorphic variables
- Check for derived values
- Eliminate redundant invariants
- Set statistical threshold for relevance
- Verify correctness with static analysis
As long as some property can be observed, it can be added as a pattern

Pattern vocabulary determines the invariants that can be found ("sum()", etc.)

Checking all patterns (and combinations!) is expensive

Trivial invariants must be eliminated
# Techniques

<table>
<thead>
<tr>
<th>Dynamic Invariants</th>
<th>Value Ranges</th>
<th>Sampled Values</th>
</tr>
</thead>
</table>

23
Dynamic Invariants

Can we check this on the fly?

Invariant
At f(), x is odd

Property
At f(), x = 2
Diduce

- Determines invariants and violations
- Written by Sudheendra Hangal and Monica Lam (2001)
- Java bytecode
- analyzed > 30,000 lines of code
Diduce

Training mode

Invariant

Property

Checking mode
Training Mode

- Start with empty set of invariants
- Adjust invariants according to values found during run
Invariants in Diduce

For each variable, Diduce has a pair \((V, M)\)

- \(V = \text{initial value of variable}\)
- \(M = \text{range of values}: \text{i-th bit of } M \text{ is cleared if value change in i-th bit was observed}\)

- With each assignment of a new value \(W\), 
  \(M\) is updated to 
  \[M := M \land \neg (W \oplus V)\]

- *Differences are stored in same format*
## Training Example

<table>
<thead>
<tr>
<th>Code</th>
<th>i</th>
<th>Values</th>
<th>Differences</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = 10</td>
<td>1010</td>
<td>1010</td>
<td>1111</td>
<td>i = 10</td>
</tr>
<tr>
<td>i += 1</td>
<td>1011</td>
<td>1010</td>
<td>1110</td>
<td>10 ≤ i ≤ 11 ∧</td>
</tr>
<tr>
<td>i += 1</td>
<td>1100</td>
<td>1010</td>
<td>1000</td>
<td>8 ≤ i ≤ 15 ∧</td>
</tr>
<tr>
<td>i += 1</td>
<td>1101</td>
<td>1010</td>
<td>1000</td>
<td>8 ≤ i ≤ 15 ∧</td>
</tr>
<tr>
<td>i += 2</td>
<td>1111</td>
<td>1010</td>
<td>1000</td>
<td>8 ≤ i ≤ 15 ∧</td>
</tr>
</tbody>
</table>

During checking, clearing an M-bit is an anomaly.
Diduce vs. Daikon

- Less space and time requirements
- Invariants are computed on the fly
- Smaller set of invariants
- Less precise invariants
Techniques

<table>
<thead>
<tr>
<th>Dynamic Invariants</th>
<th>Value Ranges</th>
<th>Sampled Values</th>
</tr>
</thead>
</table>
Detecting Anomalies

How do we collect data in the field?

Properties

Differences correlate with failure
Liblit’s Sampling

• We want properties of runs in the field
• Collecting all this data is too expensive
• Would a sample suffice?
• Sampling experiment by Liblit et al. (2003)
Return Values

• Hypothesis: *function return values* correlate with failure or success

• Classified into positive / zero / negative
CCRYPT fails

- CCRYPT is an interactive encryption tool
- When CCRYPT asks user for information before overwriting a file, and user responds with EOF, CCRYPT crashes
- 3,000 random runs
- Of 1,170 predicates, only `file_exists() > 0` and `xreadline() == 0` correlate with failure
Liblit’s Sampling

- Can we apply this technique to remote runs, too?
- 1 out of 1000 return values was sampled
- Performance loss <4%
After 3,000 runs, only five predicates are left that correlate with failure.
Web Services

- Sampling is first choice for web services
- Have 1 out of 100 users run an instrumented version of the web service
- Correlate instrumentation data with failure
- After sufficient number of runs, we can automatically identify the anomaly
## Techniques

<table>
<thead>
<tr>
<th>Dynamic Invariants</th>
<th>Value Ranges</th>
<th>Sampled Values</th>
</tr>
</thead>
</table>


Anomalies and Causes

• An anomaly is not a cause, but a correlation
• Although correlation ≠ causation, anomalies can be excellent hints
• Future belongs to those who exploit
  • Correlations in multiple runs
  • Causation in experiments
Locating Defects

- NN (Renieris + Reiss, ASE 2003)
- SD (Liblit et al., PLDI 2005)
- CT (Cleve + Zeller, ICSE 2005)
- SOBER (Liu et al, ESEC 2005)

Results obtained from Siemens test suite; can not be generalized.

- 5,542 runs
- 2 runs

Source code to examine
Concepts

- Comparing data abstractions shows anomalies correlated with failure
- Variety of abstractions and implementations
- Anomalies can be excellent hints
- Future: Integration of anomalies + causes