Agenda

- Motivation
- Fault Localization
- Evaluation
- Future Work
1 MOTIVATION
1 Motivation

- verifier determine whether an error location is reachable
- the verifier produces a counterexample
- the root cause of the error is still not obvious
- fault localization reduces the locations to look at
1 Motivation

1 int test(int nondet){
2     int x = nondet; // failing input
3     int result = 1;
4     for (int i = 2; i <= x/2 + 1; i++)
5         if (x % i == 0) result = 0;
6     if(result == 0 && isPrime(x))
9         goto ERROR;
8     EXIT: return 0;
9     ERROR: return 1;
10 }

■ counterexample contains every edge
■ fixes are possible on lines 4 and 5 (marked red)
■ fault localization identifies these positions
2 ALGORITHMS
2 Algorithms **Overview**

- **precondition** $\Psi$ (variable assignment)
- **trace** $\pi$ (all transitions)
- **post-condition** $\Phi$ (violated assertion)

**fault localization algorithms**
- SingleUnsatCore
- MAX-SAT
- Error Invariants

**visual report**
counterexample produces a list of CFAEdges representing the path to a reachable error location

- the precondition $\Psi$ represents the initial variable assignment
- the post-condition $\Phi$ equals the negation of the Boolean representation of the last assume edge
- the trace $\pi$ contains all remaining edges
- the trace formula $\text{TF}(\Psi, \pi, \Phi)$ equals the conjunct of all three parts
2 Algorithms  Trace Formula

- **precondition** $\Psi$ (variable assignment)
- **trace** $\pi$ (all transitions)
- **post-condition** $\Phi$ (violated assertion)

**trace formula**

$$\text{TF}(\psi, \pi, \phi) = \psi \land \bigwedge_{i=0}^{n} (\pi[i]) \land \phi$$

**selector formula**

$$\text{TF}_{\Lambda}(\psi, \pi, \phi) = \psi \land \bigwedge_{i=0}^{n} (\lambda_i \Rightarrow \pi[i]) \land \phi$$
2 Algorithms MAX-SAT

- finds every possible MIN-UNSAT core
- guarantees to mark a line where a fix is possible
- however, the fix might not be suitable

- MIN-UNSAT core: minimal and unsatisfiable set of clauses
- in our case: a minimal subset $S$ of selectors such that:

$$TF_A \land \bigwedge_{\lambda \in S} \lambda$$

is unsatisfiable

Based on "Error Localization using Maximum Satisfiability" by Manu Jose and Rupak Majumdar
### 2 Algorithms

**Example MAX-SAT**

```c
int test(int nondet) {
    int x = nondet;
    if (x > 0)
        if (x < 5)
            if (x > 1)
                if (x != 6 && x != -1)
                    goto ERROR;
    EXIT: return 0;
    ERROR: return 1;
}
```

**$\psi$:** $(\text{nondet} = 2) \land$

**$\pi$:** $(\text{S0} \Rightarrow x = \text{nondet}) \land$

$\quad (\text{S1} \Rightarrow x > 0) \land$

$\quad (\text{S2} \Rightarrow x < 5) \land$

$\quad (\text{S3} \Rightarrow x > 1) \land$

**$\phi$:** $x = 6 \lor x = -1$
2 Algorithms  Example MAX-SAT

\[ \Psi: (\text{non} \text{det} = 2) \land \]
\[ \pi: (S_0 \Rightarrow x = \text{non} \text{det}) \land \]
\[ (S_1 \Rightarrow x > 0) \land \]
\[ (S_2 \Rightarrow x < 5) \land \]
\[ (S_3 \Rightarrow x > 1) \land \]
\[ \Phi: x = 6 \lor x = -1 \]

\[ H = \{ \} \]
\[ S = \{S_0, S_1, S_2, S_3\} \]

**Step 0:** create a copy of all selectors (S)

**Step 1:** already tested each selector once?

**Step 2:** remove a selector \( \lambda \) from S

**Step 3:** check if S is a super- or subset of any set in H

**Step 4:** check if \( TF_\Lambda \land \bigwedge_{\lambda \in S} \lambda \) is UNSAT
2 Algorithms

Example MAX-SAT

\[ \Psi: (\text{nondet} = 2) \land \]
\[ \pi: (S0 \Rightarrow x = \text{nondet}) \land \]
\[ (S1 \Rightarrow x > 0) \land \]
\[ (S2 \Rightarrow x < 5) \land \]
\[ (S3 \Rightarrow x > 1) \land \]
\[ \Phi: x = 6 \lor x = -1 \]

\[ H = \{\{S0\}, \{S1, S2\}, \{S2, S3\}, \{S1, S2, S3\}\} \]

\[ S = \{S0, S1, S2, S3\} \]

Step 0: create a copy of all selectors (S)
Step 1: already tested each selector once?
Step 2: remove a selector \( \lambda \) from S
Step 3: check if S is a super- or subset of any set in H
Step 4: check if \( TF_\Lambda \land \bigwedge_{\lambda \in S} \lambda \) is UNSAT
2 Algorithms Example MAX-SAT

```c
int test(int nondet) {
    int x = nondet; // failing input
    int result = 1;
    for (int i = 2; i <= x/2 + 1; i++)
        if (x % i == 0) result = 0;
    if (result == 0 && isPrime(x))
        goto ERROR;
    EXIT: return 0;
    ERROR: return 1;
}
```
2 Algorithms Error Invariants

- abstracts the counterexample
- concise explanation of the faulty behavior
- summarizes parts of the program with inductive interpolants (error invariants)

Error invariants (ERRINV)

- Craig interpolant $I$:
  - $A \Rightarrow I$ is valid
  - $I \land B$ is UNSAT
  - free variables in $I$ are free in $A$ and $B$

- we can split the trace formula on all positions $0 \leq i < n$ to obtain $A$ and $B$

Based on "Error Invariants" by Evrin Ermis, Martin Schäf and Thomas Wies
## 2 Algorithms

### Example ERRINV

<table>
<thead>
<tr>
<th></th>
<th>int test() {</th>
<th>Interpolants:</th>
<th>Borders:</th>
<th>Trace:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int test() {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>int x = 5; //Ψ</td>
<td>x = 5</td>
<td>I₁: [2;2]</td>
<td>I₁: x = 5</td>
</tr>
<tr>
<td>3</td>
<td>x = x + 5;</td>
<td>x = 10</td>
<td>I₂: [3;3]</td>
<td>L₃: x = x + 5</td>
</tr>
<tr>
<td>4</td>
<td>x = x + 1;</td>
<td>x = 11</td>
<td>I₃: [4;4]</td>
<td>I₄: x = 10</td>
</tr>
<tr>
<td>5</td>
<td>x = x - 1;</td>
<td>x = 10</td>
<td>I₄: [3;5]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>if (x == 10) //Φ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>goto ERROR;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>EXIT: return 0;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ERROR: return 1;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2 Algorithms

Example ERRINV

```c
int test(int nondet) {
    int x = nondet; // failing input
    int result = 1;
    for (int i = 2; i <= x/2 + 1; i++)
        if (x % i == 0) result = 0;
    if (result == 0 && isPrime(x))
        goto ERROR;
    EXIT: return 0;
    ERROR: return 1;
}
```

Abstract Trace:
I₁: nondet = 2
L₄: i = 2
I₂: x = 2
L₅: result = 0
I₃: result = 0 && isPrime(x)
2 Algorithms

Visual Report

- All algorithms output a set of faults.
- The data structure for fault localization uses this set to create the visual report.
- Every fault maintains a list of additional information.
- All found faults are ranked.
- The report displays the information in an interactive HTML page.
3 EVALUATION
3 Evaluation

Survey: Setting

- 18 participants (students, scientists, senior and junior developers)
- participants fixed bugs in 4 programs
- fault localization (FL) report with ERRINV
- evaluation of correctness and needed time
- evaluation of the grading of the features and the feedback

<table>
<thead>
<tr>
<th>Group</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no FL</td>
<td>no FL</td>
<td>FL</td>
<td>FL</td>
</tr>
<tr>
<td>2</td>
<td>FL</td>
<td>FL</td>
<td>no FL</td>
<td>no FL</td>
</tr>
</tbody>
</table>
3 Evaluation \textit{Survey: Effectiveness}

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{task1}
\caption{Task 1}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{task2}
\caption{Task 2}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{task3}
\caption{Task 3}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{task4}
\caption{Task 4}
\end{subfigure}
\end{figure}
3 Evaluation Survey: Feedback and Grading

- estimated benefit: \( \varnothing 6.4/10 \)
- clearness of the report: \( \varnothing 7.9/10 \)
- most liked features: current values, line numbers
- critique: redundant information, marking of unnecessary lines, unreadable formulas
- summary:
  - the chosen visual representation fits the needs of the users and is intuitive
  - fault localization improves the needed time and the correctness of fixes
  - the final organization of the gathered data can be improved
3 Evaluation  Critical Reflection

Problems of both techniques:
■ both tend to mark iteration variables
■ both tend to mark test variables
■ both tend to mark function call edges
■ cannot identify missing method calls
■ bad runtime

Solutions:
■ useful options (ban, ignore, filter uniqueSelectors)
■ memoization

```java
1  //Check if a and b are even
2  int test(a = 3, b = 2){
3      int expResult = 0;
4      int check = isEven(b);
5      if(expResult != check)
6          goto ERROR;
7      EXIT: return 0;
8      ERROR: return 1;
9  }```

3 Evaluation

Benchmarks
4 FUTURE WORK
4 Future Work

- implement flow-sensitive trace formula for ERRINV
  - assumes of if statements imply all statements of the if-block
  - allows to identify important and unimportant if-blocks

- implement auto repair mechanism for MAX-SAT
  - find off-by-one errors
  - increase or decrease the value of promising variables and restart the analysis

- adapt the report
  - improve visualization of formulas
  - remove redundant information
QUESTIONS?
APPENDIX
Evaluation

Survey: Normalized Effectiveness

![Graphs showing normalized effectiveness for different tasks with FL disabled and enabled, with time and correctness metrics.](image-url)
Evaluation

Survey: Normalized Effectiveness

[Box plots comparing time for incorrect and correct responses with and without FL enabled]