Underspecified Harnesses and Interleaved Bugs

Seminar talk of Program Analysis

Yu-Wen Chen
Introduction

- **Motivation**
  - imprecise precondition
  - reduce false alarms

- **Goal**
  - filter out uninteresting bugs

- **How?**
  - providing an algorithm
Introduction

- Harness
  - precondition
- Underspecified Harness
  - causing false alarm
Introduction

- **Sequential execution**
  - $s_1 \ s_2 \ t_1 \ t_2$
  - $t_1 \ t_2 \ s_1 \ s_2$

- **Concurrent execution**
  - $s_1 \ t_1 \ s_2 \ t_2$
  - $t_1 \ s_1 \ t_2 \ s_2$

- **Interleaved bugs**
  - bugs appear in concurrent execution but not in sequential execution
To find interleaved bugs
  - problem: differential error checking
  - solution: an algorithm
  - tool: CBugs
Introduction

- Conjecture
  - Interesting and uninteresting bugs
  - bugs in sequential execution are usually…
    • cased by underspecified harness
    • non-interleaved bugs
  - interleaved bugs are more interesting
Agenda

- Introduction
- **Differential Error Checking**
- Interleaved Bugs
- Evaluation
- Summary
Differential Error Checking

- $\text{DiffError}(P_1, P_2)$
  - holds if there exists an input state $s$ s.t.
    - violates an assertion in $P_2$ and
    - does not violate an assertion in $P_1$
  - input-filter

Example

- $\text{DiffError}(P_s, P)$
  - $P$: concurrent program
  - $P_s$: sequential execution of $P$
Differential Error Checking

- **FindBug**
  - static analysis tool
  - returns
    - **NoBug**
    - **TRACE(t, i)**
      - t: execution trace
      - i: input

Underspecified Harnesses and Interleaved Bugs
Differential Error Checking

- Algorithm for solving ∩DIFFERROR(\(P_1, P_2\))

Require: Programs \(P_1\) and \(P_2\)

1: \hspace{1em} \textbf{loop}
2: \hspace{1em} \(r_1 := \text{FINDBUG}(P_2)\)
3: \hspace{1em} \textbf{if} \(r_1 = \text{NoBug}\) \textbf{then}
4: \hspace{2em} \textbf{return} \text{NoBug}
5: \hspace{1em} \textbf{end if}
6: \hspace{1em} \text{Let TRACE}(t, i) = r_1
7: \hspace{1em} \(r_2 := \text{FINDBUG}(P_1, i)\)
8: \hspace{1em} \textbf{if} \(r_2 := \text{NoBug}\) \textbf{then}
9: \hspace{2em} \textbf{return} \(r_1\)
10: \hspace{1em} \textbf{end if}
11: \hspace{1em} \text{Let TRACE}(t', i) = r_2
12: \hspace{1em} \(\varphi := \text{path\_cond}(\text{Determine}(t'), i)\)
13: \hspace{1em} \(P_2 := \text{assume} \lnot \varphi; P_2\)
14: \hspace{1em} \textbf{end loop}
Differential Error Checking

- \textit{Determinize}(t')
  - non-deterministic assignment
  - \( x ::= c \)

- \textit{path\_cond}
  - path condition

8: \hspace{1cm} \textbf{if} \ \( r_2 ::= \text{NoBug} \) \textbf{then}
9: \hspace{1cm} \textbf{return} \ \( r_1 \)
10: \hspace{1cm} \textbf{end if}
11: \hspace{1cm} \textbf{Let \ Trace}(t', i) = r_2
12: \hspace{1cm} \varphi ::= \text{path\_cond}(\text{Determinize}(t'), i)
13: \hspace{1cm} P_2 ::= \text{assume } \neg \varphi; P_2
14: \hspace{1cm} \textbf{end loop}
Differential Error Checking

Start

FindBug(P2)

NOBUG

Return NOBUG

FindBug(P1, i)

NOBUG

Return TRACE(t, i)

TRACE(t, i)

TRACE(t’, i)

Determinize

path_cond
Differential Error Checking

- Result of the algorithm

**NoBug**
- \( \text{DiffError}(P_1, P_2) \) does not hold

**Trace(\( t, i \))**
- \( \text{DiffError}(P_1, P_2) \) holds
Agenda

- Introduction
- Differential Error Checking
- **Interleaved Bugs**
- Evaluation
- Summary
Interleaved Bugs

- $F_p$: input-output relation of program $P$
  - $(s, t) \in F_p$
- $F_{Ps}$: subset of $F_p$
- Failed($t$) holds
  - failed state
- Reminder
  - $P$: concurrent program
  - $P_s$: sequential execution of $P$
Definition of interleaved bug

- A program $P$ has an interleaved bug if there is a pair of states $(s, t) \in F_P$ such that $\text{Failed}(t)$ holds and for all $(s, t') \in F_{Ps}$, $\text{Failed}(t')$ does not hold.
Interleaved Bugs

- Two concurrent programs $P$ and $Q$
- To prove the absence of interleaved bugs
  - underapproximations of $F_{Ps}$

$F_Q \subseteq F_{Ps}$ and $\text{DIFFERROR}(Q, P)$ does not hold

$P$ has no interleaved bugs
Interleaved Bugs

- Two concurrent programs $P$ and $Q$
- To prove the presence of interleaved bugs
  - overapproximations of $F_{Ps}$

$F_{Ps} \subseteq F_{Q}$ and $\text{DiffError}(Q, P)$ holds

• $P$ has an interleaved bug
Agenda

- Introduction
- Differential Error Checking
- Interleaved Bugs
- **Evaluation**
- Summary
Evaluation

- **CBugs**
  - algorithm implementation

- **Windows device drivers**
  - no precise harness
  - bugs are known
  - suffix “_bug”
## Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Buggy?</th>
<th>CBugs</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Asserts)</td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>daytona</td>
<td>485(10)</td>
<td>No</td>
<td>1 0 0</td>
<td>122</td>
</tr>
<tr>
<td>daytona_bug1</td>
<td>484(10)</td>
<td>Seq.</td>
<td>2 0 1</td>
<td>136</td>
</tr>
<tr>
<td>daytona_bug2</td>
<td>485(10)</td>
<td>Int.</td>
<td>1 0 0</td>
<td>110</td>
</tr>
<tr>
<td>daytona_bug3</td>
<td>484(10)</td>
<td>Seq.</td>
<td>2 0 1</td>
<td>161</td>
</tr>
<tr>
<td>daytona_bug5</td>
<td>485(10)</td>
<td>Int.</td>
<td>1 0 0</td>
<td>126</td>
</tr>
<tr>
<td>ndisprot_read</td>
<td>588(35)</td>
<td>No</td>
<td>10 0 0</td>
<td>2894</td>
</tr>
<tr>
<td>mouclass_bug1</td>
<td>582(19)</td>
<td>Seq.</td>
<td>10 0 1</td>
<td>905</td>
</tr>
<tr>
<td>mouclass_bug2</td>
<td>582(19)</td>
<td>Int.</td>
<td>10 0 0</td>
<td>944</td>
</tr>
</tbody>
</table>
Agenda

- Introduction
- Differential Error Checking
- Interleaved Bugs
- Evaluation

Summary
Summary

- Highlight the problem of false alarm
  - focus on interleaved bugs
  - find bugs with underspecified harness
- Use DiffError to find interleaved bug
- CBugs is able to remove false alarms due to underspecified harness
Saurabh Joshi, Shuvendu Lahiri, and Akash Lal. Underspecified Harnesses and Interleaved Bugs.
Thank you.