Underspecified Harnesses and Interleaved Bugs

Seminar talk of Program Analysis

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Yu-Wen Chen

Introduction

UNI FREIBURG

- Motivation
  - imprecise precondition
  - reduce false alarms
- Goal
  - filter out uninteresting bugs
- How?
  - providing an algorithm

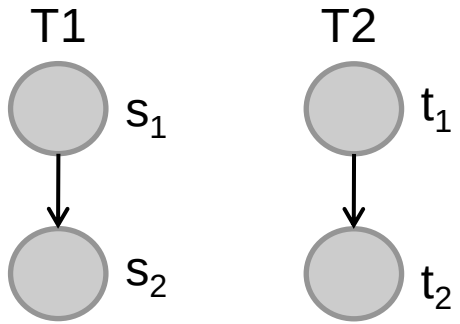
Introduction

UNI FREIBURG

- 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



Introduction

UNI FREIBURG

- To find interleaved bugs
  - problem: differential error checking
  - solution: an algorithm
  - tool: CBugs

Introduction

UNI FREIBURG

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

Agenda

UNI FREIBURG

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

Differential Error Checking

UNI FREIBURG

- 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
  - DIFFERROR(P_s, P)
    - P: concurrent program
    - P_s: sequential execution of P

Differential Error Checking

UNI FREIBURG

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

Differential Error Checking

UNI FREIBURG

- Algorithm for solving DIFFERROR(P_1, P_2)

Require: Programs P_1 and P_2

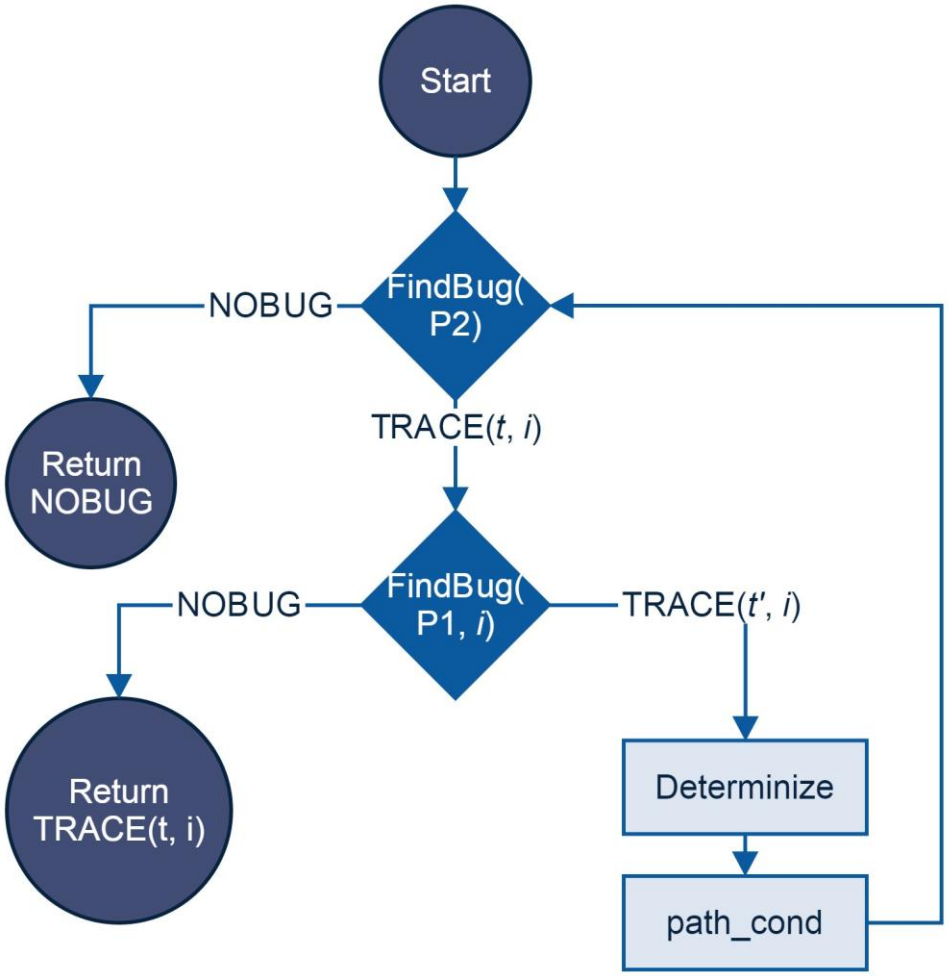
1: **loop**
2: r_1 := FINDBUG (P_2)
3: **if** r_1 = NOBUG **then**
4: **return** NOBUG
5: **end if**
6: Let TRACE(t, i) = r_1
7: r_2 := FINDBUG (P_1, i)
8: **if** r_2 := NOBUG **then**
9: **return** r_1
10: **end if**
11: Let TRACE (t', i) = r_2
12: φ := *path_cond*(*Determinize*(t'), i)
13: P_2 := assume $\neg\varphi$; P_2
14: **end loop**

Differential Error Checking

- *Determinize*(*t'*)
  - non-deterministic assignment
  - x ::= *c*
- *path_cond*
  - path condition

```
8:    if r2 := NoBug then
9:        return r1
10:   end if
11:   Let Trace (t', i) = r2
12:   φ := path_cond(Determinize(t'), i)
13:   P2 := assume ¬φ; P2
14: end loop
```

Differential Error Checking



Differential Error Checking

UNI FREIBURG

- Result of the algorithm

NoBug	• DiffError(P_1, P_2) does not hold
Trace(t, i)	• DiffError(P_1, P_2) holds

Agenda

UNI FREIBURG

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

Interleaved Bugs

UNI FREIBURG

- 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

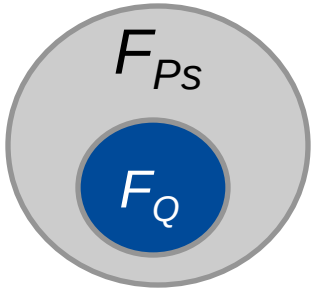
Interleaved Bugs

UNI FREIBURG

- 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 *Failed*(*t*) holds and for all $(s, t') \in F_{Ps}$, *Failed*(*t* ') does not hold.

Interleaved Bugs

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

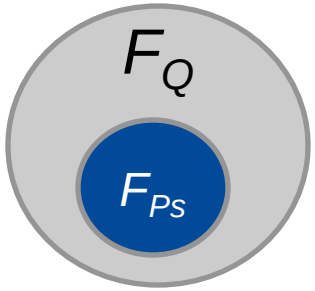


$F_Q \subseteq F_{Ps}$ and DIFFERROR(*Q*, *P*) does not hold

- *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 DIFFERROR(Q, P) holds

- *P* has an interleaved bug

Agenda

UNI FREIBURG

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

Evaluation

UNI FREIBURG

- CBugs
  - algorithm implementation
- Windows device drivers
  - no precise harness
  - bugs are known
  - suffix “_bug”

Evaluation

Name	LOC	Buggy?	CBugs				Static Analysis		
	(Asserts)		#LF	#FP	#FN	Time	#FP	#FN	Time
daytona	485(10)	No	1	0	0	122	1	0	17
daytona_bug1	484(10)	Seq.	2	0	1	136	1	0	21
daytona_bug2	485(10)	Int.	1	0	0	110	1	0	34
daytona_bug3	484(10)	Seq.	2	0	1	161	3	0	41
daytona_bug5	485(10)	Int.	1	0	0	126	1	0	29
ndisprot_read	588(35)	No	10	0	0	2894	3	0	38
mouclass_bug1	582(19)	Seq.	10	0	1	905	4	0	131
mouclass_bug2	582(19)	Int.	10	0	0	944	3	1	186

Agenda

UNI FREIBURG

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

Summary

UNI FREIBURG

- 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

Reference

UNI FREIBURG

- Saurabh Joshi, Shuvendu Lahiri, and Akash Lal. Underspecified Harnesses and Interleaved Bugs.

Q&A

UNI FREIBURG

Thank you.