Modular Verification of Interrupt-Driven Software

Chungha Sung | Markus Kusano | Chao Wang

University of Southern California | Virginia Tech
Interrupt-driven Software
Verification of Interrupts?

Without verification

With verification
Verification of Interrupts?
Prior Works

• Testing – Hard to explore all possible combinations
  
  ex) [Regehr ICES 2005]
  [Wang et al. ISSTA 2017]

• Bounded Model Checking - Cannot prove validity of assertions
  
  ex) [Kroening et al. DATE 2015]

Cannot provide proof!
Our Approach

• Abstract interpretation – good for obtaining proofs
  [Cousot & Cousot *POPL* 1977]

• Modular analysis – Only for thread behavior
  ex) [Miné *VMCAI* 2014]
  [Kusano & Wang *FSE* 2016/2017]
Behavior of Interrupts

Behavior of threads

Behavior of interrupts
Behavior of Interrupts

Under thread behavior

\[
T1() \{ \\
\quad a = 1; \\
\quad x = a; \\
\}\;
\]

\[
T2() \{ \\
\quad a = 2; \\
\}\;
\]

Under interrupt behavior (T1’s priority > T2’ priority)

\[
T1() \{ \\
\quad a = 1; \\
\quad x = a; \\
\}\;
\]

\[
T2() \{ \\
\quad a = 2; \\
\}\;
\]
Outline

Motivation

Contribution
(The first modular verification method for interrupt-driven software)

Experiments

Conclusion
Overview

Modular analysis + Data-flow feasibility based on interrupt behavior

Modular analysis for interrupts

More accurate!!
Overview

Interrupt-driven programs → Abstract Interpretation with inter-interrupt propagation → Invariants

Abstract Interpretation with inter-interrupt propagation

Invariants

Interrupt behavior modeling

interrupt-driven programs

Abstract Interpretation with inter-interrupt propagation

Invariants

Interrupt behavior modeling

Checking the feasibility of Dataflow between interrupts

Query

CFG

LLVM Front-end

Interrupt behavior modeling
Feasibility Checking

Abstract Interpretation with inter-interrupt propagation
Motivating Example 1

Irq_L() {
    x = 1;
};

Irq_H() {
    x = 0;
    assert(x == 0);
};
Motivating Example 1

Priority: L < H

Irq_L() {
    x = 1;
};

Irq_H() {
    x = 0;
    assert(x == 0);
};

Thread behavior: The assertion can be violated!
Motivating Example 1

Priority: L < H

\begin{align*}
\text{Irq\_L()} & \{ \\
& \quad x = 1; \\
& \}
\end{align*}

\begin{align*}
\text{Irq\_H()} & \{ \\
& \quad x = 0; \\
& \quad \text{assert}(x == 0); \\
& \}
\end{align*}

*Interrupt behavior: The assertion holds!*
Motivating Example 2

Irq_L() {
    x = 1;
};

Irq_H() {
    assert(x == 0);
};

Priority: L < H
Motivating Example 2

Priority: L < H

Thread behavior: The assertion can be violated!
Motivating Example 2

Priority: L < H

Irq_L() {
    x = 1;
};

Irq_H() {
    assert(x == 0);
};

Thread behavior: The assertion can be violated!

Interrupt behavior: The assertion can be violated as well!
Motivating Example 3

\begin{align*}
\text{Irq\_L}() & \{ \\
& \text{assert}(x == 0); \\
& \}; \\
\text{Irq\_H}() & \{ \\
& \text{if (\ldots)} \\
& \quad x = 1; \\
& \quad x = 0; \\
& \};
\end{align*}

Priority: $L < H$
Motivating Example 3

Priority: L < H

Thread behavior: The assertion can be violated!

```
Irq_L() {
    assert(x == 0);
}

Irq_H() {
    if (…)
        x = 1;
    x = 0;
}
```
Interrupt behavior: The assertion holds!
## Summary

<table>
<thead>
<tr>
<th>Example</th>
<th>Thread behavior (Existing)</th>
<th>Interrupt behavior (Our approach)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example1</td>
<td>Warning</td>
<td><strong>Proof</strong></td>
</tr>
<tr>
<td>Example2</td>
<td>Warning</td>
<td>Warning</td>
</tr>
<tr>
<td>Example3</td>
<td>Warning</td>
<td><strong>Proof</strong></td>
</tr>
</tbody>
</table>
Interrupt-driven programs

Abstract Interpretation with inter-interrupt propagation

Invariants

Interrupt behavior modeling

CFG

Datalog Facts

Datalog Rules

Feasibility Checking (Z3 fixed-point)

LLVM Front-end
Interrupt-driven programs

Abstract Interpretation with inter-interrupt propagation

Invariants

Interrupt behavior modeling

CFG

Datalog Facts

Datalog Rules

Feasibility Checking (Z3 fixed-point)

Query

LLVM Front-end
Program Analysis in Datalog

Interrupt-driven software $\rightarrow$ Datalog facts

Datalog rules

Data-flow Feasibility between interrupts

[Whaley & Lam, 2004]
[Livshits & Lam, 2005]
What is Datalog?

Declarative language for deductive databases
[Ullman 1989]

**Facts**
parent (bill, mary)
parent (mary, john)

**Rules**
ancestor (X, Y) ← parent (X, Y)
ancestor (X, Y) ← parent (X, Z), ancestor (Z, Y)

*New relationship: ancestor (bill, john)*
Datalog Translation

\[ \text{Irq\_L()} \{ \]
\[ \quad x = 1; \]
\[ \}; \]

\[ \text{Irq\_H()} \{ \]
\[ \quad x = 0; \]
\[ \quad \text{assert}(x == 0); \]
\[ \}; \]

\[ \text{NoPreempt (s1, s2) } \leftarrow \text{Pri}(s1, p1) \& \text{Pri}(s2, p2) \& (p2 \geq p1) \]

\[ \text{NoPreempt (x=1, x==0) } \leftarrow \text{Pri}(x=1, L) \& \text{Pri}(x==0, H) \& (H \geq L) \]
Datalog Translation

\[ \text{Irq}_\text{L}() \{ \]
\[ \quad x = 1; \]
\[ \}
\]

\[ \text{Irq}_\text{H}() \{ \]
\[ \quad x = 0; \]
\[ \quad \text{assert}(x == 0); \]
\[ \}
\]

CoveredLoad(l) <- Load(l, v) & Store(s, v) & Dom(s, l)

CoveredLoad(x==0) <- Load(x==0) & Store(x=0) & Dom(x=0, x==0)
Datalog Translation

Irq_L() {
    x = 1;
};

Irq_H() {
    x = 0;
    assert(x == 0);
};

MustNotReadFrom(l, s) <-
CoveredLoad(l) & NoPreempt (s, l) for the same variable

MustNotReadFrom(x==0, x=1) <-
CoveredLoad(x==0) & NoPreempt (x=1, x==0) for x
Datalog Translation

\[
\text{Irq\_L}() \{ \\
\quad \text{assert}(x == 0); \\
\}
\]

\[
\text{Irq\_H}() \{ \\
\quad \text{if (…)} \\
\quad \quad \text{x} = 1; \\
\quad \text{x} = 0; \\
\}
\]

\[
\text{NoPreempt \ (s1, s2) <- Pri(s1, p1) \& Pri(s2, p2) \& (p2 \geq p1)}
\]

\[
\text{NoPreempt \ (x==0, x=1) <- Pri(x==0, L) \& Pri(x=1, H) \& (H \geq L)}
\]
Datalog Translation

Irq_L() {
    assert(x == 0);
};

Irq_H() {
    if (…) 
    InterceptedStore x = 1;
    x = 0;
};

InterceptedStore(s1) <- Store(s1, v) & Store(s2, v) & PostDom(s1, s2)

InterceptedStore(x=1) <- Store(x=1) & Store(x=0) & PostDom(x=0, x=1)
**Datalog Translation**

```
Irq_L() {
    assert(x == 0);
}

Irq_H() {
    if (...)
        x = 1;
    x = 0;
}
```

MustNotReadFrom(l, s) <-
InterceptedStore(s) & NoPreempt(l, s) for the same variable

MustNotReadFrom(x==0, x=1) <-
InterceptedStore(x=1) & NoPreempt(x==0, x=1) for x
Implementation

Interrupt-driven programs

Abstract Interpretation with inter-interrupt propagation

Invariants

Interrupt behavior modeling

CFG

Dataflow Facts

Dataflow Rules

Feasibility Checking (Z3 fixed-point)

LLVM Front-end
Feasibility Check

Abstract Interpretation with inter-interrupt propagation

MustNotReadFrom(L1, S1)
MustNotReadFrom(L3, S3)
# Experimental Results 1

## Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of Benchmarks</td>
<td>35</td>
</tr>
<tr>
<td>Total LOC</td>
<td>22,541 lines</td>
</tr>
<tr>
<td>Total number of pairs</td>
<td>5,116</td>
</tr>
<tr>
<td>Number of filtered pairs</td>
<td>3,560</td>
</tr>
<tr>
<td>Analysis time</td>
<td>64.21 s</td>
</tr>
</tbody>
</table>

69%
Comparison

- Bounded Model Checking for interrupts
  [Kroening et al. DATE 2015]

- Modular analysis for threads
  [Miné VMCAI 2014]
Experimental Results 2

Number of warnings & proofs w.r.t each method

- **BMC [DATE 15]**
  - Violations: 50
  - Proofs: 100

- **Modular [VMCAI 14]**
  - Violations: 0
  - Proofs: 250

- **IntAbs (Our method)**
  - Violations: 0
  - Proofs: 100

Legend:
- Blue bar: Violations
- Orange bar: Proofs
Experimental Results 2

Number of warnings & proofs w.r.t each method

BMC [DATE 15]
Modular [VMCAI 14]
IntAbs (Our method)
Experimental Results 2

Number of warnings & proofs w.r.t each method

- BMC [DATE 15]
- Modular [VMCAI 14]
- IntAbs (Our method)
Conclusions

• Proposed the first modular static analysis method for sound verification of interrupt-driven software

• Precisely identified infeasible data flows between interrupts with a declarative interrupt model

• Showed significant precision and performance improvements
Thank you!

https://github.com/chunghasung/intabs