Reasoning About Exceptions
Using Model Checking

Reid Simmons
David Garlan
Jeannette M. Wing

George Fairbanks, Gil Tolle, Balaji Sarapeshkar, Joe Jiang

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA
Outline

• Why Model Check Exceptions?

• Approach
  – IEL
  – MOPED
  – Translations
  – “Gotchas”

• Example
  – Vending Machine
Exceptions in Programming Languages

• **Raising Exceptions**
  – Throw of named/typed exceptions
  – Catch by nearest matching handler
  – Exceptions may form inheritance hierarchy

• **“Clean up” Construct**
  – Finally / Unwind-protect
  – Executed in both nominal and exceptional situations

• **Semantics**
  – **Termination** (C++, Java, Lisp)
  – **Resumption** (Mesa, Eiffel, TDL)
Why Hard?

- Non-Local Flow of Control
- Context of “Catch” Frames Determined Dynamically
- “Clean Up” Construct Adds Additional Pathways
- Hard to Reason About All Possible Execution Paths
- Impossible To Do So Purely Locally
Overview of Our Approach
Intermediate Exception Language (IEL)

• Captures Commonalities of Exception Handling Among Different Languages

• Focuses on Control-Flow Constructs Relevant to Reasoning About Exceptions
  – Catch/Throw/Finally
  – Iteration and Conditionals
  – Break and Return
  – Assignment
  – Procedures
  – Hierarchical Exceptions

• Minimal Data Representation
  – No value-returning functions
IEL Example: Resource Locking

```plaintext
var locked: int
exception e1
procedure main () {
    locked := 0
    while true {
        try {
            lock()
            randomException()
            unlock()
        } catch e1 {
            /* unlock() */
        }
    }
}

procedure lock () {
    if locked = 1 then
        error()
    if locked = 0 then
        locked := 1
}

procedure randomException () {
    if (p) throw e1
}
```
MOPED

• Model Checker for Push-Down Automata
  – Stefan Schwoon’s PhD Thesis
  – Symbolic Model Checker
  – Handles Procedures with Local Variables
  – Need to Explicitly Handle Frame Axioms

• Verifies LTL State Reachability Formulae
  – Currently cannot handle LTL formulae involving variables

• Minimal Data Representation
Translating IEL → MOPED

• **Create Local Translation Rules for Each IEL Construct**

  • Assign (x := y)
    
    $q <\text{procN}> \rightarrow q <\text{procN+1}> (x' = y \& \text{frameAllExcept}(x))$

  • Conditional (if p then stmt)
    
    $q <\text{procN}> \rightarrow q <\text{procN+1}> \text{“if true” } (p = 1 \& \text{frameAll})$
    $q <\text{procN}> \rightarrow q <\text{procN’}> \text{“if false” } (p = 0 \& \text{frameAll})$
    ...
    ...
    $q<\text{procN’}> ...$

  • Throw (throw ex)
    
    $q <\text{procN}> \rightarrow q <\text{procEx}> (ex' = 1 \& ...)$

  • Try/Catch (try {tryblock} catch ex {catchblock})
    
    $q <\text{procN}> \rightarrow q<\text{procTry}> \text{“jump past catch” } (\text{frameAll})$
    $q <\text{procEx}> \rightarrow q<\text{procCatch}> \text{“caught ex” } (ex = 1 \& ...)$
    $q <\text{procEx}> \rightarrow q<\text{procN}> \text{“didn’t catch ex” } (ex = 0 \& ...)$
    ...
    ...
    ...catchBlock translation...
    ...
    ...
    ...
    $q <\text{procN}> ...$
“Gotchas”

- Exception Hierarchy
- (Nested) Finally Blocks
- Break and Return Statements
Modeling Exception Hierarchy

exception e0
exception e1 extends e0
try {
    throw e1
} catch e0 {}
Modeling Finally Blocks

```java
try {
    if q then throw e1
} finally {
    try {
        if p then throw e2
    } catch e1 { x := x / 0 }
    x := x + 1
}
```

- Store State of Exceptions Upon Entering Finally Block
- Clear All Exceptions Before Executing Finally Block
- Restore Exception State at End of Finally Block, Unless a New Exception Was Raised
Modeling Nested Finally Blocks

- Store State of Exceptions Upon Entering Finally
  Plus Keep Track of “nesting level” and “exception level”
- Increment “nesting level” on Entering, and Decrement on Exit
- Clear All Exceptions Before Executing Finally Block
- Before Entering Finally Block with an Exception, Set “exception level” to “nesting level”
- Propagate Exception on Exit if “exception level” equals “nesting level”
Modeling Break and Return Statements

• “Break” and “Return” Interact in Interesting Ways with Exceptions and Finally Blocks

```java
while (x < 5) {
    try {
        if x = 3 then {
            try {
                throw e1
            } finally {
                x := x * 5
            }
        }
        try {
            throw e1
        } finally {
            break
        }
    } finally {
    }
    try {
        break
    } finally {
    }
    try {
        throw e1
    } finally {
    }
}
```

Approach: Treat “Break” and “Return” as Exceptions that do not Propagate!!
Example: Vending Machine

- Model of Vending Machine
  - Machine vends product, if it has that product in stock and sufficient $$ put in
  - Represented by Java program that has significant exceptions
  - From TSE article by Sinha & Harrold

- Verify that if Money is Put in the Machine, Eventually it will either Return Money or Vend Product
  - Static analysis is insufficient
  - Hand-coded IEL program from Java source; Automated IEL $\rightarrow$ MOPED translation
  - Found bug in the program (could keep on adding money indefinitely – no limit)
Ongoing and Future Work

- **Translating Java \( \rightarrow \)** IEL
  - Work in Progress
  - Parser and Most of Translator Exist
  - Main Difficulty in Dealing with Objects
    - Complex data structures, Inheritance, Dynamic memory allocation

- **Add Specification Language to IEL**

- **Translate Other Languages (C++, Lisp)**

- **Model Resumption Model of Exceptions**

- **Test on Software with Significant Exceptions**