Building a Symbolic Execution Engine for Haskell

William Hallahan
Anton Xue
Ruzica Piskac

Yale University
Functional languages - why?

● A different way of problem solving
  ○ Pattern matching, Higher Order Functions, Algebraic Data Types…

● Functional languages allow for easier equational reasoning
  ○ Objects are described by what they are rather than how they are constructed

● Strong static type system catches many errors at compile time
  ○ Many safeguards (e.g. null pointer checks) can be encoded as types
Extraction from source code

- Use Glasgow Haskell Compiler API to extract **Core Haskell** from source
  - GHC Pipeline: Source → AST → Core Haskell → …

<table>
<thead>
<tr>
<th>Feature</th>
<th>Full Language AST</th>
<th>Core Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traceable from Source</td>
<td>Yes</td>
<td><em>Somewhat</em></td>
</tr>
<tr>
<td>Concise Representation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Easily Manipulatable</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Further translate Core Haskell to custom language (G2 Core)
  - Close one-to-one representation of Core Haskell
  - Simplifies and discards extraneous data present in Core Haskell annotations
Execution

- **General functional language**: run reductions until a normal form is reached

- **Challenge**: symbolic execution requires symbolic variables
  - Augment Haskell lazy evaluation semantics with reduction rules for symbolic variables
  - **Semantics**: *Making a Fast Curry: Push/Enter vs Eval/Apply* ... [SPJ, SM 2004]

- **Approach**: treat symbolic execution as a bounded model-checking problem
  - Implement *reduce* function that applies augmented reduction execution rules one at a time
  - Apply reduction rules repeatedly to perform execution
    - Regular Haskell: apply until normal form is reached
    - Symbolic execution: apply until normal form is reached or we hit a counter limit
Constraint solving

- Most basic feature of symbolic execution is reachability testing
  - Can convert many problems such as assertion violation into state reachability problems

- Constraint solving: interface with SMT solver
  - Convert path constraints from execution to SMT-LIB2 files
    - SMT-LIB2 format supports all the constructs necessary
      - Equivalents for primitives such as Int, Float, Rational, etc
      - Can declare new algebraic data types
  - Run a SMT solver on these files