Institute of Operating Systems and Computer Networks
Reliable System Software

Deutsches Forschungszentrum für Künstliche Intelligenz
German Research Center for Artificial Intelligence

University of Bremen

Accurate and Extensible Symbolic Execution of Binary Code based on Formal ISA Semantics

Sören Tempel, Tobias Brandt, Christoph Lüth, Christian Dietrich, Rolf Drechsler

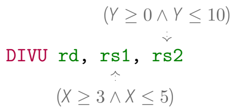
soeren.tempel@tu-braunschweig.de

Supported by the BMBF within projects Scale4Edge (no. 16ME0127), ECXL (no. 01IW22002), and VE-HEP (no. 16KIS1342).

Motivation

Symbolic Execution: SMT-based software verification and testing technique

- Enumerates execution through the SUT by reasoning about branches
- Requires custom software simulation to operate on symbolic input values
- Goal: Employing symbolic execution for testing low-level binary code



Challenges:

1. Requires a *correct* symbolic implementation of ISA instruction semantics

↪ Given the complexity of modern ISAs, a manual implementation is error-prone

2. The analysis must be easily *extendable* to support design space exploration

Formal Specifications of ISA Semantics

Idea: Describe ISA in a formal, machine-readable language

- Advantageous for various use-cases, e.g. code generation, fault injection, …
- Newer ISAs (e.g. RISC-V) provide an official formal specification
- All instructions are formally described in terms of several *language primitives*

Formal Specifications of ISA Semantics

Idea: Describe ISA in a formal, machine-readable language

- Advantageous for various use-cases, e.g. code generation, fault injection, …
- Newer ISAs (e.g. RISC-V) provide an official formal specification
- All instructions are formally described in terms of several *language primitives*

```
instrSemantics DIVU = do
  (rs1-val, rs2-val, rd) <- decodeAndReadRType

  runIfElse (rs2-val `EqInt` 0x00000000)
   do $ WriteRegister rd 0xffffffff
   do $ WriteRegister rd (rs1-val `UDiv` rs2-val)
```

The Case for Executable Formal Specifications

Problem: Formal specifications are not directly executable

- Requires maintaining compiler tooling for each use-case
- Executable formal specification have emerged recently

$\hookrightarrow$ We facilitate our own prior work on LIBRISCV

Approach: Implement symbolic execution as an ISA specification interpreter

$$\begin{tikzcd}\texttt{Decode} \arrow[r] & \texttt{ReadRegister} \arrow[r] & \texttt{RunIfElse} \arrow[r] & \texttt{UDiv} \arrow[r] & \cdots\end{tikzcd}$$

The Case for Executable Formal Specifications

Problem: Formal specifications are not directly executable

- Requires maintaining compiler tooling for each use-case
- Executable formal specification have emerged recently

↪ We facilitate our own prior work on LIBRISCV

Approach: Implement symbolic execution as an ISA specification interpreter

$$\begin{tikzcd}
\texttt{Decode} \arrow[r] & \texttt{ReadRegister} \arrow[r] \arrow[d, dotted] & \texttt{RunIfElse} \arrow[r] & \texttt{UDiv} \arrow[r] & \cdots \\
 & \text{SMT Representation} & & &
\end{tikzcd}$$

The Case for Executable Formal Specifications

Problem: Formal specifications are not directly executable

- Requires maintaining compiler tooling for each use-case
- Executable formal specification have emerged recently

↪ We facilitate our own prior work on LibRISCV

Approach: Implement symbolic execution as an ISA specification interpreter

$$\begin{tikzcd}
\texttt{Decode} \arrow[r] & \texttt{ReadRegister} \arrow[r] \arrow[d, dotted] & \texttt{RunIfElse} \arrow[r] \arrow[d, dotted] & \texttt{UDiv} \arrow[r] & \cdots \\
& \text{SMT Representation} & \text{SMT Representation} & &
\end{tikzcd}$$

The Case for Executable Formal Specifications

Problem: Formal specifications are not directly executable

- Requires maintaining compiler tooling for each use-case
- Executable formal specification have emerged recently

↪ We facilitate our own prior work on LIBRISCV

Approach: Implement symbolic execution as an ISA specification interpreter

$$\begin{tikzcd}
\texttt{Decode} \arrow[r] & \texttt{ReadRegister} \arrow[r] \arrow[d, dotted] & \texttt{RunIfElse} \arrow[r] \arrow[d, dotted] & \texttt{UDiv} \arrow[r] \arrow[d, dotted] & \cdots \\
& \text{SMT Representation} & \text{SMT Representation} & \text{SMT Representation} &
\end{tikzcd}$$

Contribution: BinSym

BinSym: Prototype implementation of the proposed approach for the RISC-V ISA

- Maps LibRISCV language primitives to SMT bit-vector theory with Z3
- Provides symbolic implementations of the register file, memory, …

$\Rightarrow$ Supports any custom instr. described in term of existing primitives

Source Code: `https://github.com/agra-uni-bremen/binsym`

- Provides an implementation of Dynamic Symbolic Execution (DSE)
- Written in roughly 1000 LOC in Haskell (excluding the formal model)!

Empirical Evaluation

- Empirical comparison with prior work on symbolic execution of RV32 binaries
- Research questions:

RQ1 Do we discover the same amount of paths as prior work?
RQ2 Does our work achieve competitive SE performance?

Empirical Evaluation

- Empirical comparison with prior work on symbolic execution of RV32 binaries
- Research questions:

RQ1 Do we discover the same amount of paths as prior work?

RQ2 Does our work achieve competitive SE performance?

Empirical Evaluation

- Empirical comparison with prior work on symbolic execution of RV32 binaries
- Research questions:

RQ1 Do we discover the same amount of paths as prior work?

RQ2 Does our work achieve competitive SE performance?

Benchmark	angr	BinSec	SymEx-VP	BinSym
`base64-encode`	**125**	6250	6250	6250
`bubble-sort`	720	720	720	720
`clif-parser`	11424	11424	11424	11424
`insertion-sort`	5040	5040	5040	5040
`uri-parser`	**8194**	8240	8240	8240

Table: Empirical comparison with prior work on synthetic benchmarks.

Empirical Evaluation

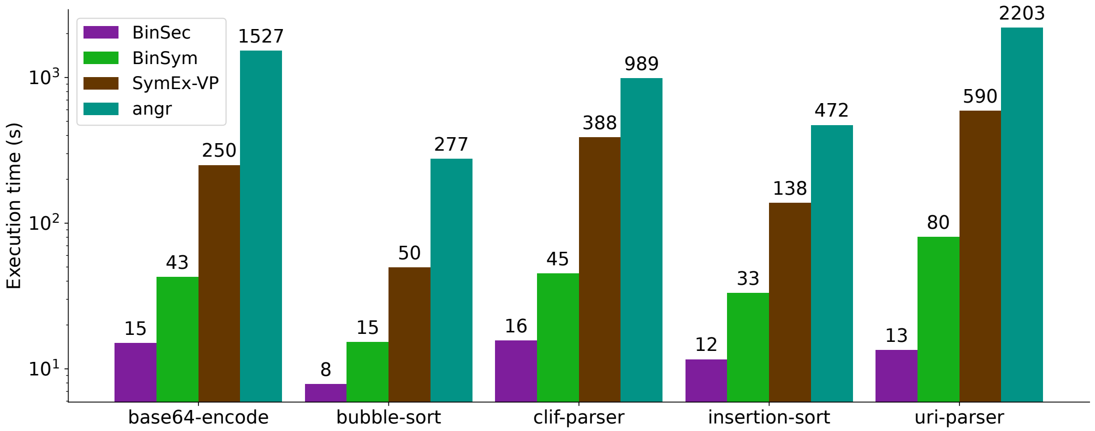
- Empirical comparison with prior work on symbolic execution of RV32 binaries
- Research questions:

RQ1 Do we discover the same amount of paths as prior work?
RQ2 Does our work achieve competitive SE performance?

Benchmark	angr	BinSec	SymEx-VP	BinSym
`base64-encode`	**125**	6250	6250	6250
`bubble-sort`	720	720	720	720
`clif-parser`	11424	11424	11424	11424
`insertion-sort`	5040	5040	5040	5040
`uri-parser`	**8194**	8240	8240	8240

Table: Empirical comparison with prior work on synthetic benchmarks.

Empirical Evaluation



Conclusion

Key Insights:

1. Executable formal semantics reduce manual effort and the margin for errors
2. Formal semantics ease extending the analysis to additional instructions
3. Execution speed with executable formal ISA specifications is competitive

Future Work: Formally prove the correctness of the symbolic semantics
$\hookrightarrow$ Using theorem-prover definitions provided by prior work

Institute of Operating Systems and Computer Networks
Reliable System Software

Deutsches Forschungszentrum für Künstliche Intelligenz
German Research Center for Artificial Intelligence

University of Bremen

Accurate and Extensible Symbolic Execution of Binary Code based on Formal ISA Semantics

Sören Tempel, Tobias Brandt, Christoph Lüth, Christian Dietrich, Rolf Drechsler
soeren.tempel@tu-braunschweig.de

Supported by the BMBF within projects Scale4Edge (no. 16ME0127), ECXL (no. 01IW22002), and VE-HEP (no. 16KIS1342).