Deutsches Forschungszentrum
für Künstliche Intelligenz
German Research Center for Artificial Intelligence

University of Bremen

Symbolic Execution of Binary Code based on Formal ISA Semantics

Sören Tempel, Tobias Brandt, Christoph Lüth, Rolf Drechsler

tempel@uni-bremen.de

April 15th, 2024

Supported by the Scale4Edge BMBF project under contract no. 16ME0127.

Motivation

Goal: Employ symbolic execution for testing embedded software

Challenges

- Tight integration between software and hardware components
- Software interacts on a low abstraction level with the hardware
- Example: Low-level instruction for configuring interrupt handlers

$\Rightarrow$ KLEE and tools build on it (e.g. FIE) do not support these

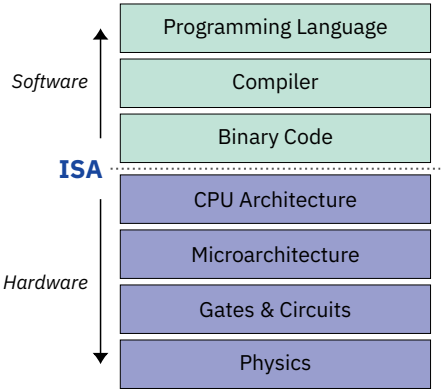
Background: Instruction Set Architectures (ISAs)

ISA: Central interface between hard- and software

- Specifies the instructions supported by the CPU
- Instructions are used in binaries $\rightarrow$ binary analysis

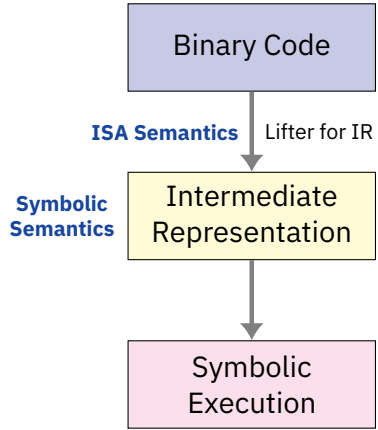
Observation: ISA specifications are very complex

- Thousands of pages of (mostly) natural language
- Problem: Need symbolic instruction semantics



Prior Work: IR-based Execution of Binary Code

Idea: Lift binaries to an intermediate representation (IR)
⇒ Symbolically execute the resulting, simpler IR



Limitations:

1. Lifting is complex and may induce inaccuracies
2. No symbolic execution of ISA-specific instructions

Formal Specifications of ISA Semantics

Idea: Describe ISA in a formal machine-readable language

- Eases supporting code generation and similar use-cases
- Newer ISAs (e.g. RISC-V) provide an official formal spec

Formal Specifications of ISA Semantics

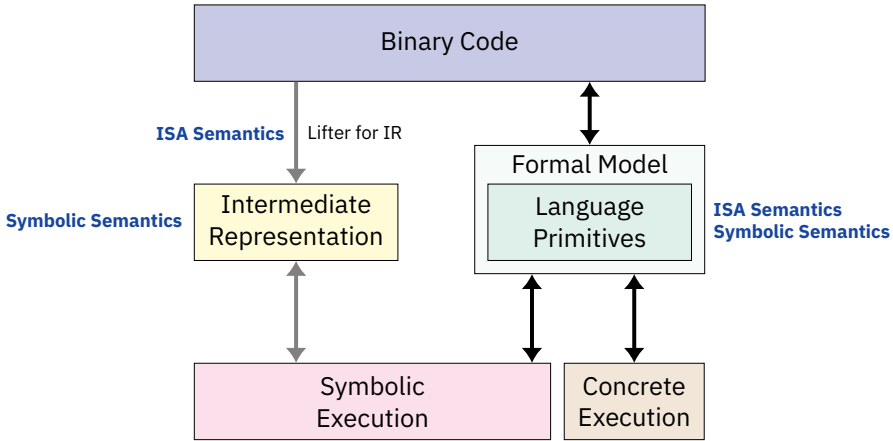
Idea: Describe ISA in a formal machine-readable language

- Eases supporting code generation and similar use-cases
- Newer ISAs (e.g. RISC-V) provide an official formal spec

Example

```
instrSemantics LB{..} = do
    base <- readRegister rs1
    byte <- loadByte (base `Add` imm)
    writeRegister rd (SExtByte $ FromImm byte)
```

Symbolic Execution using Formal ISA Semantics



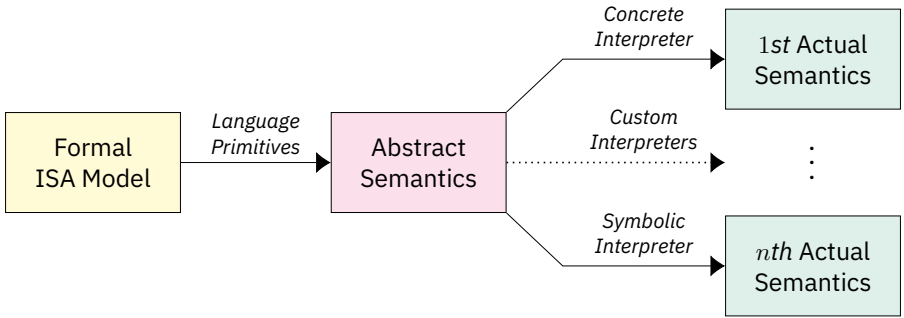
LibRISCV: A versatile formal model for RISC-V

Contribution: Formal model tailored to creation of ISA interpreters

$\Rightarrow$ Models instruction semantics independent of value representation

LibRISCV: A versatile formal model for RISC-V

Contribution: Formal model tailored to creation of ISA interpreters
⇒ Models instruction semantics independent of value representation



BinSym: Symbolic Semantics for LibRISCV

Contribution: Actual semantics for symbolic execution of RISC-V binaries

- Provides symbolic implementations of register file, memory, …
- Maps LibRISCV expression language to Z3 bit-vectors

BinSym: Symbolic Semantics for LibRISCV

Contribution: Actual semantics for symbolic execution of RISC-V binaries

- Provides symbolic implementations of register file, memory, ...
- Maps LibRISCV expression language to Z3 bit-vectors

Implementation: `https://github.com/agra-uni-bremen/BinSym`
⇒ Standard DSE implementation in Haskell

BinSym: Symbolic Semantics for LibRISCV

Contribution: Actual semantics for symbolic execution of RISC-V binaries

- Provides symbolic implementations of register file, memory, ...
- Maps LibRISCV expression language to Z3 bit-vectors

Properties

1. Operates on the binary-level without IR-lifting
2. More faithful to the (formal) ISA specification
3. Supports low-level interactions with the architectural state

Evaluation

Observation from prior work:

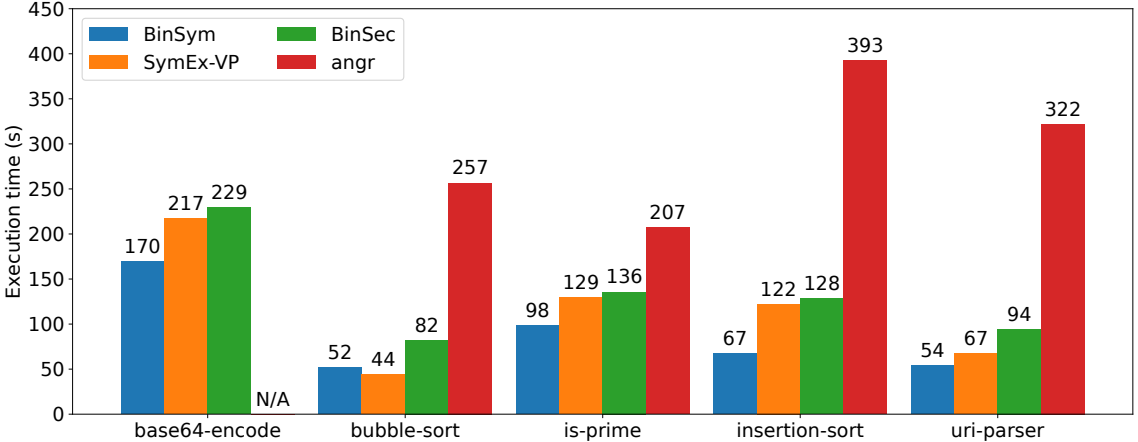
[Binary lifting] tends to inflate the code by a factor between 3 and 7.

— Poeplau et al., Systematic comparison of symbolic execution systems

RQ: What is the impact of code size inflation on *symbolic* execution speed?

$\Rightarrow$ Compare symbolic execution speed on synthetic benchmarks

Preliminary Results



Conclusion

Key Insights:

1. Formal ISA semantics are beneficial for symbolic execution
2. Ease supporting architecture-specific low-level instructions
3. Eliminating the IR abstraction might improve query complexity

More Information:

LibRISCV `https://doi.org/10.1007/978-3-031-38938-2_2`

BinSym `https://doi.org/10.48550/arXiv.2404.04132`

Source code and artifacts are referenced in the papers.