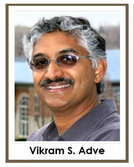
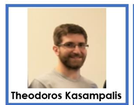

Vikram S. Adve


Grigore Rosu


Sandeep Dasgupta


Theodoros Kasampalis


Daejun Park

A Complete Formal Semantics of x86-64 User-Level ISA

github.com/kframework/X86-64-semantics

Accepted in PLDI'19

ISA Semantics is Useful

- Enables direct analysis of binary
- Enables automated binary formal reasoning
- Checking accuracy of ISA specification
- Post-silicon processor validation
- Security vulnerability on binary code
- Compiler verification (e.g. Compcert, CakeML)

Defining formal semantics of user-level x86-64

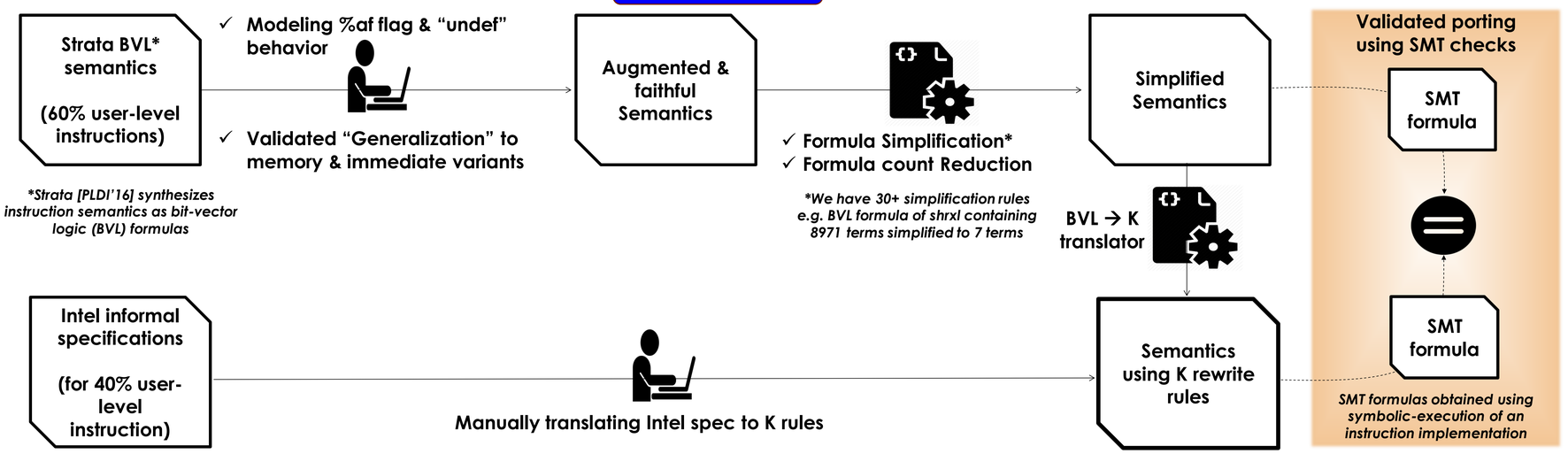
x86-64 Spec Challenges

- 996 unique mnemonics with 3736 variants
- Inconsistent behavior of instruction variants
- 3000+ pages of informal prose + pseudo-code
- Implementation defined behavior
- Ambiguous specification

Previous Work

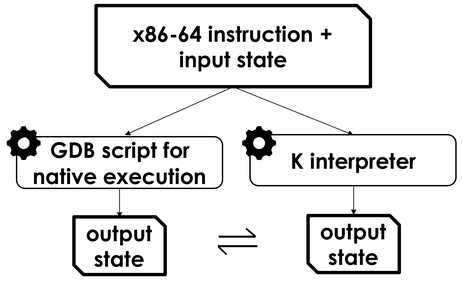
Project Name	Details
x86-64 semantics *by* Goel et al.	33% user-level support
x86-64 semantics *by* Heule et al. [PLDI'16]	60% user-level support
Projects hosting x86-32 spec	❖ Compcert ❖ Rocksalt [PLDI'12] ❖ Myreen et. al. [FMCAD'08]
Indirect semantics of x86-64	e.g. BAP, Angr etc.

Our Approach



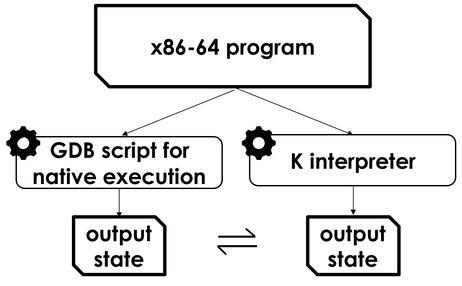
Validation

Instruction level testing



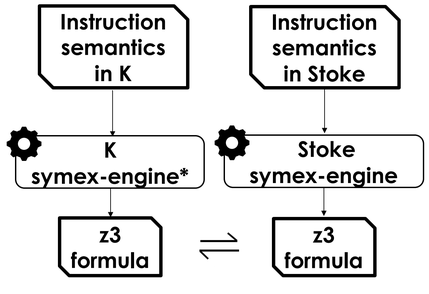
Each instruction's semantics is tested against hardware using 7000+ input states

Program level testing



All the programs in GCC-C torture are co-simulated against hardware

Comparing against Stoke



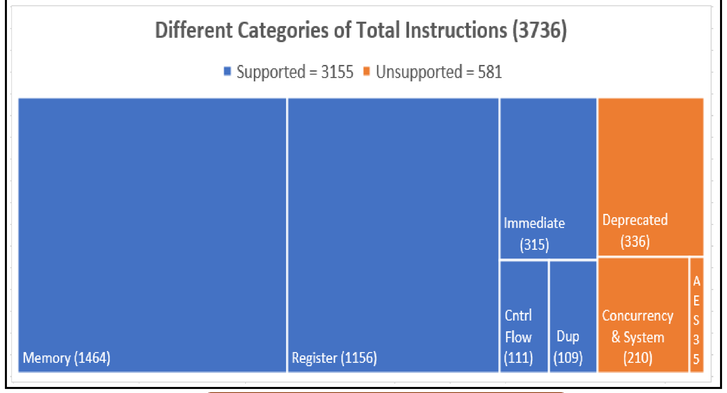
z3 summaries, obtained from K rules, of ~336 instructions are compared against that Stoke's

**symex-engine: symbolic execution engine*

Bugs Found

- Intel Manual (8+ instances)
- Strata's simplification rules (2 instances)
- Stoke's semantics (40+ instances)

Scope



Potential Applications

- Program verification
- Security vulnerability tracking
- Translation validation of compiler optimization

What's Next ?

Translation validation of binary decompilation