Automated Code Proofs on a Formal Model of the X86

Shilpi Goel and Warren A. Hunt, Jr.

The University of Texas at Austin

May 18, 2013

OUTLINE

Outline

Our Goals

1. Develop an accurate model of the x86 Instruction Set Architecture (ISA)

2. Develop automated procedures for reasoning about x86 machine code

WHY DO WE CARE?

- Analysis of high-level programs is not good enough.
- High-level programs are not always available.
- **Formal verification** of machine code!
  - Formal model of the x86 ISA
  - Reason about machine code on this model

Our Goals, revisited

1. Develop a **formal** and **executable** model of the x86 ISA
   - **Accurate** and **unsimplified** model
   - Specifications: Intel's Software Developer's Manuals
   - ~3000 pages of prose
   - **Co-simulations**
   - Need executability to do co-simulations

A **single model** for simulation and formal analysis enables us to **validate** it with co-simulations so that we can **trust** it for our proofs.

OUR GOALS, REVISITED

1. Develop an accurate, formal, and executable model of the x86 ISA
2. Develop **automated procedures** for reasoning about x86 machine code
   - **Functional correctness** of machine code
   - Minimize lemma construction

Outline

Outline

FORMALIZING X86 ISA IN ACL2

ACL2:

- *A Computational Logic for Applicative Common Lisp*
- Descendant of the Boyer-Moore theorem prover
- Programming language
- Mathematical logic
- Mechanical theorem prover

FORMALIZING X86 ISA IN ACL2

- Our x86 ISA model has been formalized using an *interpreter approach to operational semantics.*
- Semantics of a program is given by the effect it has on the state of the machine.
- State-transition function is characterized by a recursively defined interpreter.

X86 STATE

Component	Description
registers	general-purpose, segment, debug, control, model-specific registers
rip	instruction pointer
flg	64-bit flags register
mem	physical memory

Run Function

Recursively defined interpreter that specifies the x86 model

```
run (n, x86):

if n == 0:
   return (x86)
else
   if halt instruction encountered:
      return (x86)
   else
      run (n - 1, step (x86))
```

STEP FUNCTION

```
step (x86):

pc = rip (x86)

[prefixes, opcode, ... , imm] = Fetch-and-Decode (pc, x86)

case opcode:
     #x00 -> add-semantic-fn (prefixes, ... , imm, x86)

     ...      ...

     #xFF -> inc-semantic-fn (prefixes, ... , imm, x86)
```

INSTRUCTION SEMANTIC FUNCTIONS

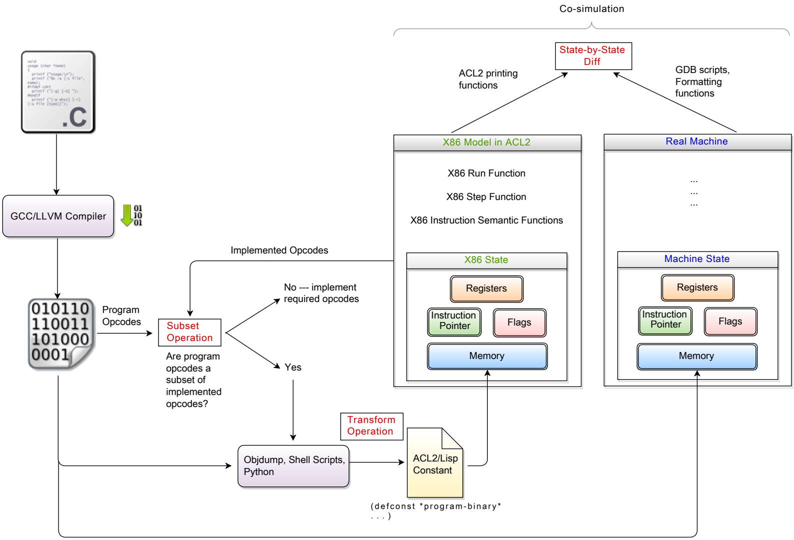
- INPUT: x86 state
  Decoded instruction
  OUTPUT: Next x86 state
- A semantic function describes the effects of executing an instruction.
- Every instruction in the model has its own semantic function.

X86 MODEL

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory
- All addressing modes
- Supports IA-32e paging
- 118 user-mode instructions (219 opcodes)
- +40,000 lines of code

Outline

Executing Binary Programs on X86 model



EXECUTING PROGRAMS ON X86 MODEL

- Execute a contemporary SAT solver on our model
  - Produces exactly the ***same effects*** on model's registers and memory as those produced on the real x86 processor.
- Execution speed: [1]
  - Paging excluded: ~3.3 million instructions/second
  - Paging included: ~300,000 instructions/second
- ACL2 has features that help us **avoid trade-offs** between **efficiency** and **reasoning**.

[1] on an Intel Xeon CPU @ 3.50GHz

Outline

Outline

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data
- **GL**: framework **verified in ACL2** for proving theorems involving **finite symbolic objects** via **bit-blasting**
- Symbolic object: **any** ACL2 object (like lists, numbers, etc.)

OUTLINE

DEMO

Automatic correctness proof for an x86 *popcount* binary program, for **counting** the number of **non-zero bits** in the bit-level representation of an unsigned integer input.

CODE PROOFS: SYMBOLIC EXECUTION APPROACH

- Write the program's **specification**
- Prove that the **program satisfies the specification** (fully automatic)

CODE PROOFS: SYMBOLIC EXECUTION APPROACH

- No lemma construction needed — proof done **fully automatically**
- Reason directly about semantics of programs
  - Account for the complicated x86 **decoding process**
- Proofs of correctness of larger programs to be obtained **compositionally** using traditional theorem proving techniques

Outline

CONCLUSION

- Executable, formal model of a significant subset of x86 ISA
- No simplification of the semantics of x86 instructions
- Validation of the x86 model using efficient co-simulation
- x86 model capable of running as well as reasoning about real x86 binary programs

Future Work

- Add system calls to enable reasoning about I/O (`open`, `read`, `write`, etc.)
- Extend the model with more instructions
- Infrastructure for verification of linux utilities

Automated Code Proofs on a Formal Model of the X86

Shilpi Goel and Warren A. Hunt, Jr.

The University of Texas at Austin

May 18, 2013

Questions?