A Formal Model of the X86 ISA for Binary Program Verification

Shilpi Goel

The University of Texas at Austin

April 2, 2013

Outline

OUTLINE

OUR GOALS

1. Develop an accurate, non-idealized model of the x86 Instruction Set Architecture (ISA)

OUR GOALS

1. Develop an accurate, non-idealized model of the x86 Instruction Set Architecture (ISA)

2. Develop automated procedures for reasoning about x86 machine code

OUR GOALS

1. Develop an accurate, non-idealized model of the x86 Instruction Set Architecture (ISA)

2. Develop automated procedures for reasoning about x86 machine code

Infrastructure for verification of linux utilities like *cat* and *od*

WHY DO WE CARE?

- Analysis of high-level programs is not good enough.

Why Do We Care?

- Analysis of high-level programs is not good enough.
- High-level programs are not always available.

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!

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!

WHY DO WE CARE?

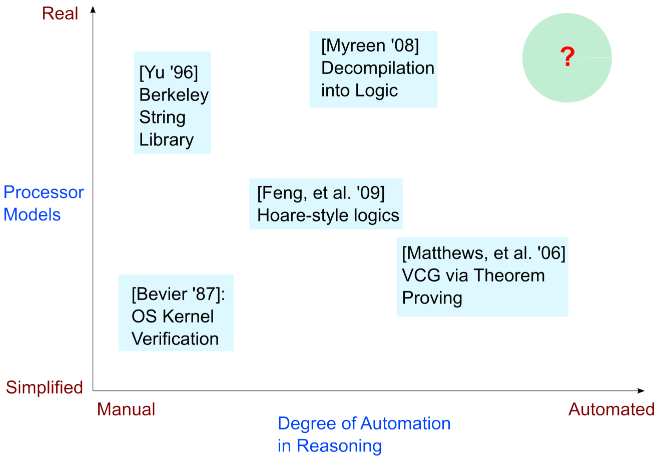
- Analysis of high-level programs is not good enough.
- High-level programs are not always available.
- **Formal verification** of machine code!
  - Formal Model of the x86 ISA

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

OUTLINE

MACHINE CODE VERIFICATION ON FORMAL PROCESSOR MODELS



OUR GOALS, REVISITED

1. Develop an **accurate**, non-idealized, formal, and executable model of the x86 ISA

Our Goals, revisited

1. Develop an accurate, **non-idealized**, formal, and executable model of the x86 ISA

Our Goals, revisited

1. Develop an accurate, non-idealized, **formal**, and executable model of the x86 ISA

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA
   - Specifications: Intel's Software Developer's Manuals

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA
   - Specifications: Intel's Software Developer's Manuals
   - ~4000 pages of prose

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA
   - Specifications: Intel's Software Developer's Manuals
   - ~4000 pages of prose
   - Model should emulate the real machine

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA
   - Specifications: Intel's Software Developer's Manuals
   - ~4000 pages of prose
   - Model should emulate the real machine
   - **Co-simulations**

Our Goals, revisited

1. Develop an accurate, non-idealized, formal, and **executable** model of the x86 ISA
    - Specifications: Intel's Software Developer's Manuals
    - ~4000 pages of prose
    - Model should emulate the real machine
    - **Co-simulations**
    - Need executability to do co-simulations

OUR GOALS, REVISITED

1. Develop an accurate, non-idealized, formal, and executable model of the x86 ISA

OUR GOALS, REVISITED

1. Develop an accurate, non-idealized, formal, and executable model of the x86 ISA

2. Develop automated procedures for reasoning about x86 machine code

OUR GOALS, REVISITED

1. Develop an accurate, non-idealized, formal, and executable model of the x86 ISA

2. Develop automated procedures for reasoning about x86 machine code

    - Functional correctness of machine code

Our Goals, revisited

1. Develop an accurate, non-idealized, 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?

FORMALIZING X86 ISA IN ACL2

ACL2?

- *A Computational Logic for Applicative Common Lisp*

FORMALIZING X86 ISA IN ACL2

ACL2?

- *A Computational Logic for Applicative Common Lisp*
- Programming language

FORMALIZING X86 ISA IN ACL2

ACL2?

- *A Computational Logic for Applicative Common Lisp*
- Programming language
- Mathematical logic

FORMALIZING X86 ISA IN ACL2

ACL2?

- *A Computational Logic for Applicative Common Lisp*
- 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*.

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.

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 (4096 TB)

Run Function

Recursively defined interpreter that specifies the x86 model

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 components of the instruction
  OUTPUT: Next x86 state

INSTRUCTION SEMANTIC FUNCTIONS

- INPUT: x86 state
  Decoded components of the instruction
  OUTPUT: Next x86 state
- A semantic function describes the effects of executing an instruction.

INSTRUCTION SEMANTIC FUNCTIONS

- INPUT: x86 state
  Decoded components of the 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

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode

X86 MODEL

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory

X86 MODEL

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory
- All addressing modes

X86 MODEL

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory
- All addressing modes
- 118 user-mode instructions (219 opcodes)

X86 MODEL

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory
- All addressing modes
- 118 user-mode instructions (219 opcodes)
- Execution speed: ~3.3 million instructions/second

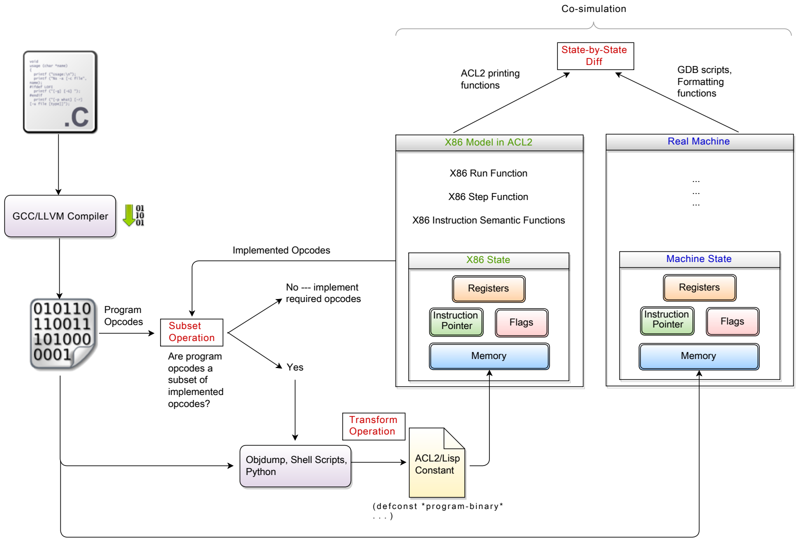
X86 Model

We use Intel's Software Developer's Manuals as our specification.

- 64-bit mode
- Model entire 2^{52} bytes (4096 TB) of memory
- All addressing modes
- 118 user-mode instructions (219 opcodes)
- Execution speed: ~3.3 million instructions/second
- +40,000 lines of code

OUTLINE

Executing Binary Programs on X86 model



OUTLINE

Outline

CODE PROOFS: CLOCK FUNCTION APPROACH

- Write the program's **specification**

CODE PROOFS: CLOCK FUNCTION APPROACH

- Write the program's **specification**
- Write the **algorithm** used in the program

Code Proofs: Clock Function Approach

- Write the program's **specification**
- Write the **algorithm** used in the program
- Prove that the **algorithm satisfies the specification**

CODE PROOFS: CLOCK FUNCTION APPROACH

- Write the program's **specification**
- Write the **algorithm** used in the program
- Prove that the **algorithm satisfies the specification**
- Define **clock functions**

CODE PROOFS: CLOCK FUNCTION APPROACH

- Write the program's **specification**
- Write the **algorithm** used in the program
- Prove that the **algorithm satisfies the specification**
- Define **clock functions**
- Prove that the **program implements the algorithm**

Code Proofs: Clock Function Approach

- Write the program's **specification**
- Write the **algorithm** used in the program
- Prove that the **algorithm satisfies the specification**
- Define **clock functions**
- Prove that the **program implements the algorithm**
- Prove that the **program satisfies the specification**

OUTLINE

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*

Symbolic Execution in ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*
- **GL**: verified framework for proving ACL2 theorems involving **finite objects**

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*
- **GL**: verified framework for proving ACL2 theorems involving finite objects

Symbolic Execution in ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*
- **GL**: verified framework for proving ACL2 theorems involving finite objects
- **Symbolic objects**: finite objects represented by boolean expressions

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*
- **GL**: verified framework for proving ACL2 theorems involving finite objects
- Symbolic objects: finite objects represented by boolean expressions

SYMBOLIC EXECUTION IN ACL2

- **Symbolic Execution**: Executing functions on symbolic data; can be used as a *proof procedure*
- **GL**: verified framework for proving ACL2 theorems involving finite objects
- Symbolic objects: finite objects represented by boolean expressions
- Computations involving these symbolic objects done using **verified BDD operations**

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**

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

CODE PROOFS: SYMBOLIC EXECUTION APPROACH

- No lemma construction needed; proof done fully automatically
- Reason *directly* about semantics of programs (+40,000 LoC)

Code Proofs: Symbolic Execution Approach

- No lemma construction needed; proof done fully automatically
- Reason *directly* about semantics of programs (+40,000 LoC)
- 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

Conclusion

- Executable, formal model of a significant subset of x86 ISA
- No simplification of the semantics of x86 instructions

CONCLUSION

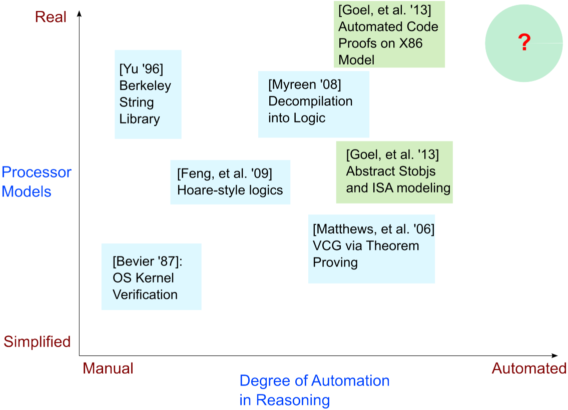
- Executable, formal model of a significant subset of x86 ISA
- No simplification of the semantics of x86 instructions
- X86 ISA model capable of running and reasoning about *real* x86 binary programs

PAPERS

- [ACL2 Workshop'13]: **S. Goel**, W. Hunt, and M. Kaufmann
  Abstract Stobjs and Their Application to ISA Modeling
- [VSTTE'13]: **S. Goel** and W. Hunt
  Automated Code Proofs on a Formal Model of the X86

Papers

- [ACL2 Workshop'13]: **S. Goel**, W. Hunt, and M. Kaufmann
  Abstract Stobjs and Their Application to ISA Modeling
- [VSTTE'13]: **S. Goel** and W. Hunt
  Automated Code Proofs on a Formal Model of the X86



Future Work

- Add **system calls** to enable reasoning about I/O (`open`, `read`, `write`, etc.)

Future Work

- Add **system calls** to enable reasoning about I/O (`open`, `read`, `write`, etc.)
- Further **automate** the **co-simulation** framework
  - Automated test case generation
  - Enhance GDB mode framework

Future Work

- Add **system calls** to enable reasoning about I/O (`open`, `read`, `write`, etc.)
- Further **automate** the **co-simulation** framework
  - Automated test case generation
  - Enhance GDB mode framework
- Build automated binary **program annotation and instrumentation** tools

FUTURE WORK

- Add **system calls** to enable reasoning about I/O (`open`, `read`, `write`, etc.)
- Further **automate** the **co-simulation** framework
  - Automated test case generation
  - Enhance GDB mode framework
- Build automated binary **program annotation and instrumentation** tools
- Infrastructure for verification of linux utilities like *cat* and *od*

A Formal Model of the X86 ISA for Binary Program Verification

Shilpi Goel

The University of Texas at Austin

April 2, 2013