A Formal Model of x86 for Machine-Code Proofs

Shilpi Goel
shigoel@cs.utexas.edu
The University of Texas at Austin
**Goal:** Build a program verification framework that can be deployed in the software industry.
Goal: Build a program verification framework that can be deployed in the software industry.

Observations:

- *Machine code verification* frameworks can serve as general-purpose program analysis frameworks.
Goal: Build a program verification framework that can be deployed in the software industry.

Observations:

- **Machine code verification** frameworks can serve as general-purpose program analysis frameworks.

- Analysis of program behavior is done by both *simulation* and *formal verification*.
Goal: Build a program verification framework that can be deployed in the software industry.

Observations:

- *Machine code verification* frameworks can serve as general-purpose program analysis frameworks.

- Analysis of program behavior is done by both *simulation* and *formal verification*.

- There are *separate tools* for simulation and formal verification.
Approach

Develop a **formal** and **executable** model of the x86 instruction set architecture in the **ACL2** theorem proving system.
Approach

Develop a **formal** and **executable** model of the x86 instruction set architecture in the [ACL2](https://ACL2.org) theorem proving system.

- **Simulate x86 machine code programs produced by GCC/LLVM compilers.**
  - **Co-simulations** are done to validate the model.
    - We believe that we have the **fastest formal x86 simulator**.
      (~580K - 2.4 million instructions/sec)
Approach

Develop a **formal** and **executable** model of the x86 instruction set architecture in the **ACL2** theorem proving system.

Simulate x86 machine code programs produced by GCC/LLVM compilers.

- **Co-simulations** are done to validate the model.
  - We believe that we have the **fastest formal x86 simulator**.
    (~580K - 2.4 million instructions/sec)

Prove or disprove the **correctness of machine code programs** with respect to their specifications.

- Reason about straight-line code automatically using a verified bit-blasting library in ACL2.
Automated reasoning about machine code

- Mechanically verify non-trivial programs like `cat`, standard library functions, operating system processes, etc.
Future Work

Automated reasoning about machine code
- Mechanically verify non-trivial programs like `cat`, standard library functions, operating system processes, etc.

Analysis of resource usage
- Memory consumption
Automated reasoning about machine code
- Mechanically verify non-trivial programs like `cat`, standard library functions, operating system processes, etc.

Analysis of resource usage
- Memory consumption

Program comprehension and bug identification
- Is there any set of inputs for a program that can produce a desired output?
A Formal Model of x86 for Machine-Code Proofs

Shilpi Goel
shigoel@cs.utexas.edu
The University of Texas at Austin