UT Austin CRASH Project

Nathalie Beavers, Soumava Ghosh, Shilpi Goel, Marijn J. H. Heule, Warren A. Hunt, Jr., Matt Kaufmann, Robert B. Krug, J Strother Moore, Ben Selfridge, Nathan Wetzler

May, 2013

Computer Science Department
1 University Way, M/S C0500
University of Texas
Austin, TX 78712-0233

hunt@cs.utexas.edu
TEL: +1 512 471 9748
FAX: +1 512 471 8885

Outline

Outline

Introduction

To enable the modeling and analysis of industrial-sized systems:

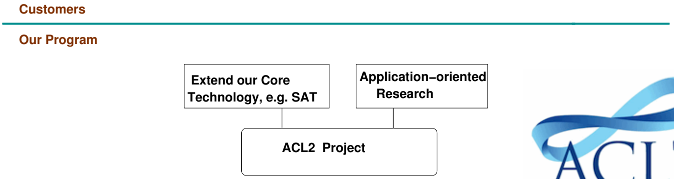
- We are developing an x86 model suitable for code analysis.
- We are extending our ACL2-based analysis toolsuite with SAT.
- We are vetting our tools on commercial-sized problems.
- We are improving our ACL2 theorem proving environment.

Our ACL2-based modeling and analysis toolsuite is in use by AMD, Centaur, IBM, Rockwell-Collins, and others.

Today, we present our approach for modeling the x86 ISA and our use of SAT for proof by symbolic execution.

Ecosystem

We have significant collaboration with the industry.

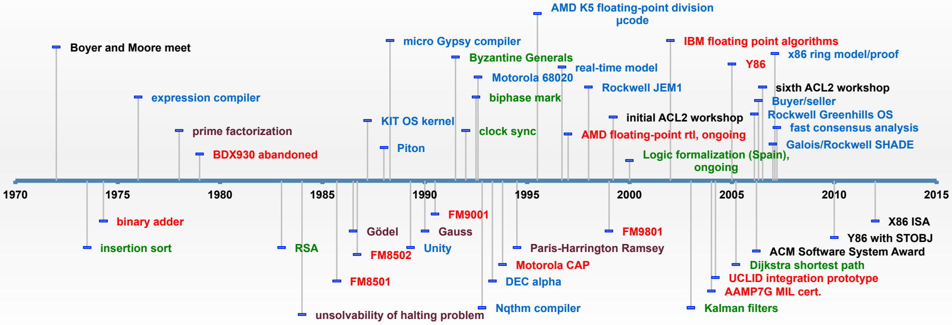


Our own research includes:

- **Development** of core technologies
- **Application** of these technologies on different verification domains
- **Commercial Driver:** validation for Centaur's x86 design

Timeline

- Our group has been working on the development and deployment of reasoning systems for 40 years.

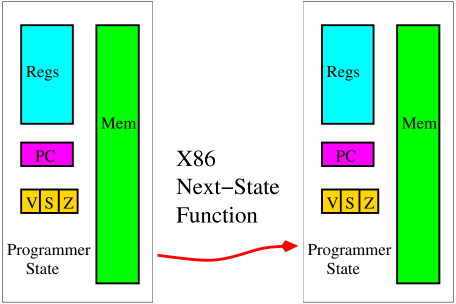


Outline

Evolving X86 ISA Model

We are developing a **formal**, **executable** x86 ISA model.

- Our x86 model implements almost all one- and two-byte instructions.
- For all defined instructions, model implements all addressing modes.
- It can emulate many x86 binary programs emitted by GCC/LLVM.
- We continue to do co-simulations to gain confidence in our model.



X86 Top-Level Model

We have a formal, executable implementation of **118 user-level x86 instructions** (219 opcodes).

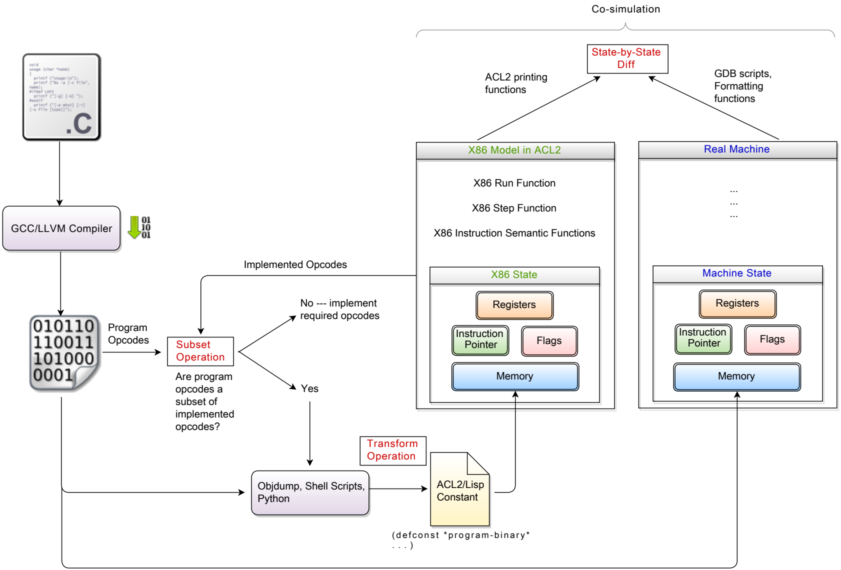
```
(defun x86-run (n x86)
 ; Returns x86 obtained by executing n instructions (or until halting).
  (cond ((ms x86) x86)

        ((zp n) x86)

        (t (let ((x86 (x86-fetch-decode-execute x86)))
             (x86-run (1- n) x86)))))
```

- x86 model about 40,000 lines in size, including our evolving 64-bit paging model
- Execution speed with paging included: 300,000 instructions/second
- Execution speed with paging excluded: 3 million instructions/second

Emulating X86 Programs



Emulating X86 Programs

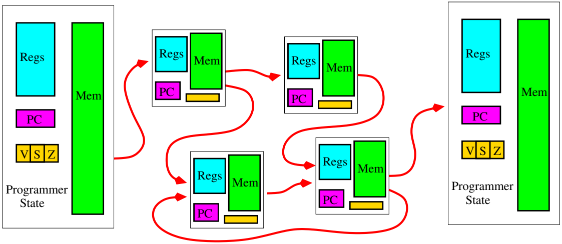
We run x86 binary programs on our x86 model.

- We can run a contemporary **SAT solver** on our x86 model.
- We have modified this solver so that it doesn't require system calls...
- We solve SAT competition benchmarks on our model; the largest example tried so far: *cmu-bmc-barrel6.cnf*.
  - Number of variables: 2306
  - Number of clauses: 8931
  - Number of x86 instructions executed: **9,142,833,444**
- Co-simulation: Our model produced exactly the **same effects** on the memory and registers as those produced by a physical x86 processor.

Verifying X86 Programs

We use ACL2(h) to symbolically execute our x86 model.

- We compile C-code with GCC/LLVM.
- We load binary code into the memory of our x86 model.
- As appropriate, we initialize the registers, etc. with symbolic values.
- Previously, we have demonstrated a **fully automatic proof** of correctness of a x86 binary program using symbolic execution.



Outline

BDDs versus SAT for Symbolic Execution

Our symbolic execution framework uses either BDDs or AIGs. AIGs are transformed into CNF formulas, and checked by SAT solvers.

Binary Decision Diagrams (BDDs)

- A mature technology; however,
- Memory problems arise for large problems
- Fully integrated into ACL2(h)

Satisfiability (SAT) solving

- Powerful technology that is improved yearly
- Problems can be solved with millions of clauses
- Used as external tool

For some problems, SAT is demonstratively more effective.

Matrix Root Problem in ACL2

Given a 2×2 matrix M, does there exist a 2×2 matrix R with only natural numbers such that $R^2 = M$?

$$\begin{pmatrix} \mathtt{a} & \mathtt{b} \\ \mathtt{c} & \mathtt{d} \end{pmatrix}^2 = \begin{pmatrix} \mathtt{w} & \mathtt{x} \\ \mathtt{y} & \mathtt{z} \end{pmatrix}$$

```
(defun matrix-root? (a b c d  w x y z)
  (declare (xargs :guard (and (natp a) (natp b)
                              (natp c) (natp d))))
  (let* ((ww (+ (* a a) (* b c))) (xx (+ (* a b) (* b d)))
         (yy (+ (* c a) (* d c))) (zz (+ (* c b) (* d d))))
    (and (equal w ww) (equal x xx)
         (equal y yy) (equal z zz))))
```

Matrix Root Problem with a Solution

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^2 = \begin{pmatrix} 229452 & 269434 \\ 326414 & 385740 \end{pmatrix} = \begin{pmatrix} 311 & 331 \\ 401 & 503 \end{pmatrix}^2$$

Proof fails: using BDDs in 181 secs; however, with SAT in 0.03 secs!

```
(def-gl-thm matrix-root-large-fails
  :hyp (let ((m 512))
         (and (natp a) (natp b) (natp c) (natp d)
              (< a m)  (< b m)  (< c m)  (< d m)))
  :concl
  (not (matrix-root? a b c d 229452 269434 326414 385740))


  :g-bindings
  `((a (:g-number ,(gl-int  0 1 10)))
    (b (:g-number ,(gl-int 10 1 10)))
    (c (:g-number ,(gl-int 20 1 10)))
    (d (:g-number ,(gl-int 30 1 10)))))
```

Matrix Root Problem with No Solutions

$$\begin{pmatrix} \texttt{a} & \texttt{b} \\ \texttt{c} & \texttt{d} \end{pmatrix}^2 = \begin{pmatrix} 229450 & 269434 \\ 326414 & 385740 \end{pmatrix} \quad \text{has no solutions}$$

Proof succeeds: using BDDs in 177 secs; however, with SAT in 0.01 secs!

```
(def-gl-thm matrix-root-large-succeeds
  :hyp (let ((m 512))
         (and (natp a) (natp b) (natp c) (natp d)
              (< a m)  (< b m)  (< c m)  (< d m)))
  :concl
  (not (matrix-root? a b c d 229450 269434 326414 385740))

  :g-bindings
  `((a (:g-number ,(gl-int  0 1 10)))
    (b (:g-number ,(gl-int 10 1 10)))
    (c (:g-number ,(gl-int 20 1 10)))
    (d (:g-number ,(gl-int 30 1 10)))))
```

Verification of SAT results

SAT solving is a powerful technique, but presently external to ACL2:

- SAT solutions are easy to check (in linear time)
- Clausal proofs are popular but expensive to check
- Even a SAT proof checker is hard to mechanically verify

How can we deal with a claim that no solution exists?

- Our solution: filter clausal proofs
- Then, check the filtered proof with a verified proof checker

Tool Chain for Checking Unsatisfiability Results

Input: Boolean formula constructed by ACL2

↙ ↓ ↘

Black box solver *Proof filter* *Verified checker*

→

unsatisfiable?

↘ ↑

Output: refutation proof emitted by SAT solver

Matrix Root Problem with No Solutions Verified

$$\begin{pmatrix} \mathtt{a} & \mathtt{b} \\ \mathtt{c} & \mathtt{d} \end{pmatrix}^2 = \begin{pmatrix} 229450 & 269434 \\ 326414 & 385740 \end{pmatrix} \quad \text{has no solutions}$$

Proof succeeds: using BDDs in 177 secs; however, with SAT in 0.01 secs!

Results on proof checking:

- Boolean formula contained 4365 variables and 14749 clauses
- Glucose solves the formula emitting a proof of 333 lemmas
- Proof filter reduced the proof to 2 lemmas
- The reduced proof is checked using a verified checker in 1.3 seconds
- Future work: verify faster proof checker!

Outline

ACL2 Enhancements

ACL2 has been under continuous development for 20+ years, largely in response to user requests. Release notes document 100s of enhancements.

ACL2 is freely available at: http://www.cs.utexas.edu/users/moore/acl2/

- Released ACL2 Version 6.0 in December, 2012
- Released ACL2 Version 6.1 in February, 2013
- Sample developments:
  - Change in license: From GPL Version 2 to a *3-clause BSD license*
  - High functionality data structures: *Abstract and Nested Stobjs*
  - Better feedback from the prover: *Case split reports*
  - Heuristic improvements: *Arithmetic bounders for tau*

In addition, we developed a proof format for SAT solvers to facilitate easy generation and efficient verification of compact proofs. We plan to enhance ACL2 with SAT technology like we did with our BDD package.

Outline

Future Work

Extending the ACL2 system:

- Integrate SAT mechanisms into ACL2 proof infrastructure
- Improve efficiency of our symbolic simulation techniques
- In general, improve the ACL2 system, supporting our x86 ISA modeling and proof efforts

Extending our x86 ISA model:

- Develop infrastructure for binary code proofs
- Integrate x86 memory management into our model
- Add system calls to the x86 model
- Continue extending the number of instructions modeled
- Further automate our co-simulation environment for model validation

Outline

Conclusion

We continue to expand our modeling and analysis capabilities.

- We have developed a 64-bit data and 52-bit address memory model.
- We have specified most integer instructions with all addressing modes.
- We are developing a co-simulation mechanism for model validation.
- We have started verifying x86 binary programs.
- Our model can be used as a build-to and a compile-to specification.
- We have developed a path to use SAT with our proofs.
- We have extended and improved our x86 model.
- Our model can be used to safely explore all manner of malware.
- We continue to enhance the ACL2 system.

We perform our work in an environment where we can prove or disprove theorems about our models.

Outline

Publications

Since November, 2012:

- *Automated Reencoding of Boolean Formulas*
  Norbert Manthey and **Marijn J. H. Heule** and Armin Biere
- *Revisiting Hyper Binary Resolution*
  Marijn J. H. Heule, Matti Jarvisalo, and Armin Biere
- *Enhancements to ACL2 in Versions 5.0, 6.0, and 6.1*
  Matt Kaufmann and **J Strother Moore**
- *A Parallelized Theorem Prover for a Logic with Parallel Execution*
  David L. Rager, **Warren A. Hunt, Jr.**, and **Matt Kaufmann**
- *Abstract Stobjs and Their Application to ISA Modeling*
  Shilpi Goel, **Warren A. Hunt, Jr.**, and **Matt Kaufmann**
- *Automated Code Proofs on a Formal Model of the X86*
  Shilpi Goel and **Warren A. Hunt, Jr.**
- *Verifying Refutations with Extended Resolution*
  Marijn J. H. Heule, **Warren A. Hunt, Jr.**, and **Nathan Wetzler**
- *Mechanical Verification of SAT Refutations with Extended Resolution*
  Nathan Wetzler, **Marijn J. H. Heule**, and **Warren A. Hunt, Jr.**
- *A SAT Approach to Clique-Width*
  Marijn J. H. Heule and Stefan Szeider