Making

Formal Methods

Disappear

Ganesh Gopalakrishnan
With acknowledgements to his students and colleagues, especially Mike Kirby

http://www.cs.utah.edu/fv

University of Utah
We are in a complex world of digital designs
We are in a complex world of digital designs

From the mindboggling complexity of Hardware...

“I meant to put in 3B transistors; how do I know they are all there?”

“If I count them one transistor a second, I’ll be dead before I finish counting!”
We are in a complex world of digital designs

From the mindboggling complexity of Hardware...

To the mind-numbing complexity and variety of software...

"I meant to put in 3B transistors; how do I know they are all there?"

"If I count them one transistor a second, I’ll be dead before I finish counting!"

<table>
<thead>
<tr>
<th>S/W solutions</th>
<th>H/W platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC</td>
<td>MPI, UPC, OpenCL, CUDA</td>
</tr>
<tr>
<td>Server / Desktop</td>
<td>Pthreads, TBB, TPL, OpenMP</td>
</tr>
<tr>
<td>Embedded Computing</td>
<td>MCAPI, MRAPI</td>
</tr>
</tbody>
</table>
We are in a complex world of digital designs

From the mindboggling complexity of Hardware...

To the mind-numbing complexity and variety of software...

...correctness and reliability are CENTRAL CHALLENGES underlying whatever we do!

<table>
<thead>
<tr>
<th></th>
<th>S/W solutions</th>
<th>H/W platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC</td>
<td>MPI, UPC, OpenCL, CUDA</td>
<td>Cluster Supercomputers</td>
</tr>
<tr>
<td>Server / Desktop</td>
<td>Pthreads, TBB, TPL, OpenMP</td>
<td>Desktop machines</td>
</tr>
<tr>
<td>Embedded Computing</td>
<td>MCAPI, MRAPI</td>
<td>FPGAs, SoC</td>
</tr>
</tbody>
</table>
### Example

i.e. There is Trouble in the “Engine Room!”

<table>
<thead>
<tr>
<th>“AI”</th>
<th>“ML”</th>
<th>“Graphics”</th>
<th>“Big Data”</th>
<th>“Robotics”</th>
<th>“Web”</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>S/W solutions</th>
<th>H/W platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC</td>
<td>MPI, UPC, OpenCL, CUDA</td>
</tr>
<tr>
<td>Server / Desktop</td>
<td>Pthreads, TBB, TPL, OpenMP</td>
</tr>
<tr>
<td>Embedded Computing</td>
<td>MCAPI, MRAPI</td>
</tr>
</tbody>
</table>
Correctness of computing systems is essential

• Underemphasized so far in CS education
  – Emphasis varies with university / department

• Disruptive technologies (parallel and concurrent hardware and software) makes correctness harder to define / achieve:
  – Heterogeneous concurrent programming
    • “A bad idea whose time has come” (PACT talk title)
  – Traditional “Software Engineering” has largely ignored concurrency
    • Conferences such as FSE, ASE, ICSE beginning to respond
What does ‘Formal Methods’ Address?

• The correctness of digital computing systems
  – Hardware
  – Software
• Formal methods can also address performance
• Correctness/performance separation often good
  – Can only carry so much in one’s head
  – You may fixate on inconsequential performance losses
    • Profiling TRULY shows where performance mattered!
  – Those who aim for correctness can later aim for performance THAT REALLY MATTERS
Why do we need Formal Methods?

• Today’s testing methods are
  – Unreliable and wasteful
    • Glaring omissions occur
    • Redundant tests are administered
    • Yet, no metric on coverage attained
  – Especially for concurrent / parallel systems
  – Unbounded number of pitfalls
  – Formal Methods must be used EARLY during the design
    • Bug caught early may make less news (smaller bonus checks)
    • Fortunately, engineers still care to get it right the first time IF ONLY THEY KNEW HOW TO DO IT (even in simple cases).
Why do we need to “hide” Formal Methods?

• Bob Colwell’s story of the “12 transistor radio”
Why do we need to “hide” Formal Methods?

• Engineers need math to understand/conquer complexity
• After exerting a formative role, the math must stay out
  – Context-free Grammars to build Parsers
  – Differential Calculus to build Bridges
  – Pringle aerodynamics
  – Navier-Stokes equations may help design the best diapers
    • But impresses the least # of parents in diaper aisles

To be excessively wedded to the math once it has served its purpose can dissuade practitioners.
Hence one must “hide” formal methods into good tool flows, design practices, clear documentation, etc.
BUT first, we must use math to grow many FM areas !!
Why do we need to “hide” Formal Methods?

• Engineers need math to understand/conquer complexity
• After exerting a formative role, the math must stay out
  – Context-free Grammars to build Parsers
  – Differential Calculus to build Bridges
  – Pringle aerodynamics
  – Navier-Stokes equations may help design the best diapers
    • But impresses the least # of parents in diaper aisles

To be excessively wedded to the math once it has served its purpose can dissuade practitioners.
Hence one must “hide” formal methods into good tool flows, design practices, clear documentation, etc.
BUT first, we must use math to grow many FM areas!!
Who uses FM? Some Hardware Successes...

• Intel’s Pentium FDIV bug of 1995 spurred a LOT of interest
• Ariane’s $2B explosion added to the interest
  • After 12 years: Intel i7 floating-point unit correctness FV-ed!
    ─ A lot of simulation work was completely eliminated!
      • Real $ savings + winning the trust of real engineers that FV works!
      • Symbolic Trajectory Evaluation tools provide coverage for ALL inputs
        ─ Not just the ones you picked in your dreams
• Cache coherency hardware in all your computers
  ─ Starting from UltraSparc-1 in 1995, they have been FVed at the protocol state machine level
    • Hardware that runs at GHz may not run into known bugs for months
    • The smallest schedule perturbation / porting → bugs erupt!!
• All the CAD tools you use to build circuits
  ─ Formal Equivalence Verification tools do HEAVY LIFTING
Who uses FM? Some Hardware Successes...

- Intel’s Pentium FDIV bug of 1995 spurred a LOT of interest
- Ariane’s $2B explosion added to the interest
- After 12 years: Intel i7 floating-point unit correctness FV-ed!
  - A lot of simulation work was completely eliminated!
    - Real $ savings + winning the trust of real engineers that FV works!
    - Symbolic Trajectory Evaluation tools provide coverage for ALL inputs
      - Not just the ones you picked in your dreams
- Cache coherency hardware in all your computers
  - Starting from UltraSparc-1 in 1995, they have been FVed at the protocol state machine level
    - Hardware that runs at GHz may not run into known bugs for months
    - The smallest schedule perturbation / porting → bugs erupt!!
- All the CAD tools you use to build circuits
  - Formal Equivalence Verification tools do HEAVY LIFTING
Who uses FM? Some Hardware Successes...

- Intel’s Pentium FDIV bug of 1995 spurred a LOT of interest
- Ariane’s $2B explosion added to the interest
- After 12 years: Intel i7 floating-point unit correctness FV-ed!
  - A lot of simulation work was completely eliminated!
    - Real $ savings + winning the trust of real engineers that FV works!
    - Symbolic Trajectory Evaluation tools provide coverage for ALL inputs
      - Not just the ones you picked in your dreams
- Cache coherency hardware in all your computers
  - Starting from UltraSparc-1 in 1995, they have been FVed at the protocol state machine level
    - Hardware that runs at GHz may not run into known bugs for months
    - The smallest schedule perturbation / porting → bugs erupt !!
- All the CAD tools you use to build circuits
  - Formal Equivalence Verification tools do HEAVY LIFTING
Who uses them? Software Successes...

- Bell Labs pioneered early use in switch protocols
- Microsoft Device Driver Certification tools
- Coding practices to check in new codes into builds requires designers to write assertions
  - Pertaining to parameters and side effects
  - Types for atomicity (to cheaply check for races)
- Testing for browser vulnerability
  - Manufacturers discover attacks ahead of competition
    - Using First Order Decision Procedures
  - Often don’t release patches – “why muddy the water”?
  - “Patch” magically appears in a day!
    - FV had helped calculate and keep it ready!
- JPL NASA, NSA, car companies, Airbus, Rockwell-Collins, NEC, Fujitsu, Intel, AMD, IBM, ... all use FV for HW, SW, and Microcode
Who uses them? Software Successes...

- Bell Labs pioneered early use in switch protocols
- Microsoft Device Driver Certification tools
- Coding practices to check in new codes into builds requires designers to write assertions
  - Pertaining to parameters and side effects
  - Types for atomicity (to cheaply check for races)
- Testing for browser vulnerability
  - Manufacturers discover attacks ahead of competition
    - Using First Order Decision Procedures
  - Often don’t release patches – “why muddy the water”?
  - “Patch” magically appears in days
    - Perhaps FV had helped calculate and keep them ready?
- JPL NASA, NSA, car companies, Airbus, Rockwell-Collins, NEC, Fujitsu, Intel, AMD, IBM, ... all use FV for HW, SW, and Microcode
Who uses them? Software Successes...

• Bell Labs pioneered early use in switch protocols
• Microsoft Device Driver Certification tools
• New codes can’t be checked in unless they have assertions
  – Pertaining to parameters and side effects
  – Types for atomicity (to cheaply check for races)
• Testing for browser vulnerability
  – Manufacturers discover attacks ahead of competition
    • Using First Order Decision Procedures
  – Often don’t release patches – “why muddy the water”?
  – “Patch” magically appears in a day!
    • FV had helped calculate and keep it ready!

• JPL NASA, NSA, car companies, Airbus, Rockwell-Collins, NEC, Fujitsu, Intel, AMD, IBM, … all use FV for HW, SW, and Microcode
The very idea of verification seems a non-starter

(much like the Bumble bee is not supposed to fly..)
The very idea of verification seems a non-starter

(much like the Bumble bee is not supposed to fly..)

• Most problems are undecidable!
• Easier ones: Non Primitive Recursive
• Still easier: “Ordinary Exp.”

Solution:

Don’t bother! Do it anyway!
Find representations that are linear (most cases)
Develop skills to accommodate real problems!

Don’t misinterpret complexity theory results!
The very idea of verification seems a non-starter

(much like the Bumble bee is not supposed to fly..)

• Most problems are undecidable!
• Easier ones: Non Primitive Recursive
• Still easier: “Ordinary Exp.”

Solution:
Don’t bother! Do it anyway!
Find representations that are linear (most cases)
Develop skills to accommodate real problems!

Don’t misinterpret complexity theory results!
I’ve heard all that before; how does FM really work?

- I’ll show you FM through six real examples
- Each example will touch upon fundamental questions
  - Wasn’t it supposed to be NP-complete?
  - Or in some cases non Primitive Recursive?
  - Or in some cases semi-decidable?
  - Or in some cases undecidable?
I’ve heard all that before; how does FM really work?

• I’ll show you FM through six real examples
• Each example will touch upon fundamental questions
  – Wasn’t it supposed to be NP-complete?
  – Or in some cases non Primitive Recursive?
  – Or in some cases semi-decidable?
  – Or in some cases undecidable?
• FV answer : Go away! I’ll do it anyhow!
  – i.e. find Exp. Succinct ways to represent / compute!
  – ...with a dash of empirical facts and randomization
What are some Exp. Succinct representations?

• Positional number system
  – Those Indians invented NOTHING!
  – Knuth’s number story

• NFA vs. DFA

• Quantified Boolean formulae versus ordinary Boolean formulae

• … what others... ?
Demos #1

• How large Boolean circuits (think FPUs) are verified relying upon compact representations
  – Minimized DFAs can compactly encode Boolean functions!
    • DFAs not exp succinct UNLESS a lot of “common prefix sharing” goes on
    • Such DFA hash-tables can store GBs in KBs
  – Canonical (Myhill / Nerode) – equality
  – Heuristic required : pick variable decoding order!
    • Maximizes common prefix sharing likelihood
Example demonstrated

<table>
<thead>
<tr>
<th>Solving b7 b6 b5 b4 b3 b2 b1 b0 = a7 a6 a5 a4 a3 a2 a1 a0</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.e. equality comparison</td>
</tr>
<tr>
<td>• Truth-table for a 64-bit equality comparator (\rightarrow) 2^128 entries</td>
</tr>
<tr>
<td>• BDD for it (\rightarrow) about 128 entries</td>
</tr>
</tbody>
</table>
Demos #2

- How logical reasoning can be supported with counterexample generation describing missed facts
  - Counterexample generation is one of the nicest byproducts of symbolic verification
Puzzle from Lewis Carroll

- All who neither dance on tight ropes nor eat penny-buns are old.
- Pigs, that are liable to giddiness, are treated with respect.
- A wise balloonist takes an umbrella with him.
- No one ought to lunch in public who looks ridiculous and eats penny-buns.
- Young creatures, who go up in balloons, are liable to giddiness.
- Fat creatures, who look ridiculous, may lunch in public, provided that they do not dance on tight ropes.
- No wise creatures dance on tight ropes, if liable to giddiness.
- A pig looks ridiculous carrying an umbrella.
- All who do not dance on tight ropes and who are treated with respect are fat.

Show that no wise young pigs go up in balloons.
Encoding the puzzle

- let A1 = ((not dance) and (not eats)) => old;
- let A2 = (pig and giddy) => respect;
- let A3 = (wise and balloon) => umbrella;
- let A4 = (ridic and eats) => (not public);
- let A5 = (young and balloon) => giddy;
- let A6 = (fat and ridic and (not dance)) => public;
- let A7 = (wise and giddy) => (not dance);
- let A8 = (pig and umbrella) => ridic;
- let A9 = ((not dance) and respect) => fat;
- let P0 = wise;
- let P1 = young;
- let P2 = pig;
- let P3 = balloon;
- let goal = A1 and A2 and A3 and A4 and A5 and A6 and A7 and A8 and A9 and P0 and P1 and P2 and P3;
- upall goal;
- view goal; --- must be FALSE. Then we have a proof by contradiction!
Demos #3

• How C semantics can be symbolically encoded
  – Again shows the power of symbolic reasoning
  – Modern developments in this area are in the area of Satisfiability Modulo Theories
Example demonstrated

• How logical reasoning can be supported with counterexample generation describing missed facts

  – Counterexample generation is one of the nicest byproducts of symbolic verification

    main(){
        int Z1, Z2, Z3;
        int x1, x2;
        int z11, z12, z13, z21, z22, z23;
        /* x1 = x2; */
        z11 = z21; z12 = z22; z13 = z23;
        if (x1 == 1) z11 = Z1; if (x1 == 2) z12 = Z2; if (x1 == 3) z13 = Z3;
        if (x2 == 1) z21 = Z1; else if (x2 == 2) z22 = Z2; else if (x2 == 3) z23 = Z3;
        assert((z11 + z12 + z13) == (z21 + z22 + z23));
    }
Demos #4

• How Pthread / C programs can be verified
  – Symbolic encodings become too unwieldy
  – We need good “explicit search” methods
    • Let there be $P$ processes executing $K$ atomic steps each
    • Need heuristics to bound the number of interleavings which can grow as
      $$(K \cdot P)! / (K!)^P$$ which is over $10^B$ for $K=5, \, P=5$$
Pthread deadlock due to “lost signal” (monitor)

if (qsize == 0)
    pthread_cond_wait(&cond_empty, &mux);

• FIXED TO

while (qsize == 0)
    pthread_cond_wait(&cond_empty, &mux);
We have built a tool for Thread App. Verification - Inspect

- Multithreaded C Program
- Program Instrumentor
- Instrumented Program
- Thread Library Wrapper
- Program Analyzer
- Analysis result
- Executable
  - thread 1
  - thread n
- Scheduler

Compile
request/permit
request/permit
Demos #5

• How MPI programs can be formally verified
  – Capture MPI semantics in Search Algorithms
  – Again severely bound the number of interleavings examined without losing ANY coverage
Our tool for Msg Passing App Verification - ISP

- MPI Program
- Interposition Layer

Executable

Proc_1
Proc_2
......
Proc_n

Scheduler

MPI Runtime

- Hijack MPI Calls
- Scheduler decides how they are sent to the MPI runtime
- Scheduler plays out only the RELEVANT interleavings (to detect safety violations such as deadlocks and assertion violations)
Demos #6

• How are large Boolean circuits (think FPUs, GPUs, hybrid systems, ...) are verified relying upon compact representations
  – An example of GPU inter-iteration race detection
  – Random testing almost guaranteed to miss these
Long-term view of CUDA / OpenCL FV

C Application Containing Multiple Kernels

Analyzer

Kernel Descriptions

PUG Analyzer for Races and Assertions

CPU / GPU Communication Codes

Kernel Invocation Contexts

CPU / GPU Communication Verifier (CGV)

Verification Results

Verification Results
PUG’s Symbolic Approach

C Application Containing Multiple Kernels → Analyzer supported by LLNL Rose → Verification Conditions i.e. “Constraints” → Constraint solver (Fast Logical Decision Procedures)

UNSAT: The instance is “OK” – i.e.
- Race-free
- No mismatched barriers
- Passes user Assertions

SAT: The instance has bugs
- Puts out “bread crumbs” to help debug
  (SAT instance)
Demo : real race (GPU class)

```c
__global__ void computeKernel(int *d_in, int *d_out, int *d_sum) {

    d_out[threadIdx.x] = 0;
    for (int i=0; i<SIZE/BLOCKSIZE; i++) {
        d_out[threadIdx.x] += compare(d_in[i*BLOCKSIZE+threadIdx.x],6);
    }

    __syncthreads();
    assume(blockDim.x <= BLOCKSIZE / 2); // for testing
    if(threadIdx.x%2==0) {
        for(int i=0; i<SIZE/BLOCKSIZE; i++) {
            d_out[threadIdx.x+SIZE/BLOCKSIZE*i]+=d_out[threadIdx.x+SIZE/BLOCKSIZE*i+1];
        }
    }

    /* The counter example given by PUG is: TRY HITTING THIS VIA RANDOM TESTING!
    t1.x = 2, t2.x = 10, i@t1 = 1, i@t2 = 0,
    that is,
    d_out[threadIdx.x+8*i]+=d_out[threadIdx.x+8*i+1];
    d_out[2+8*1]+=d_out[10+8*0+1];
    */
```
**Sample results: Bug-free Examples**

<table>
<thead>
<tr>
<th>Kernels (in CUDA SDK)</th>
<th>loc</th>
<th>+O</th>
<th>+C</th>
<th>+R</th>
<th>B.C.</th>
<th>Time (sec.) (pass)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitonic Sort</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td>HIGH</td>
<td>2.2</td>
</tr>
<tr>
<td>MatrixMult</td>
<td>102</td>
<td>*</td>
<td>*</td>
<td></td>
<td>HIGH</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Histogram64</td>
<td>136</td>
<td></td>
<td></td>
<td></td>
<td>LOW</td>
<td>2.9</td>
</tr>
<tr>
<td>Sobel</td>
<td>130</td>
<td>*</td>
<td></td>
<td></td>
<td>HIGH</td>
<td>5.6</td>
</tr>
<tr>
<td>Reduction</td>
<td>315</td>
<td></td>
<td></td>
<td></td>
<td>HIGH</td>
<td>3.4</td>
</tr>
<tr>
<td>Scan</td>
<td>255</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>LOW</td>
<td>3.5</td>
</tr>
<tr>
<td>Scan Large</td>
<td>237</td>
<td>*</td>
<td>*</td>
<td></td>
<td>LOW</td>
<td>5.7</td>
</tr>
<tr>
<td>Nbody</td>
<td>206</td>
<td>*</td>
<td></td>
<td></td>
<td>HIGH</td>
<td>7.4</td>
</tr>
<tr>
<td>Particles</td>
<td>320</td>
<td>*</td>
<td>*</td>
<td></td>
<td>HIGH</td>
<td>6.3</td>
</tr>
<tr>
<td>Bisect Large</td>
<td>1400</td>
<td>*</td>
<td>*</td>
<td></td>
<td>HIGH</td>
<td>44</td>
</tr>
<tr>
<td>Radix Sort</td>
<td>1150</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>LOW</td>
<td>39</td>
</tr>
<tr>
<td>Eigenvalues</td>
<td>2300</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>HIGH</td>
<td>68</td>
</tr>
</tbody>
</table>

+ O: required assertions to specify that bit-vector computations don’t overflow

+C: required constraints on the input values

+R: required manual loop refinement

B.C.: measures how serious the bank conflicts are

Time: SMT solving time in seconds to confirm absence of issues.
**Sample results: Buggy Examples**

We tested 57 assignment submissions from a recently completed graduate GPU class taught in our department.

<table>
<thead>
<tr>
<th>Defects</th>
<th>Barrier Error or Race</th>
<th>Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>benign</td>
<td>fatal</td>
</tr>
<tr>
<td>13 (23%)</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Defects**: Indicates how many kernels are not well parameterized, i.e. work only in certain configurations

**Refinement**: Measures how many loops need automatic refinement.
How to make Formal Methods Disappear?

• Our GEM plug-in for MPI dynamic model checking is a good example

• Seems like a debugger
• Yet under the hoods provides formal coverage guarantees

• Another good example is LineUp (MSR)
Concluding Remarks

• FM has matured
• In many cases, it is SO MATURE that it is being hidden into countless realistic tools
• In other cases, its math is still the primary item of interest
• FM community size is miniscule compared to “ad hoc testing” team sizes
• Education is key to progress
• Demos such as these are essential, or otherwise our area will continue to suffer from neglect
• Will be teaching BDDs in CS 3100
Various FV tool design activities in our group

Integrated Eclipse Based Framework (PTP)

Conventional Tools

ISI

Distributed MPI Analyzer

PUG

? Inspect

MCA API verifiers

MPI

CUDA / OpenCL

OpenMP

Pthreads

Multicore Association APIs

High End Machines for HPC / Cloud

Desktop Servers and Compute Servers

Embedded Systems and Devices
What is Exp. Succinct?
What is Exp. Succinct?

• It perhaps started with Indians...
• They invented NOTHING!
What is Exp. Succinct?

• It perhaps started with Indians...
• They invented NOTHING!
• Yes, NOTHING, or Zero
  – Positional Number System born
  – Exponentially succinct!
What is Exp. Succinct?

• It perhaps started with Indians...
• They invented NOTHING!
• Yes, NOTHING, or Zero
  – Positional Number System born
  – Exponentially succinct!

• Example: Knuth’s paper on paths in a grid
  – Can you write the number of paths on a grid from [0,0] to [N,N] within the rectangle [0,0], [N,N]?
    • In Unary?
    • In Decimal / Binary?
What is Exp. Succinct?

• It perhaps started with Indians...
• They invented NOTHING!
• Yes, NOTHING, or Zero
  – Positional Number System born
  – Exponentially succinct!

• Example: Knuth’s paper on paths in a grid
  – Can you write the number of paths on a grid from [0,0] to [N,N] within the rectangle [0,0] , [N,N] ?
    • In Unary?
    • In Decimal / Binary?

• Conquering Verification Complexity:
  – Use Exp. Succinct representations / searches!
What is Exp. Succinct?

• It perhaps started with Indians...
• They invented NOTHING!
• Yes, NOTHING, or Zero
  – Positional Number System born
  – Exponentially succinct!
• Knuth’s paper on paths in a grid
  – Can you write the number of paths on a grid from [0,0] to [N,N] within the rectangle [0,0], [N,N]?
    • In Unary?
    • In Decimal / Binary?

• Conquering Verification Complexity often requires the use of Exp Succinct representations / searches