Correctness
via
Compilation to Logic
and
Automated Theorem Provers

Microsoft Research
July 2014
Reduction to Logic

In **theory**, **all** problems of program correctness can be reduced to problems of logic.
Reduction to Logic

Is execution path $P$ feasible?

Testing

Is assertion $X$ violated?

Verification

Is Formula $F$ Satisfiable?
public static void Puzzle(int x) {
    int res = x;
    res = res + (res << 10);
    res = res ^ (res >> 6);
    if (x > 0 && res == x + 1)
        throw new Exception("bug");
}

(declare-const x (_ BitVec 32))
(assert (bvsgt x #x00000000))
(assert (= (bvadd x #x00000001)
    (bxor (bvadd x (bvshl x #x0000000A))
        (bvsahr (bvadd x (bvshl x #x0000000A)) #x00000006))))
(check-sat)
(get-model)

http://rise4fun.com/Z3/j1BB
Logic/Complexity Classes

Undecidable (FOL + LIA)

Semi Decidable (FOL)

NEXPTIME (EPR)

PSPACE (QBF)

NP (SAT)

Practical problems often have structure that can be exploited.

Algorithmic advances

Large-scale evaluation and careful engineering
In practice, many problems of program correctness can be compiled to problems of logic and solved by automated theorem provers.
Z3 reasons over a combination of theories

- Boolean Algebra
- Bit Vectors
- Linear Arithmetic
- Floating Point
- First-order Axiomizations
- Sets/Maps/...
- Algebraic Data Types
- Non-linear, Reals
Results and Contributions

Algorithms
Decidable Fragments
Data structures
Heuristics

Won 19/21 divisions in SMT 2011 Competition

The most influential tool paper in the first 20 years of TACAS (2014)
Program Correctness via Compilation to Logic

Automated Test Generation

Automated Safety/Termination Checkers

Interactive Functional Verification

Z3
SAGE: Binary File Fuzzing
- Symbolic execution of x86 traces
- 1/3 of all file fuzzing security bugs during Windows 7
- Z3 theories: primarily bit vectors, also arrays
- Patrice Godefroid, David Molnar, Ella Bounimova

Pex: Parameterized Unit Testing
- Symbolic execution of .NET traces
- Many external users via VS; powers www.codehunt.com
- Z3 theories: automata, bit vectors, maps, algebraic data types, ...
- Nikolai Tillman, Peli de Halleux
Program Correctness via Compilation to Logic

- Automated Test Generation
- Automated Safety/Termination Checkers
- Interactive Functional Verification

Boogie IR/Verifier

Z3
SymDiff

- Modular comparison of procedures for behavioral differences
- Used to test many versions of .NET JIT
- Implemented at Boogie level, supports C/C++, x86, ...
- Shuvendu Lahiri, Chris Hawblitzel

Corral

- Whole program analysis engine based on stratified inlining
- Powers Static Driver Verifier
- Implemented at Boogie level, supports C/C++ and .NET
- Akask Lal (MSRI), Shaz Qadeer
Interactive Functional Verification

**Dafny**
- Object-oriented language with specification language and verifier
- Extensive use in education and MS
- Implemented using **Boogie**
- Rustan Leino, Michal Moskal

**F***
- ML-like functional language with powerful type system and verifier
- Certified TLS implementation
- Combines type checking with Z3
- N. Swamy, C. Fournet (MSRC), + MSR-INRIA, INRIA and IMDEA colleagues

http://www.rise4fun.com/dafny/tutorial/  
http://www.rise4fun.com/fstar/tutorial/
Correctness via Compilation to Logic

• F*
  – Formalize programming language semantics via logic

• Z3: The Next Generation
  – Program analysis to logic

• Network verification
  – Eliminate datacenter misconfiguration errors
BUGS ACROSS THE BOARD!  Arguably, all are language design failures

Heartbleed OpenSSL
Internet Explorer 1776
...

Facebook API OAuth
OWASP CSRFGuard
...

• Buffer overrun
• Use-after-free

• Dynamic type-safety violation
• Dynamic type-safety violation

ACM Software Systems Award 2013
The most advanced and robust program logic in the world
Widely recognized as the gold standard for reliability in PL academic circles
Proofs of 4-color theorem, Feit-Thompson theorem, CompCert C compiler, ...

But, even Coq is flawed: 2 soundness bugs in termination checker in the last 6 months!

• A flaw in the logic lurking for the past 15+ years
SECURING THE SPECTRUM OF PROGRAMMING LANGUAGES WITH F*
F*: a semantic framework in which to
• model,
• implement,
• and certify software

across the spectrum of programming languages

http://research.microsoft.com/fstar
http://rise4fun.com/fstar/

Developed collaboratively by RiSE (Redmond), PPT (Cambridge) and MSR-INRIA since 2008
F* source resembles F#, but with richer specifications via types

Uses an SMT solver to automatically prove user-provided specifications

Multiple backends for cross-platform support

val counter: unit -> Writer x:int{x >= 0}
let counter = let c = ref 0 in
  fun () -> c := !c + 1; !c
THE 1\textsuperscript{ST} \textit{(SELF-)}CERTIFIED PROGRAM VERIFIER

• Why trust a program verifier?
• In 2012, we programmed the F* verifier in F*, and proved it correct
• Bootstrapped the correctness proof using Coq (avoid the termination bug)
• Involved checking the largest known Coq proof
  • 8GB proof
  • verified by Coq in 24 machine-days
A PERFECTLY SECURE COMPILER FROM F* TO JS

• Increasingly, JavaScript is the target for many compilers
• But, the semantics of JS is wildly different from these languages

In 2013, beyond JavaScript:
• Developed a formal semantics of JavaScript within F*
• Proved a compiler from F* to JavaScript "fully abstract"
  • Full abstraction is the semantically perfect property for a translation
A VERIFIED IMPLEMENTATION OF TLS

miTLS-1.0:
• A full reference implementation of TLS (SSL) implemented in F7 (a subset of F*)
• A proof of its security: TLS establishes a secure channel between its endpoints
• But, performance overhead of 10x

miTLS-2.0: Currently underway
• A high-performance variant, using verified low-level memory management
  • Beyond C++: using F* for safe, performant low-level code
• Goal: A drop-in replacement for OpenSSL with certified security
Impacts theory:

- Robust logics and tools for program verification and mathematical proofs

And practice, through certified security for

- Key elements of internet infrastructure (TLS) and
- Web-programming (JavaScript)
Z3: The Next Generation

Interpolants
Objective Functions

New Queries

New Engines

Z3 as Cloud Service

https://www.github.com/leodemoura/lean/

Leonardo de Moura, Nikolaj Bjorner, Christoph Wintersteiger, Ken McMillan, Margus Veanes, Andrey Rybalchenko

Arithmetic, Bit-Vectors, Booleans, Arrays, Datatypes, Quantifiers

Symbolic automata Horn solvers

New Deployment
Interactive Theorem Prover
Horn Clause Satisfiability Modulo Theories

(Program Analysis to Logic)

Ken McMillan, Nikolaj Bjørner, Andrey Rybalchenko
Program Analysis Architecture

Many queries
Inefficient interface

Different analyzers interpret IR differently
The Logic Alternative

- Program
  - Transform
    - IR
      - Interpret/Analyze
        - Solve
  - Transform
    - IR
      - VC Gen
        - Translate IR to logic (one semantic interpretation)
      - Logic
        - Solve A
          - Solve B
            - Easy to share information
Program Analysis as Higher-order Verification

Verification

• Requires decorating a program with auxiliary assertions, such as
  – Loop invariants
  – Procedure summaries

• VC generation of decorated program yields
  – first-order logic formulae

• Leaving the auxiliary assertions as unknowns in VC generation yields
  – second-order logic formulae

• Automatically discovering sufficient auxiliary assertions is the problem of Program Analysis
Example

var $i : int := 0$;
while $i < N$ invariant $R(i)$ do
  $i := i + 2$;
done
assert $i \neq 41$;

• The verification conditions are:
  
  $(i = 0) \Rightarrow R(i)$  
  invariant holds on entry

  $R(i) \land (i < N) \Rightarrow R(i + 2)$  
  loop preserves invariant

  $R(i) \land \neg(i < N) \Rightarrow (i \neq 41)$  
  assertion holds on loop exit

• To analyze the program, we solve for $R$:

  $R(i) \iff (i \mod 2) = 0$
Horn Solvers

• Apply to various program analysis scenarios
  – Interprocedural analysis
  – Concurrent programs

• Use standard logics and formats
  – Many applications

• Now implemented in Z3 using
  – Property-driven reachability (PDR)
  – Interpolation methods (Duality)
Performance

- **Corral** is currently the most effective analyzer in SDV, using stratified inlining and many calls to Z3.

- **Duality** is an interpolation-based Horn solver, integrated inside Z3.

- On this hard example, Duality dominates Corral in performance.

<table>
<thead>
<tr>
<th></th>
<th>Timeouts</th>
<th>Wins</th>
<th>Defects</th>
<th>Warnings</th>
<th>Passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corral</td>
<td>38</td>
<td>1</td>
<td>28</td>
<td>8</td>
<td>49</td>
</tr>
<tr>
<td>Duality</td>
<td>14</td>
<td>23</td>
<td>29</td>
<td>9</td>
<td>69</td>
</tr>
</tbody>
</table>
Advantages of Reduction to Logic

1) Separation of concerns
2) Simplified tools
3) Allows interoperation
4) Established standards

Program

Transform

IR

VC Gen

Logic

Solve

interpreting program semantics

SMTLIB

common language and model

process logic not IR
Network Verification

Symbolic Analysis via Z3
SecGuru Validation Tool for Network ACLs
Flow Analysis Solver for Network Reachability

Program Verification Networks Useful SDN Tools

Tony Hoare Vincent Cerf
SecGuru: Validating Network Connectivity Restrictions

Karthick Jayaraman, Charlie Kaufman

Nikolaj Bjørner
Network Policies: Complexity, Challenge and Opportunity

Several devices, vendors, formats
- Net filters
- Firewalls
- Routers

Challenge in the field
- Do devices enforce policy?
- Ripple effect of policy changes

Arcane
- Low-level configuration files
- Mostly manual effort

Human Errors by Activity
- 1: 74%
- 2: 13%
- 3: 13%

Pie chart showing the distribution of human errors by activity.
A Data-center Architecture
SecGuru Workflow

Windows Azure Network Monitoring Infrastructure
## Contracts/Policies

<table>
<thead>
<tr>
<th>Status</th>
<th>Source Address</th>
<th>Src Port</th>
<th>Destination Address</th>
<th>Dst Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permit</td>
<td>10.20.0.0/19</td>
<td>Any</td>
<td>157.55.252.0/30</td>
<td>Any</td>
<td>6</td>
</tr>
<tr>
<td>Deny</td>
<td>Any</td>
<td>Any</td>
<td>65.52.244.0/27</td>
<td>Any</td>
<td>4</td>
</tr>
</tbody>
</table>
Policies as Logical Formulas

Traditional Low level of Configuration network managers use

Precise Semantics as formulas

Allow: \((10.20.0.0 \leq srcIp 10.20.31.255) \land (157.55.252.0 \leq dstIp \leq 157.55.252.255) \land (protocol = 6)\)

Deny: \((65.52.244.0 \leq dstIp \leq 65.52.247.255) \land (protocol = 4)\)

Combining semantics

\(\bigvee_{i} \text{Allow}_i \land \bigwedge_{j} \neg \text{Deny}_j\)

Contracts/Policies
Policies as Logical Formulas

Traditional Low level of Configuration network managers use

Contracts/ Policies

Precise Semantics as formulas

(10.20.0.0 ≤ srcIp 10.20.31.255) ∧
Allow: (157.55.252.0 ≤ dstIp ≤ 157.55.252.255) ∧
(protocol = 6)

Deny: (65.52.244.0 ≤ dstIp ≤ 65.52.247.255) ∧
(protocol = 4)

Combining semantics

Policy_0 = false
Policy_{i+1} = if IsAllow_i then Rule_i ∨ Policy_i
else ¬Rule_i ∧ Policy_i
Policy ≡ Policy_{N+1}
Beyond Z3: a new idea to go from one violation to all violations

\[ (\bigvee_i \text{Allow}_i) \land (\bigwedge_j \neg \text{Deny}_j) \rightarrow \neg \left[ (\bigvee_m \text{Allow}_m) \land (\bigwedge_n \neg \text{Deny}_n) \right] \]

Semantic Diffs

\[ \text{srcIp} = 10.20.0.0/16, 10.22.0.0/16 \]
\[ \text{dstIp} = 157.55.252.000/24, 157.56.252.000/24 \]
\[ \text{port} = 80,443 \]

Representing solutions
- \( 2 \times 2^{16} \times 2 \times 2^8 \times 2 = 2^{27} \) single solutions, or
- 8 products of contiguous ranges, or
- A single product of ranges

SecGuru contains optimized algorithm for turning single solutions into all (product of ranges)
# Other Domains Powered by Z3

<table>
<thead>
<tr>
<th>Domain</th>
<th>Title</th>
<th>Conference</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog circuit design</td>
<td>Verifying global convergence for a digital phase-locked loop</td>
<td>FMCAD 2013</td>
<td>UBC</td>
</tr>
<tr>
<td>Model-based testing of OS performance</td>
<td>Designing scalable software for multicore processors</td>
<td>SOSP 2014</td>
<td>MIT</td>
</tr>
<tr>
<td>Program optimization</td>
<td>Consolidation of Queries with User-Defined Functions</td>
<td>PLDI 2014</td>
<td>UT Austin, U. Oxford, MSR</td>
</tr>
<tr>
<td>Computational biology</td>
<td>Analyzing and synthesizing genomic logic functions</td>
<td>CAV 2014</td>
<td>U. Oxford, MSR</td>
</tr>
</tbody>
</table>
Correctness via
Compilation to Logic and
Automated Theorem Provers

Automated Test Generation
Automated Safety/Termination Checkers
Interactive Functional Verification

New Queries
New Engines

Z3