Welcome to the 2019 DeepSpec Workshop!
The Science of Deep Specification

Benjamin C. Pierce
University of Pennsylvania

DeepSpec Workshop @ PLDI
June, 2019
“We can’t build software that works!”
Or...?
How did that happen?
• Better **programming languages**
  • Powerful mechanisms for *abstraction* and *modularity*

• Better **software development methodology**
  • Agile workflows, unit testing, …

• Stable **platforms and frameworks**
  • Posix, Win32, Android, iOS, apache, DOM/JS, …
Are we done?

Nope
What about secure software?
Grounds for hope...

- Better **programming languages**
  - Basic safety guarantees built in
- Better **understanding of risks** and vulnerabilities
- Better **system architectures for security**
  - Separation kernels, hypervisors, sandboxing, TPMs, …
- Success stories of formal specification and machine-checked verification of critical software at scale
  - Not a panacea (side channels, etc.)
  - But a big step in the right direction!
Fixpoint compile (e : exp) : list instr :=
match e with
| Num n => [PUSH n]
| Plus e1 e2 => compile e1 ++ compile e2 ++ [PLUS]
| Minus e1 e2 => compile e1 ++ compile e2 ++ [MINUS]
| Mult e1 e2 => compile e1 ++ compile e2 ++ [MULT]
end.

Example e3 :
assert (compiles_correctly (Plus (Num 2) (Num 2))).

Example e4 :
assert (compiles_correctly (Plus (Num 5) (Num 3))).

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. …
Are logical specifications practical?
- Accepts most of ISO C 99
- Produces machine code for PowerPC, ARM, x86 (32-bit), and RISC-V architectures
- 90% of the performance of GCC (v4, opt. level 1)
Verification really works!

Regehr’s Csmith project used random testing to assess all popular C compilers, and reported:

“The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.”
- Real-world operating-system kernel
- With an end-to-end proof of implementation correctness and security enforcement
- Verified down to machine code
And many, many more!

- Bedrock system
- Ur/Web compiler
- CompCert TSO compiler
- CompCert static analysis tools
- Jitk and Data6 verified filesystems
- Fscq file system from MIT
- Verdi distributed system framework
- Testable formal spec for AutoSAR
- CakeML compiler
- Vellvm: Verified LLVM optimizations

- IronClad Apps
- Full-scale formal specifications of critical system interfaces
  - X86 instruction set
  - TCP protocol suite
  - Posix file system interface
  - Weak memory consistency models for x86, ARM, PowerPC
  - ISO C / C++ concurrency
  - Elf loader format
  - C language (Cerberus – also see Krebbers, K semantics, …)
Verified Textbooks!

SoftwareFoundations.org

... and several others!
Why now?

Urgent need for increased confidence

+ Diminishing value of “paper proofs”

+ Progress on enabling technologies
Enabling Technologies

- **Logics**
  - Concurrent separation logic, ...

- **Proof assistants**
  - Coq, Isabelle, ACL2, Twelf, HOL-light, ...

- **Testing tools and methodologies**
  - QuickCheck, QuickChick, ...

- **DSLs for writing specifications**
  - OTT, Lem, Redex, ...

- **Languages with integrated specifications**
  - Dafny, Boogie, JML, F*, Liquid Types, Verilog PSL, Dependent Haskell, ...
Enabling Technologies
So are we done?

Nope.
Lessons from CompCert

- Shao
- Appel
- Leroy
- Sewell
Lessons from CompCert
Lessons from CompCert

- Shao
- Appel
- Sewell
- Leroy

- OS client interface
  - CertiKOS hypervisor kernel
  - C language
- Program Logic
  - Verifiable C System
  - C language
- C language
- CompCert Compiler
- PowerPC ISA
- PowerPC ISA
- IBM's CPU
- Transistors

IBM's CPU

CompCert Compiler

C language

CertiKOS hypervisor kernel

PowerPC ISA

IBM's CPU

Transistors

Lessons from CompCert
Lessons from seL4

• Original specification and correctness proof for seL4 kernel took ~20 person years

• Later, the same team added a tool for setting up secure system configurations
  • where processes at different security levels were guaranteed not to interfere

• Proving correctness of this tool took ~4 person years, of which 1.5 years were devoted to upgrading the kernel specification (and proof) to eliminate unwanted nondeterminism
Two-sided specifications

Verified components must connect at specification boundaries
“Deep” specifications:

- **Formal**: mathematically rigorous
- **Rich**: precisely expressing intended behavior of complex software
- **Live**: automatically checked against actual code (not just a model)
- **Two-sided**: exercised by both “implementors” and “clients”
The Science of Deep Specification

Andrew Appel
Princeton

Lennart Beringer
Princeton

Adam Chlipala
MIT

Yours truly
University of Pennsylvania

Zhong Shao
Yale

Stephanie Weirich
University of Pennsylvania

Steve Zdancewic
University of Pennsylvania
And more importantly…

Andres Erbsen
Antal Spector-Zabusky
Antoine Voizard
Benjamin Sherman
Christine Rizkallah
David Costanzo
David Kaloper Meršinjak
Dmitri Garbuzov
Hernán Vanzetto
Jade Philipoom
Jason Gross
Ji-Yong Shin
Jieung Kim
Joachim Breitner
Joonwon Choi
Joshua Lockerman
Jérémie Koenig

Leonidas Lampropoulos
Li-yao Xia
Lionel Rieg
Lucas Paul
Matthew Weaver
Mengqi Liu
Mirai Ikebuchi
Murali Vijayaraghavan
Nick Giannarakis
Olivier Savary Belanger
Pedro Henrique Avezedo de Amorim
Paul He
Pierre Wilke
Qinxiang Cao
Quentin Carbonneaux

Richard Zhang
Ronghui Gu
Samuel Gruetter
Santiago Cuellar
Unsung Lee
Vilhelm Sjöberg
William Mansky
Wolf Honore
Xiongnan (Newman) Wu
Yao Li
Yishuai Li
Yuanfeng Peng
Yuting Wang
Zoe Paraskevopoulou
Goal:

Move from point success stories to sustainable engineering practice at industrially relevant scale
Verified Software Toolchain

Program logic for proving correctness of (concurrent) C programs

Proof automation tools for applying the program logic

Demo projects:
crypto primitives,
“mailbox” communication system,
garbage collector for Certicoq
B+ Trees DBMS
web server
“Certified Abstraction Layers”

a new refinement-based methodology for software correctness proofs

... of programs with low-level concerns such as interrupts, virtual-memory mapping, scheduling, ...
LLVM compiler intermediate language

Vellvm: Formal specification of LLVM; proofs of correctness of LLVM compiler phases

Demo project:

Use as basis for testing correctness of GHC, using QuickChick

Steve Zdancewic
Haskell: widely used pure functional programming language with lazy evaluation

Haskell Core: near-source-level intermediate language inside GHC compiler

Haskell Core Spec:
Formal specification of semantics of the Haskell core language

Demo projects:
Prove correctness of some GHC phases using hs-to-coq

Use as basis for testing correctness of GHC, using QuickChick
Gallina: functional programming language inside Coq

“Extraction:” Translate Gallina to ML, compile with Ocaml compiler

Extraction is quite good, but it’s not verified correct

Verified Compiler for Coq programs

CertiCoq:
A verified compiler for Gallina

Write your software as a pure functional program in Coq, prove its correctness using Coq, use CertiCoq to compile to efficient machine code

Demo projects:
Resolution theorem prover for Separation Logic
(?) CompCert
(?) database query optimization
(?) parts of web server
Verified processor design

Old way:
- Write reference manual for ISA
- Write RTL program in VHDL
- Compile VHDL into transistors

New way:
- Formal specification of ISA
- Write RTL program in Bluespec
- Compile Bluespec into VHDL
- Compile VHDL into transistors

Use existing tools to verify correctness

Demo project:
specification / verification of RISC-V processor implementation

Adam Chlipala
Old way:
Fuzz testing

Recent ways:
Semantic fuzz testing

Tool: QuickCheck, for Haskell and Erlang; fuzzes over (tree) data structures, automatically reduces bugs found into minimal input cases

QuickChick:
Semantic fuzz testing based on conformance to formal specification in Coq

Demo projects:
Apache web server
DeepSpec web server
Haskell compiler
Verification of cryptographic primitives

Demo projects: these crypto applications serve as demo projects for several of our other tools:

High-level cryptographic specs ("pseudorandom function, cryptographic advantage"),
Message authentication,
Random number generation

High-level functional specs (elliptic curves in finite fields)

Low-level functional specs (multibit carry)

Efficient imperative implementations
Verified Software Toolchain

Program logic for proving correctness of (concurrent) C programs

Proof automation tools for applying the program logic

Demo projects:
- crypto primitives,
- “mailbox” communication system
- garbage collector for Certicoq
- B+ Trees DBMS
- web server
Goal: Rich, formal, live, 2-sided specs
Application demo?
Application demos!

Web Server
Operating system
C programming
File system
Crypto
Self-driving car
C Programming
Hypervisor
DeepWeb

A web server built on DeepSpec
Many parts

One whole
Challenges

• Extreme vertical integration
  • Make progress through a sequence of “integration experiments”

• Multiple levels and styles of specifications
  • Need a “lingua franca” for writing a variety of specs
  • → Interaction trees

• Combining testing and verification

• Reasoning about server behavior “modulo the network”
Challenge: Vertical Integration
Goal: A “single QED” encompassing the whole stack

Executable high-level specification of HTTP(S) protocols and web services
System call interface specification
Instruction-set specification
RTL-level description of circuit behaviors
Executable high-level specification of HTTP(S) protocols and web services

Functional program with same observable behavior as C web server

System call interface specification (separation logic Hoare triples)

System call interface specification (CertiKOS “layer interface”)

Instruction-set specification (assembly level, structured memory model)

Instruction-set specification (machine-code level, flat memory model)

RTL-level description of circuit behaviors
Challenge:
Disparate Specification Styles
Too many metalanguages!

• Network-level HTTP spec
  • Nondeterministic “model implementation” (functional program)
  • Client-side acceptance tester (functional program)

• Web server implementation
  • CompCert “observation traces”

• VST C verification tool
  • Hoare triples in separation logic

• CertiKOS
  • “Layer interfaces”
Interaction trees
Reasoning
“Modulo the Network”
Swap server specification

Server

Bat
Cat
Dog
Elk

Client 1

Cat
Bat

Client 2

Dog

Cat

Elk

Client 3

Dog
Swap server: in the real world

- Messages on different connections can be reordered
- Messages can be delayed indefinitely
Network refinement

- Specification

\( \text{network-refines} \)

Implementation

Network semantics

Observable behavior by clients

U1

Observable behavior by clients

Adaptation of Observational refinement/Linearizability
Challenge:
Testable High-Level Specifications
Because (1) we want to test our C code and (2) the tester also needs to work with stock web servers.

**What we have**

- Specification
- Implementation

**network-refines**

**Network semantics**

**Observable behavior by clients**

**Where we have to stand for testing**
Main challenge: nondeterminism
• introduced by the network
• … or present in the original spec
Final theorem

• “If you put these bits (produced by compiling CertiKOS and the web server using CompCert) into a memory connected to this connection of transistors (produced by compiling a RISC-V implementation using Kami), the behaviors of the resulting system will network refine the behaviors described by the model implementation.”
Progress

• Vertical integration
  • See CPP 2019 paper about testing and VST verification of a “swap server”

• Interaction trees
  • See talks by Steve Zdancewic today and by Yann Régis-Giannis and Gil Hur tomorrow

• Connecting VST and CertiKOS
  • See talk by William Mansky today

• Connecting CertiKOS and Risc-V
  • Ongoing work at Yale and MIT on a “flat memory” semantics for CompCert
The demo is not the (only) scientific result!

DeepSpec is not “build a verified stack”
DeepSpec is . . .

a coherent collection of tools and techniques . . .

…that can be connected, combined, and configured to allow users to build and foundationally verify high assurance, functionally correct software and hardware.
DeeoSpec Workshop Overview
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 - 09:45</td>
<td>Welcome and overview</td>
</tr>
<tr>
<td>09:45 - 10:30</td>
<td>Compiler Verification</td>
</tr>
<tr>
<td>11:00 - 11:30</td>
<td>Modular Reasoning</td>
</tr>
<tr>
<td>11:30 - 12:00</td>
<td>Interaction Trees and Algebraic Effects I</td>
</tr>
<tr>
<td>12:00 - 12:30</td>
<td>- Overview of the DeepSpec Expedition and its Capstone Application</td>
</tr>
<tr>
<td></td>
<td>Benjamirn C. Pierce University of Pennsylvania</td>
</tr>
<tr>
<td>11:00 - 11:30</td>
<td>- Closure Conversion is Safe for Space</td>
</tr>
<tr>
<td></td>
<td>Zoe Paraskevopoulou Princeton University, Andrew Appel Princeton</td>
</tr>
<tr>
<td>11:30 - 12:00</td>
<td>- Fast, Verified Partial Evaluation</td>
</tr>
<tr>
<td></td>
<td>Adam Chlipala Massachusetts Institute of Technology, USA</td>
</tr>
<tr>
<td>12:00 - 12:30</td>
<td>- Stack-Aware CompCert</td>
</tr>
<tr>
<td></td>
<td>Yuling Wang Yale University</td>
</tr>
<tr>
<td>14:00 - 14:30</td>
<td>- Abstraction, Subsumption, and Linking in VST</td>
</tr>
<tr>
<td></td>
<td>Lennart Beringer Princeton University, Andrew Appel Princeton</td>
</tr>
<tr>
<td>14:30 - 15:00</td>
<td>- Compositional Verification of Preemptive OS Kernels with Temporal and</td>
</tr>
<tr>
<td></td>
<td>Spatial Isolation</td>
</tr>
<tr>
<td></td>
<td>Mengqi Liu Yale University</td>
</tr>
<tr>
<td>15:00 - 15:30</td>
<td>- Modular Correctness Proofs at the Hardware-Software Interface</td>
</tr>
<tr>
<td></td>
<td>Joonwon Choi Massachusetts Institute of Technology, USA</td>
</tr>
<tr>
<td>16:00 - 16:20</td>
<td>- Interaction Trees: Representing Recursive and Impure Programs in Coq</td>
</tr>
<tr>
<td></td>
<td>Steve Zdancewic University of Pennsylvania</td>
</tr>
<tr>
<td>16:20 - 16:45</td>
<td>- Connecting Separation Logic with First-Order Reasoning on Memory</td>
</tr>
<tr>
<td></td>
<td>William Mansky University of Illinois at Chicago</td>
</tr>
<tr>
<td>16:45 - 17:30</td>
<td>- Typed Programming with Algebraic Effects (in terms of ambient values,</td>
</tr>
<tr>
<td></td>
<td>functions, and control</td>
</tr>
<tr>
<td></td>
<td>Daan Leijen Microsoft Research, USA</td>
</tr>
</tbody>
</table>
Implementation and Verification of Modular Effectful Systems in Coq using FreeSpec
Yann Régis-Gianas IRIF, University Paris Diderot and CNRS, France / INRIA PLR2

Names, Places, and Things: Generic Traversals over Generic Syntax with Binding
James McKinna University of Edinburgh

Development of the RISC-V ISA Formal Specification
Rishiyur Nikhil

Automated Formal Memory Consistency Verification of Hardware
Yatin Manerkar Princeton University

Project Oak: Control Data in Distributed Systems, Verify All The Things
Ben Laurie Google Research

Refinement-Based Game Semantics for CompCert
Jérémie Koenig Yale University

Coinductive Reasoning about Interaction Trees
Chung-Kil Hur Seoul National University

Coverage Guided, Property Based Testing
Leonidas Lampropoulos University of Pennsylvania

Interaction Trees and Algebraic Effects II

Hardware / Software Interface Specifications

Verifying all the things

Coinduction and testing
Join us!

Teaching materials
Summer schools
PhD and postdoc positions

Technical workshops
(like this one :-)

visitors program

Visit deepspec.org to see what’s happening and join our mailing list

Thank you!
(any (more) questions?)