SOFTWARE FOUNDATIONS

VOLUME 2: PROGRAMMING LANGUAGE FOUNDATIONS

TABLE OF CONTENTS INDEX ROADMAP

POSTSCRIPT

Congratulations: We've made it to the end!

Looking Back

We've covered a lot of ground. Here's a quick review of the whole trajectory we've followed, starting at the beginning of *Logical Foundations*...

- *Functional programming*:
  - "declarative" programming style (recursion over persistent data structures, rather than looping over mutable arrays or pointer structures)
  - higher-order functions
  - polymorphism

- *Logic*, the mathematical basis for software engineering:

```
        logic                          calculus
--------------------   ~   ----------------------------
software engineering       mechanical/civil engineering
```

  - inductively defined sets and relations
  - inductive proofs
  - proof objects

- *Coq*, an industrial-strength proof assistant
  - functional core language
  - core tactics
  - automation

- *Foundations of programming languages*
  - notations and definitional techniques for precisely specifying
    - abstract syntax
    - operational semantics
      - big-step style

        - small-step style
      - type systems
    - program equivalence
    - Hoare logic
    - fundamental metatheory of type systems
      - progress and preservation
    - theory of subtyping

Looking Around

Large-scale applications of these core topics can be found everywhere, both in ongoing research projects and in real-world software systems. Here are a few recent examples involving formal, machine-checked verification of real-world software and hardware systems, to give a sense of what is being done today...

CompCert

CompCert is a fully verified optimizing compiler for almost all of the ISO C_{90} / ANSI C language, generating code for x_{86}, ARM, and PowerPC processors. The whole of CompCert is is written in Gallina and extracted to an efficient OCaml program using Coq's extraction facilities.

"The CompCert project investigates the formal verification of realistic compilers usable for critical embedded software. Such verified compilers come with a mathematical, machine-checked proof that the generated executable code behaves exactly as prescribed by the semantics of the source program. By ruling out the possibility of compiler-introduced bugs, verified compilers strengthen the guarantees that can be obtained by applying formal methods to source programs."

In 2011, CompCert was included in a landmark study on fuzz-testing a large number of real-world C compilers using the CSmith tool. The CSmith authors wrote:

- 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.

http://compcert.inria.fr

seL4

seL4 is a fully verified microkernel, considered to be the world's first OS kernel with an end-to-end proof of implementation correctness and security enforcement. It is implemented in C and ARM assembly and specified and verified using Isabelle. The code is available as open source.

"seL4 has been comprehensively formally verified: a rigorous process to prove mathematically that its executable code, as it runs on hardware, correctly implements the behaviour allowed by the specification, and no others. Furthermore, we have proved that the specification has the desired safety and security properties (integrity and confidentiality)... The verification was achieved at a cost that is significantly less than that of traditional high-assurance development approaches, while giving guarantees traditional approaches cannot provide."

https://sel4.systems.

CertiKOS

CertiKOS is a clean-slate, fully verified hypervisor, written in CompCert C and verified in Coq.

"The CertiKOS project aims to develop a novel and practical programming infrastructure for constructing large-scale certified system software. By combining recent advances in programming languages, operating systems, and formal methods, we hope to attack the following research questions: (1) what OS kernel structure can offer the best support for extensibility, security, and resilience? (2) which semantic models and program logics can best capture these abstractions? (3) what are the right programming languages and environments for developing such certified kernels? and (4) how to build automation facilities to make certified software development really scale?"

http://flint.cs.yale.edu/certikos/

Ironclad

Ironclad Apps is a collection of fully verified web applications, including a "notary" for securely signing statements, a password hasher, a multi-user trusted counter, and a differentially-private database.

The system is coded in the verification-oriented programming language Dafny and verified using Boogie, a verification tool based on Hoare logic.

"An Ironclad App lets a user securely transmit her data to a remote machine with the guarantee that every instruction executed on that machine adheres to a formal abstract specification of the app’s behavior. This does more than eliminate implementation vulnerabilities such as buffer overflows, parsing errors, or data leaks; it tells the user exactly how the app will behave at all times. We provide these guarantees via complete, low-level software verification. We then use cryptography and secure hardware to enable secure channels from the verified software to remote users."

https://github.com/Microsoft/Ironclad/tree/master/ironclad-apps

Verdi

Verdi is a framework for implementing and formally verifying distributed systems.

"Verdi supports several different fault models ranging from idealistic to realistic. Verdi's verified system transformers (VSTs) encapsulate common fault tolerance techniques. Developers can verify an application in an idealized fault model, and then apply a VST to obtain an application that is guaranteed to have analogous properties in a more adversarial environment. Verdi is developed using the Coq proof assistant, and systems are extracted to OCaml for execution. Verdi systems, including a fault-tolerant key-value store, achieve comparable performance to unverified counterparts."

http://verdi.uwplse.org

DeepSpec

The Science of Deep Specification is an NSF "Expedition" project (running from 2016 to 2020) that focuses on the specification and verification of full functional correctness of both software and hardware. It also sponsors workshops and summer schools.

- Website: http://deepspec.org/
- Overview presentations:
    - http://deepspec.org/about/
    - https://www.youtube.com/watch?v=IPNdsnRWBkk

REMS

REMS is a european project on Rigorous Engineering of Mainstream Systems. It has produced detailed formal specifications of a wide range of critical real-world interfaces, protocols, and APIs, including the C language, the ELF linker format, the ARM, Power, MIPS, CHERI, and RISC-V instruction sets, the weak memory models of ARM and Power processors, and POSIX filesystems.

"The project is focussed on lightweight rigorous methods: precise specification (post hoc and during design) and testing against specifications, with full verification only in some cases. The project emphasises building useful (and reusable) semantics and tools. We are building accurate full-scale mathematical models of some of the key computational abstractions (processor architectures, programming languages, concurrent OS interfaces, and network protocols), studying how this can be done, and investigating how such models can be used for new verification research and in new systems and programming language research. Supporting all this, we are also working on new specification tools and their foundations."

http://www.cl.cam.ac.uk/~pes20/rems/

Others

There's much more. Other projects worth checking out include:

- Vellvm (formal specification and verification of LLVM optimization passes)
- Zach Tatlock's formally certified browser
- Tobias Nipkow's formalization of most of Java
- The CakeML verified ML compiler
- Greg Morrisett's formal specification of the x_{86} instruction set and the RockSalt Software Fault Isolation tool (a better, faster, more secure version of Google's Native Client)
- Ur/Web, a programming language for verified web applications embedded in Coq
- the Princeton Verified Software Toolchain

Looking Forward

Some good places to learn more...

- This book includes several optional chapters covering topics that you may find useful. Take a look at the table of contents and the chapter dependency diagram to find them.

- More on Hoare logic and program verification
    - The Formal Semantics of Programming Languages: An Introduction, by Glynn Winskel [Winskel 1993].
    - Many practical verification tools, e.g. Microsoft's Boogie system, Java Extended Static Checking, etc.

- More on the foundations of programming languages:
    - Concrete Semantics with Isabelle/HOL, by Tobias Nipkow and Gerwin Klein [Nipkow 2014]
    - Types and Programming Languages, by Benjamin C. Pierce [Pierce 2002].
    - Practical Foundations for Programming Languages, by Robert Harper [Harper 2016].
    - Foundations for Programming Languages, by John C. Mitchell [Mitchell 1996].

- Iron Lambda (http://iron.ouroborus.net/) is a collection of Coq formalisations for functional languages of increasing complexity. It fills part of the gap between the end of the Software Foundations course and what appears in current research papers. The collection has at least Progress and Preservation theorems for a number of variants of STLC and the polymorphic lambda-calculus (System F).

- Finally, here are some of the main conferences on programming languages and formal verification:
    - Principles of Programming Langauges (POPL)
    - Programming Language Design and Implementation (PLDI)
    - International Conference on Functional Programming (ICFP)

- Computer Aided Verification (CAV)
- Interactive Theorem Proving (ITP)
- Certified Programs and Proofs (CPP)
- SPLASH/OOPSLA conferences
- Principles in Practice workshop (PiP)
- CoqPL workshop