CompCert
Most impressive formal verification effort to date

Atze van der Ploeg

Some slides stolen from: link

What is CompCert?

- A formally verified compiler from (a subset of) C to PowerPC assembly
- Programmed and proven (largely) in the Coq Systems
- Most impresive software verification effort to date (?)
- Authors: Xavier Leroy et al
- Disclaimer: I have nothing to do with this project :)

More info

Why CompCert?

- Most avionics, car, space embedded software written in C
- Need high-assurance: bugs can cost lives!
- Lots of effort spend getting certainty about this software *source code* (tests, formal methods, code review etc.)
- Then send to compiler
- What kind of certainty do we have about the resulting *machine code*?

Why CompCert?

To improve the quality of C compilers, we created Csmith, a randomized test-case generation tool, and spent three years using it to find compiler bugs. During this period we reported more than 325 previously unknown bugs to compiler developers. Every compiler we tested was found to crash and also to silently generate wrong code when presented with valid input.

X. Yang, Y. Chen, E. Eide & J. Regehr, PLDI 2011

What does a formally verified compiler mean?

$S \Downarrow B$

Relies on a formal specification of the C language!

Source program S (in C) can exhibit behavior B.

Behavior includes system calls with their input/results, errors, non-termination.

Example behavior:

1. Read 0 from terminal
2. Send network message 80234 to 238924
3. Read 55 from terminal
4. Crash

What does a formally verified compiler mean?

$T \Downarrow B$

Target program T (in assembly) can exhibit behavior B.

Relies on a formal specification of the assembly language!

The compiler is a mathematical function, *compile*, of type: C -> assembly

Mathematical statement, 1 st attempt:

A compiler is correct iff: $\forall\ b : Behavior, S \Downarrow B <-> compile(S) \Downarrow B$

What is a formally verified compiler?

Mathematical statement, 1 st attempt:

A compiler is correct iff: $\forall$ b : Behavior, S $\Downarrow$ B <-> compile(S) $\Downarrow$ B

Too strict! Compiler may makes choices, optimize errors away!

Mathematical statement:

A compiler is correct iff:

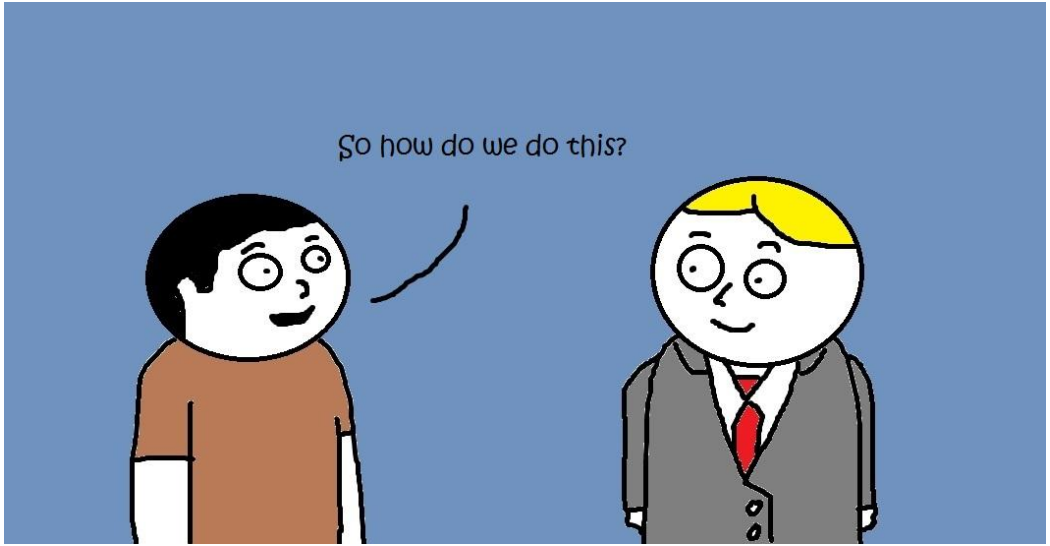
safe S $\rightarrow$ $\forall$ b : Behavior, compile(S) $\Downarrow$ B $\rightarrow$ S $\Downarrow$ B

Safe means program cannot crash

A formally verified compiler is:

- A computer program implementing the function compile
- A proof that: safe S → ∀ b : Behavior, compile(S) ⇓ B → S ⇓ B

How do we do that?

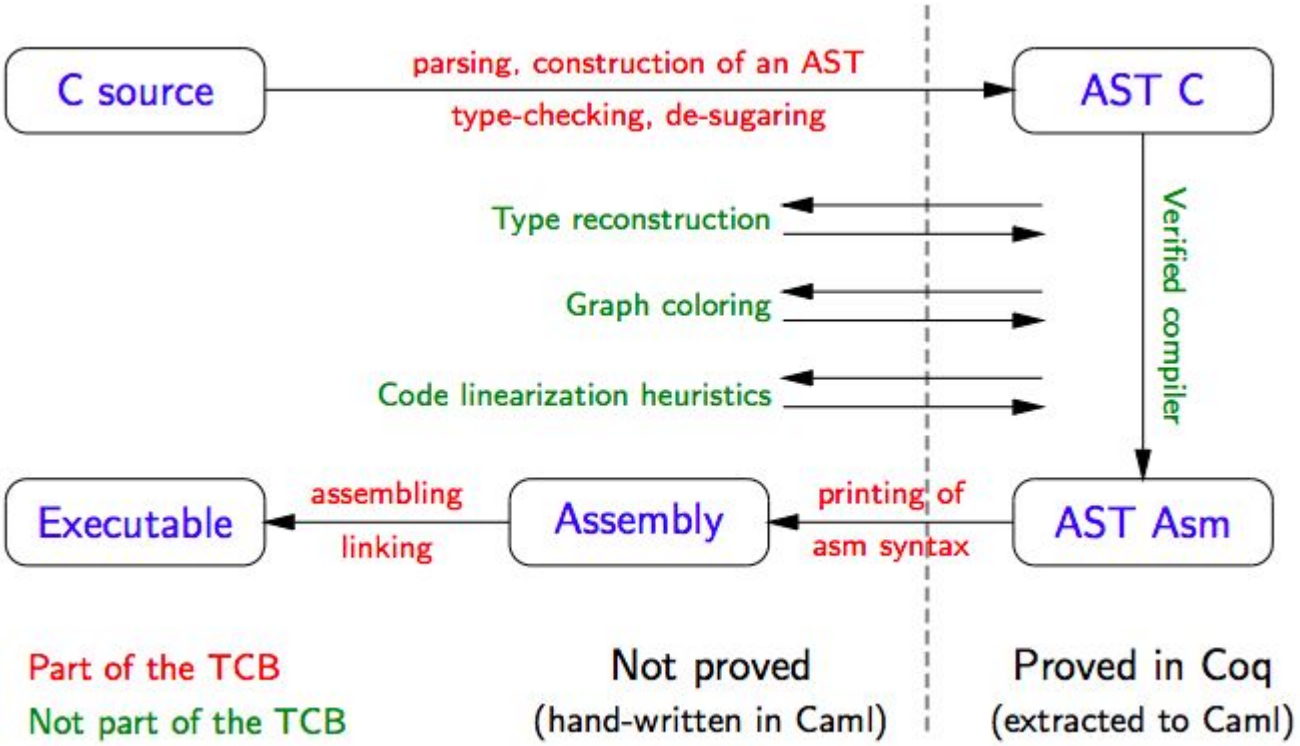


- Program compiler and prove the statement in Coq
- Extract OCaml code from Coq (erase proof)
- Compile OCaml code -> compiler

Does it work?

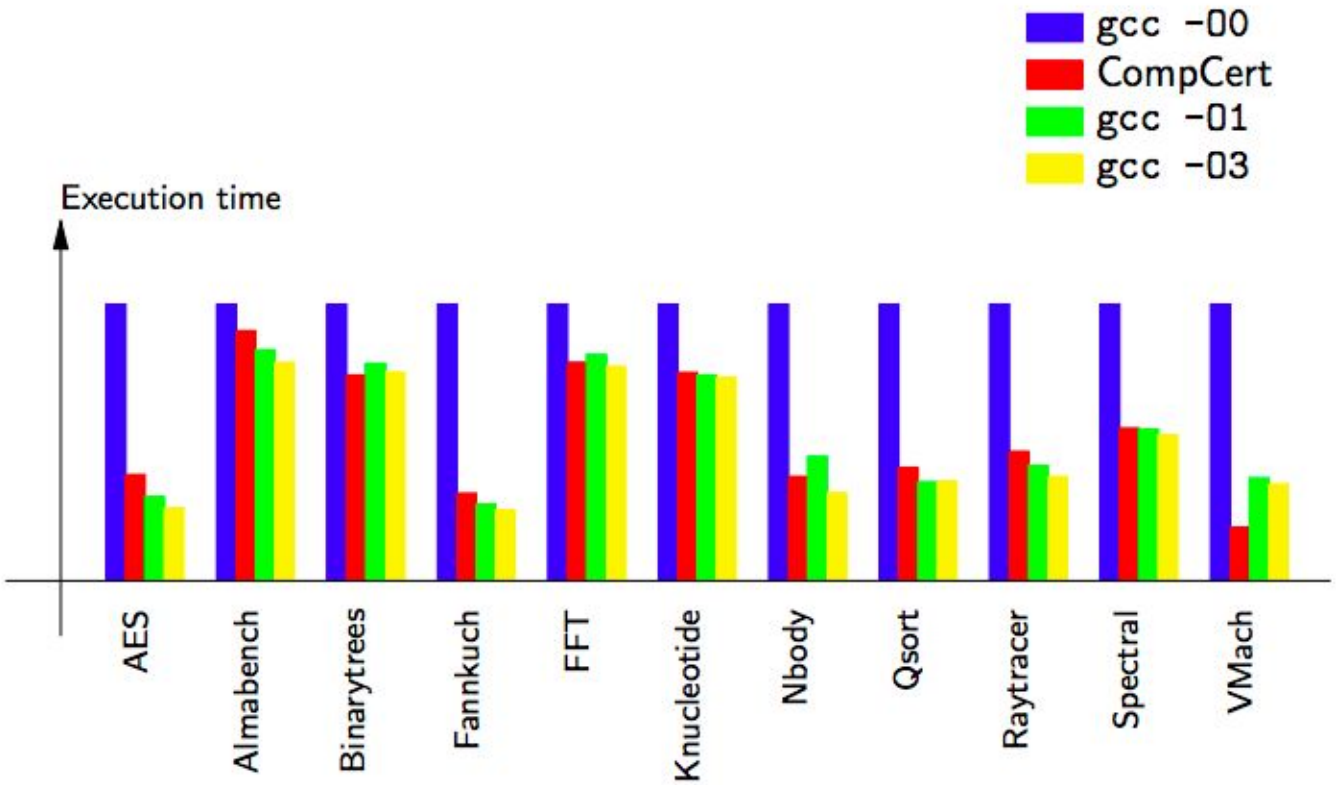
The striking thing about our CompCert results is that the middleend bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have 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. **(PLDI 2011)**

CompCert overview

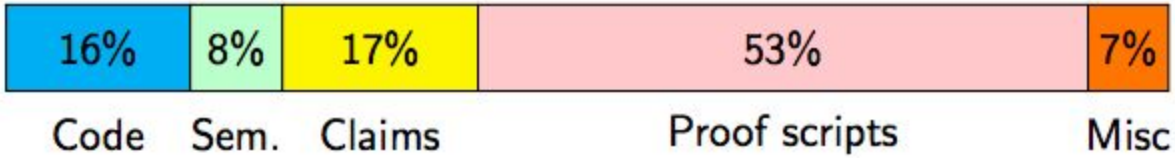


Performance of generated code

(On a PowerPC G5 processor)



Proof effort



50,000 lines of Coq.

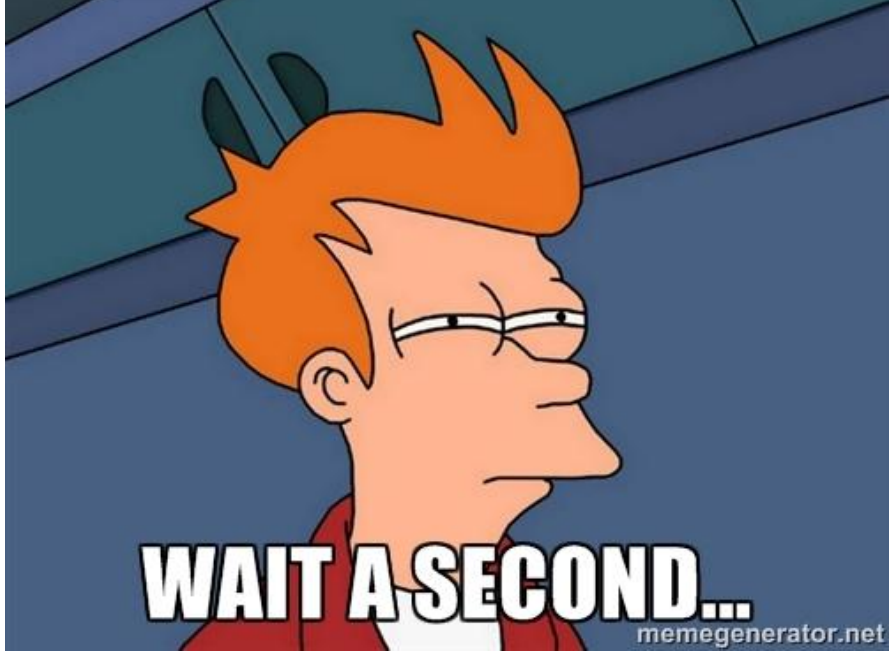
Including 8000 lines of "source code" ($\approx$ 40,000 lines of Java or C#).

4 person.years

Low proof automation (could be improved).

Still not verified

- Parsing & Printing
- Assembler
- OCaml compiler!
- Hardware(?)
- OS



But still... a lot more certainty than with GCC

CompCert

- Verified C compiler
- Relies on trusted assembler, OCaml compiler
- Seems much more bug-free than other compilers!