# How much is a mechanized proof worth, certification-wise?

Xavier Leroy

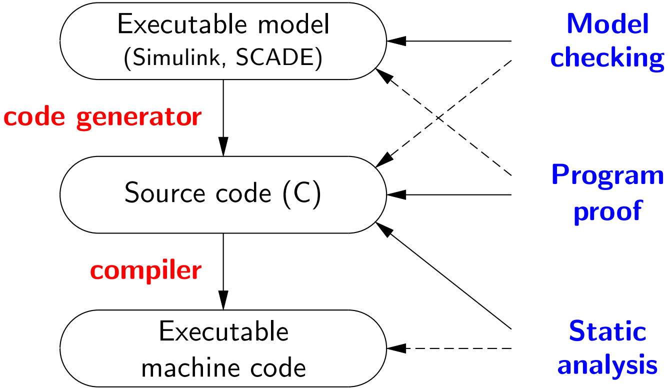
Inria Paris-Rocquencourt

PiP 2014: Principles in Practice

# In this talk. . .

Some feedback from the aircraft industry concerning the potential usefulness of the CompCert formally-verified C compiler.

# The initial plan




"Pitch" CompCert and its proof as a radical way to

- establish confidence in the C→asm compilation process;
- preserve guarantees obtained by C-level formal verification.

# Things not to say

## when pitching your research to the critical software industry

"It's obviously the right thing to do."

*(So you say. Have you ever built an airplane?)*

# Things not to say

## when pitching your research to the critical software industry

"It's obviously the right thing to do."

*(So you say. Have you ever built an airplane?)*

"The maths are beautiful."

*(We are into business, not aesthetics.)*

# Things not to say

## when pitching your research to the critical software industry

"It's obviously the right thing to do."

*(So you say. Have you ever built an airplane?)*

"The maths are beautiful."

*(We are into business, not aesthetics.)*

"You'll get stronger guarantees this way than by testing."

*(Our avionics software has no known flaws.)*
*(Besides, we perfected testing to an art.)*

# Things you might say

## when pitching your research to the critical software industry

"You'll get evidence that is complementary to testing."

*(Asymmetric redundancy is good!)*

# Things you might say

## when pitching your research to the critical software industry

"You'll get evidence that is complementary to testing."

*(Asymmetric redundancy is good!)*

"You could save money on testing."

*(Unit testing is costly, indeed.)*

# Things you might say

## when pitching your research to the critical software industry

"You'll get evidence that is complementary to testing."

*(Asymmetric redundancy is good!)*

"You could save money on testing."

*(Unit testing is costly, indeed.)*

"You could save time on re-testing after changes"

*(We must sometimes react quicky, indeed.)*

# Things you might say

## when pitching your research to the critical software industry

"You'll get evidence that is complementary to testing."

*(Asymmetric redundancy is good!)*

"You could save money on testing."

*(Unit testing is costly, indeed.)*

"You could save time on re-testing after changes"

*(We must sometimes react quicky, indeed.)*

"You could gain performance by using better algorithms
that you could not test well enough."

*(We have performance issues, indeed.)*

# Things you will hear

## when pitching your research to the critical software industry

*Maybe your stuff has some value.*
*But do you have a certification plan?*

Huh? It's proved in Coq!

*A formal verification is not a certification.*

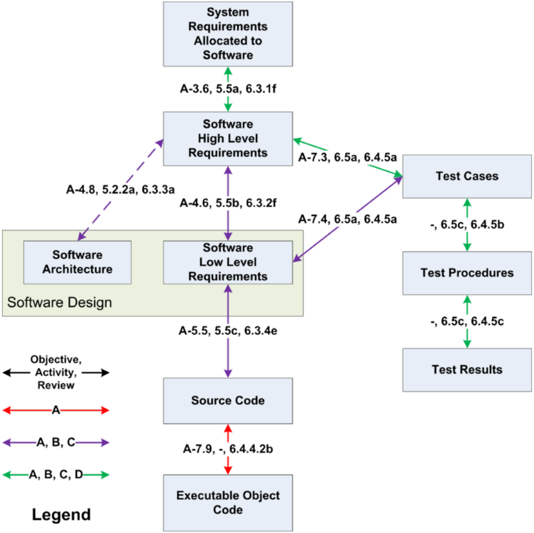
# Certification


Awarded by certification authorities (FAA, EASA) following domain-specific regulations, e.g. DO-178C for avionics (*Software Considerations in Airborne Systems and Equipment Certification*).

DO-178C specifies:

- several levels of assurance;
- the corresponding requirements to meet;
- the verification activities to conduct;
- but not any particular technology (prog. lang., tools, etc).

# DO-178C process and traceability



# Verification techniques

Three major kinds:

- Reviews (qualitative)
- Analyses (quantitative)
- Testing.

Analyses welcome maths & physics:

- High levels: aerodynamics, control theory.
- Low levels: use of software verification tools
  (e.g. static analyzers, deductive program provers).

But: tools must be qualified to get certification credit.

# Tool qualification (DO-330)

Purpose: obtain appropriate assurance that the tools are at least as dependable as the manual processes that they are replacing.

- Criteria 1: a tool whose output is part of the airborne software and thus could insert an error.
- Criteria 2: a tool that automates verification processes and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of other verification or development processes.
- Criteria 3: a tool that could fail to detect an error.

Determines how stringent the tool qualification is:

| | **Criteria** | | |
|---|---|---|---|
| **Software Level** | **1** | **2** | **3** |
| A | TQL-1 | TQL-4 | TQL-5 |
| B | TQL-2 | TQL-4 | TQL-5 |
| C | TQL-3 | TQL-5 | TQL-5 |
| D | TQL-4 | TQL-5 | TQL-5 |

# How much is CompCert's proof worth, certification/qualification-wise?

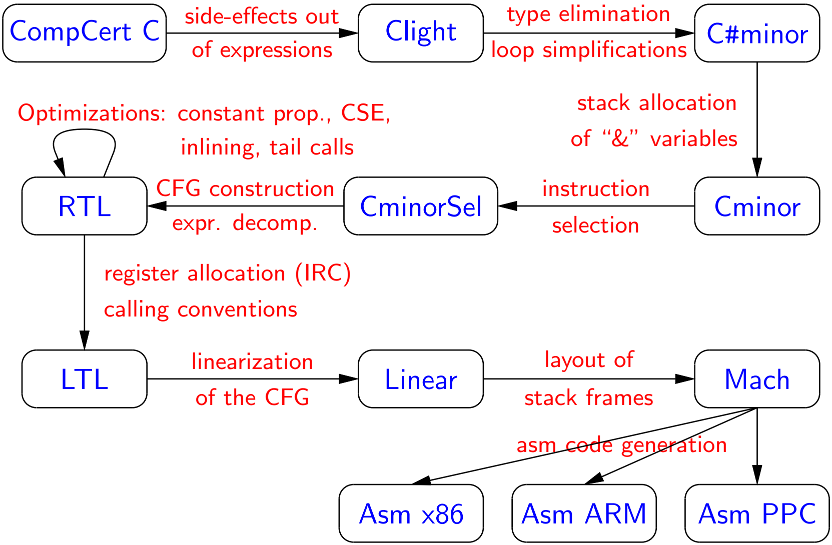
(A very hypothetical question: qualification of a C compiler has never been attempted before, and might not be economically viable.)

At first sight, a plausible match:

| | | |
|---|---|---|
| Coq specifications | $\approx$ | parts of the high-level requirements |
| Coq functions | $\approx$ | parts of the low-level requirements |
| Coq proofs | $\approx$ | automated verification activity |

But this leaves many things unaccounted for. . .

# The formally-verified part of CompCert



# The formally-verified part of CompCert

The high-level specifications comprise:

- Operational semantics of CompCert C (big, complex)
- Operational semantics of PowerPC Asm (large, simple)
- The statement of semantic preservation: simulation diagram or (simpler) inclusion between whole-program behaviors.
- Supporting theories: machine integers, floats, memory model, I/O model.

# The formally-verified part of CompCert

Thanks to the proof, no need to talk about:

- Intermediate languages.
- Compilation algorithms.
- Optimizations and their supporting static analyses.

(Note: optimizations, being context-dependent, cannot be validated by traditional testing.)

# Validating an operational semantics

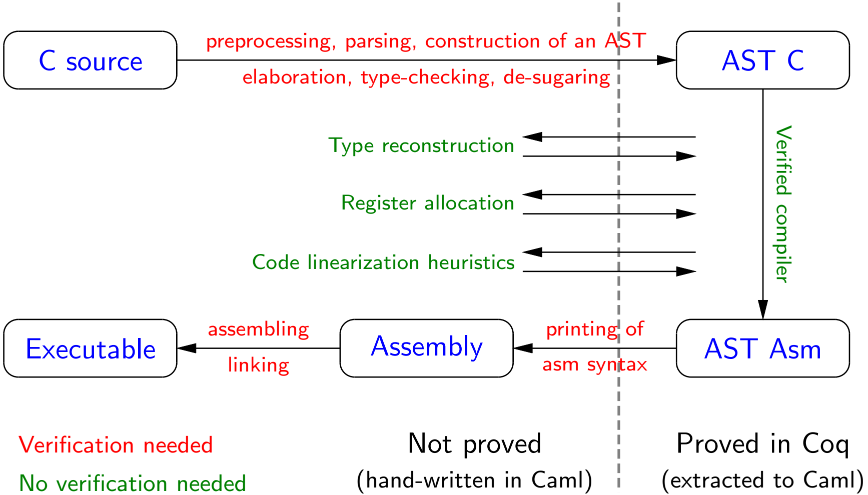
Most plausible approach: testing on an executable form of the semantics.

The CompCert C reference interpreter:
Coq functions that are proved equivalent to one-step transitions.
(Approach suggested by Brian Campbell.)

Other techniques: PLT Redex, Ott, Jakarta, ...

Example of use: 3-way differential random testing with Csmith
(CompCert interpreter / CompCert compiler / GCC).
(But: no value for certification.)

# The full CompCert compiler



# What to do with the unproved parts?

Assembly and linking:

- An unverified validation tool that matches the ELF executable against the Asm AST.

From C source to CompCert C AST:

- Testing? (mostly compositional transformations)
- Proving more things? (e.g. lexing and parsing)
- Lack of high-level formal specifications.

# When spec = implementation...

Example: interpreting the "tag soup" (G. Necula).

```
volatile long unsigned inline long static * const f(void)
{ ... }
```

$\rightarrow$

Function declaration:

name: `f`
storage class: `static`
inline: `true`
result type: `TPtr(TInt(IULongLong, volatile), const)`
parameters: none
varargs: no
body: ...

# Trusting Coq

As a verification tool:

- The “de Bruijn” architecture is appreciated (production of independently-checkable proof terms).
- Multiple independent checkers would be a big plus (e.g. `coqchk` that works and is developed independently).

As a code generation tool: (extraction)

- Highly suspect.
- Manual review of extracted Caml code probably needed.

Doubts on the OCaml compiler and runtime:

- A simplified version used in Scade KCG6, passed level-A qualification.
- Multiple implementations could help (e.g. OCamlJava, F#).

# Various cognitive dissonances

DO-178 traceability $=$ refinement $\wedge$ no additional functionality.

vs.

Soundness proofs cannot show that no dead code was introduced.

# Various cognitive dissonances

DO-178 traceability = refinement $\wedge$ no additional functionality.

vs.

Soundness proofs cannot show that no dead code was introduced.

Good mathematical style: define things then immediately prove some properties about them.

vs.

Orthodox V&V practice: development and verification are distinct activities, preferably done by different teams.

# Concluding remarks

A formal verification is not a certification.

# Concluding remarks

A formal verification is not a certification.

A formal verification can contribute towards a certification.

# Concluding remarks

A formal verification is not a certification.

A formal verification can contribute towards a certification.

Focus on your specifications.
(A clearer spec is worth a 2x increase in proof effort.)

# Concluding remarks

A formal verification is not a certification.

A formal verification can contribute towards a certification.

Focus on your specifications.
(A clearer spec is worth a 2x increase in proof effort.)

Make provisions for testing your specifications.
(Semantics that are executable, for example.)

# Concluding remarks

A formal verification is not a certification.

A formal verification can contribute towards a certification.

Focus on your specifications.
(A clearer spec is worth a 2x increase in proof effort.)

Make provisions for testing your specifications.
(Semantics that are executable, for example.)

Focus your verification efforts on parts that cannot be adequately verified by testing.

# Concluding remarks

A formal verification is not a certification.

A formal verification can contribute towards a certification.

Focus on your specifications.
(A clearer spec is worth a 2x increase in proof effort.)

Make provisions for testing your specifications.
(Semantics that are executable, for example.)

Focus your verification efforts on parts that cannot be adequately verified by testing.

Rome wasn't built in a day. Keep bringing nice stones!