Trustworthy Software and AI

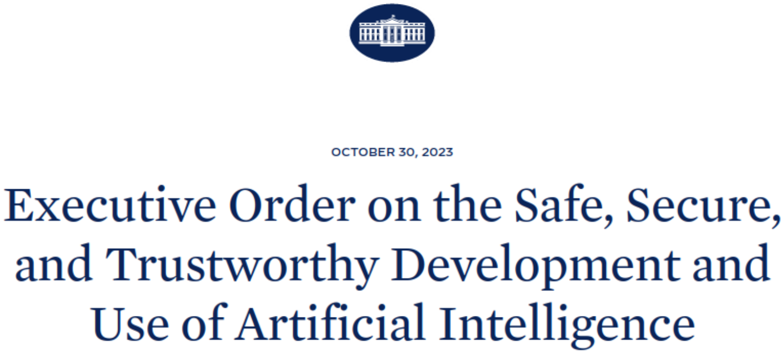
George Stelle

Los Alamos National Laboratory

November 1, 2023

Motivation

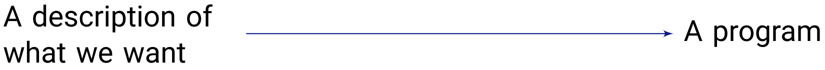
Motivation



OCTOBER 30, 2023

Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence

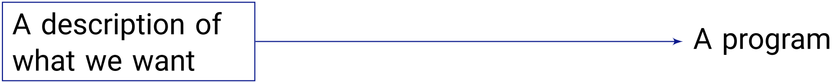
Overview



A description of what we want

A program

Overview



A description of what we want

A program

Background

> A program without a specification cannot be wrong, it can only be surprising
>
> *Young et al.*

Specification

What is a specification?

- **The old, bad ways:**

Specification

What is a specification?

- **The old, bad ways:**
  - Natural language descriptions of desired behavior (imprecise, not checkable)

Specification

What is a specification?

- **The old, bad ways:**
  - Natural language descriptions of desired behavior (imprecise, not checkable)
  - Testing (incomplete)

Specification

What is a specification?

- **The old, bad ways:**
  - Natural language descriptions of desired behavior (imprecise, not checkable)
  - Testing (incomplete)
  - A program written in a programming language (low level, depends on the above)

Specification

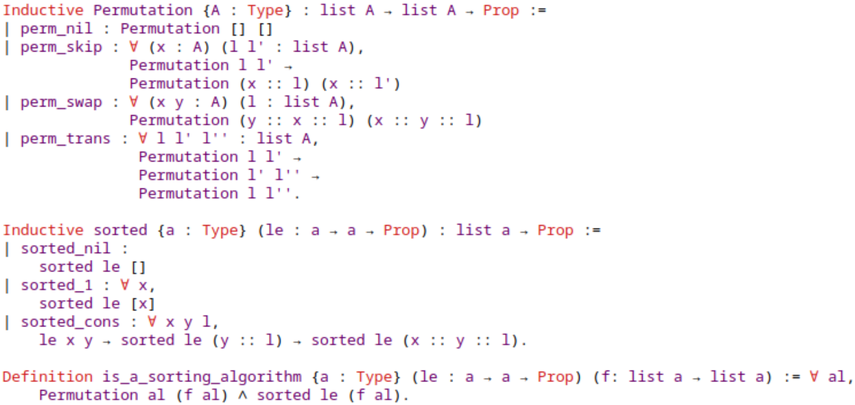
What is a specification?

- **The old, bad ways:**
  - Natural language descriptions of desired behavior (imprecise, not checkable)
  - Testing (incomplete)
  - A program written in a programming language (low level, depends on the above)
- **The new, good way:** formally specify *all* desired properties over *all* possible inputs, without implementation details, and be checkable.

Type Theory

$$\begin{aligned} \mathit{Propositions} &\subset \mathit{Types} \\ \mathit{Proofs} &\subset \mathit{Programs} \end{aligned}$$

Example



```
Inductive Permutation {A : Type} : list A → list A → Prop :=
| perm_nil : Permutation [] []
| perm_skip : ∀ (x : A) (l l' : list A),
            Permutation l l' →
            Permutation (x :: l) (x :: l')
| perm_swap : ∀ (x y : A) (l : list A),
            Permutation (y :: x :: l) (x :: y :: l)
| perm_trans : ∀ l l' l'' : list A,
             Permutation l l' →
             Permutation l' l'' →
             Permutation l l''.

Inductive sorted {a : Type} (le : a → a → Prop) : list a → Prop :=
| sorted_nil :
    sorted le []
| sorted_1 : ∀ x,
    sorted le [x]
| sorted_cons : ∀ x y l,
    le x y → sorted le (y :: l) → sorted le (x :: y :: l).

Definition is_a_sorting_algorithm {a : Type} (le : a → a → Prop) (f: list a → list a) := ∀ al,
    Permutation al (f al) ∧ sorted le (f al).
```

Mathematics

As a consequence of my #Lean4 formalization project I have found a small (but non-trivial) bug in my paper!

-Terence Tao[1]

Precision

“The (Compcert’s) semantics is deterministic and makes precise a number of behaviors left unspecified or undefined in the ISO C standard”

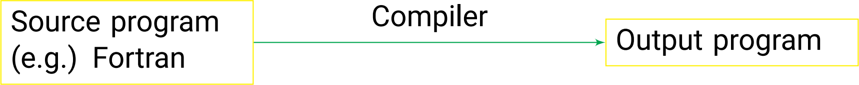
-Xavier Leroy[2]

Empirical Evidence

“So far we have reported 79 GCC bugs and 202 LLVM bugs . . . 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.”

-Yang et al.[3]

Current Software

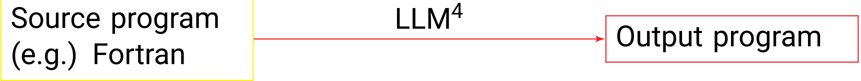


Source program (e.g.) Fortran

Compiler

Output program

LLM Compilation

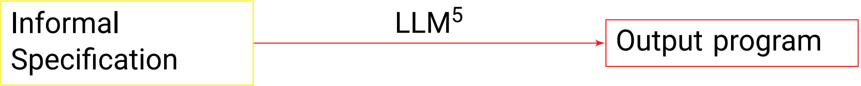


Source program (e.g.) Fortran

LLM[4]

Output program

LLM Synthesis



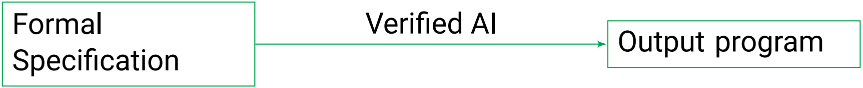
Informal Specification

LLM[5]

Output program

But I can verify!

Goal



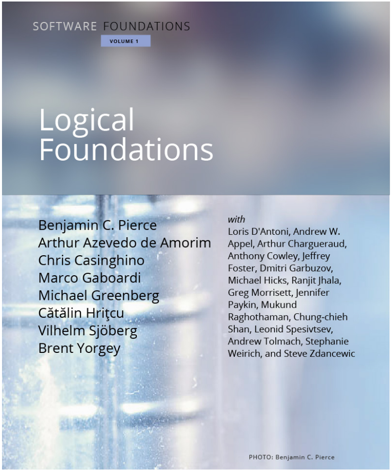
Formal Specification

Verified AI

Output program

Motivation

Further Reading



SOFTWARE FOUNDATIONS

VOLUME 1

Logical Foundations

Benjamin C. Pierce
Arthur Azevedo de Amorim
Chris Casinghino
Marco Gaboardi
Michael Greenberg
Cătălin Hriţcu
Vilhelm Sjöberg
Brent Yorgey

with
Loris D'Antoni, Andrew W. Appel, Arthur Chargueraud, Anthony Cowley, Jeffrey Foster, Dmitri Garbuzov, Michael Hicks, Ranjit Jhala, Greg Morrisett, Jennifer Paykin, Mukund Raghothaman, Chung-chieh Shan, Leonid Spesivtsev, Andrew Tolmach, Stephanie Weirich, and Steve Zdancewic

PHOTO: Benjamin C. Pierce

ASC/NNSA

- First NNSA workshop on formal verification to be held in December in Santa Fe.

ASC/NNSA

- First NNSA workshop on formal verification to be held in December in Santa Fe.
- Sandia has been funding this work for at least 10 years, has built expertise.

ASC/NNSA

- First NNSA workshop on formal verification to be held in December in Santa Fe.
- Sandia has been funding this work for at least 10 years, has built expertise.
- We (LANL) should try and catch up.

ASC/NNSA

- First NNSA workshop on formal verification to be held in December in Santa Fe.
- Sandia has been funding this work for at least 10 years, has built expertise.
- We (LANL) should try and catch up.
- AI4SS

ASC/NNSA

- First NNSA workshop on formal verification to be held in December in Santa Fe.
- Sandia has been funding this work for at least 10 years, has built expertise.
- We (LANL) should try and catch up.
- AI4SS
- Come talk to me!

References

[1] URL: `https://mathstodon.xyz/@tao/111287749336059662`.

[2] Xavier Leroy. “Formal verification of a realistic compiler”. In: *Communications of the ACM* 52.7 (2009), pp. 107–115.

[3] Xuejun Yang et al. “Finding and understanding bugs in C compilers”. In: *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*. 2011, pp. 283–294.

[4] Chris Cummins et al. “Large language models for compiler optimization”. In: *arXiv preprint arXiv:2309.07062* (2023).

[5] Naman Jain et al. “Jigsaw: Large language models meet program synthesis”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 1219–1231.