Introduction to Type Theories

introductory lecture

Anja Petković Komel

OPLSS 2025

Why Type Theories?

Why Type Theories?

- **Answer for mathematicians**: because they are beautiful mathematical foundations that can express a constructive approach.

Why Type Theories?

- **Answer for mathematicians**: because they are beautiful mathematical foundations that can express a constructive approach.
- **Answer for the rest of the world**: because proof assistants are built on them.

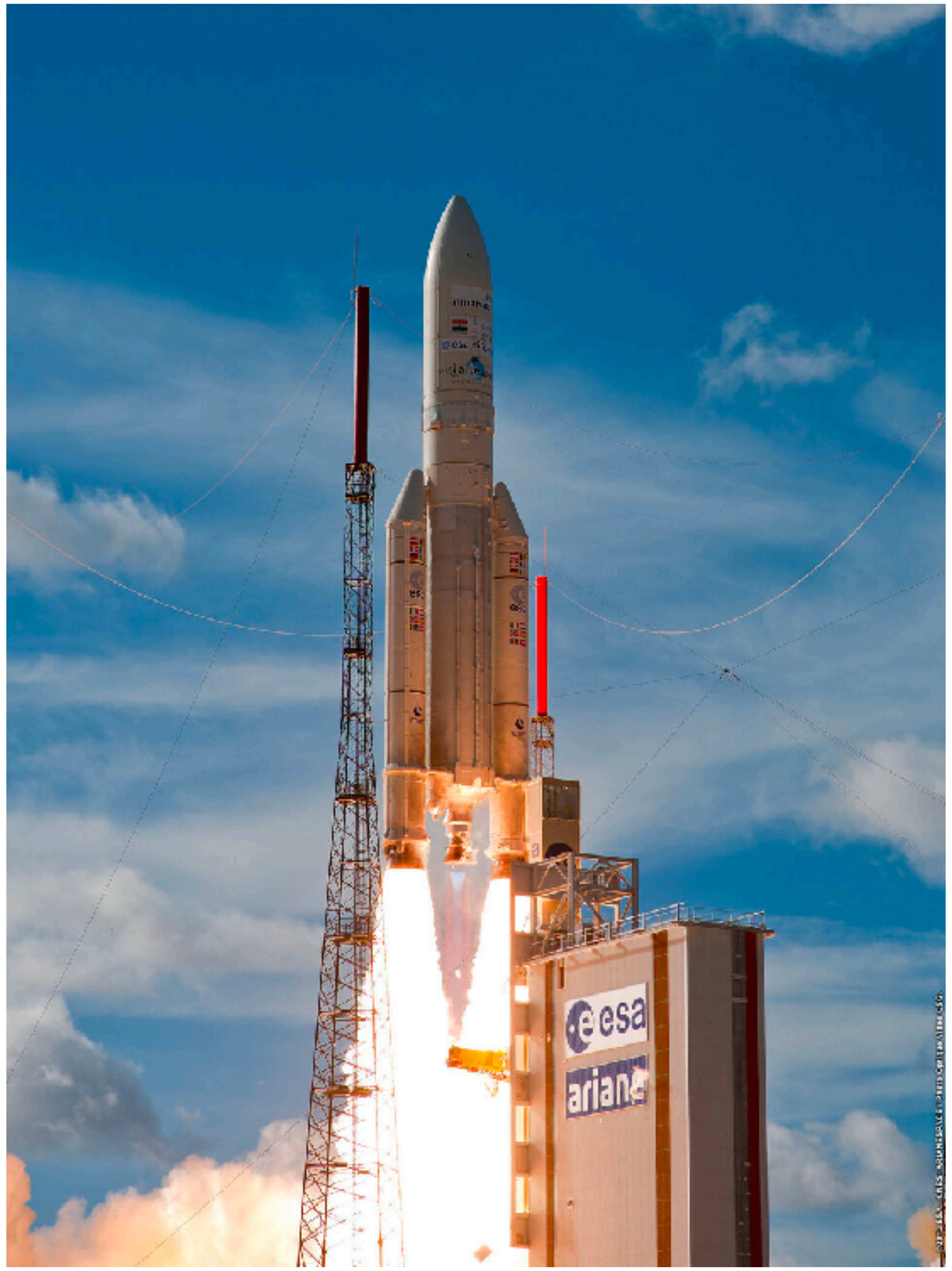
Why Type Theories?

- **Answer for mathematicians**: because they are beautiful mathematical foundations that can express a constructive approach.

- **Answer for the rest of the world**: because proof assistants are built on them.

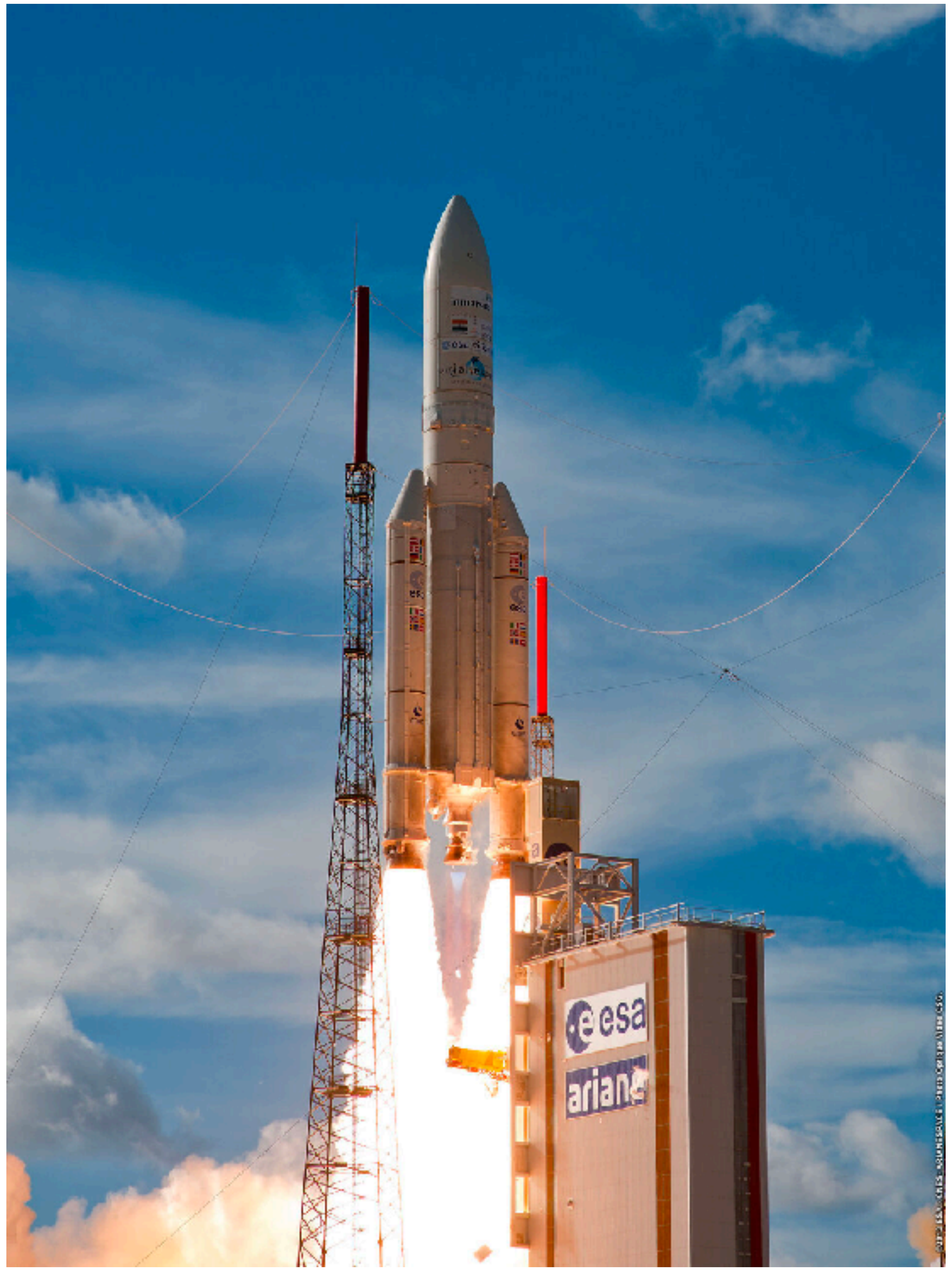
so, why proof assistants?

Verifying Software Correctness



Ariane 5 rocket, 4 June 1996

Verifying Software Correctness



Ariane 5 rocket, 4 June 1996

Extracting Smart Contracts Tested and Verified in Coq

Danil Annenkov[1], Mikkel Milo[2], Jakob Botsch Nielsen[1], and Bas Spitters[1]

[1] Concordium Blockchain Research Center, Aarhus University
[2] Department of Computer Science, Aarhus University, Denmark

Abstract

We implement extraction of Coq programs to functional languages based on MetaCoq's certified erasure. As part of this, we implement an optimisation pass removing unused arguments. We prove the pass correct wrt. a conventional call-by-value operational semantics of functional languages. We apply this to two functional smart contract languages, Liquidity and Midlang, and to the functional language Elm. Our development is done in the context of the ConCert framework that enables smart contract verification. We contribute a verified boardroom voting smart contract featuring maximum voter privacy such that each vote is kept private except under collusion of all other parties. We also integrate property-based testing into ConCert using QuickChick and our development is the first to support testing properties of interacting smart contracts. We test several complex contracts such as a DAO-like contract, an escrow contract, an implementation of a Decentralized Finance (DeFi) contract which includes a custom token standard (Tezos FA2), and more. In total, this gives us a way to write dependent programs in Coq, test them semi-automatically, verify, and then extract to functional smart contract languages, while retaining a small trusted computing base of only MetaCoq and the pretty-printers into these languages.

1 Introduction

Smart contracts are programs running on top of a blockchain. They often control large amounts of cryptocurrency and cannot be changed after deployment. Unfortunately, many vulnerabilities have [cs.PL] 26 Apr 2021

Extracting Smart Contracts Tested and Verified in Coq

Danil Annenkov[1], Mikkel Milo[2], Jakob Botsch Nielsen[1], and Bas Spitters[1]

[1] Concordium Blockchain Research Center, Aarhus University
[2] Department of Computer Science, Aarhus University, Denmark

Abstract

We implement extraction of Coq programs to functional languages based on MetaCoq's certified erasure. part of this, we implement an optimisation pass removing unused arguments. We prove the pass correct wr conventional call-by-value operational semantics of functional languages. We apply this to two functional sm contract languages, Liquidity and Midlang, and to the functional language Elm. Our development is done in context of the ConCert framework that enables smart contract verification. We contribute a verified boardro voting smart contract featuring maximum voter privacy such that each vote is kept private except under collus of all other parties. We also integrate property-based testing into ConCert using QuickChick and our developm is the first to support testing properties of interacting smart contracts. We test several complex contracts s as a DAO-like contract, an escrow contract, an implementation of a Decentralized Finance (DeFi) contract wh includes a custom token standard (Tezos FA2), and more. In total, this gives us a way to write depend programs in Coq, test them semi-automatically, verify, and then extract to functional smart contract languag while retaining a small trusted computing base of only MetaCoq and the pretty-printers into these languages.

1 Introduction

Smart contracts are programs running on top of a blockchain. They often control large amounts cryptocurrency and cannot be changed after deployment. Unfortunately, many vulnerabilities h

Formal Certification of a Compiler Back-end

or: Programming a Compiler with a Proof Assistant

Xavier Leroy
INRIA Rocquencourt
Xavier.Leroy@inria.fr

Abstract

This paper reports on the development and formal certification (proof of semantic preservation) of a compiler from Cminor (a C-like imperative language) to PowerPC assembly code, using the Coq proof assistant both for programming the compiler and for proving its correctness. Such a certified compiler is useful in the context of formal methods applied to the certification of critical software: the certification of the compiler guarantees that the safety properties proved on the source code hold for the executable compiled code as well.

Categories and Subject Descriptors F.3.1 [*Logics and meanings of programs*]: Specifying and verifying and reasoning about programs—Mechanical verification.; D.2.4 [*Software engineering*]: Software/program verification—Correctness proofs, formal methods, reliability; D.3.4 [*Programming languages*]: Processors—Compilers, optimization

General Terms Languages, Reliability, Security, Verification.

Keywords Certified compilation, semantic preservation, program proof, compiler transformations and optimizations, the Coq theorem prover.

can potentially invalidate all the guarantees so painfully obtained using formal methods. In other terms, from a formal methods perspective, the compiler is a weak link between a source program that has been formally verified and a hardware processor that, more and more often, has also been formally verified. The safety-critical software industry is aware of this issue and uses a variety of techniques to alleviate it, such as conducting manual code reviews of the generated assembly code after having turned all compiler optimizations off. These techniques do not fully address the issue, and are costly in terms of development time and program performance.

An obviously better approach is to apply formal methods to the compiler itself in order to gain assurance that it preserves the semantics of the source programs. Many different approaches have been proposed and investigated, including on-paper and on-machine proofs of semantic preservation, proof-carrying code, credible compilation, translation validation, and type-preserving compilers. (These approaches are compared in section 2.) For the last two years, we have been working on the development of a *realistic*, *certified* compiler. By *certified*, we mean a compiler that is accompanied by a machine-checked proof of semantic preservation. By *realistic*, we mean a compiler that compiles a language commonly used for critical embedded software (a subset of C) down to assembly code for a processor commonly used in

Extracting Smart Contracts Tested and Verified in Coq

Danil Annenkov[1], Mikkel Milo[2], Jakob Botsch Nielsen[1], and Bas Spitters[1]

[1] Concordium Blockchain Research Center, Aarhus University
[2] Department of Computer Science, Aarhus University, Denmark

Abstract

We implement extraction of Coq programs to functional languages based on MetaCoq's certified erasure. part of this, we implement an optimisation pass removing unused arguments. We prove the pass correct wrt conventional call-by-value operational semantics of functional languages. We apply this to two functional sm contract languages, Liquidity and Midlang, and to the functional language Elm. Our development is done in context of the ConCert framework that enables smart contract verification. We contribute a verified boardro voting smart contract featuring maximum voter privacy such that each vote is kept private except under collus of all other parties. We also integrate property-based testing into ConCert using QuickChick and our developm is the first to support testing properties of interacting smart contracts. We test severa as a DAO-like contract, an escrow contract, an implementation of a Decentralized Finar includes a custom token standard (Tezos FA2), and more. In total, this gives us a programs in Coq, test them semi-automatically, verify, and then extract to functional while retaining a small trusted computing base of only MetaCoq and the pretty-printer

1 Introduction

Smart contracts are programs running on top of a blockchain. They often cc cryptocurrency and cannot be changed after deployment. Unfortunately, ma

Formal Certification of a Compiler Back-end

or: Programming a Compiler with a Proof Assistant

Xavier Leroy
INRIA Rocquencourt
Xavier.Leroy@inria.fr

COMPCERT

COMPILERS YOU CAN *FORMALLY* TRUST

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.

The main result of the project is the CompCert C verified compiler, a high-assurance compiler for almost all of the C language (ISO C 2011), generating efficient code for the ARM, PowerPC, RISC-V and x86 processors.

ACM Software System Award 2021

Download CompCert C

Read the manual

MENU

- Home
- Partners
- Motivations
- Research
- The Compcert C compiler
- Downloads
- Publications

Extracting Smart Contracts Tested and Verified in Coq

Danil Annenkov[1], Mikkel Milo[2], Jakob Botsch Nielsen[1], and Bas Spitters[1]

[1] Concordium Blockchain Research Center, Aarhus University
[2] Department of Computer Science, Aarhus University, Denmark

Abstract

We implement extraction of Coq programs to functional languages based on MetaCoq's certified erasure.

[cs.PL] 26 Apr 2021

Formal Certification of a Compiler Back-end

or: Programming a Compiler with a Proof Assistant

Xavier Leroy

INRIA Rocquencourt

Xavier.Leroy@inria.fr

fscq

fscq

A Formally Certified Crash-proof File System

Overview

FSCQ is the first file system with a machine-checkable proof (using the Coq proof assistant) that its implementation meets its specification and whose specification includes crashes. FSCQ provably avoids bugs that have plagued previous file systems, such as performing disk writes without sufficient barriers or forgetting to zero out directory blocks. If a crash happens at an inopportune time, these bugs can lead to data loss. FSCQ's theorems prove that, under any sequence of crashes followed by reboots, FSCQ will recover the file system correctly without losing data.

People

- Haogang Chen
- Daniel Ziegler
- Tej Chajed
- Adam Chlipala
- M. Frans Kaashoek
- Nickolai Zeldovich

RT

COMPILERS YOU CAN *FORMALLY* TRUST

roject investigates the formal verification of realistic compilers usable for
d software. Such verified compilers come with a mathematical, machine-
hat the generated executable code behaves exactly as prescribed by the
source program. By ruling out the possibility of compiler-introduced bugs,
rs strengthen the guarantees that can be obtained by applying formal
ce programs.

of the project is the CompCert C verified compiler, a
ompiler for almost all of the C language (ISO C 2011),
ent code for the ARM, PowerPC, RISC-V and x86

ACM Software System Award 2021

ompCert C | Read the manual

MENU

- Home
- Partners
- Motivations
- Research
- The Compcert C compiler
- Downloads
- Publications

Formalised Mathematics

Formalised Mathematics

A formal proof of the four color theorem

Limin Xiang
Department of Information Science,
Kyushu Sangyo University
3-1 Matsukadai 2-Chome, Higashi-ku, Fukuoka 813-8503, Japan
E-mail: xiang@is.kyusan-u.ac.jp
Tel: +81-92-673-5400, Fax: +81-92-673-5454

Manuscript, April 16, 2009

Abstract: A formal proof has not been found for the four color theorem since 1852 when Francis Guthrie first conjectured the four color theorem. Why? A bad idea, we think, directed people to a rough road. Using a similar method to that for the formal proof of the five color theorem, a formal proof is proposed in this paper of the four color theorem, namely, every planar graph is four-colorable. The formal proof proposed can also be regarded as an algorithm to color a planar graph using four colors so that no two adjacent vertices receive the same color.

1. Introduction

Since 1852 when Francis Guthrie first conjectured the four color theorem [1], a formal proof has not been found for the four color theorem. The **four color theorem**, or the **four color map theorem**, states that given any separation of the plane into contiguous

A **planar graph** G is a Graph that may be embedded in the plane without intersecting edges.

A graph G is said to be n **-colorable**, denoted by $c(G) = n$, if it's possible to assign one of n colors to each vertex in such a way that no two connected vertices have the same color.

Formalised Mathematics

A formal proof of the four color theorem

Limin Xiang
Department of Information Science,
Kyushu Sangyo University
3-1 Matsukadai 2-Chome, Higashi-ku, Fukuoka 813-8503, Japan
E-mail: xiang@is.kyusan-u.ac.jp
Tel: +81-92-673-5400, Fax: +81-92-673-5454

Manuscript, April 16, 2009

Abstract: A formal proof has not been found for the four color theorem since 1852 when Francis Guthrie firs conjectured the four color theorem. Why? A bad idea, we think, directed people to a rough road. Using a sim method to that for the formal proof of the five color theorem, a formal proof is proposed in this paper of the fo theorem, namely, every planar graph is four-colorable. The formal proof proposed can also be regarded as a algorithm to color a planar graph using four colors so that no two adjacent vertices receive the same color.

1. Introduction

Since 1852 when Francis Guthrie first conjectured the four color theorem [1], a formal proof has not been found for the four color theorem. The **four color theorem**, or the **four color map theorem**, states that given any separation of the plane into contiguous

A **planar graph** G is a Graph that may be em in the plane without intersecting edges.

A graph G is said to be n **-colorable**, den $c(G)=n$, if it's possible to assign one of n c each vertex in such a way that no two co vertices have the same color.

A Machine-Checked Proof of the Odd Order Theorem

Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry

Microsoft Research - Inria Joint Centre

Abstract. This paper reports on a six-year collaborative effort that culminated in a complete formalization of a proof of the Feit-Thompson Odd Order Theorem in the Coq proof assistant. The formalized proof is constructive, and relies on nothing but the axioms and rules of the foundational framework implemented by Coq. To support the formalization, we developed a comprehensive set of reusable libraries of formalized mathematics, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers.

1 Introduction

The Odd Order Theorem asserts that every finite group of odd order is solvable.

Formalised Mathematics

A forma

Limin Xiang
Department of Inf
Kyushu Sangyo U
3-1 Matsukadai 2-
E-mail: xiang@is.
Tel: +81-92-673-5

Manuscript, April

Abstract: A forma
conjectured the fo
method to that for
theorem, namely,
algorithm to color

1. Introduction

Since 1852 wh
the four color thec
found for the fo
theorem, or the f
given any separ

Mathematics in Lean

ince 1852 when Francis Guthrie firs
people to a rough road. Using a sim
of is proposed in this paper of the fc
proposed can also be regarded as a
t vertices receive the same color.

raph G is a Graph that may be em
without intersecting edges.

is said to be n **-colorable**, den
if it's possible to assign one of n
x in such a way that no two co
e the same color.

A Machine-Checked Proof of the Odd Order Theorem

ges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen,
ançois Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor,
Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi,
and Laurent Théry

Microsoft Research - Inria Joint Centre

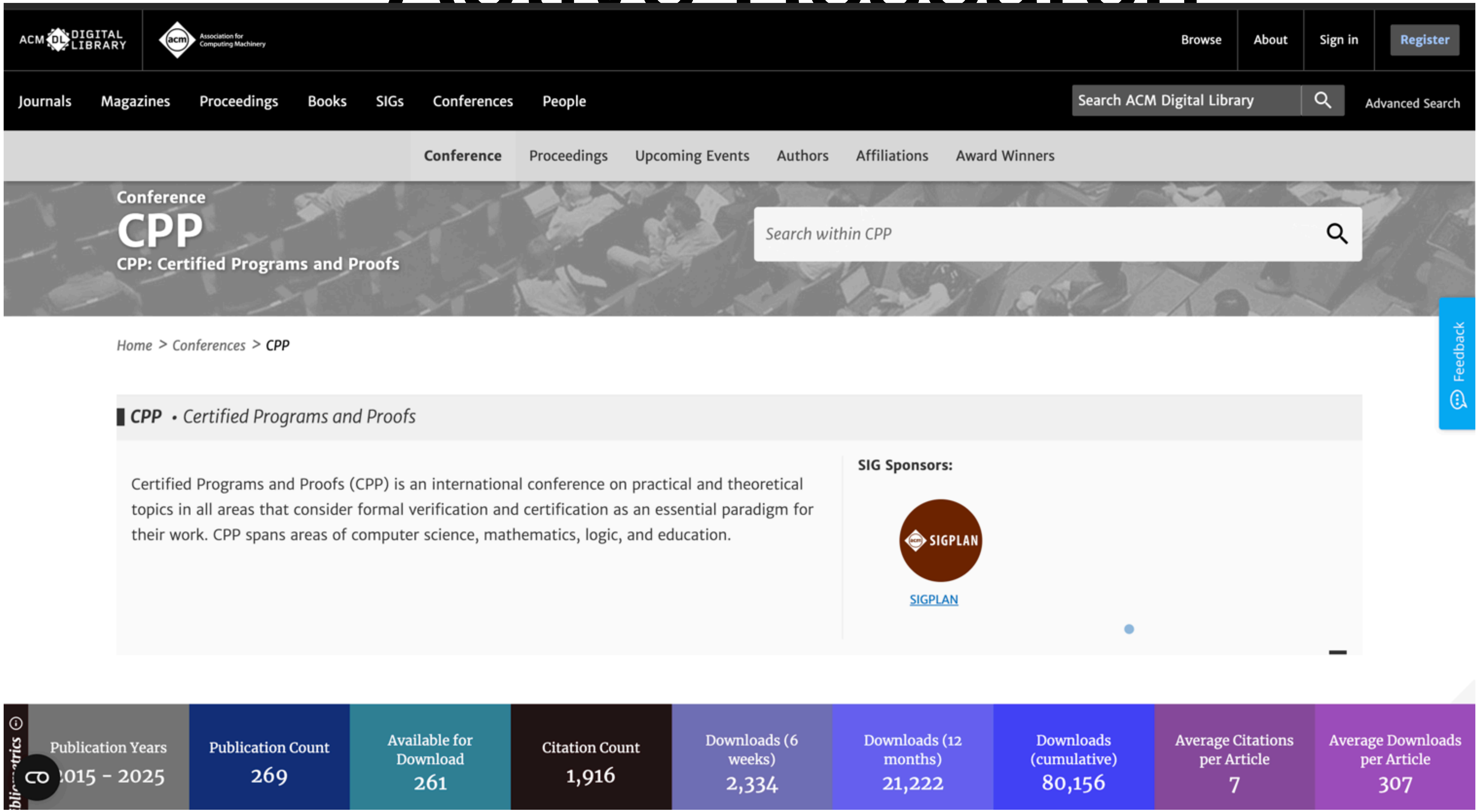
Abstract. This paper reports on a six-year collaborative effort that culminated in a complete formalization of a proof of the Feit-Thompson Odd Order Theorem in the Coq proof assistant. The formalized proof is constructive, and relies on nothing but the axioms and rules of the foundational framework implemented by Coq. To support the formalization, we developed a comprehensive set of reusable libraries of formalized mathematics, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers.

1 Introduction

The Odd Order Theorem asserts that every finite group of odd order is solvable.

Active Research

Active Research



Active Resear

ACM DIGITAL LIBRARY | Association for Computing Machinery

Journals | Magazines | Proceedings | Books | SIGs | Conferences | People

Conference | Proceedings | Upcoming Events | Authors | Affiliations | Award Winners

Conference

CPP

CPP: Certified Programs and Proofs

Search within CPP

Home > Conferences > CPP

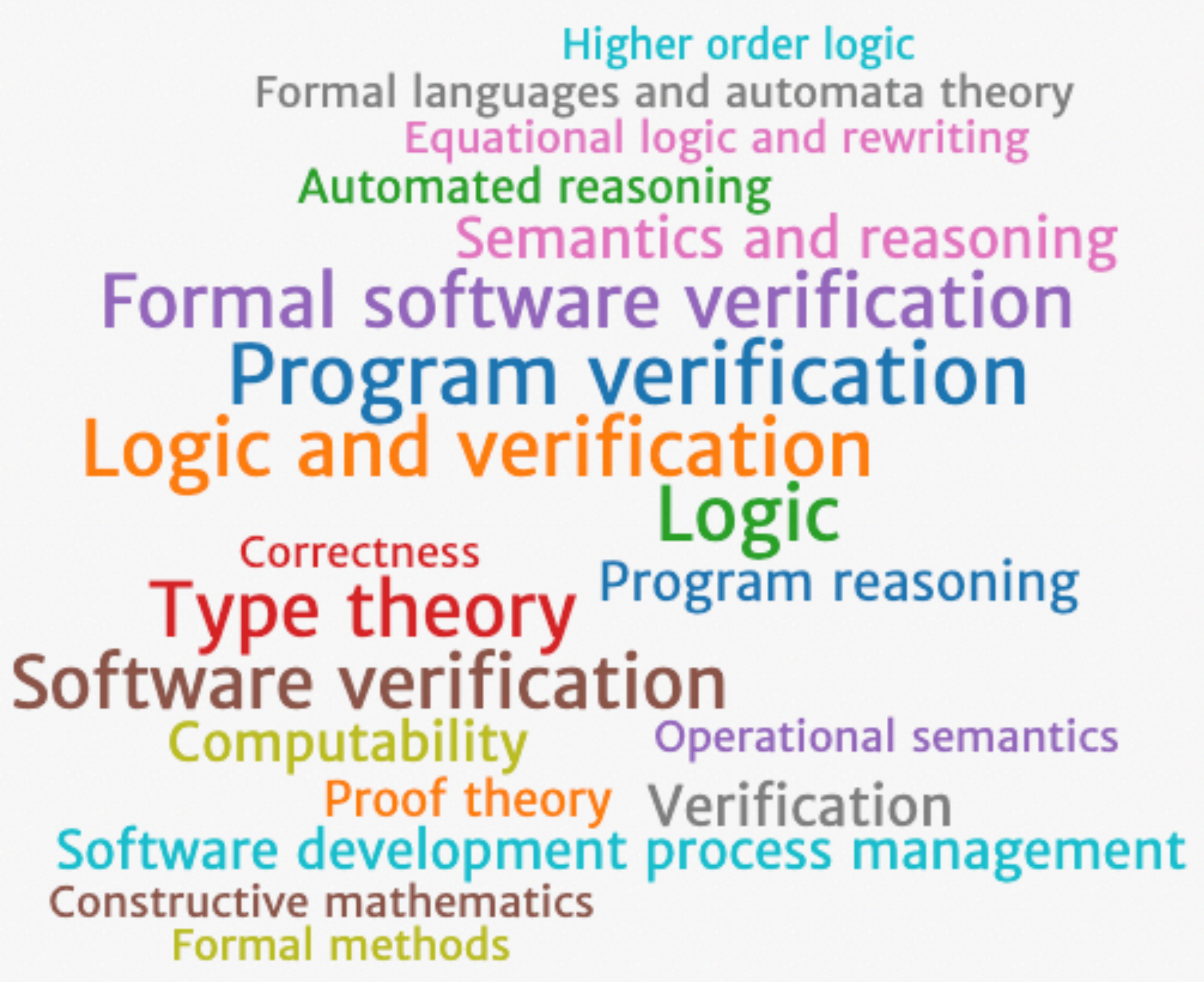
CPP • *Certified Programs and Proofs*

Certified Programs and Proofs (CPP) is an international conference on practical and theoretical topics in all areas that consider formal verification and certification as an essential paradigm for their work. CPP spans areas of computer science, mathematics, logic, and education.

SIG Sponsors:

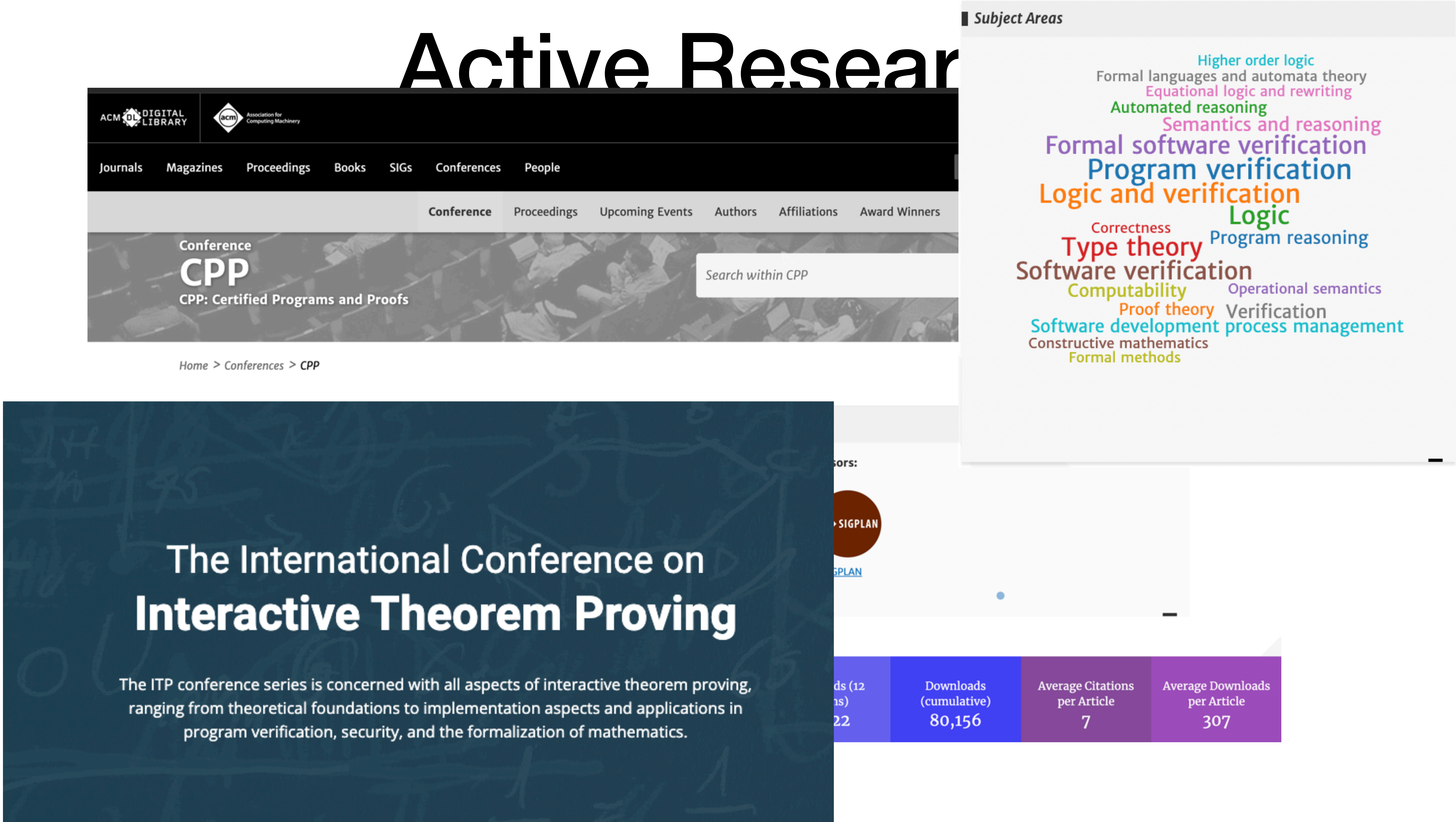
SIGPLAN

Subject Areas



Publication Years	Publication Count	Available for Download	Citation Count	Downloads (6 weeks)	Downloads (12 months)	Downloads (cumulative)	Average Citations per Article	Average Downloads per Article
2015 - 2025	269	261	1,916	2,334	21,222	80,156	7	307

Active Resear



Active Resear

ACM DL DIGITAL LIBRARY | acm Association for Computing Machinery

Journals | Magazines | Proceedings | Books | SIGs | Conferences | People

Conference | Proceedings | Upcoming Events | Au

Conference

CPP

CPP: Certified Programs and Proofs

Sear

Home > Conferences > CPP

Subject Areas

Higher order logic
Formal languages and automata theory
Equational logic and rewriting
Automated reasoning
Semantics and reasoning
Formal software verification

TYPES 2025 | Call for contributions | Committees | Invited speakers | Accepted talks | Registration | Programme | Practical information | Book of abstracts | Group photo

TYPES 2025

University of Strathclyde, Glasgow, Scotland • 9–13 June 2025

The 31st International Conference on *Types for Proofs and Programs* will take place at the University of Strathclyde from Monday 9 June to Friday 13 June 2025, and is organised by the Mathematically Structured Programming group. On Tuesday 10 June, there will be a co-located Women in EuroProofNet event dedicated to gender balance in our community. The week after, CALCO/MFPS will also take place at the University of Strathclyde.

The TYPES conference is a forum to present new and ongoing work in all aspects of type theory and its applications, especially in formalised and computer assisted reasoning and computer programming. Areas of interest include, but are not limited to:

- foundations of type theory and constructive mathematics;
- applications of type theory;
- dependently typed programming;
- industrial uses of type theory technology;
- meta-theoretic studies of type systems;
- proof assistants and proof technology;
- automation in computer-assisted reasoning;
- links between type theory and functional programming;
- formalizing mathematics using type theory

TYPES 2025

The International Conference on Interactive Theorem Proving

The ITP conference series is concerned with all aspects of interactive theorem provi
ranging from theoretical foundations to implementation aspects and applications i
program verification, security, and the formalization of mathematics.

Vladimir Voevodsky, 1966 - 2017

Proof Assistants

Calculus of
inductive
constructions

Rocq-HoTT:
Homotopy type theory

Unimath:
Univalent foundations

Agda

Martin-Löf
type theory

- - cubical:
Cubical type theory

LEAN THEOREM PROVER

Calculus of
inductive
constructions

More Proof Assistants

Nuprl:
Computational type theory

Isabelle/HOL:
Higher order logic

Mizar:
Tarski-Grothendick
set theory with syntactical
weak types

redTT:
(Cartesian) cubical
type theory

cubicaltt:
Cubical type theory

F*
Andromeda:
Extensional type theory

Andromeda 2:
General type theories

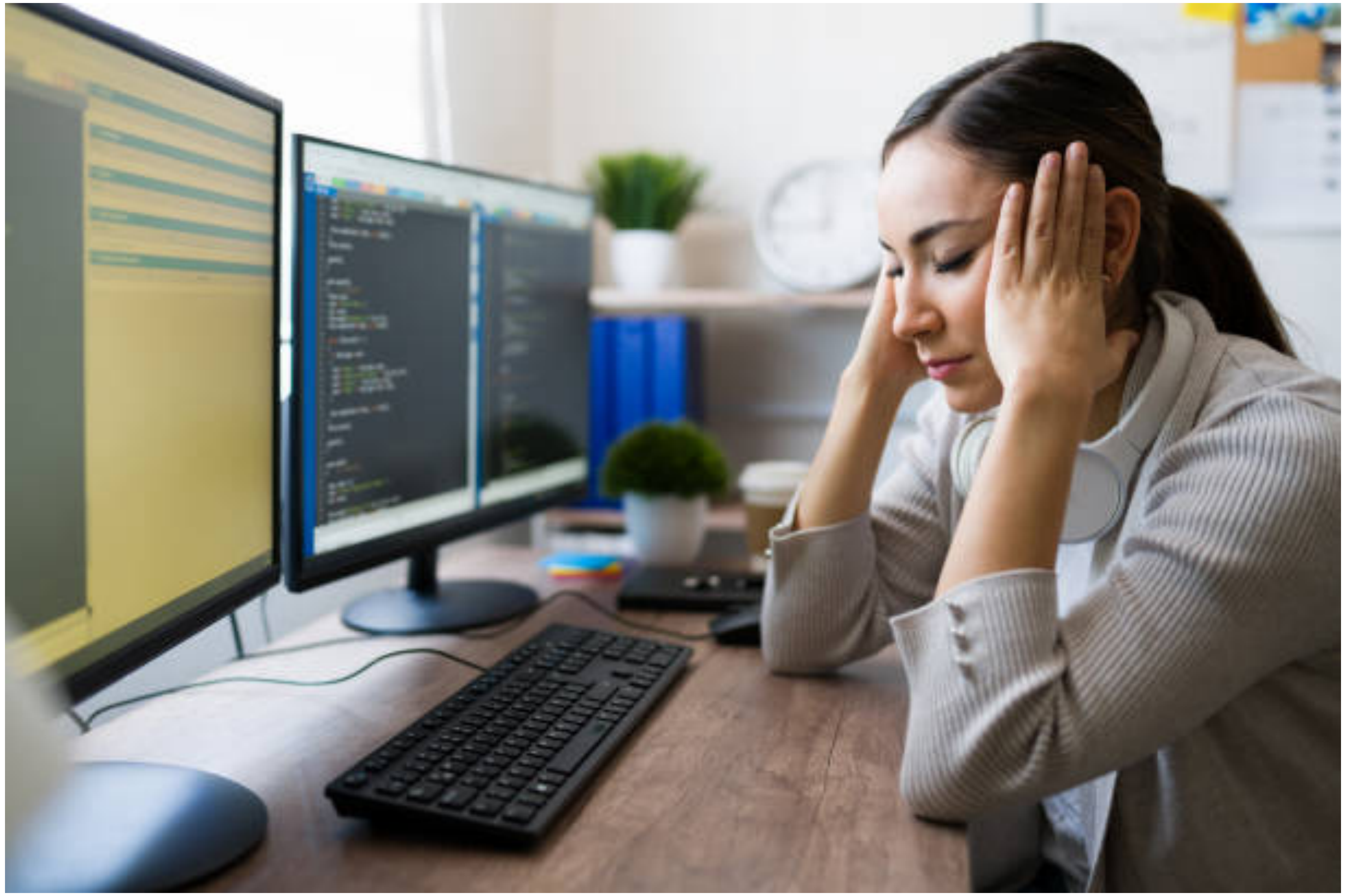
Arend:
Cubical type theory

Why are we learning about type theories,

rather than using a proof assistant?

Why are we learning about type theories,

rather than using a proof assistant?



Structure of Lectures

Structure of Lectures

1. **Martin-Löf type theory (MLTT)**
   following chapter 1 of
   Egbert Rijke: Introduction to Homotopy Type Theory

Structure of Lectures

1. **Martin-Löf type theory (MLTT)**
   following chapter 1 of
   Egbert Rijke: Introduction to Homotopy Type Theory

2. **Calculus of Inductive Constructions (CIC) & Rocq**
   Interactively proving (easy) theorems
   Live coding + slides

Structure of Lectures

1. **Martin-Löf type theory (MLTT)**
   following chapter 1 of
   Egbert Rijke: Introduction to Homotopy Type Theory

2. **Calculus of Inductive Constructions (CIC) & Rocq**
   Interactively proving (easy) theorems
   Live coding + slides

3. If time permits: **more Rocq** or **Meta-Theory of type theories**
   Haselwarter, P. G. and Bauer, A., “Finitary type theories with and without contexts”

Disclaimer:

we will discuss type theories **syntactically**.

Disclaimer:
we will discuss type theories **syntactically**.

There is a whole lot of research about semantic models of type theories.

If interested, ask for references (or better yet, ask Paige Randall North)

- knowledge of category theory is a prerequisite.