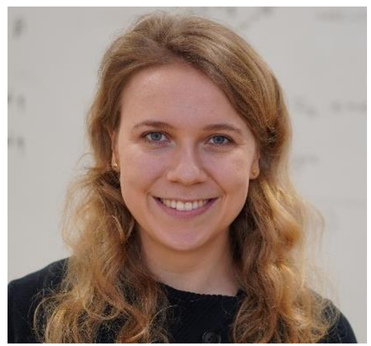
# Proofs are Programs

## Winter 2024-25 @ Ruhr Uni Bochum



**Lecturers:** Cătălin Hrițcu and Clara Schneidewind

**TAs:** Cezar Andrici and Sebastian Harwig

Max Planck Institute for Security and Privacy (MPI-SP)

# This course in a nutshell

- Logic and proofs
- Verifying functional programs
- Formalizing simple imperative language
- Secure information flow
- Using the Coq proof assistant
  - gentle introduction to both <u>functional programming</u> and <u>formal proofs</u> (both in Coq)

["Le coq mécanisé" picture by Lilia Anisimova]

# The Coq proof assistant

- Developed at Inria in France since 1983 (in OCaml)
- **Coq helps you build formal proofs interactively**
- **Proving in Coq is like programming**
  - gamified, addictive, and lots of fun
  - if you like programming, you will also like Coq proofs
- **This helps you deeply understand proofs**
- **In fact, formal Coq proofs are just purely functional programs**
  - **Curry-Howard correspondence**:
    deep connection between logic and functional programming

# Functional programming

$\lambda$

- **Write computations as recursive functions**
  - **using recursion, immutable datatypes, and pattern matching**
  - **limit side-effects**, such as **mutating** stateful data structures
- **Coq is purely functional = zero side-effects**
  - all computations are mathematical functions (in particular terminating)
- **Functional programming languages** like OCaml, Haskell, ...
  - not completely pure, but still try to reduce and control side-effects
  - they still make it as easy as possible to write functional code
- **Course previous semester using OCaml** (again in couple of semesters)
  - Not a prerequisite for this course

OCaml

# Functional Programming in practice λ

- **Functional languages have some practical success**
  - **Meta** (OCaml, Haskell, Rust), **Microsoft** (OCaml, F#, F*, and Rust), **X** (Scala), **Mozilla** (Rust), **Google** (Rust), **Amazon** (Rust), **Financial industry**, **Blockchains and smart contracts** (Clara's group does research on this), ...
- **Not yet fully mainstream, but ...**
  - **Many cool ideas already adopted by mainstream languages**:

    - Lambdas, Generics in Java/C#, Rust's type system, datatypes, pattern matching (most admired language on Stack Overflow for the last 11 years!)
  - **Functional programmers often earn more** (Stack Overflow developer survey)
  - **Functional programs are concise, elegant, beautiful**
    - This makes understanding and reasoning about programs much easier, both informally and formally
- **For this course this last point is very important**
  - formal verification by proving theorems about purely functional programs in Coq

# Coq proof that append is associative

```
Fixpoint app {a:Type} (l1 l2 : list a) : list a :=
  match l1 with
  | [] => l2
  | hd :: tl => hd :: (app tl l2)
  end.

Theorem app_assoc : forall (a:Type) (l1 l2 l3 : list a),
  app (app l1 l2) l3 = app l1 (app l2 l3).
Proof.
  intros a l1 l2 l3. induction l1 as [| hd tl IH].
  - (* l1 = [] *) simpl. reflexivity.
  - (* l1 = hd :: tl *) simpl. rewrite -> IH. reflexivity.
Qed.
```

```
Fixpoint app {a:Type} (l1 l2 : list a) : list a :=
  match l1 with
  | [] => l2
  | hd :: tl => hd :: (app tl l2)
  end.

Theorem app_assoc : forall (a:Type) (l1 l2 l3 : list a),
  app (app l1 l2) l3 = app l1 (app l2 l3).
Proof.
  intros a l1 l2 l3. induction l1 as [| hd tl IH].
  - (* l1 = [] *) simpl. reflexivity.
  - (* l1 = hd :: tl *) simpl. rewrite -> IH. reflexivity.
Qed.
```

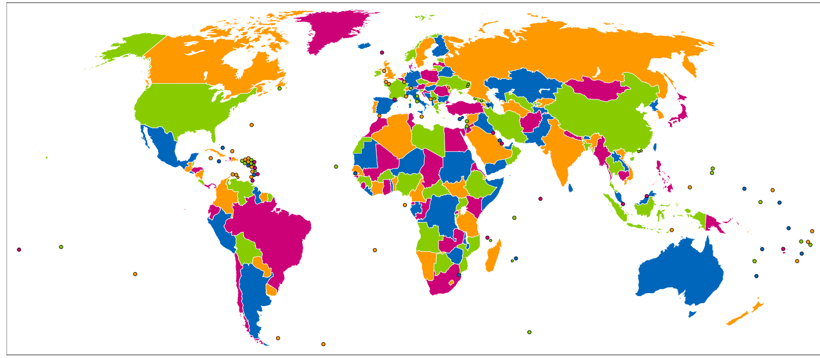
-:--- app.v Bot (14,29) (Coq Script(1-) +6 Holes Fly/en)

```
1 goal (ID 25)

- a : Type
- hd : a
- tl, l2, l3 : list a
- IH : app (app tl l2) l3 = app tl (app l2 l3)
  ============================
  hd :: app (app tl l2) l3 = hd :: app tl (app l2 l3)
```

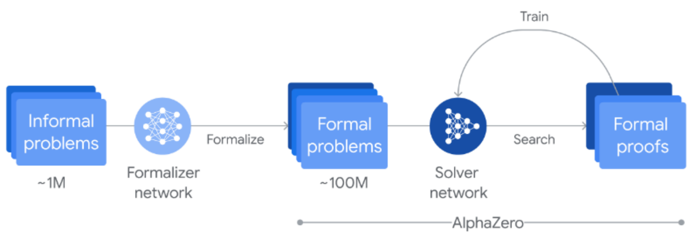
# Machine-checked proofs of math theorems

- **The 4-color theorem**
  - proved in 1976, simplified in 1996
  - proofs rely on a computer
  - infeasible for a human to check
  - initially not accepted by all mathematicians
  - mechanized in Coq by Georges Gonthier in 2005



[World map from Wikipedia]

- **Construction of perfectoid spaces** (Peter Scholze, Lean community, 2020)
  - Mathlib math library for Lean has over 100,000 theorems, over 1,000,000 lines of code, and an active community including many mathematicians (e.g. Terence Tao)
- **AlphaProof: Google DeepMind AI achieves silver medalist level in IMO (2024)**
  - Key ingredient: the Lean proof assistant checks each step of the AI



# Formalizing simple imperative language

$$\frac{}{\texttt{st =[ skip ]=> st}}\ \text{(E\_Skip)}$$

$$\frac{\texttt{aeval st a = n}}{\texttt{st =[ x := a ]=> (x !-> n ; st)}}\ \text{(E\_Asgn)}$$

$$\frac{\begin{array}{c}\texttt{st =[ }c_1\texttt{ ]=> st'}\\ \texttt{st' =[ }c_2\texttt{ ]=> st''}\end{array}}{\texttt{st =[ }c_1\texttt{;}c_2\texttt{ ]=> st''}}\ \text{(E\_Seq)}$$

$$\frac{\begin{array}{c}\texttt{beval st b = true}\\ \texttt{st =[ }c_1\texttt{ ]=> st'}\end{array}}{\texttt{st =[ if b then }c_1\texttt{ else }c_2\texttt{ end ]=> st'}}\ \text{(E\_IfTrue)}$$

$$\frac{\begin{array}{c}\texttt{beval st b = false}\\ \texttt{st =[ }c_2\texttt{ ]=> st'}\end{array}}{\texttt{st =[ if b then }c_1\texttt{ else }c_2\texttt{ end ]=> st'}}\ \text{(E\_IfFalse)}$$

$$\frac{\texttt{beval st b = false}}{\texttt{st =[ while b do c end ]=> st}}\ \text{(E\_WhileFalse)}$$

$$\frac{\begin{array}{c}\texttt{beval st b = true}\\ \texttt{st =[ c ]=> st'}\\ \texttt{st' =[ while b do c end ]=> st''}\end{array}}{\texttt{st =[ while b do c end ]=> st''}}\ \text{(E\_WhileTrue)}$$

```
Inductive ceval : com -> state -> state -> Prop :=
  | E_Skip : forall st,
      st =[ skip ]=> st
  | E_Asgn  : forall st a n x,
      aeval st a = n ->
      st =[ x := a ]=> (x !-> n ; st)
  | E_Seq : forall c1 c2 st st' st'',
      st  =[ c1 ]=> st'  ->
      st' =[ c2 ]=> st'' ->
      st  =[ c1 ; c2 ]=> st''
  | E_IfTrue : forall st st' b c1 c2,
      beval st b = true ->
      st =[ c1 ]=> st' ->
      st =[ if b then c1 else c2 end]=> st'
  | E_IfFalse : forall st st' b c1 c2,
      beval st b = false ->
      st =[ c2 ]=> st' ->
      st =[ if b then c1 else c2 end]=> st'
  | E_WhileFalse : forall b st c,
      beval st b = false ->
      st =[ while b do c end ]=> st
  | E_WhileTrue : forall st st' st'' b c,
      beval st b = true ->
      st  =[ c ]=> st' ->
      st' =[ while b do c end ]=> st'' ->
      st  =[ while b do c end ]=> st''

  where "st =[ c ]=> st'" := (ceval c st st').
```

# Why formalize programming languages?

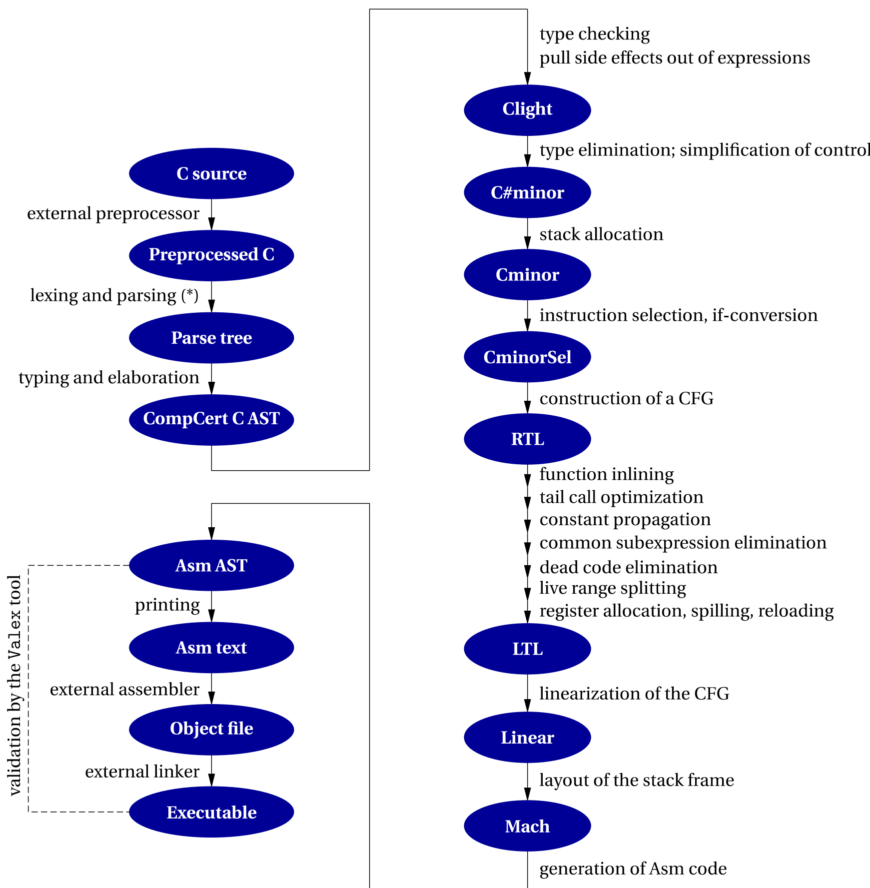
## CompCert C compiler

verified in Coq to compile correctly

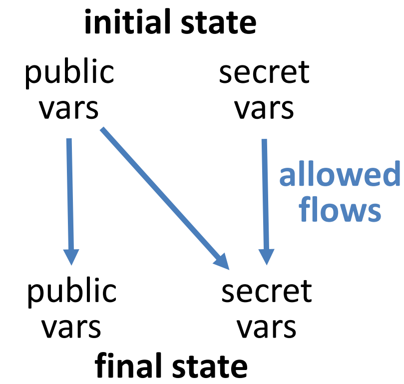
- each language given a semantics
- transformations and optimizations
  - implemented as pure functions
  - proved to preserve semantics

Cezar, Sebastian, Cătălin, et al building secure compilers in Coq and F*

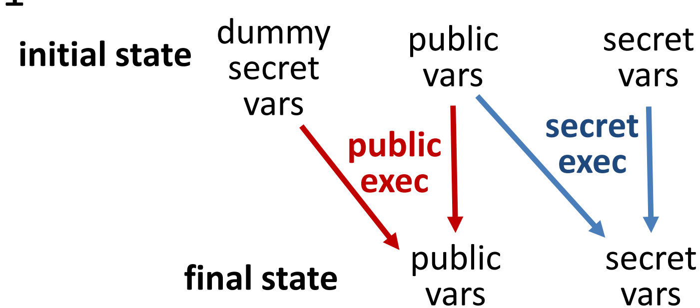
- including a secure variant of CompCert
- a secure compiler from F* to OCaml
- side-channel defenses, etc.



# Secure information flow




- **What does it mean that a program doesn't leak secrets?**
- **Noninterference** (for imperative programs):
  – secrets don't flow from secret variables to public variables
  – executing the program twice with different initial values for the secret variables produces two final states whose public variables are still equal
- **Can be enforced statically by simple type system**
  – Prevent explicit flows (simple taint tracking): public := 2*secret + 1
  – Prevent implicit flows, via the control flow of the program:
  **if** secret=0 **then** public := 0 **else** public := 1
- **Can also be enforced dynamically**
  – e.g. Secure Multi-Execution

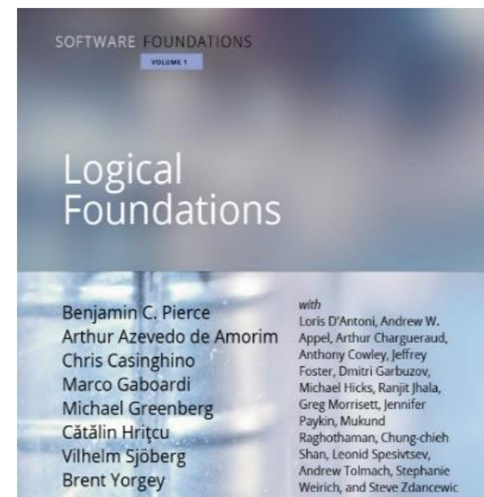


# In this course you will learn ...

- **to write purely functional programs in Coq**
  - natural numbers, lists, maps, trees, program syntax
- **to verify these programs by proving theorems about them**
  - case analysis, induction, inversion, proof automation, ...
- **about the Curry-Howard correspondence** (optional for BSc)
  - proofs = typed purely functional programs
- **to formalize a simple imperative programming language**
  - syntax and operational semantics (evaluation derivations)
- **about secure information-flow**
  - enforcing noninterference statically or dynamically

# This course is very hands on

- **Coq is your personal proof assistant**
  - Taking gamification to the next level!
- **Course based on two textbooks**
  - lots of exercises in Coq
  - our Logical Foundations version available at: https://mpi-sp-pap-2024-25.github.io/book-lf
- **You will understand proving**
  - It's not hard, it's just programming!
  - If you like programming, you will like formal proofs too!
- **Advice: ask questions, interact**

# Lecture logistics

- 14 lectures: roughly first 1/2 Clara, second 1/2 Cătălin
- Vacation 21 Dec to 6 Jan, so no lecture, no tutorials
- We hope for a mostly in-person course
  - **So please attend physically whenever possible!**
  - When you **really cannot** attend physically you can use Zoom or watch the recording

# Exercises

- **This is a very hands on course, so exercising <u>strongly</u> recommended**
  - you will learn the most by writing programs and proofs in Coq
  - very strong correlation between exercise scores and exam scores
- **Exercises count for up to 20% of bonus points**
  - not required to do the optional exercises; they don't count for grade
- **New exercise sheet will be released on Moodle after most courses**
  - 1(-2) Coq file(s) with holes you have to fill (basically 1-2 book chapters)
  - there will be around 10 exercise sheets in total
- You have to **turn in your solution on Moodle before next course**
  - up to Wednesday at 23:59
- **Exercises are individual, please don't share solutions in any way!**
- **Exercises highly recommended even if you're not taking this for credit**

# Tutorials: Q&A about the exercise sheets

- **TAs:** Cezar Andrici and Sebastian Harwig
- **Wednesday 14:15-15:45**
- You can come and ask existing questions
  - Can also ask about old assignments, but solutions anyway on Moodle
- You can also work on your own during tutorials
  - and ask questions as they arise
- If you manage to solve an exercise sheet and don't have any questions, then no problem, you are not forced to come
- Zoom participation in Q&A sessions possible (same Zoom room)
  - if you cannot make it in person, but in-person participants get priority

# Exams

- **Midterm exam** (optional)
  - practice for the final exam
  - also written, on paper
  - duration: 60 minutes
  - bonus points: up to 10%
  - date: Thu, 28 November
    - time: 10:30-11:30

- **Final exam**
  - written, on paper
    - so we will also teach you how to write down proofs informally
  - duration: 120 minutes
  - 100% of the grade
  - date: 21 February
  - re-exam: 26 March

# Credit and grade

## CASA PhD students

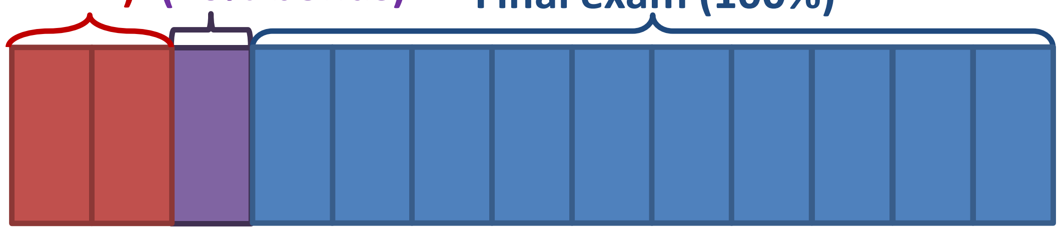
- attendance requirement is 50% of the exercises score

## You can get credit if you study

- Computer Science BSc and MSc
- IT Security BSc* and MSc

  *ITS BSc only as "free elective" (sorry!)
- Mathematics MSc

**Exercises (20% bonus)** **Midterm (10% bonus)** **Final exam (100%)**



**Adding up everything, you need 50% to pass and get credit,**

**and you need at least 95% to get highest grade**

# Follow up course next semester

## Foundations of Programming Languages, Verification, and Security

# Foundations of ...

- **Programming Languages**
  - formalize simple imperative and functional languages in Coq
  - type systems, program transformations, simple compilers
  - semantics, metatheory (e.g. type safety of the language)
- **Verification**
  - Hoare Logic: verify imperative programs
    - against logical / functional specifications
  - Relational Hoare Logic: program equivalence and security

- **Security**
  - Preventing timing side channels
    - for crypto code: constant time, speculative constant time
    - for arbitrary programs: relative security

# BACKUP SLIDES

# Logics and proofs

Q: How do we know something is true?

A: We prove it

Q: How do we know that we have a **proof**?

A: We need to know what it means for something to be a proof.
First cut: A proof is a “logical” sequence of arguments, starting from some initial assumptions

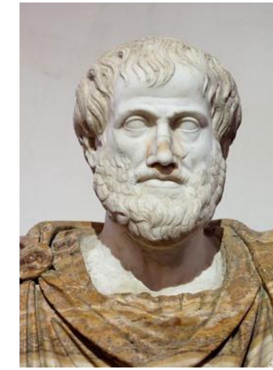
Q: How do we agree on what is a **valid** sequence of arguments?
Can any sequence be a proof? E.g.

All humans are mortal

All Greeks are human

Therefore I am a Greek!

A: No, no, no! We need to think harder about **valid** ways of reasoning...

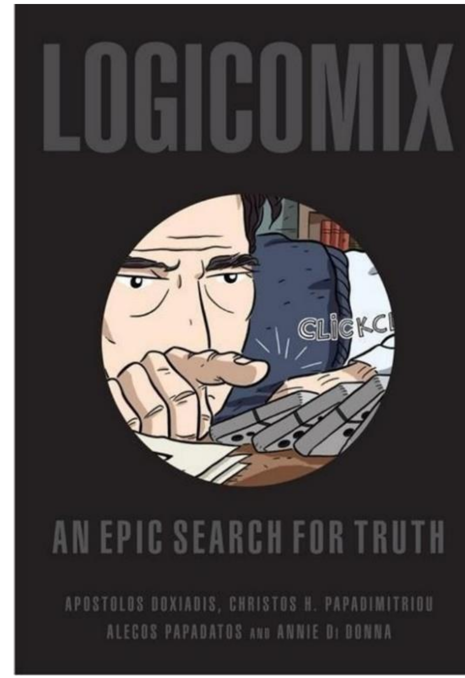


Aristotle
384 – 322 BC

Euclid
~300 BC

# The story of logics is quite fascinating

- **David Hilbert** (1862 – 1943)
- **Gottlob Frege** (1848-1925)
  - first-order logic (∀x. P x /\ R x x)
  - derive the laws of arithmetic from first principles (Hilbert's 2nd problem)
- **Bertrand Russell** (1872 - 1970)
  - paradox, inconsistency in Frege's logical system, tried to fix it
  - Principia Mathematica
- **Kurt Gödel** (1906 - 1978)
  - incompleteness theorems




- **Gerhard Gentzen** (1909-1945) -- consistency of arithmetic
- **Alonzo Church** (1903 - 1995) -- lambda calculus, simple theory of types
- **Alan Turing** (1912 - 1954) -- undecidability of arithmetic, halting problem
- ...

# Logics and computer science

- Logics is a foundation of **mathematics** and **computer science**
  - **precise proofs** with respect to **valid inference steps/rules**
- **Logics and CS greatly influenced each other**, e.g.:
  - **automated theorem provers** (e.g., SAT and SMT solvers)
  - **model checkers**
  - **proof assistants**: Coq, Isabelle, HOL family, F*, ACL2, etc.
    - interactively constructed, machine-checked proofs
    - addictive, gamification of proofs

# Logics and proofs

- Foundation of **mathematics** and **computer science**
  - **formal proofs** with respect to **valid inference steps/rules**
- **This course: typed constructive higher-order logic**
  - **higher-order**
    - can quantify not only over individuals (∀x:nat. P x)
    - but also over propositions (∀P:Prop. P),
      predicates (∀Q:nat->Prop. Q x), relations (∀R:nat->nat->Prop. R x y),
      functions (∀f:nat->bool, ∀g:(nat->bool)->bool), ...
  - **constructive**, aka **intuitionistic logic**:
    - a proposition is valid if one can construct a proof
    - philosophically rejects excluded middle (P ∨ ¬P, classical logic)
  - **typed** (dependently typed)

# Verifying realistic <u>imperative</u> programs

- **the seL4 OS microkernel** (Isabelle/HOL), **the CertiKOS hypervisor** (Coq)
- **EverCrypt cryptographic library** (F*) -- shipping in Firefox and the Linux kernel
- **EverParse3D verified binary parsers** (F*) -- shipping in the Windows kernel
  - Cătălin, Cezar, et al designing and using F* proof assistant, which combines features of Coq (dependent types) with Hoare Logic verification (Dijkstra monads)
- **Libjade high-assurance post-quantum crypto library** (EasyCrypt, MPI-SP et al)
  - EasyCrypt proof assistant using <u>Probabilistic</u> Relational Hoare Logic (MPI-SP et al)
- **Verification of smart contracts** (Coq, F*, ...)
  - Clara, Jana, and their groups at MPI-SP working on this hot topic