Foundations of Programming Languages, Verification and Security (SoSe 2025)

Lecturers: Jana Hoffman and Cătălin Hrițcu

Teaching assistants: Federico Badaloni and Yonghyun Kim

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

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)

λ

- **Verification**
  - Hoare Logic: verify imperative programs
  - Relational Hoare Logic: program equivalence and security

- **Security**
  - Information flow control: preventing direct + indirect leaks
  - Preventing timing side channels for crypto code: cryptographic constant time, speculative constant time

Why formalize programming languages?

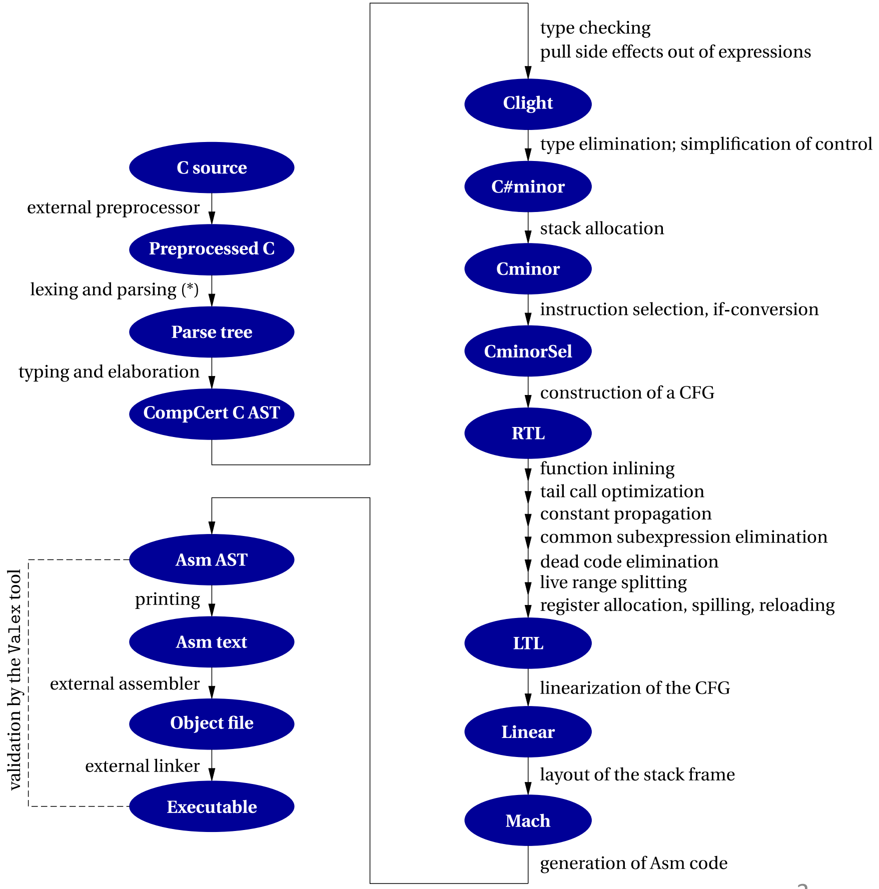
CompCert C compiler

verified in Coq to compile correctly

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

Cătălin's group building secure compilers

- including a secure variant of CompCert
- inaugural lecture on April 30, at 4pm



C source

external preprocessor

Preprocessed C

lexing and parsing (*)

Parse tree

typing and elaboration

CompCert C AST

type checking
pull side effects out of expressions

Clight

type elimination; simplification of control

C#minor

stack allocation

Cminor

instruction selection, if-conversion

CminorSel

construction of a CFG

RTL

function inlining
tail call optimization
constant propagation
common subexpression elimination
dead code elimination
live range splitting
register allocation, spilling, reloading

LTL

linearization of the CFG

Linear

layout of the stack frame

Mach

generation of Asm code

Asm AST

printing

Asm text

external assembler

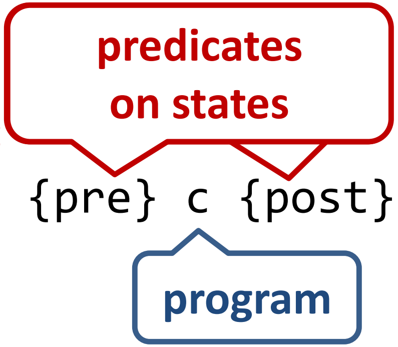
Object file

external linker

Executable

validation by the Valex tool

- **The CompCert compiler is a purely functional program in Coq**
  - verification of purely functional programs often much easier
  - yet some programs are hard to implement efficiently in functional languages (e.g. crypto, operating systems)
- **Verifying imperative programs**
  - proving formally (in Coq) that an **imperative program** satisfies a **functional/logical specification**
  - **Hoare Logic specifications, in terms of pre- and post-conditions :**
    - if the **pre-condition** holds for the initial state, then running the program will produce a final state satisfying the **post-condition**
  - **Relational Hoare Logic specifications:**
    - relating 2+ executions: information flow properties (more on next slide)
    - relating 2+ programs: program equivalence (compiler optimizations)

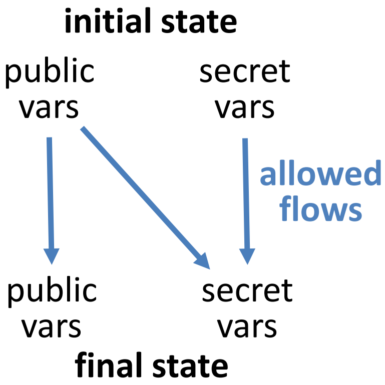


$$\{rel\_pre\}\ c_1 \sim c_2\ \{rel\_post\}$$

relations on states

Secure information flow

- **What does it mean that a program doesn't leak secrets?**
- **Noninterference** for simple imperative programs:
  - secrets don't flow from secret variables to public variables (assumed observable)



  - Formally: executing the program twice with different initial values for the secret variables produces two final states whose public variables are still equal
- **Can be e.g. enforced statically by simple type system (more today)**

More realistic leakage models for information flow

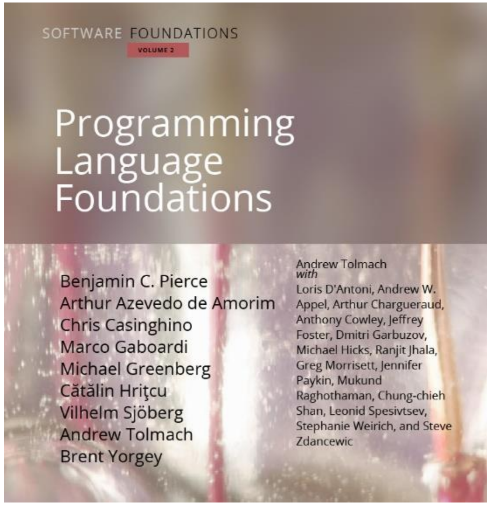
- **Cryptographic constant time** (i.e. secret independent timing)
  - **widely-used programming discipline for writing cryptographic code** without leaking secrets via obvious cache side channels:
    - no secret-dependent branches and no secret-dependent memory accesses
  - the obtained guarantees formalized as a **variant of noninterference**
  - can also be checked by a **simple type system**
- **Speculative constant time** (MPI-SP folks, including Jana and Cătălin)
  - Spectre: constant time code can still leak because of speculative execution
  - **Stronger noninterference variant** that prevents leaks in speculative executions
  - **Speculative load hardening** transformation enforcing this security property

SPECTRE

This course is very hands on

- **Coq proofs can be lots of fun!**
- **Course based on two textbook volumes**
  - lots of exercises in Coq
  - our book versions linked from Moodle:
    https://mpi-sp-foe-2025.github.io/book-plf
    https://mpi-sp-foe-2025.github.io/book-secf

- **Prerequisite: Proofs are Programs**
  - Having attended the course last semester
  - or knowing to use Coq and having (self-)studied the Logical Foundations book:
    https://mpi-sp-pap-2024-25.github.io/book-lf



SOFTWARE FOUNDATIONS
VOLUME 2

Programming Language Foundations

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

Andrew Tolmach
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, Stephanie Weirich, and Steve Zdancewic

SOFTWARE FOUNDATIONS
VOLUME 7

Security Foundations

Cătălin Hrițcu
Roberto Blanco

with contributions from
Santiago Arranz Olmos, Gilles Barthe, Lionel Blatter, Léon Ducruet, Sebastian Harwig, Yonghyun Kim, and Jérémy Thibault

Lecture logistics

- 13 lectures: roughly first 1/2 Jana, second 1/2 Cătălin
  - exceptions: first lecture Catalin, before midterm Rob on RHL
- Pentecost Vacation 9-13 June, 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 (see Moodle)
- **Join on Moodle for all materials**
  - **If external to RUB create account with any email address**
- **Advice: ask questions, interact during the lecture**

Exercises

- **Solving 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
  - **highly recommended even if you're not taking this for credit**
- **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**
  - 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 <u>11:59 AM</u> (new time, right before noon!)**
- **Exercises are individual, please don't share solutions in any way!**
- **Using generative AI to solve homework exercises is not allowed!**

Tutorials: Q&A about the exercise sheets

- **TAs:** Federico Badaloni and Yonghyun Kim
- **Tuesdays at 10:15-11:45**
- 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

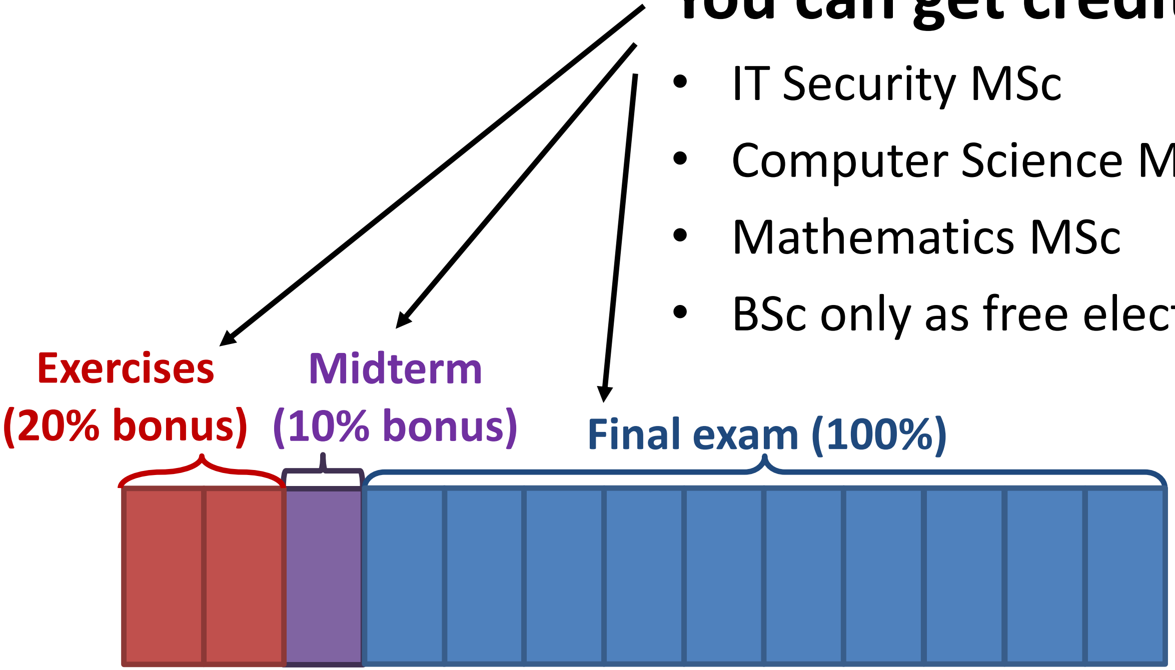
- **Midterm exam** (optional)
  - practice for the final exam
  - also written, on paper
  - duration: 60 minutes
  - bonus points: up to 10%
  - date: Wed, **28 May**
    - time: 14:30-15:30
    - usual lecture slot,
just in larger lecture hall

- **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: **8 August**
  - re-exam: **16 September**

Credit and grade

You can get credit if you study

- IT Security MSc
- Computer Science MSc
- Mathematics MSc
- BSc only as free elective /



Adding up everything, you need 49.01% to pass and get credit,

and you need at least 94.01% to get highest grade