

CS-550

**Synthesis, analysis and verification**

Cursus	Sem.	Type
Computer and Communication Sciences		Opt.
Computer engineering minor	E	Opt.
Computer science	MA2	Opt.
SC master EPFL	MA2, MA4	Opt.

Language of teaching	English
Credits	6
Session	Summer
Semester	Spring
Exam	During the semester
Workload	180h
Weeks	14
<b>Hours</b>	<b>6 weekly</b>
Courses	2 weekly
Exercises	2 weekly
Project	2 weekly
<b>Number of positions</b>	

**Remark**

pas donné en 2017-18

**Summary**

The course presents theory, algorithms, and tools for reasoning about computer systems, including techniques for software and hardware verification and synthesis.

**Content**

Motivation:

Tools for automated analysis and verification of software can improve reliability of software that we use every day. The underlying techniques are also used for compiler optimizations and program understanding. In recent years, new algorithms and combinations of existing techniques have made such tools more effective than in the past. This course will give an overview of basic techniques, as well as the recent advances that made this progress possible. In many years the course also contains guest lectures presenting recent research results.

Topics covered include:

- Logic and relational program semantics
- Verification condition generation and Hoare logic
- Synthesis of programs from relations
- Abstract interpretation and data flow analysis
- Predicate abstraction
- Modular verification
- Decision procedures, SMT solvers, and resolution-based provers

**Learning Prerequisites****Required courses**

Theoretical computer science and discrete mathematics course, or equivalent background and fluency in discrete mathematics and introductory theoretical computer science concepts (e.g. M. Sipser textbook)  
Functional programming in Scala, or ability to pick up Scala quickly (students knowing Haskell or ML generally have no trouble).

**Recommended courses**

The knowledge of mathematical logic and combinatorial optimization is beneficial

### Learning Outcomes

By the end of the course, the student must be able to:

- Formalize program correctness
- Prove correctness of programs on paper
- Sketch an automated verification algorithm
- Interpret results of verification systems
- Create a simple program verifier
- Construct a constraint solver
- Systematize approaches to software correctness
- Choose an appropriate technique for improving software reliability

### Transversal skills

- Assess progress against the plan, and adapt the plan as appropriate.
- Respect the rules of the institution in which you are working.
- Demonstrate a capacity for creativity.
- Make an oral presentation.
- Summarize an article or a technical report.
- Write a scientific or technical report.
- Communicate effectively with professionals from other disciplines.
- Identify the different roles that are involved in well-functioning teams and assume different roles, including leadership roles.

### Teaching methods

Ex catedra

Exercise sessions

Practical work on projects under supervision of teaching assistants

### Expected student activities

Attending lectures

Exercises in class

Homeworks

Mid-term exam

Practical project on modifying a verification system

### Assessment methods

- 30% common project in first part of semester (in stages and feedback after each, but grade only after all of them)
- 40% quiz in 2nd part of semester
- 30% individual projects by the project deadline

### Supervision

Office hours	Yes
Assistants	Yes

Forum                      Yes

## Resources

### Bibliography

**The Calculus of Computation: Decision Procedures with Applications to Verification.** **Bradley**, Aaron R., **Manna**, Zohar, Springer, 2007. ISBN 978-3-540-74113-8.

### Ressources en bibliothèque

- [The Calculus of Computation / Bradley](#)

### Notes/Handbook

<http://lara.epfl.ch/w/sav13:top>

### Websites

- <http://lara.epfl.ch/w/sav>

### Videos

- [https://www.youtube.com/watch?v=rm\\_kqt61JQ8](https://www.youtube.com/watch?v=rm_kqt61JQ8)