What is going to be taught?

Using and applying so-called « formal methods »
- Prototype programs/systems (functional programming)
- Define the expected properties of the programs (logical formulas)
- Use tools to
  - prove that the properties are true and, otherwise,
  - automatically find counterexamples to the properties
- (Bonus) Export prototypes into the Scala programming language
- (Bonus) Integrate the verified Scala programs into Java applications

Why teaching this? What are the motivations?

Why teaching this in Master 1 of computer science?
- ACO=: programming in the large
- ACF=: programming in the small
- Because finding the right solution to a problem at the first attempt is almost impossible (TP0):
  - We should be able to rapidly prototype a solution
  - We should be able to rapidly detect bad solutions
  - We should be able to know why the solution is bad (counterexamples)
  - We should have guarantees on the solution when no counterexample is found (proof)
- Because it is commonly used in high-technology software industry
- Because it is mature to become popular

Who claims that teaching formal methods is important?

- All the world leading universities in computer science:
  (MIT, Stanford, Berkeley, Rice, . . ., Oxford, Cambridge, ETH Zürich, EPFL, TUM München, . . .)
- High-tech critical software industry:
  (Amazon, Microsoft, NASA, Intel, Airbus, RATP, Gemalto, . . .)
  (In Rennes : DGA, Orange, Mitsubishi, Technicolor, . . .)
- All your Master teachers
- Me!
  - 20 years of research on formal verification:
    - developing verification tools
    - modelization and verification of industrial systems
  - Experience of industrial use of formal methods:
    - Orange
    - Technicolor
Some companies using formal methods

- Transportation
  - line 14 of Paris subway (RATP)
  - primary flight control software of A340 and A380 (Airbus)
  - AILS (Airbone Information for Lateral Spacing) (NASA)

- Army: military secured communication software (DGA)

- Security for Communications/Trading/Banks
  - online payment protocols (Orange)
  - cloud computing (Amazon Web Services)
  - blockchain applications (CEA/Ethereum/Tezos/Legicash/Symbiont)

- Consumer software
  - processors (Intel)
  - Windows drivers, secured web protocols (Microsoft)
  - Smartcards, Java cards (Gemalto, Fime)
  - home networks, secured movie editing devices (Technicolor)

Objectives of ACF

- (Re)-introduce functional programming
  - Good and fast prototyping/modeling language
  - Renewed programming paradigm (Ocaml, F#, Scala)

What Languages Are Associated with the Highest Salaries Worldwide?

<table>
<thead>
<tr>
<th>Language</th>
<th>Global</th>
<th>United States</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>F#</td>
<td>$74,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ocaml</td>
<td>$73,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clojure</td>
<td>$72,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groovy</td>
<td>$72,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perl</td>
<td>$69,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rust</td>
<td>$69,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erlang</td>
<td>$67,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scala</td>
<td>$67,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How does ACF relates to other M1/L3 courses?

- ACO (M1)
  - ACF is only about programming in the small
  - ACF focuses on the validity of a solution/program

- MVFA (M1)
  - Restricted to the verification of (large) finite models in MVFA
  - Verification of properties is always automatic in MVFA

- LOG (L3)
  - Same core logical language as LOG, extended in ACF
  - ACF does not focus on proofs
  - Automation of many aspects of LOG

- PRG2 (L3)
  - Functional programming instead of imperative (Why)
  - More complex programs in ACF
  - . . . and more complex properties that you can prove!
  - Integration of verified code in Java project

Evaluation

- Terminal exam (1/2 of the final mark)
- 3 projects (1/2 of the final mark)
  - Model/prototype a software using functional programming
  - Define the expected properties of the software using logic
  - Check that the software satisfies the properties
  - Export a Scala program corresponding to the model
  - Integrate it into a Java program
Practical work (demo)

- Model/prototype a program \((\text{demo.thy} + \text{solTP0.thy})\)
- Formally define its property using logic formula
- Search for counterexamples
- Prove the property
- Export an executable version in Scala
- Integrate into a Java program \((\text{CM1 project + TP0Isabelle})\)

Tools to be used (all freely available):
- Isabelle/HOL (2020)
- Scala 2.13 with SBT (Scala Build Tool)
- IntelliJ bundle/Visual Studio/YourFavoriteEditor

Instructions: http://people.irisa.fr/Thomas.Genet/ACF/#tools

Lessons and practical classes (expected) schedule

- Propositional and first order logics
- Terms, functions and types
- Recursive functions
- Interactive proof basics
- Scala
- Specification and verification methodology
- Principles of verification techniques

0...done
1 Propositional and first order logic
2 and 3 Logic, recursive functions, interactive proof
4 Scala
5 and 5bis Project 1: “Glob” simplification
6 and 7 Project 2: Certified static analyzer
8, 9 and 10 Project 3: Price negotiation (web) applet

What about « Travaux dirigés »?

No classical « Travaux dirigés » (TD)!
- ACF has only lessons and practical classes!
- You have to ask questions during the lessons (CM/TP)
- You have to ask Isabelle/HOL many questions and at any time
- You have to ask questions on ACF’s Moodle
- You have to read Isabelle/HOL and Scala documentation

You can ask questions during « open » TDs:
- Send questions on Moodle’s forum
- If some questions need more interaction, I schedule a TD slot
- When all questions have been answered, the TD is over