ACF : Analyse et Conception Formelles

Introduction

Thomas Genet

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, Thales, Gemalto, . . .)
  (In Rennes : DGA, ANSSI, 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)
- Consumer software
  - processors (Intel)
  - Windows drivers, secured web protocols (Microsoft)
  - Smartcards, Javacards (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)
  - Proofs are far easier on a functional program than on an imperative one (*e.g.* Why3 in ProgC)
- Use logic to formally define the properties of a software
  - « The most precise, concise and expressive programming language »
  - Proving one formula can replace infinitely many tests !
  - Testing one formula can replace thousands of tests !

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
- ProgC (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 (`demo.thy` + `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 (`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

7 lessons :

1. Propositional and first order logics
2. Terms, functions and types
3. Recursive functions
4. Interactive proof basics
5. Scala
6. Specification and verification methodology
7. Principles of verification techniques

12 practical classes :

- 0 ... done
- 1 Propositional and first order logic
- 2 and 3 Logic, recursive functions, interactive proof
- 4 Scala, evaluators and random testing
- 5 Project 1 : Equivalence of security policies
- 6 and 7 Project 2 : Certified static analyzer
- 8 and 9 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