Formalisation of mathematical proofs in Lean (Laboratorio di formalizzazione di dimostrazioni matematiche tramite il software Lean)

Instructors: Alessandro Iraci, Davide Lombardo, Marcello Mamino

2023-2024
(first semester)

1 Contents

The course aims to introduce Lean, a software based on ideas from type theory that can be used to formalise even very advanced mathematical proofs. Lean is both a programming language and a language for specifying mathematical arguments in an unambiguous way. After covering some preliminaries, necessary to understand the way statements and proofs are formalised in Lean, the course will focus on the practical use of the language, relying also on *mathlib*, a constantly growing, publicly available library of formal proofs.

2 Practical information

The course lasts 20 hours and takes place in the first semester. It is essentially a laboratory: students are encouraged to bring their own laptops and will spend some time in each lecture formalising various mathematical proofs with the help of the lecturers.

For the final assessment, students will be asked to formalise a non-trivial result (e.g. a theorem from one of their first or second year courses). There will be an oral examination in which the student will be asked to explain parts of their proof/code.

The course has no prerequisites other than familiarity with the contents of the first year courses.

For enquiries please contact davide.lombardo@unipi.it, alessandro.iraci@unipi.it or marcello.mamino@unipi.it.

3 Highlights

This is what a Lean proof done with bare hands might look like (this shows the equivalence $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$):

```
variable (p q r : Prop)

-- distributivity
example : p ∧ (q ∨ r) ↔ (p ∧ q) ∨ (p ∧ r) := by
  constructor
  · intro hmp
    have h1 := hmp.1
    rcases hmp.2 with hq | hr
    · left
      exact ⟨ h1, hq ⟩
    · right
      exact ⟨ h1, hr ⟩
  intro hmpr
  rcases hmpr with hpq | hpr
  · constructor
    exact hpq.1
    left
    exact hpq.2
  · constructor
    exact hpr.1
    right
    exact hpr.2
```

For a perhaps more readable and probably more intriguing example, here is the barber paradox:

```
axiom Man : Type                        -- create mankind
axiom shaves : Man → Man → Prop         -- give men the ability to shave men
infix:100 " shaves "  => shaves         -- use infix notation for convenience

example: ¬ ∃ barber : Man , ∀ guy : Man , barber shaves guy ↔ ¬ guy shaves guy :=
  fun ⟨ barber, hp ⟩ =>                 -- get a barber that satisfies the hypothesis
    have ⟨ r_arr, l_arr ⟩ := hp barber -- apply hypothesis to the barber himself
    have ns := fun x => r_arr x x       -- deduce that the barber doesn't shave himself
    ns (l_arr ns)                       -- hence he does --> contradiction
```