Introduction to Interactive Proof of Software

J.-F. Monin

Univ. Joseph Fourier and
LIAMA-FORMES, Tsinghua Univ., Beijing

2012, Semester 1

Lecture 1

Outline

Outline

Software Engineering among Engineering

Classical engineering:
for bridges, airplane wings, electric or chemical devices

Engineering

- Heavy and expensive material
- Continuous phenomena

Software

- Cheap and easy to change
- Discrete phenomena

Easy to change

Software is

- easy to type (to design?)
- easy to debug
- easy to introduce bugs while correcting bugs...

Easy then complex

Continuous/discrete

Analog systems

Changing a little bit the input or to the device makes the output change just a little bit

Mathematical model is continuous

Exhaustive testing easy: check the bounds

Discrete (or digital) systems

Changing a little bit the input or to the program makes the output completely different

Mathematical model is discrete

Exhaustive testing: impossible

Example: Ariane 5, flight 501

Let us see...

Firework (June 4, 1996)

Firework (detail)

What happened

A *software* problem

Yet another story

Syntax: FORTRAN

```
DO 10 I = 1, 10
body of the loop
```

```
DO 10 I = 1. 10
body of the loop
```

Reported to make Mariner 1 lossed

Answers

Quality engineering?

Means: reviews, documentation, processes

Why not, but...

Aircraft industry already implements the strongest quality processes

Answers

Formal Methods

Prove that some piece of software behaves accordingly to a given specification

Boilds down to theorem proving: programs and specifications are represented by logical formulas

Hand waving not allowed

Better programming languages

Programming languages relying on solid foundations

Overview of Formal Methods

Understanding Formal Methods, J.F. Monin, Springer, 2003

- Static analysis
- Model Checking
- Deductive techniques
- Soundness: LCF architecture, proof terms (can be checked independantly)

Trade off

- pencil-paper / tool support
- automatization / generality
- ...

In summary

- Describe a model
- Explain it
- Reason about it
- Be clean and precise

Use math and logic... and make it funny!

Coq teaching in the world

Now routinely taught in many highly ranked universities

- France: Paris, Grenoble, Lyon, Bordeaux, Strasbourg...
- Europe: UK, Italy,...
- USA: Harvard, Yale, U. Pennsyvania, MIT, Princeton...
- Australia
- China: Coq Summer School Tsinghua, Suzhou, Shanghai

Some industrial uses

Spacecrafts, airplanes (Airbus, Boing)

Microsoft
Intel
French railways
Telecom Operators
Nuclear power plants
Banks
Cryptography

Contents of this course

Discover 3 aspects of Coq

1. Coq as a proof assistant
    - write precise and clear definitions
    - how to state meaningful theorems
    - how to prove them in a perfectly rigorous way
      this task is interactive: tedious parts can be discharged by the machine but creative part need input from a human.

Contents of this course

Discover 3 aspects of Coq

2. Coq as a challenging programming language

- many applications of Coq to problems arising in computer science
- even for mathematics: benefits of a computer scientist way of thinking

Contents of this course

Discover 3 aspects of Coq

3. Applications to reasoning about non-trivial programs
   - lists, trees...
   - data-structures implemented with pointers

Expected benefits from this course

Learn basics on successful Formal Methods

Introduction to Coq, one of the major proof assistants

Deep insights on programming languages

Applications to the mathematical modeling of a simple sequential programming language

Introduction to functional programming

Understand rigorous foundations for software

Outline

Organization of the course

Lectures

- $11 \times 3 \times 45mn$
- Theory + practice
- Laptop preferably with Linux (e.g. Ubuntu) with the following software:
  - coq-8.4 (package or `http://coq.inria.fr/download`
  - emacs + proofgeneral-coq (for professionals)
  - or coq-ide should be already included in coq-8.4

Project

- Will start around week 6
- important for evaluation

Scoring

Examinations

- Mid term exam (around week 4): 20%
- Final exam: 30 %
  Most scoring is on typical questions
- Project: 50%

Bonus

- homework
- challenging exercises

Results (from past experience)

- several scores > 97

Outline

Firsts steps to Coq

Make mathematics funny

Key idea: abstraction

- take a concrete expression
- make some value (repeated or not) a parameter
- that's it

But far reaching

The abstract thing can be

- a data, a function, a program, a type
- a family of them
- subtle combinations
  e.g. a program may depend on a previously abstracted value; programs may depend on pgms, or on types, or conversely.

A very powerful logic

- Statements
- Proofs: concrete data
- More powerful than Peano arithmetic: Goodstein sequences

A way to compute proofs for given statements

$\Rightarrow$ Programming comes first

A strange programming language

- Without state!!
- Called functional programming

2 questions

- What can we do with it,
  in particular can we do as much as with states
  in particular can we simulate states
- is it realistic?
  - implementation: exists? efficient?
  - is it easy to use?

A strange programming language

State is a burden for reasoning

Immutable values are much more convenient

All proof assistants are related to a functional programming language

In the case of Coq (and others e.g. Agda, Matita, Lego, Nuprl) the relationship is very tight

Live Demo

Types everywhere

Very powerful types
Everything has a type, even types
We can compute on types and on values at the same time.

Examples: families of types.

- Example: n-tuples, with $n = 1, 2\ldots$ even 0.

... So it will become complex...

We start with a graphical syntax

Types having finitely many values

The simplest are called an enumeration

Example

```
Red : color             Red_f : rgb
Orange : color
Yellow : color
Green : color           Green_f : rgb
Blue : color            Blue_f: rgb
Indigo : color
Violet : color
```

Warning: a value has only one type

What is the type of color and rgb?

```
color : Set
rgb : Set
```

What is the type of Set?

```
Set : Type
```

What is the type of Type?

$\texttt{Type}(i)$: $\texttt{Type}(i+1)$

Graphical syntax

`Red` `color`

`Orange` `color`

`Yellow` `color`

`Green` `color`

`Blue` `color`

`Indigo` `color`

`Violet` `color`

Outline

Graphical syntax

$$\frac{}{\texttt{Red}:\texttt{color}} \qquad \frac{}{\texttt{Orange}:\texttt{color}} \qquad \frac{}{\texttt{Yellow}:\texttt{color}} \qquad \frac{}{\texttt{Green}:\texttt{color}}$$

$$\frac{}{\texttt{Blue}:\texttt{color}} \qquad \frac{}{\texttt{Indigo}:\texttt{color}} \qquad \frac{}{\texttt{Violet}:\texttt{color}}$$

The horizontal bar means: MAKES

`Red`, `Orange`,... are called CONSTRUCTORS

At the same time we get $\frac{}{\texttt{color}:\texttt{Set}}$

Intermezzo: definitions

In order to save space, we use definitions.
E.g. *(Coq syntax)*

```
Definition R := Red.
```

means that `R` is definitionally the same as `Red`.

```
Definition co := color.
```

means that `co` is definitionally the same as `color`.

Hence
`Red:color`, `Red:co`, `R:color` and `R:co`
are all the same judgement

Graphical syntax

```
Definition O := Orange.   Definition Y := Yellow.
Definition G := Green.    Definition B := Blue.
Definition I := Indigo.   Definition V := Violet.
```

$\dfrac{}{\texttt{co}}\,\texttt{R}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{O}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{Y}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{G}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{B}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{I}$ $\quad$ $\dfrac{}{\texttt{co}}\,\texttt{V}$

```
Definition Rf := Red_f.   Definition Gf := Green_f.
Definition Bf := Blue_f.
```

$\dfrac{}{\texttt{rgb}}\,\texttt{Rf}$ $\quad$ $\dfrac{}{\texttt{rgb}}\,\texttt{Gf}$ $\quad$ $\dfrac{}{\texttt{rgb}}\,\texttt{Bf}$

Outline

Using a value

We know how to make (or construct) a value in `color` or in `rgb`.

Next issue: how to use a value

- use a given value
- use a (still) unknown value

Composition of n-tuples

Making a 4-tuple of rgb

$$\frac{\texttt{rgb} \quad \texttt{rgb} \quad \texttt{rgb} \quad \texttt{rgb}}{\texttt{tuple4}}\ \texttt{Mk4}$$

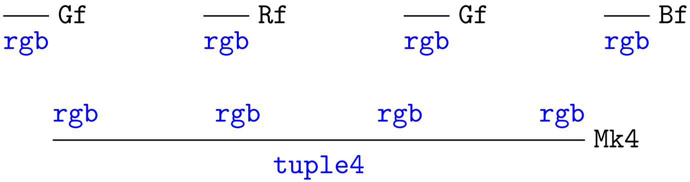
The constructor `Mk4` makes a `tuple4` from

- a `rgb`
- a `rgb`
- a `rgb`
- a `rgb`

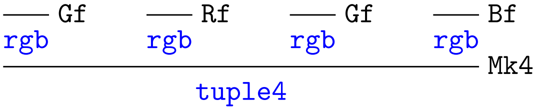
At the same time we get $\frac{}{\texttt{Set}}\ \texttt{tuple4}$

Pluging rgb into Mk4

Building blocks



Connecting them yields the concrete 4-tuple of rgb



Others trees for 4-tuples

$$\dfrac{\dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Bf}}{\texttt{tuple4}}\,\texttt{Mk4}$$

$$\dfrac{\dfrac{}{\texttt{rgb}}\,\texttt{Bf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Bf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Gf}}{\texttt{tuple4}}\,\texttt{Mk4}$$

$$\dfrac{\dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf}}{\texttt{tuple4}}\,\texttt{Mk4}$$

Another view on Mk4

As a building block

$$\frac{\texttt{rgb} \qquad \texttt{rgb} \qquad \texttt{rgb} \qquad \texttt{rgb}}{\texttt{tuple4}}\ \texttt{Mk4}$$

As a tree

$$\frac{\overbrace{\texttt{rgb}}^{x_1} \qquad \overbrace{\texttt{rgb}}^{x_2} \qquad \overbrace{\texttt{rgb}}^{x_3} \qquad \overbrace{\texttt{rgb}}^{x_4}}{\texttt{tuple4}}\ \texttt{Mk4}$$

This is called an open tree

Outline

Closed and open trees

The meaning (or value) of

$$\frac{\dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Bf}}{\texttt{tuple4}}\,\texttt{Mk4}$$

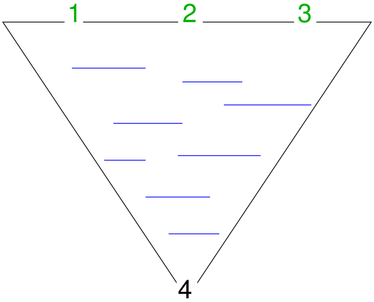
is completely defined: this is called a closed tree.

In contrast, the meaning of the open tree

$$\frac{\dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \overbrace{\texttt{rgb}}^{x_2} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \overbrace{\texttt{rgb}}^{x_4}}{\texttt{tuple4}}\,\texttt{Mk4}$$

depends on x_2 and x_4.

General shape: trees

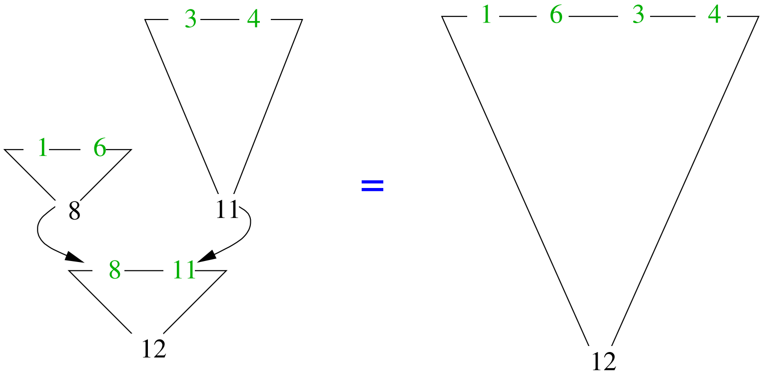


Interpretation

- At positions 1, 2, 3, 4: types
- 1, 2, 3: inputs
- 4: output (or result)

Makes the output from the inputs

Pluging trees



Environment

The tree

$$\dfrac{\dfrac{}{\texttt{rgb}}\;\texttt{Gf} \qquad \overbrace{\texttt{rgb}}^{x_2} \qquad \dfrac{}{\texttt{rgb}}\;\texttt{Rf} \qquad \overbrace{\texttt{rgb}}^{x_4}}{\texttt{tuple4}}\;\texttt{Mk4}$$

has a meaning for all trees plugged into x_2 and x_4.

The variables $x_2 : \texttt{rgb}$ and $x_4 : \texttt{rgb}$ make up the environment of this tree

Outline

More 4-tuples

WRONG

4-tuple of rgb

$$\frac{\frac{}{\texttt{rgb}}\,\texttt{Gf} \quad \frac{}{\texttt{rgb}}\,\texttt{Rf} \quad \frac{}{\texttt{rgb}}\,\texttt{Gf} \quad \frac{}{\texttt{rgb}}\,\texttt{Bf}}{\texttt{tuple4}}\,\texttt{Mk4}$$

4-tuple of color

$$\frac{\frac{}{\texttt{co}}\,\texttt{O} \quad \frac{}{\texttt{co}}\,\texttt{Y} \quad \frac{}{\texttt{co}}\,\texttt{B} \quad \frac{}{\texttt{co}}\,\texttt{V}}{\texttt{tuple4}}\,\texttt{Mk4}$$

Mk4 must be applied to arguments of a given type

How to make 4-tuples more general

Solution 1: have different constructors

$$\frac{\overbrace{\texttt{rgb}}^{x_1}\quad\overbrace{\texttt{rgb}}^{x_2}\quad\overbrace{\texttt{rgb}}^{x_3}\quad\overbrace{\texttt{rgb}}^{x_4}}{\texttt{tuple4}}\;\texttt{Mk4rgb}$$

$$\frac{\overbrace{\texttt{color}}^{x_1}\quad\overbrace{\texttt{color}}^{x_2}\quad\overbrace{\texttt{color}}^{x_3}\quad\overbrace{\texttt{color}}^{x_4}}{\texttt{tuple4}}\;\texttt{Mk4co}$$

$$\frac{\overbrace{\texttt{tuple4}}^{x_1}\quad\overbrace{\texttt{tuple4}}^{x_2}\quad\overbrace{\texttt{tuple4}}^{x_3}\quad\overbrace{\texttt{tuple4}}^{x_4}}{\texttt{tuple4}}\;\texttt{Mk4t4}$$

At the same time we get $\frac{}{\texttt{Set}}\;\texttt{tuple4}$

How to make 4-tuples more general

Remark

Beyond `Mk4rgb`, `Mk4co`, `Mk4t4`, we can imagine hererogeneous 4-tuples, for instance:

$$\frac{\overbrace{\texttt{rgb}}^{x_2}\quad\overbrace{\texttt{color}}^{x_5}\quad\overbrace{\texttt{tuple4}}^{x_3}\quad\overbrace{\texttt{rgb}}^{x_6}}{\texttt{tuple4}}\;\texttt{Mk4het}$$

Many possibilities... to be considered again later.

How to make 4-tuples more general

Solution 2: only one constructor, but more general

$$\frac{\texttt{A} \quad \texttt{A} \quad \texttt{A} \quad \texttt{A}}{\texttt{gtuple4}}\ \texttt{Mk4}$$

But where does `A` come from?

We want the previous tree for all `A`...

The magic

Solution 2: only one constructor, but more general

$$\frac{\overbrace{\texttt{Set}}^{\texttt{A}} \quad \overbrace{\texttt{A}}^{x_1} \quad \overbrace{\texttt{A}}^{x_2} \quad \overbrace{\texttt{A}}^{x_3} \quad \overbrace{\texttt{A}}^{x_4}}{\texttt{gtuple4}}\ \texttt{Mk4}$$

As usual, at the same time we get

$$\frac{}{\texttt{Set}}\ \texttt{gtuple4}$$

Intermezzo: a shorthand for trees

$$\dfrac{\dfrac{}{\texttt{tuple4}}\,\texttt{t1} \qquad \dfrac{}{\texttt{tuple4}}\,\texttt{t2} \qquad \dfrac{}{\texttt{tuple4}}\,\texttt{t3} \qquad \dfrac{}{\texttt{tuple4}}\,\texttt{t4}}{\texttt{tuple4}}\,\texttt{Mk4t4}$$

Where `t1` is for example defined as

$$\dfrac{\dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Rf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\,\texttt{Bf}}{\texttt{tuple4}}\,\texttt{Mk4rgb}$$

And so on for `t2`, etc.

Concrete example

```
—— rgb      ══ u1      ══ u2      ══ u3      ══ u4
Set         rgb        rgb        rgb        rgb
────────────────────────────────────────────────── Mk4
                      gtuple4
```

General homogeneous 4-tuples

$$\dfrac{\dfrac{}{\texttt{Set}}\,\texttt{A} \qquad \dfrac{}{\texttt{A}}\,\texttt{u1} \qquad \dfrac{}{\texttt{A}}\,\texttt{u2} \qquad \dfrac{}{\texttt{A}}\,\texttt{u3} \qquad \dfrac{}{\texttt{A}}\,\texttt{u4}}{\texttt{gtuple4}}\,\texttt{Mk4}$$

The type `A` can be many things beyond `rgb`

- `gtuple4`
- a complex tree

The trees `u1`, `u2`, `u3`, `u4` and `A` can be open
(they can depend on variables).

Exercises

1) Write trees for examples of 4-tuples of 4-tuples using `tuple4` and `gtuple4`.
Some of them, closed, some of them open
E.g. $\langle\langle \mathtt{R}, \mathtt{Y}, \mathtt{B}, \mathtt{B}\rangle, \langle \mathtt{B}, \mathtt{O}, x_4, \mathtt{R}\rangle, \langle x_7, x_7, x_7, \mathtt{V}\rangle, \langle \mathtt{V}, \mathtt{Y}, \mathtt{O}, \mathtt{R}\rangle\rangle$

2) Trees for heterogeneous pairs (2-tuples) and for heterogeneous triples.

3) Trees for homogeneous n-tuples, where n can be 1, 2 or 3.

4) Trees for heterogeneous n-tuples, where n can be 0, 1 or 2.