# Programming = proving?
# The Curry-Howard correspondence today

## Introduction

Xavier Leroy

Collège de France

2018-11-21


COLLÈGE
DE FRANCE
1530

# Computer science and mathematical logic

Since its inception, computer science has taken many ideas and many principles from mathematical logic.

(Church, Turing, von Neumann, and many other founders of computer science were logicians or had beed trained in logic.)

The 2018 lectures are devoted to an especially strong and seminal interaction between logic and computer science, and more precisely between proof theory and programming languages.

# Programming with logic

Many programming languages treat programs as formulas of a logic:

- Logic programming (Prolog, Mercury, ...)
- Constraint programming (Prolog III, CHIP, Oz, ...)
- Queries in relational databases.

Executing the program amounts to proving or refuting the corresponding logical formula.

# An example in Prolog

A Prolog program:

```
append([], L, L).
append([H|T], L, [H|M]) :- append(T, L, M).

?- append(X, Y, [1,2,3]).
```

These Horn clauses and the final query define a logical formula:

$$\begin{array}{rl} & (\forall L, \mathrm{append}([\,], L, L)) \\ \wedge & (\forall H, T, L, M, \mathrm{append}(T, L, M) \Rightarrow \mathrm{append}([H|T], L, [H|M])) \\ \Rightarrow & \exists X, Y, \mathrm{append}(X, Y, [1, 2, 3]) \end{array}$$

Executing the program amounts to proving this proposition by finding appropriate $X, Y$ that satisfy the $\exists$.

# Proposition = program

Logic programming falls within a rather natural correspondence …
which is not the Curry-Howard correspondence:

| programming language | mathematical logic |
|---|---|
| program | proposition |
| execution | proof |

# The Curry-Howard correspondence

An isomorphism between an intuitionistic logic and the simply-typed $\lambda$-calculus (the core of a functional programming language).

| simply-typed $\lambda$-calculus | intuitionistic logic |
|---|---|
| type | proposition |
| term (programme) | proof |
| reduction (execution) | cut elimination (normalization) |

Complete change of viewpoint: programs are no longer propositions, but the proofs of propositions.

# The Curry-Howard correspondence

Inspires a new perspective on logics and on programming languages, summarized by the “PAT principle”:

Propositions As Types

Proofs As Terms

# Programming = proving?

Far from being a coincidence, the Curry-Howard correspondence has developed into a deep structural link between languages and logics, and between programming and proving.

- Correspondences between other, more expressive logics and languages, e.g. second-order arithmetic and polymorphic lambda-calculus.
- Formalisms that can be used both as logics and as programming languages: Martin-Löf type theory, the Calculus of Constructions, ....
- Tools that can be used both as proof assistants and as programming environments: Coq, Agda, ...

# Curry-Howard today

A guiding principle to design, understand, and formalize programming languages (mostly, but not only, functional languages).

A new perspective on classical mathematics: what is the computational contents of a proof?

New ways to program, integrating formal verification better (dependent types, etc).

New ways to do mathematics, leveraging the power of the computer.

Powerful and versatile tools such as Coq and Agda, to help us explore this border between computer science and mathematics.

# Seminar talks

- 28 nov: Pierre-Évariste Dagand, *Les types dépendants: tout un programme!*
- 5 déc: Assia Mahboubi, *Mathématiques assistées par ordinateur*
- 12 déc: Matthieu Sozeau, *Programmer avec Coq: récursion et filtrage dépendant*
- 19 déc: Guillaume Munch-Maccagnoni, *Peut-on dupliquer un objet? Linéarité et contrôle des ressources*
- 16 jan: Alexandre Miquel, *Forcing et réalisabilité: vers l'unification*
- 23 jan: Christine Tasson, *Sémantique des programmes fonctionnels probabilistes, à la lumière de la logique linéaire*
- 30 jan: Thierry Coquand, *Du calcul des constructions à la théorie des types univalente*