Programing in Coq

Guillaume Claret

http://guillaume.claret.me/
πr^2, *PPS*, *INRIA*, Paris 7

EJCP 2013
with Yann Régis-Gianas

A dream

Languages have been more and more typed to ensure correctness:

- assembly
- *C*
- *OCaml*
- subtyping, region systems, implicit complexity, ...
- *W*hyML with annotations in first-order logic
- ...

We want the limit.

A solution: the Curry-Howard correspondence

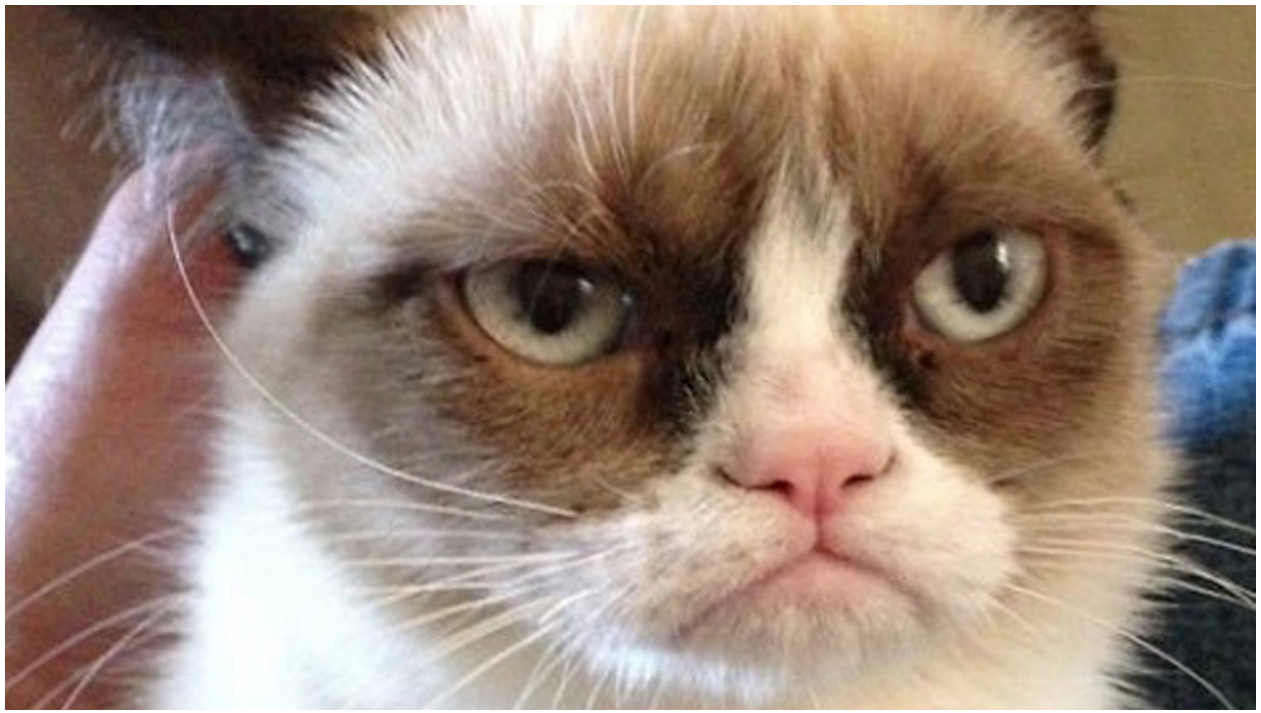
Programming	Logics
type	theorem
program	proof
$A \to B$	$A \Rightarrow B$
$A \times B$	$A \wedge B$
$A + B$	$A \vee B$
dependent types	$\forall x \in A, P(x)$

Coq is a theorem prover based on the Curry-Howard correspondence.

Main feature

In Coq, you can freely combine programs and proofs.

Testimony



"I tried to program in Coq, this was awful."

Problems

- Coq relates logics and λ-calculus, but computers are Turing machines
- programmers do not always want to prove everything, like termination or completeness
- we need speed

The solution: (1) for the user

An effect system:

- representing mutable variables, exceptions, non-termination
- maybe even continuations, co-routines, concurrency, ...
- possible to specify (Hoare logics on mutable variables, ...)
- scalable (modular)
- inferred (as easy as *OCaml*)

(2) The compiler

Compilation to *OCaml* (already exists) or a certified compiler. Key idea:

- definition of the *x86* syntax in Coq
- definition of an `eval` function in Coq (we do not leave the logics)
- `eval` function run JITing the real assembly (full speed)

On top of it, a compilation stack.

For now

A Coq plugin to simplify proof by reflection using the extraction mechanism.

Cybele

More information on: `http://cybele.gforge.inria.fr/`.

Thanks.