BRICKS

Basic Research in Informatics

www.bsik-bricks.nl

A Declarative Proof Language for the Coq Proof Assistant

Pierre CORBINEAU
BRICKS/FOCUS Project 642.000.501
ARPA: Advancing the Real use of Proof Assistants
Foundations team
Institute for Computing and Information Sciences
Radboud Universiteit Nijmegen

What is Coq ?

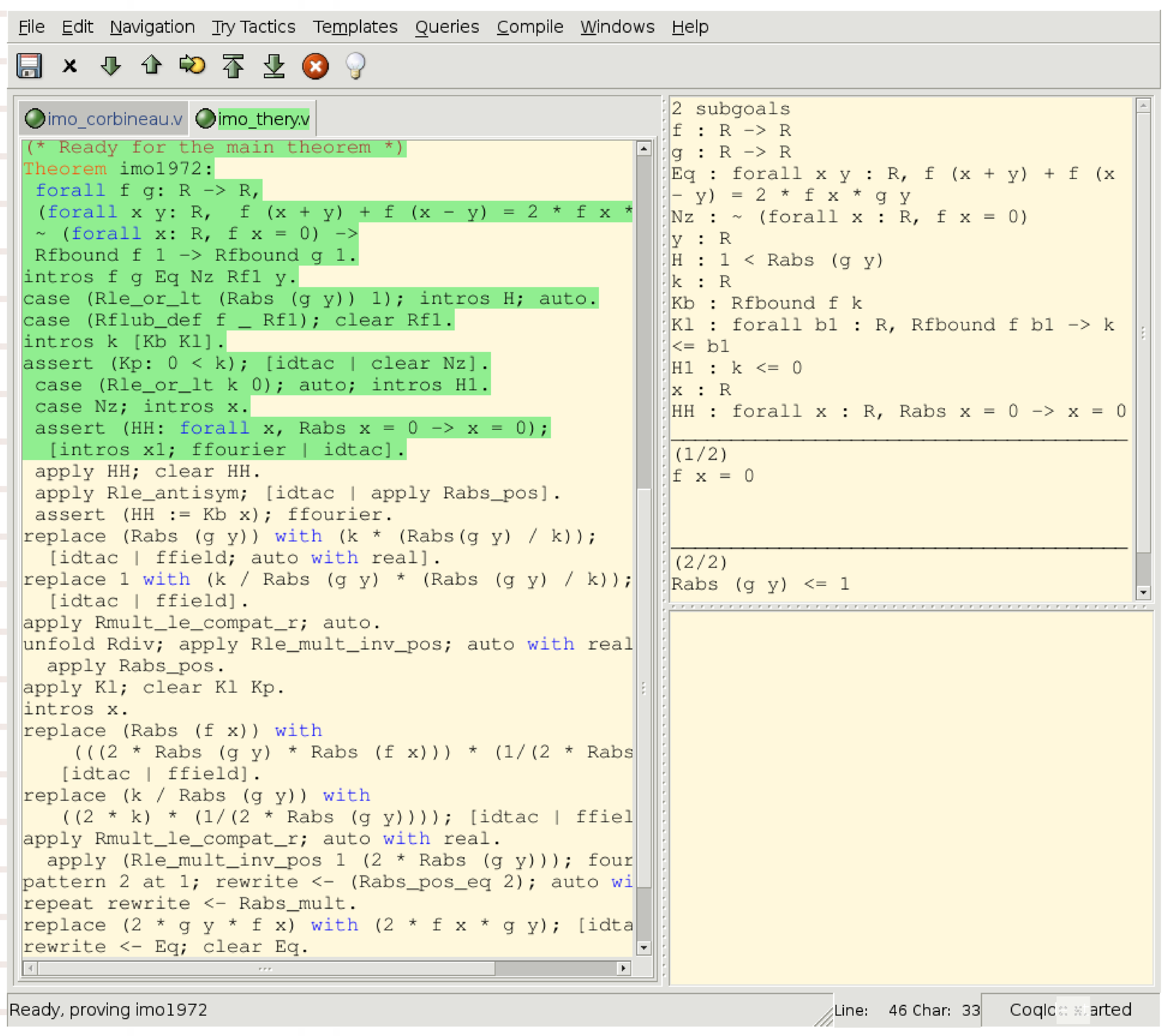
Coq is an interactive proof assistant developed by the LogiCal team at the French INRIA institute. It allows to formally define mathematical objects and helps the user prove properties of those objects. Coq is based on a variant of Type Theory called Calculus of Inductive Constructions. The architecture of the Coq proof assistant ensures the correctness of the proofs written by the user. Coq is mostly used for two kinds of applications :

- the certification of computer systems and software: Javacard virtual machine, certified compilers…
- the formalisation of mathematics : Four color theorem, C-CoRN…

The basic use of Coq follows three steps:

1. Define the objects and axioms used.
2. State a theorem.
3. Provide steps of the proof (proof script) until it is completed.

Finally, a proof object is built, double-checked by the small, trusted *kernel* and saved.



A theorem from the book

Theorem 1 (Int. Math. Olympiads 1972, B2) *Let f and g be real-valued functions defined on the real line such that for all x and y, $f(x+y)+f(x-y)=2f(x)g(y)$. If f is not identically zero and $|f(x)|\leq 1$ for all x, prove that $|g(x)|\leq 1$ for all x.*

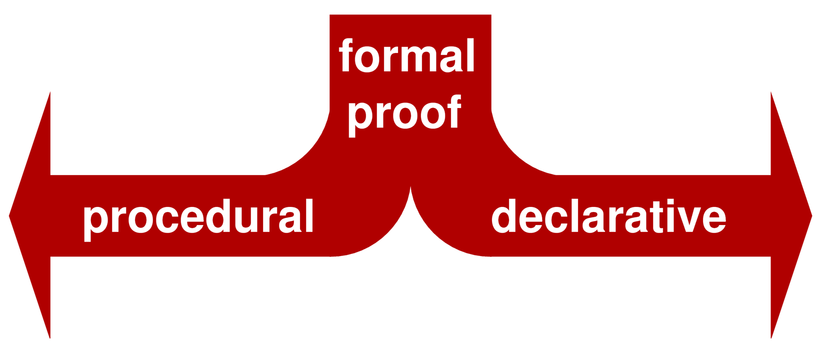
Proof: Let k be the least upper bound for $|f(x)|$. Suppose $|g(y)|>1$. Take any x with $|f(x)|>0$, then

$$\begin{aligned} 2k &\geq |f(x+y)|+|f(x-y)| \\ &\geq |f(x+y)+f(x-y)| \\ &= 2|g(y)||f(x)| \end{aligned}$$

so $|f(x)|<k/|g(y)|$. In other words, $k/|g(y)|$ is an upper bound for $|f(x)|$ which is less than k. Contradiction. □


```
Require Import Reals.

Theorem B2: forall f g : R -> R,
(forall x y,
  f (x + y) + f (x - y) = 2 * f x * g y) ->
~(forall x, f x = 0) ->
(forall x, Rabs (f x) <= 1) ->
(forall x, Rabs (g x) <= 1).
```



On the left hand side, a screenshot of the CoqIDE interface displays (on the left) a part of the proof script for the theorem B2, using the original procedural proof language. The green text has already been processed. The top-right part of the screen displays the proof state. On the right, the same theorem is proved using the new declarative language developped in the ARPA project.

Formalising Mathematics

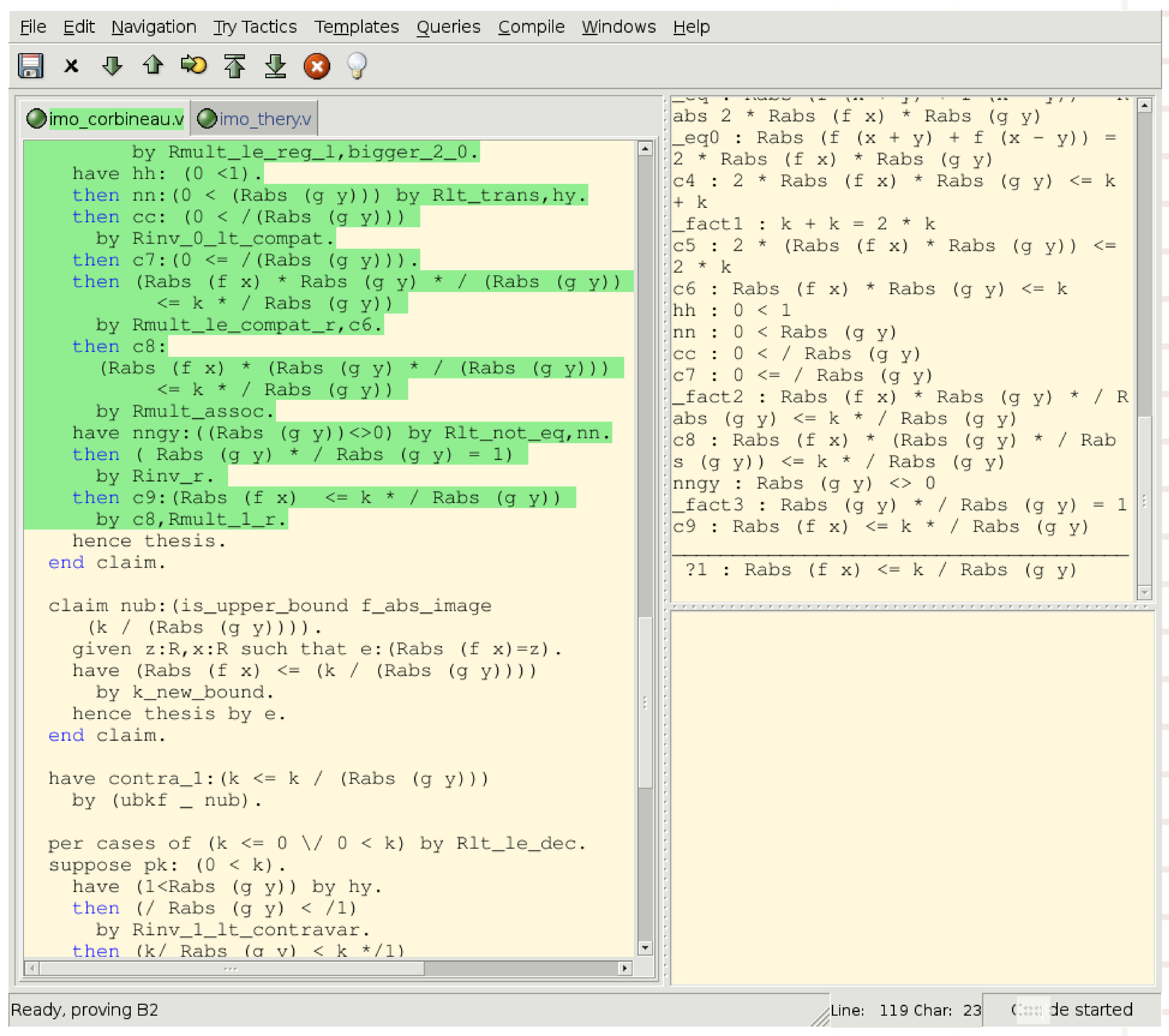
The Foundations Team in Nijmegen uses Coq to develop a library of formalised mathematics known as C-CoRN. It contains *constructive* mathematics.

The process of formalising mathematics is quite cumbersome since textbook mathematics are not written in a formal language. The difficulties arise from the formal system (Type Theory), but also to a much greater extend from the *proof language*.

Those difficulties are the impossiblility to read the proof scripts without running them in Coq, the steep learning curve to for new proof developers, the exotic look of the proof script and the trouble to keep older proofs running for newer Coq versions.

To solve these problems, our approach consists in providing a new proof language for the Coq proof assistant, which needs to be:

- readable (by casual mathematicians)
- accessible (easy to learn)
- natural (like textbook proofs)
- maintainable (resistant to software updates)



Procedural vs. Declarative style

A proof language consists of commands that modify the proof state, which keeps track of what needs proving. A procedural style language emphasises the use of specific methods to proceed, while a declarative style language emphasises the *target* proof state.

On the one hand, the declarative style is more readable because the script explicitly contains the successive proof states, more accessible because it relies on a few basic operations, more natural because commands use English words and standard logical formulae, and much maintainable because modifications affect the behaviour of the script only locally.

The declarative style on the other hand, is much more verbose than the procedural style and relies heavily on automation to verify proof steps.

Previous uses of the declarative style include the Mizar Proof assistant and the ISAR language for the Isabelle proof assistant.

Basic commands

- `assume H:(0 < 2*x).` put in the context a hypothesis named `H` asserting that `0 < 2*x`.
- `have H':(x > 0) by H.` derive a new fact asserting that `x > 0` from the fact named `H`, and name it `H'`.
- `then (x > 0).` derive a new anonymous fact asserting that `x > 0` from the fact on the previous line.
- `thus (x > 0).` derive a new anonymous fact asserting that `x > 0`, and use it to conclude the proof or part of it.
- `per cases of (x=0 \/ x<>0) by EM.` derive a new fact asserting that `x=0 \/ x<>0` from `EM` (the axiom of excluded middle), and start a case analysis from this fact.
- `suppose nzx:(x<>0).` start proving the subcase where `(x <> 0)`.

Links

- `http://coq.inria.fr/`
  The Coq proof assistant
- `http://www.cs.ru.nl/~corbineau/mmode.html`
  The Declarative Proof Language for Coq.
- `http://c-corn.cs.ru.nl/`
  The C-CoRN library of formalised mathematics.
- `http://pauillac.inria.fr/~xleroy/compcert-backend/`
  A certified compiler backend in Coq
- `http://mizar.org/`
  Mizar: the first declarative proof assistant
- `http://isabelle.in.tum.de/Isar/`
  ISAR, a declarative language for the Isabelle proof assistant

Centrum voor Wiskunde en Informatica · TU Delft · NWO Nederlandse Organisatie voor Wetenschappelijk Onderzoek · Universiteit Utrecht · TU/e technische universiteit eindhoven · Universiteit Twente