**Basic Research in Informatics** 

# 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 :

A theorem from the book

Formalising Mathematics

www.bsik-bricks.nl

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:

- 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.

<u>File Edit Navigation Try Tactics Templates Queries Compile Windows Help</u>

#### 🔚 × 🗣 🕁 轮 🕁 🖸 🍚

f : R -> R g : R -> R Eq : forall x y : R, f (x + y) + f (x - y) = 2 * f x * g y Nz : ~ (forall x : R, f x = 0) y : R H : 1 < Rabs (g y) k : R Kb : Rfbound f k K1 : forall b1 : R, Rfbound f b1 -> k <= b1 H1 : k <= 0 x : R HH : forall x : R, Rabs x = 0 -> x = 0 (1/2) f x = 0 (2/2) Rabs (g y) <= 1	
Eq : forall x y : R, f (x + y) + f (x - y) = 2 * f x * g y Nz : ~ (forall x : R, f x = 0) y : R H : 1 < Rabs (g y) k : R Kb : Rfbound f k Kl : forall b1 : R, Rfbound f b1 -> k <= b1 H1 : k <= 0 x : R HH : forall x : R, Rabs x = 0 -> x = 0 (1/2) f x = 0 (2/2)	
<pre>- y) = 2 * f x * g y Nz : ~ (forall x : R, f x = 0) y : R H : 1 &lt; Rabs (g y) k : R Kb : Rfbound f k Kl : forall bl : R, Rfbound f bl -&gt; k &lt;= b1 H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0</pre>	
<pre>Nz : ~ (forall x : R, f x = 0) y : R H : 1 &lt; Rabs (g y) k : R Kb : Rfbound f k Kl : forall bl : R, Rfbound f bl -&gt; k &lt;= b1 H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0 (2/2)</pre>	-
<pre>y : R H : 1 &lt; Rabs (g y) k : R Kb : Rfbound f k Kl : forall b1 : R, Rfbound f b1 -&gt; k &lt;= b1 H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0</pre>	)
<pre>H : 1 &lt; Rabs (g y) k : R Kb : Rfbound f k Kl : forall b1 : R, Rfbound f b1 -&gt; k &lt;= b1 H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0 (2/2)</pre>	- -
<pre>k : R Kb : Rfbound f k Kl : forall bl : R, Rfbound f bl -&gt; k &lt;= bl H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0</pre>	) -
<pre>Kb : Rfbound f k Kl : forall bl : R, Rfbound f bl -&gt; k &lt;= bl Hl : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0 (2/2)</pre>	) - -
<pre>Kl : forall bl : R, Rfbound f bl -&gt; k &lt;= bl H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0 (2/2)</pre>	-
<pre>&lt;= b1 H1 : k &lt;= 0 x : R HH : forall x : R, Rabs x = 0 -&gt; x = 0 (1/2) f x = 0 (2/2)</pre>	) - -
H1 : k <= 0 x : R HH : forall x : R, Rabs x = 0 -> x = 0 (1/2) f x = 0 (2/2)	-
x : R HH : forall x : R, Rabs x = 0 -> x = 0 (1/2) f x = 0 (2/2)	
HH : forall x : R, Rabs x = 0 -> x = 0 (1/2) f x = 0 (2/2)	
(1/2) f x = 0 (2/2)	-
f x = 0 (2/2)	-
f x = 0 (2/2)	-
(2/2)	-
	-
	-
	-
Rabs (g y) <= 1	
***************************************	

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

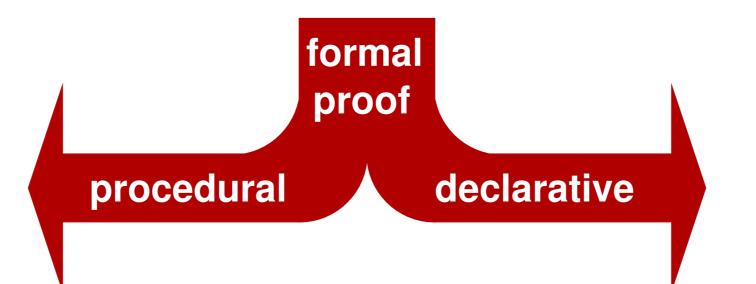
> $2k \ge |f(x+y)| + |f(x-y)|$  $\geq |f(x+y) + f(x-y)|$ = 2|g(y)||f(x)|

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.

formal statement

Require Import Reals.

```
Theorem B2: forall f g : R \rightarrow R,
(forall x y,
  f(x + y) + f(x - y) = 2 * f x * g y) \rightarrow
\tilde{}(forall x, f x = 0) ->
(forall x, Rabs (f x) <= 1) \rightarrow
(forall x, Rabs (g x) \leq 1).
```



• readable (by casual mathematicians)

• accessible (easy to learn)

natural (like textbook proofs)

• maintainable (resistant to software updates)

Edit Navigation Try Tactics Templates Queries Compile Windows Help

#### 🔚 🗙 👎 🕆 🖘 ዥ 🛂 📀

eady, proving B2

<pre>imo_corbineau.v @imo_thery.v</pre>	abs 2 * Rabs (f x) * Rabs (g y)
hu Drult le neg l bigger 2 0	eq0 : Rabs (f (x + y) + f (x - y)) =
by Rmult_le_reg_l, bigger_2_0.	2 * Rabs (f x) * Rabs (g y)
have hh: $(0 < 1)$ .	c4 : 2 * Rabs (f x) * Rabs (g y) <= k
<pre>then nn:(0 &lt; (Rabs (g y))) by Rlt_trans,</pre>	, hy. + k
then cc: $(0 < /(Rabs (g y)))$	fact1 : k + k = 2 * k
by Rinv_0_lt_compat.	<pre>[c5 : 2 * (Rabs (f x) * Rabs (g y)) &lt;=</pre>
then c7:(0 <= /(Rabs (g y))).	2 * k
then (Rabs (f x) * Rabs (g y) * / (Rabs	(g y)) c6 : Rabs (f x) * Rabs (g y) <= k
<= k * / Rabs (g y))	hh : 0 < 1
by Rmult_le_compat_r,c6.	nn : 0 < Rabs (g y)
then c8:	
(Rabs (f x) * (Rabs (g y) * / (Rabs (g	(cc: 0 < / Rabs (g y))
<= k * / Rabs (g y))	
by Rmult_assoc.	fact2: Rabs (f x) * Rabs (g y) * / R
have nngy:((Rabs (g y))<>0) by Rlt_not_e	$eg, nn$ , abs $(g y) \le k * / Rabs (g y)$
then (Rabs (g y) * / Rabs (g y) = 1)	
by Rinv_r.	[s (g y)) <= k * / Rabs (g y)
then c9: (Rabs (f x) $\leq k \times / Rabs$ (g y)	nngy : Rabs (g y) <> 0
by c8, Rmult_1_r.	-1 $(g y) - 1$
hence thesis.	(c9 : Rabs (f x) <= k * / Rabs (g y)
end claim.	
end craim.	?1 : Rabs (f x) <= k / Rabs (g y)
claim nub:(is_upper_bound f_abs_image	
(k / (Rabs (g y))).	
given z:R, x:R such that e: (Rabs (f x)=z)	
have (Rabs (f x) <= $(k / (Rabs (g y)))$	
by k_new_bound.	
hence thesis by e.	
end claim.	
end claim.	
have contra_1: $(k \le k / (Rabs (g y)))$	
by (ubkf _ nub).	
by (ubki _ nub).	
per cases of (k <= 0 $\setminus$ 0 < k) by Rlt_le_d	dog
	uec.
suppose pk: $(0 < k)$ .	
have $(1 < \text{Rabs} (g y))$ by hy.	
then (/ Rabs (g y) $<$ /1)	
by Rinv_1_lt_contravar.	
then $(k/ \text{ Rabs } (\sigma v) < k */1)$	

Ready, proving imo1972

### 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.

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.

### Basic commands

▲ assume H:(0 < 2\*x). put in the context a hypoth- INP http://coq.inria.fr/ esis named H asserting that  $0 < 2 \times x$ . The Coq proof assistant

A have H':(x > 0) by H. derive a new fact assert- I http://www.cs.ru.nl/~corbineau/mmode.html ing that x > 0 from the fact named H, and name it Н'.

 $\angle$  then (x > 0). derive a new anonymous fact asserting that x > 0 from the fact on the previous line.

### Links

Line: 119 Char: 23 🤇 🕻 🕄

de started

The Declarative Proof Language for Coq. http://c-corn.cs.ru.nl/ The C-CoRN library of formalised mathematics.

http://pauillac.inria.fr/~xleroy/

 $\not$  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  $x=0 \setminus x <>0$  from EM (the axiom of excluded middle), and start a case analysis from this fact.

start proving the subcase ▲ suppose nzx:(x<>0). where  $(x \iff 0)$ .

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









