QuickChick: Property-based testing for Coq

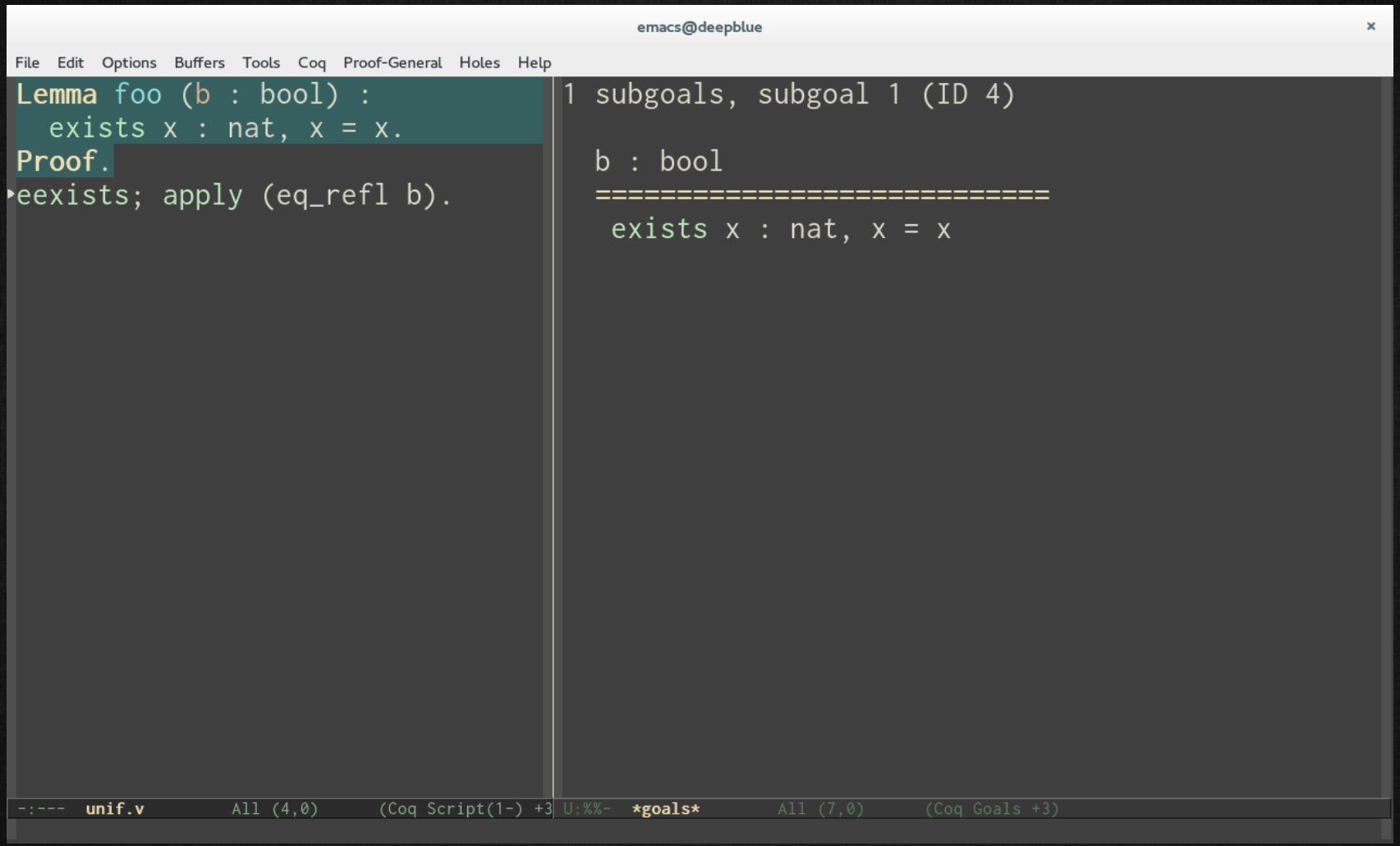
Maxime Dénès • Cătălin Hriţcu • Leonidas Lampropoulos • Zoe Paraskevopoulou • Benjamin C. Pierce

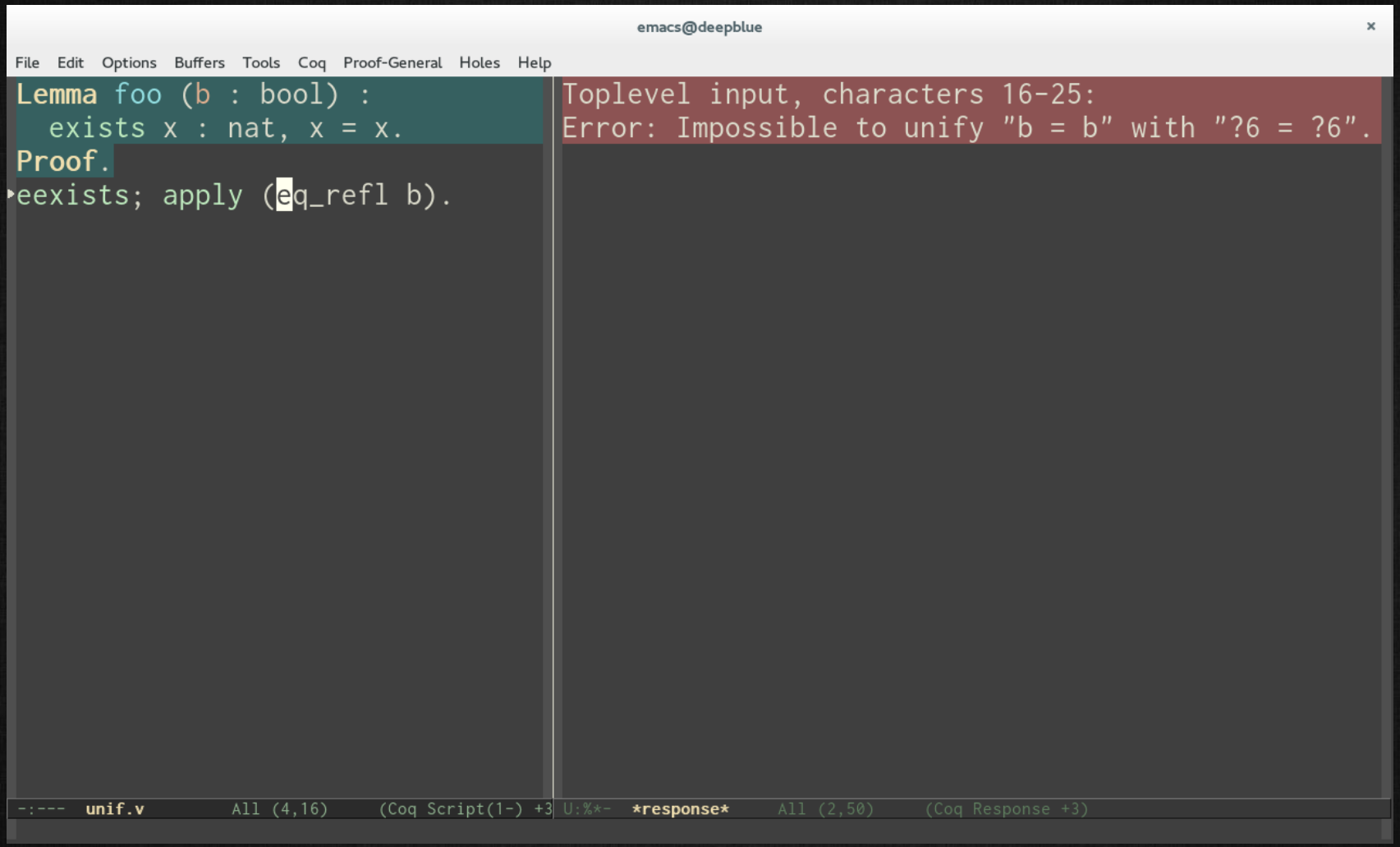
University of Pennsylvania • Inria Paris-Rocquencourt • NTU Athens


COQ CAN BE MEAN!

```
Theorem Feit_Thompson (gT : finGroupType) (G : {group gT}) :
  odd #|G| -> solvable G.
Proof. exact: (minSimpleOdd_ind no_minSimple_odd_group). Qed.
```

“Feit_Thompson is defined.”


emacs@deepblue
File Edit Options Buffers Tools Coq Proof-General Holes Help
Lemma foo (b : bool) :
exists x : nat, x = x.
Proof.
eexists; apply (eq_refl b).
1 subgoals, subgoal 1 (ID 4)
b : bool
============================
exists x : nat, x = x
-:--- unif.v All (4,0) (Coq Script(1-) +3
U:%%- *goals* All (7,0) (Coq Goals +3)


emacs@deepblue
File Edit Options Buffers Tools Coq Proof-General Holes Help
Lemma foo (b : bool) :
exists x : nat, x = x.
Proof.
eexists; apply (eq_refl b).
Toplevel input, characters 16-25:
Error: Impossible to unify "b = b" with "?6 = ?6".
-:--- unif.v All (4,16) (Coq Script(1-) +3
U:%*- *response* All (2,50) (Coq Response +3)

```
Lemma Fermat : forall a b c n : nat, 2 < n -> a^n + b^n = c^n
    -> a = b = c = 0.
Proof.
[... 1000 lines ...]
exact: my_lemma.
[... 100000 lines ...]
Qed.
```

```
Lemma Fermat : forall a b c n : nat, 2 < n -> a^n + b^n = c^n
    -> a = b = c = 0.
Proof.
[... 1000 lines ...]
exact: my_lemma.
[... 100000 lines ...]
Qed.

Lemma my_lemma : prime 4.
Proof. admit. Qed.
```

Why would you be proving a false statement?

- Some definitions could be wrong
- Conjectures are part of the proving process

Why would you be proving a false statement?

- Some definitions could be wrong
- Conjectures are part of the proving process

Murphy's law for Coq: if there is an incorrect admit, it will be the last remaining one.

Why would you be proving a false statement?

- Some definitions could be wrong
- Conjectures are part of the proving process

Murphy's law for Coq: if there is an incorrect admit, it will be the last remaining one.

Standard idea: try to catch errors early by random testing

Random testing is already popular for functional languages (QuickCheck [Claessen et al. 2000] in Haskell)

Random testing is already popular for functional languages (QuickCheck [Claessen et al. 2000] in Haskell)

and in proof assistants (Isabelle [Bulwahn 2012], Agda [Dybjer et al. 2003])

Random testing is already popular for functional languages (QuickCheck [Claessen et al. 2000] in Haskell)

and in proof assistants (Isabelle [Bulwahn 2012], Agda [Dybjer et al. 2003])

The idea is:

- Define an executable property `forall x : T, P(x)`
- Generate random elements in `T`
- Check that the property holds for these elements

We introduce QuickChick, a random testing plug-in for Coq

Status: very experimental

Status: very experimental

but already provides most of the features of Haskell's QuickCheck (except notably generation of functions).

Status: very experimental

but already provides most of the features of Haskell's QuickCheck (except notably generation of functions).

We are still studying full extension to polymorphism and dependent types.

DEMO

What can random testing reveal?

- Errors in programs (definitions)
- Errors in specifications (properties)

What can random testing reveal?

- Errors in programs (definitions)
- Errors in specifications (properties)
- Errors in generators

What can random testing reveal?

- Errors in programs (definitions)
- Errors in specifications (properties)
- Errors in generators

We provide some ways of detecting this last kind of errors:

- Mutation framework
- Formal verification of generators
- Language-based approach

Interlude: randomly testing a proof assistant

Recent work [Palka et al. 2011] showed that compilers can be tested by generating random lambda-terms.

Interlude: randomly testing a proof assistant

Recent work [Palka et al. 2011] showed that compilers can be tested by generating random lambda-terms.

We are investigating the applicability of similar approaches to test Coq's kernel.

Interlude: randomly testing a proof assistant

Recent work [Palka et al. 2011] showed that compilers can be tested by generating random lambda-terms.

We are investigating the applicability of similar approaches to test Coq's kernel.

Idea: many bugs could be find by testing partial properties on terms with little logical content.

Interlude: randomly testing a proof assistant

Recent work [Palka et al. 2011] showed that compilers can be tested by generating random lambda-terms.

We are investigating the applicability of similar approaches to test Coq's kernel.

Idea: many bugs could be find by testing partial properties on terms with little logical content.

The challenge is the generation of such terms.

Conclusion:

QuickChick is still very unstable, but you can play with it:
`https://github.com/lemonidas/QuickChick`

Not user-friendly yet, but we already applied it to non-trivial examples like testing non-interference.

THANK YOU!