The proof system Coq

Guillaume Claret

April 21st, 2015

The proof system Coq

A language to:

- state theorems
- write proofs verified by computer
- write algorithms

Raw example

```
Fixpoint fact (n:nat) : nat :=
  match n with
    | O => 1
    | S n => S n * fact n
  end.

Lemma lt_O_fact n : 0 < fact n.
Proof.
  induction n; simpl; auto with arith.
Qed.
```

Why is it important to check proofs?

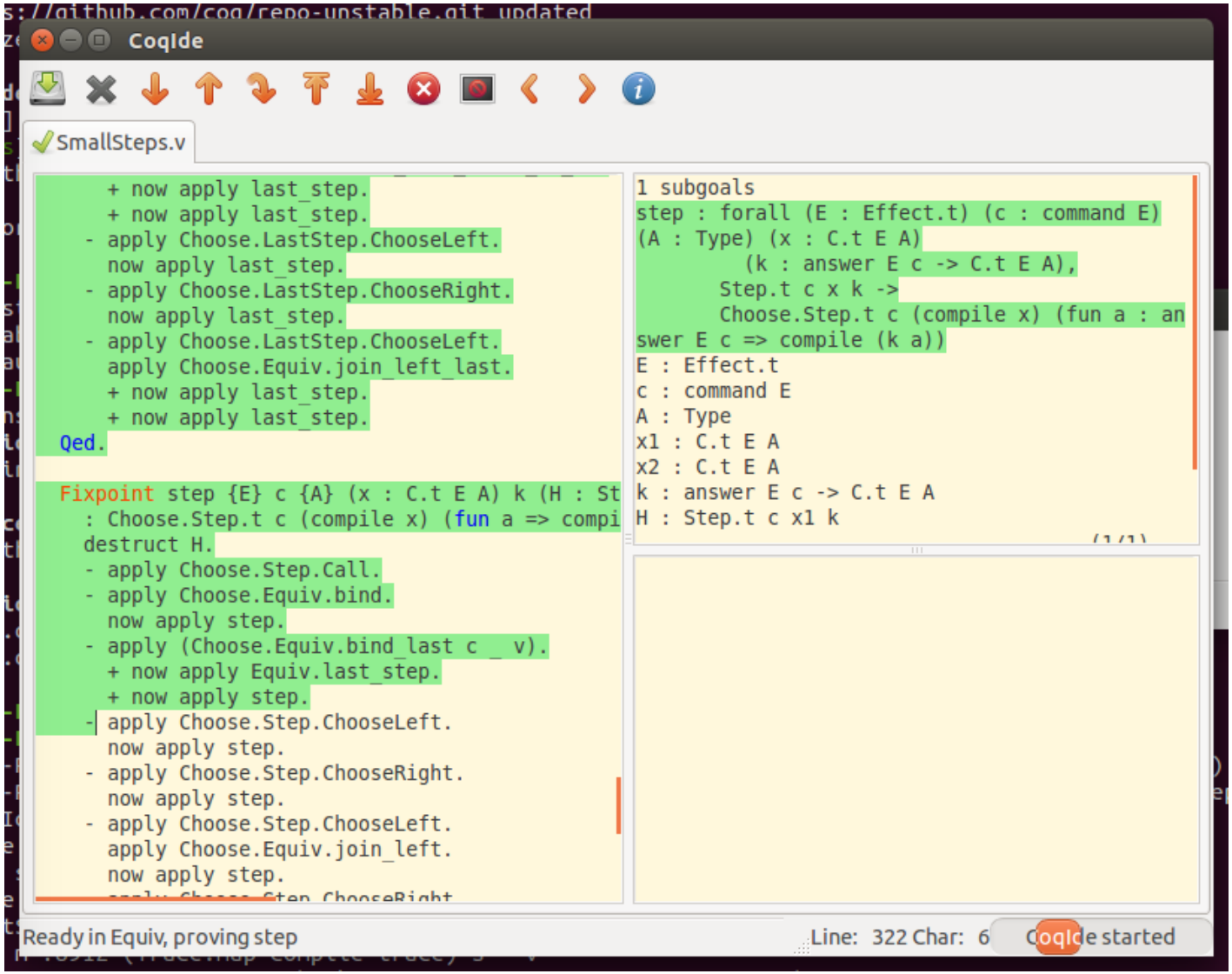
- mathematical proofs can hundreds of pages long with human mistakes
- software without proofs almost always contains bugs

The proof system Coq

An environment to:

- reason interactively
- organize and distribute proofs

IDE


CoqIde
SmallSteps.v
+ now apply last_step.
+ now apply last_step.
- apply Choose.LastStep.ChooseLeft.
now apply last_step.
- apply Choose.LastStep.ChooseRight.
now apply last_step.
- apply Choose.LastStep.ChooseLeft.
apply Choose.Equiv.join_left_last.
+ now apply last_step.
+ now apply last_step.
Qed.
Fixpoint step {E} c {A} (x : C.t E A) k (H : St
: Choose.Step.t c (compile x) (fun a => compi
destruct H.
- apply Choose.Step.Call.
- apply Choose.Equiv.bind.
now apply step.
- apply (Choose.Equiv.bind_last c _ v).
+ now apply Equiv.last_step.
+ now apply step.
- apply Choose.Step.ChooseLeft.
now apply step.
- apply Choose.Step.ChooseRight.
now apply step.
- apply Choose.Step.ChooseLeft.
apply Choose.Equiv.join_left.
now apply step.
1 subgoals
step : forall (E : Effect.t) (c : command E)
(A : Type) (x : C.t E A)
(k : answer E c -> C.t E A),
Step.t c x k ->
Choose.Step.t c (compile x) (fun a : an
swer E c => compile (k a))
E : Effect.t
c : command E
A : Type
x1 : C.t E A
x2 : C.t E A
k : answer E c -> C.t E A
H : Step.t c x1 k
Ready in Equiv, proving step
Line: 322 Char: 6
CoqIde started

Distribute

Install a package:

```
opam install -j4 package
```

Name	Version	Description
coq:color	1.1.0	A library on rewriting theory and termination.
coq:compcert	2.4.0	The CompCert C compiler.
coq:concurrency:pluto	1.0.0	A web server written in Coq.
coq:concurrency:proxy	1.0.0	A proxy to interface concurrent Coq programs with the operating system.
coq:concurrency:system	1.0.0	Experimental library to write concurrent applications in Coq.
coq:constructors	1.0.0	An example Coq plugin, defining a tactic to get the constructors of an inductive type in a list.
coq:coqeal:refinements	0.9.1	A refinement framework (for algebra).
coq:coqeal:theory	0.9.1	The theory needed by the refinement framework library.
coq:coquelicot	2.0.1	A Coq formalization of real analysis compatible with the standard library.
coq:corn	1.0.0	The CoRN library.

Usage

Maths:

- four colors theorem (Gontier 04)
- Feit – Thompson theorem (odd order theorem) (Gontier & all, 12)

Software written and proven in Coq:

- certified C compiler CompCert (Xavier Leroy & all)
- *Bedrock* library for low-level programs (Adam Chlipala & all)

Usage

Many conference papers with Coq proofs in annex.

More

Based on type theory instead of set theory:

set theory	type theory
	$+ : \mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}$
$\{n \in \mathbb{N} \mid \phi(n) \in \mathbb{P}\}$	$3 + 4 : \mathrm{nat}$

- we can mix proofs and programs
- reasoning on programming languages made simpler

Conclusion

- quick demo to give you a taste of Coq
- we hope it can be useful

Questions

Questions