Proving Theorems

and

Certifying Programs

with Coq

Stephan Boyer
@stepchowfun

Ask questions! Don’t be shy!

Logic

What’s a proof?

- **Informally:** an argument for the truth of a proposition
- **Formally:** a chain of applications of inference rules starting from axioms

Inference rules

Example: modus ponens

If **A → B** and **A** then **B**.

"→" means "**implies**"

Inference rules

Example: modus ponens

If **Stephan is at LambdaConf**
 → **Stephan is in Boulder**
and **Stephan is at LambdaConf**
then **Stephan is in Boulder**.

Programming

Inference rules

Example: function application

If $f : A \to B$ and $x : A$
then $f(x) : B$.

"→" is the type constructor for functions

Modus ponens ≅ function application

Modus ponens (logic)

If $A \to B$ and A
then B.

Application (programming)

If $f : A \to B$ and $x : A$
then $f(x) : B$.

Modus ponens ≅ function application

Modus ponens (logic)

If **f : A → B** and **x : A**
then **f(x) : B**.

f, **x**, and **f(x)** are *proofs*

Application (programming)

If **f : A → B** and **x : A**
then **f(x) : B**.

f, **x**, and **f(x)** are *programs*

Conjunctions ≅ products

Conjunction (logic)

If $p : A \land B$ then $p_1 : A$.
If $p : A \land B$ then $p_2 : B$.

p, p_1, and p_2 are *proofs*

Product (programming)

If $p : (A, B)$ then $p_1 : A$.
If $p : (A, B)$ then $p_2 : B$.

p, p_1, and p_2 are *programs*

Curry–Howard correspondence

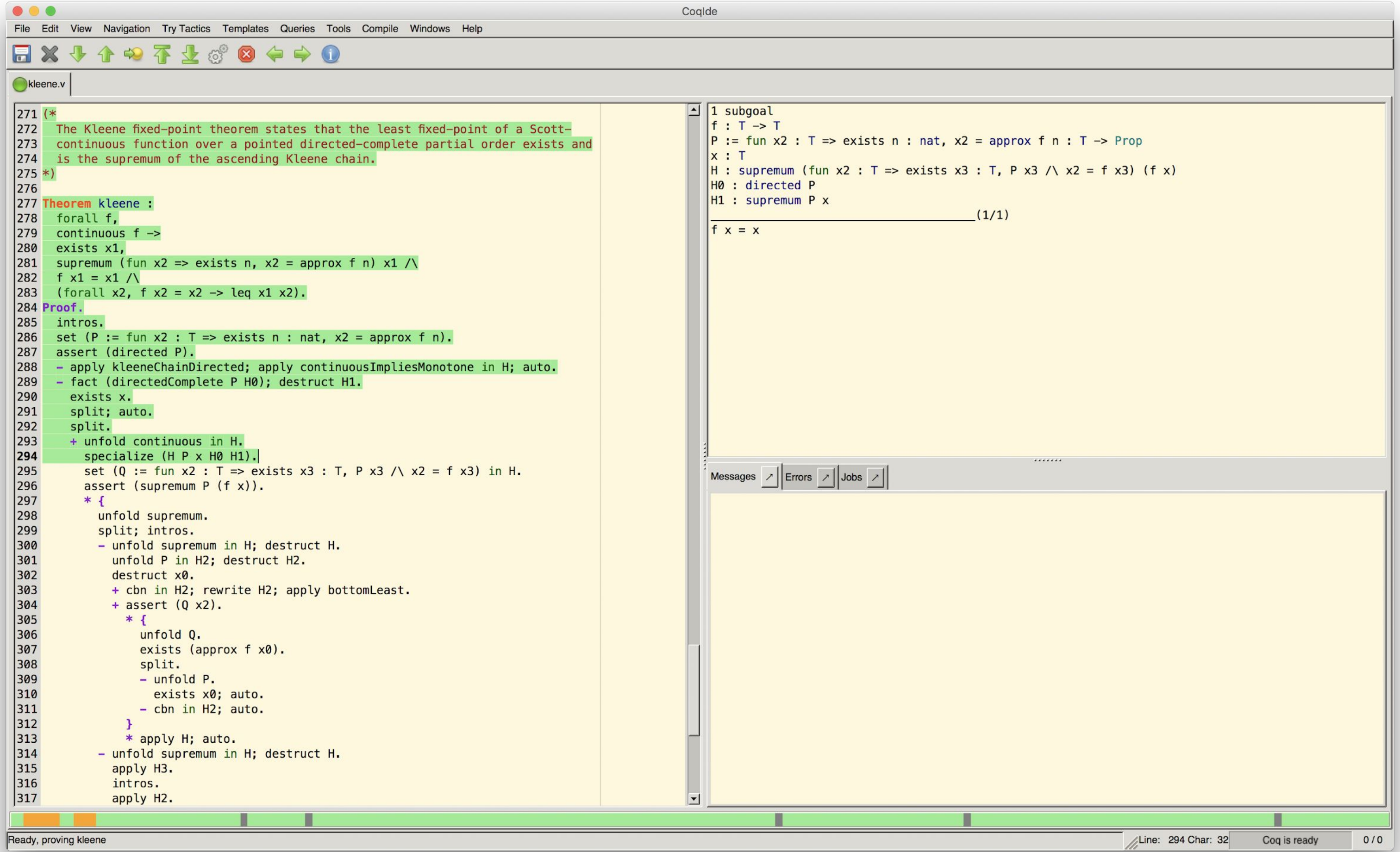
Logic		**Programming**
propositions	≅	types
proofs	≅	programs

Coq

Coq

"Rooster" in French

- A small pure functional programming language (**Gallina**)
- A scripting language for proof automation (**Ltac**)
- Proof scripts don't need to be trusted!


CoqIde
File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help
kleene.v
(*
The Kleene fixed-point theorem states that the least fixed-point of a Scott-
continuous function over a pointed directed-complete partial order exists and
is the supremum of the ascending Kleene chain.
*)

Theorem kleene :
forall f,
continuous f ->
exists x1,
supremum (fun x2 => exists n, x2 = approx f n) x1 /\
f x1 = x1 /\
(forall x2, f x2 = x2 -> leq x1 x2).
Proof.
intros.
set (P := fun x2 : T => exists n : nat, x2 = approx f n).
assert (directed P).
- apply kleeneChainDirected; apply continuousImpliesMonotone in H; auto.
- fact (directedComplete P H0); destruct H1.
exists x.
split; auto.
split.
+ unfold continuous in H.
specialize (H P x H0 H1).
set (Q := fun x2 : T => exists x3 : T, P x3 /\ x2 = f x3) in H.
assert (supremum P (f x)).
* {
unfold supremum.
split; intros.
- unfold supremum in H; destruct H.
unfold P in H2; destruct H2.
destruct x0.
+ cbn in H2; rewrite H2; apply bottomLeast.
+ assert (Q x2).
* {
unfold Q.
exists (approx f x0).
split.
- unfold P.
exists x0; auto.
- cbn in H2; auto.
}
* apply H; auto.
- unfold supremum in H; destruct H.
apply H3.
intros.
apply H2.
1 subgoal
f : T -> T
P := fun x2 : T => exists n : nat, x2 = approx f n : T -> Prop
x : T
H : supremum (fun x2 : T => exists x3 : T, P x3 /\ x2 = f x3) (f x)
H0 : directed P
H1 : supremum P x
(1/1)
f x = x
Messages
Errors
Jobs
Ready, proving kleene
Line: 294 Char: 32
Coq is ready
0 / 0

Follow along:
https://github.com/stepchowfun/coq-intro