How Computer Proves

Taehyun Eom

May 30, 2024

README License

The Four Color Theorem

Docker CI failing | contributions welcome | code of conduct | chat on zulip

This library contains a formal proof of the Four Color Theorem in Coq, along with the theories needed to support stating and then proving the Theorem. This includes an axiomatization of the setoid of classical real numbers, basic plane topology definitions, and a theory of combinatorial hypermaps.

Meta

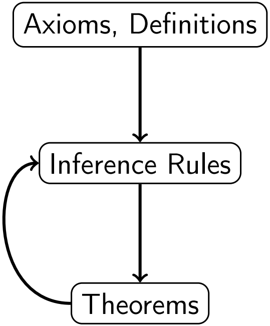
- Author(s):
  - Georges Gonthier (initial)
- Coq-community maintainer(s):
  - Yves Bertot (@ybertot)
- License: CeCILL-B
- Compatible Coq versions: 8.16 or later
- Additional dependencies:
  - MathComp ssreflect 2.0 or later
  - MathComp algebra
- Coq namespace: `fourcolor`
- Related publication(s):
  - Formal Proof—The Four-Color Theorem

https://github.com/coq-community/fourcolor/tree/master

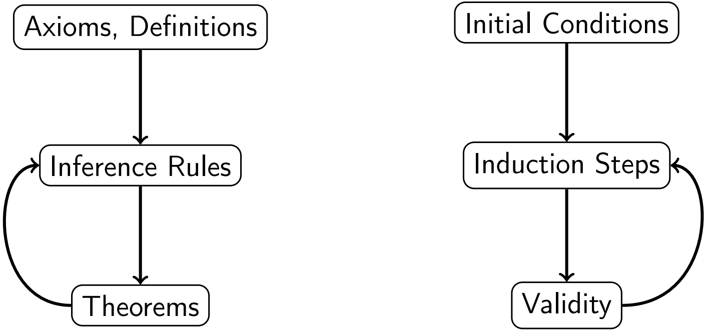
```
Section FourColorTheorem.

Variable Rmodel : Real.model.
Let R := Real.model_structure Rmodel.
Implicit Type m : map R.

Theorem four_color_finite m : finite_simple_map m -> colorable_with 4 m.
Proof.
intros fin_m.
pose proof (discretize.discretize_to_hypermap fin_m) as [G planarG colG].
exact (colG (combinatorial4ct.four_color_hypermap planarG)).
Qed.

Theorem four_color m : simple_map m -> colorable_with 4 m.
Proof. revert m; exact (finitize.compactness_extension four_color_finite). Qed.

End FourColorTheorem.
```

theories/fourcolor.v

How to prove?



How to prove?


Axioms, Definitions
Inference Rules
Theorems
Initial Conditions
Induction Steps
Validity

Inference Rules

Modus Ponens(Implication Elimination)

$$P, P \rightarrow Q \Rightarrow Q$$

Inference Rules

Modus Ponens(Implication Elimination)

$$P, P \to Q \Rightarrow Q$$

Deduction Theorem(Implication Introduction)

$$(\{H_1, \cdots, H_k, P\} \Rightarrow Q) \Rightarrow (\{H_1, \cdots, H_k\} \Rightarrow P \to Q)$$

Inference Rules

Modus Ponens(Implication Elimination)

$$P, P \to Q \vdash Q$$

Deduction Theorem(Implication Introduction)

$$(\{H_1, \cdots, H_k, P\} \vdash Q) \Rightarrow (\{H_1, \cdots, H_k\} \vdash P \to Q)$$

Inference Rules

Modus Ponens(Implication Elimination)

$$P, P \rightarrow Q \vdash Q$$

Deduction Theorem(Implication Introduction)

$$(\{H_1, \cdots, H_k, P\} \vdash Q) \Rightarrow (\{H_1, \cdots, H_k\} \vdash P \rightarrow Q)$$

Modus ponens and deduction theorem can convert inference rules into axioms.

$$(A \wedge B \vdash A) \Rightarrow (A \wedge B \rightarrow A)$$

Rule for hypothesis

P can be proved when P is assumed.

$$P \vdash P$$

If P can be proved without a hypothesis Q, then it can be proved with Q.

$$(\vdash P) \Rightarrow (Q \vdash P)$$

Modus ponens with a hypothesis.

$$(P \vdash Q \to R), (P \vdash Q) \Rightarrow (P \vdash R)$$

Three Axioms

P can be proved when P is assumed.

$$P \to P$$

If P can be proved without a hypothesis Q, then it can be proved with Q.

$$P \to [Q \to P]$$

Modus ponens with a hypothesis.

$$[P \to [Q \to R]] \to [[P \to Q] \to [P \to R]]$$

Curry-Howard correspondence

$$P \to Q, P \vdash Q$$

Curry-Howard correspondence

$$P \to Q, P \vdash Q$$

$$f : X \to Y, x : X \Rightarrow f(x) : Y$$

Curry-Howard correspondence

Modus Ponens	Function Application
Axiom	Predefined Object
Theorem	Induced Object
Hypothesis	Free Variable
$P \to Q$	Function Type
$A \wedge B \to A$, $A \wedge B \to B$	$(a, b) \mapsto a$, $(a, b) \mapsto b$
$A \wedge B$	Pair Type (Product Type)
$A \to A \vee B$, $B \to A \vee B$	$X \to X \oplus Y$, $Y \to X \oplus Y$
$A \vee B$	Union Type (Sum Type)
Syllogism	Function Composition
Three Axioms	?

Curry-Howard correspondence

$$P \to P \Rightarrow I(x) = x$$

$$\begin{aligned} P \to [Q \to P] &\Rightarrow K(x)(y) = x \\ &\rightsquigarrow K(x, y) = x \end{aligned}$$

$$S : [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]]$$

$$P \to P \Rightarrow I(x) = x$$

$$\begin{aligned} P \to [Q \to P] &\Rightarrow K(x)(y) = x \\ &\rightsquigarrow K(x, y) = x \end{aligned}$$

$$S : [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]]$$

$$x : P \to [Q \to R] \qquad S(x) : [P \to Q] \to [P \to R]$$

Curry-Howard correspondence

$$P \to P \Rightarrow I(x) = x$$

$$\begin{aligned} P \to [Q \to P] \Rightarrow & K(x)(y) = x \\ \rightsquigarrow & K(x, y) = x \end{aligned}$$

$$\begin{aligned} & & S &: [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]] \\ x &: P \to [Q \to R] & S(x) &: [P \to Q] \to [P \to R] \\ y &: P \to Q & S(x)(y) &: P \to R \end{aligned}$$

Curry-Howard correspondence

$$P \to P \Rightarrow I(x) = x$$

$$\begin{aligned} P \to [Q \to P] \Rightarrow & K(x)(y) = x \\ \rightsquigarrow & K(x, y) = x \end{aligned}$$

$$\begin{array}{ll} & S : [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]] \\ x : P \to [Q \to R] & S(x) : [P \to Q] \to [P \to R] \\ y : P \to Q & S(x)(y) : P \to R \\ z : P & S(x)(y)(z) : R = x(P)(Q) = x(z)(y(z)) \\ & \rightsquigarrow S(x, y, z) = x(z, y(z)) \end{array}$$

Three Axioms $\Leftrightarrow$ SKI combinator

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$S : [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]]$$

Program vs Proof

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$\begin{aligned} S :& [P \to [Q \to R]] \to [[P \to Q] \to [P \to R]] \\ K :& [P \to [Q \to P]] \\ & \Rightarrow R := P \end{aligned}$$

Program vs Proof

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$\begin{aligned} S :& [P \to [Q \to P]] \to [[P \to Q] \to [P \to P]] \\ K :& [P \to [Q \to P]] \end{aligned}$$

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$\begin{aligned} S :& [P \to [Q \to P]] \to [[P \to Q] \to [P \to P]] \\ K :& [P \to [Q \to P]] \\ SK :& [P \to Q] \to [P \to P] \end{aligned}$$

Program vs Proof

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$\begin{aligned} S :& [P \to [Q \to P]] \to [[P \to Q] \to [P \to P]] \\ K :& [P \to [Q \to P]] \\ SK :& [P \to Q] \to [P \to P] \\ K :& P \to [Q' \to P] \\ &\Rightarrow Q := [Q' \to P] \end{aligned}$$

Program vs Proof

Redundancy of I combinator

$$SK \bullet x \Rightarrow S(K, \bullet, x) \Rightarrow K(x, \bullet(x)) \Rightarrow x.$$

Hence, we can say

$$I = SK\bullet = SKK = SKS$$

Redundancy of I axiom

$$\begin{aligned}
S :& [P \to [[Q' \to P] \to P]] \to [[P \to [Q' \to P]] \to [P \to P]] \\
K :& [P \to [[Q' \to P] \to P]] \\
SK :& [P \to [Q' \to P] \to [P \to P] \\
K :& P \to [Q' \to P] \\
SKK :& P \to P
\end{aligned}$$

Test on Haskell

```
ghci> k x y = x
ghci> :t k
k :: p1 -> p2 -> p1
ghci>
ghci> s x y z = x z (y z)
ghci> :t s
s :: (t1 -> t2 -> t3) -> (t1 -> t2) -> t1 -> t3
ghci>
ghci> :t s k
s k :: (t3 -> t2) -> t3 -> t3
ghci> :t s k k
s k k :: t3 -> t3
ghci> s k k "Hello world!"
"Hello world!"
ghci>
ghci> :t s k (s k)
s k (s k) :: (t3 -> t2) -> t3 -> t2
```

Negation

Primitive Negation

- Primitive unary operator
- Axiom to make logic system complete :
  $[\neg P \to \neg Q] \to [Q \to P]$

Negation as an Abbreviation

- Abbreviation of $P \to \bot$
- Axiom to make logic system complete
  - $[\neg P \to \neg Q] \to [Q \to P]$
  - $\neg\neg P \to P$
  - $\neg P \vee P$
  - $[[P \to Q] \to P] \to P$

An example with Negation

$$\neg\neg P \to [\neg\neg\neg\neg P \to \neg\neg P]$$

$$\neg\neg P \vdash \neg\neg\neg\neg P \to \neg\neg P$$

$$\neg\neg P \vdash [\neg\neg\neg\neg P \to \neg\neg P] \to [\neg P \to \neg\neg\neg P]$$

$$\neg\neg P \vdash \neg P \to \neg\neg\neg P$$

$$\neg\neg P \vdash [\neg P \to \neg\neg\neg P] \to [\neg\neg P \to P]$$

$$\neg\neg P \vdash \neg\neg P \to P$$

$$\neg\neg P \vdash \neg\neg P$$

$$\neg\neg P \vdash P$$

$$\neg\neg P \to P$$

An example with Negation

$$
\begin{aligned}
K &: \neg\neg P \to [\neg\neg\neg\neg P \to \neg\neg P] \\
K(x) &: \neg\neg P \vdash \neg\neg\neg\neg P \to \neg\neg P \\
C &: \neg\neg P \vdash [\neg\neg\neg\neg P \to \neg\neg P] \to [\neg P \to \neg\neg\neg P] \\
C(K(x)) &: \neg\neg P \vdash \neg P \to \neg\neg\neg P \\
C &: \neg\neg P \vdash [\neg P \to \neg\neg\neg P] \to [\neg\neg P \to P] \\
C(C(K(x))) &: \neg\neg P \vdash \neg\neg P \to P \\
x &: \neg\neg P \vdash \neg\neg P \\
C(C(K(x)))(x) &: \neg\neg P \vdash P \\
x \mapsto C(C(K(x)))(x) &: \neg\neg P \to P
\end{aligned}
$$

Test on Haskell

```
ghci> :{
ghci| c :: ([a] -> [b]) -> (b -> a)
ghci| c = undefined
ghci| :}
ghci>
ghci> d x = c (c (k x)) x
ghci> :t d
d :: [[a]] -> a
```

For a simple graph G and its cycle $C = e_1 e_2 \cdots e_n$, define

$$f(C) = (e_1, e_n e_{n-1}, \cdots e_2) = (C[0], C[-1:0:-1]).$$

The type of f is

$f : \{\text{Cycles of } G\} \to \cup_{v,w \in V(G)}\{\text{Pairs of distinct paths from } v \text{ to } w\}$

Existence of this function proves that if the graph has no two distinct paths with common endpoints, then the graph is acyclic.

$\neg P$	V^*
$P \to \bot$	$V^* : V \to L$
$C' : [P \to Q] \to [\neg Q \to \neg P]$	$\mathcal{L}(V, W) \to \mathcal{L}(W^*, V^*)$
$C : [\neg P \to \neg Q] \to [Q \to P]$	$\mathcal{L}(V^*, W^*) \to \mathcal{L}(W, V)$ (finite)
$D' : P \to \neg\neg P$	$V \to V^{**}$
$D : \neg\neg P \to P$	$V^{**} \to V$ (finite)
K, S	$K(x), S(x)(y)$ are not linear

As a linear map

- C satisfies $f(C(\varphi)(v)) = \varphi(f)(v)$ for any $f \in V^*$, $\varphi \in \mathcal{L}(V, W)$, $v \in W$.
- C' satisfies $C'(\varphi)(f) = f \circ \varphi$.
- D satisfies $f(D(\varphi)) = \varphi(f)$.
- D' satisfies $D'(v)(\varphi) = \varphi(v)$.

As a proof, $D = CD'$ and $D' = CD$.

For Linear Algebra

$D = CD'$ as a linear map

$$f(C(D')(\varphi)) = D'(f)(\varphi) \quad f \in V^*, \varphi \in V^{**}$$
$$\Rightarrow f((CD')(\varphi)) = \varphi(f)$$

$D' = CD$ as a linear map

$$\psi(C(D)(v)) = D(\psi)(v) \quad \psi \in V^{***}, v \in V$$
$$\Rightarrow \psi((CD)(v)) = D'(v)(D(\psi)) = \psi(D'(v))$$
$$\Rightarrow CD = D'$$

```
ghci> d' = c d
ghci> :t d'
d' :: b -> [[b]]
ghci> :t c d'
c d' :: [[a]] -> a
```

For Linear Algebra

Construct C' from C, D, D'

For any linear map $f : V \to W$,

$$C'(f) = C(D' \circ f \circ D) = C(x \mapsto D'(f(D(x))))$$

```
ghci> c' f = c (d'.f.d)
ghci> :t c'
c' :: (b1 -> b2) -> [b2] -> [b1]
```

Prove C' from C, D, D'

$$
\begin{aligned}
x &: [P \to Q], \neg\neg P \vdash \neg\neg P \\
D &: [P \to Q], \neg\neg P \vdash \neg\neg P \to P \\
D(x) &: [P \to Q], \neg\neg P \vdash P \\
f &: [P \to Q], \neg\neg P \vdash P \to Q \\
f(D(x)) &: [P \to Q], \neg\neg P \vdash Q \\
D' &: [P \to Q], \neg\neg P \vdash Q \to \neg\neg Q \\
D'(f(D(x)) &: [P \to Q], \neg\neg P \vdash \neg\neg Q \\
x \mapsto D'(f(D(x)) &: [P \to Q] \vdash \neg\neg P \to \neg\neg Q \\
C &: [P \to Q] \vdash [\neg\neg P \to \neg\neg Q] \to [\neg Q \to \neg P] \\
C(x \mapsto D'(f(D(x)))) &: [P \to Q] \vdash \neg Q \to \neg P \\
C' &: [P \to Q] \to [\neg Q \to \neg P]
\end{aligned}
$$

For Enumerative Combinatorics

Bijective Proof

- What is a bijective proof?
- Why bijective proofs?

Combinatorial Species and Generating Functions

SEQ, CYC, SET, MSET, PSET, $\cdots$

From Graph Theory

Which problem is hard to understand for computers?

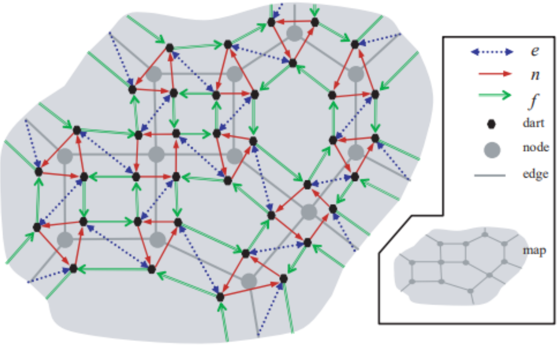


Figure 1. A hypermap.

Georges Gonthier, *Formal proof—the four color theorem*, Notices Amer. Math. Soc. 55(2008) no.11, 1382–1393
Georges Gonthier, *A computer-checked proof of the Four Color Theorem*, Inria 2023, hal-04034866

Thank you.