# Calculus of Constructions vs. Coq (Syntax)

| Calculus of Constructions | Coq |
| --- | --- |
| $\lambda x : A.\, M$ | `fun x : A => M` |
| $\lambda x : A, y : B, z : B.\, M$ | `fun (x : A) (y z : B) => M` |
| $M\,N$ | `M N` |
| $A \to B$ | `A -> B` |
| $\forall x : A.\, B$ | `forall x : A, B` |
| $\forall x : A, y : B, z : B.\, C$ | `forall (x : A) (y z : B), C` |
| $*$ | `Prop/Set/Type` |
| $\square$ | `Type` |

# A Tale of Two Universes

In CoC, all (basic) types have sort $*$ — the universe of types.
In Coq, there are two basic sorts: `Prop` and `Set`.

# A Tale of Two Universes

In CoC, all (basic) types have sort $*$ — the universe of types.
In Coq, there are two basic sorts: `Prop` and `Set`. Why?

# A Tale of Two Universes

In CoC, all (basic) types have sort $*$ — the universe of types.
In Coq, there are two basic sorts: `Prop` and `Set`. Why?

`Prop` is the universe of *propositions*. We care *whether* a proposition holds, but not *how* it holds. We just want to know if a proposition is inhabited (provable), and generally don't care about which inhabitant (proof) we have.

# A Tale of Two Universes

In CoC, all (basic) types have sort $*$ — the universe of types.
In Coq, there are two basic sorts: `Prop` and `Set`. Why?

`Prop` is the universe of *propositions*. We care *whether* a proposition holds, but not *how* it holds. We just want to know if a proposition is inhabited (provable), and generally don't care about which inhabitant (proof) we have.

`Set` is the universe of *sets*. We care about the distinction between inhabitants of a set. For instance, we care that 2 and 3 are different inhabitants of $\mathbf{nat}$. We want to *compute* with sets.

# Example: Classical Logic

In classical logic, we have the axiom

$$\forall P : \texttt{Prop}.\, P \lor \neg P$$

# Example: Classical Logic

In classical logic, we have the axiom

$$\forall P : \texttt{Prop}.\, P \vee \neg P$$

Intuitionistically, a proof of $P \vee Q$ tells you *which* of $P$ or $Q$ holds. Adding this axiom gives a new way of proving a disjunction where we don't actually know which of the two holds.

# Example: Classical Logic

In classical logic, we have the axiom

$$\forall P : \texttt{Prop}.\, P \vee \neg P$$

Intuitionistically, a proof of $P \vee Q$ tells you *which* of $P$ or $Q$ holds. Adding this axiom gives a new way of proving a disjunction where we don't actually know which of the two holds.

Should this be the same as assuming the existence of a function that determines which of $P$ or $\neg P$ holds?

# Example: Classical Logic

In classical logic, we have the axiom

$$\forall P : \texttt{Prop}.\, P \vee \neg P$$

Intuitionistically, a proof of $P \vee Q$ tells you *which* of $P$ or $Q$ holds. Adding this axiom gives a new way of proving a disjunction where we don't actually know which of the two holds.

Should this be the same as assuming the existence of a function that determines which of $P$ or $\neg P$ holds?
If so, then our functions go beyond computability. Let $H(n)$ be the proposition that the Turing machine represented by $n$ halts. Then we have a function $\forall n : \mathbf{nat}.\, H(n) \vee \neg H(n)$ which tells us whether or not an arbitrary TM halts.

# Example: Classical Logic

In classical logic, we have the axiom

$$\forall P : \texttt{Prop}.\, P \vee \neg P$$

Intuitionistically, a proof of $P \vee Q$ tells you *which* of $P$ or $Q$ holds. Adding this axiom gives a new way of proving a disjunction where we don't actually know which of the two holds.

Should this be the same as assuming the existence of a function that determines which of $P$ or $\neg P$ holds?
If so, then our functions go beyond computability. Let $H(n)$ be the proposition that the Turing machine represented by $n$ halts. Then we have a function $\forall n : \mathbf{nat}.\, H(n) \vee \neg H(n)$ which tells us whether or not an arbitrary TM halts.

Coq's solution is to separate out the propositional universe (`Prop`) from the computable universe (`Set`). The two interact, but something in `Set` cannot depend on knowing the inhabitant of something in `Prop`.