Category Theory in Coq 8.5

Amin Timany       Bart Jacobs

iMinds-Distrinet KU Leuven

7th Coq Workshop – Sophia Antipolis
June 26, 2015
List of the most important formalized notions

- **basic constructions:**
  - terminal/initial object
  - products/sums
  - equalizers/coequalizers
  - pullbacks/pushouts
  - exponentials
  - $+ \dashv \Delta \dashv \times$ and $(- \times a) \vdash a^-$

- **external constructions:**
  - comma categories
  - product category

- **for** Cat: \((\text{Obj} := \text{Category}, \text{Hom} := \text{Functor})\)
  - cartesian closure
  - initial object

- **for** Set: \((\text{Obj} := \text{Type}, \text{Hom} := \text{fun} \ A \ B \Rightarrow A \rightarrow B)\)
  - initial object
  - sums
  - equalizers
  - coequalizers
  - pullbacks
  - cartesian closure
  - local cartesian closure
  - completeness
  - co-completeness
  - sub-object classifier \((\text{Prop} : \text{Type})\)
  - topos

†uses the axioms of propositional extensionality and constructive indefinite description (axiom of choice).

- the Yoneda lemma
- adjunction
  - hom-functor adjunction, unit-counit adjunction, universal morphism adjunction and their conversions
  - duality: $F \dashv G \Rightarrow G^{op} \dashv F^{op}$
  - uniqueness up to natural isomorphism
  - category of adjunctions

- kan extensions
  - global definition
  - local definition with both hom-functor and cones (along a functor)
  - uniqueness
  - preservation by adjoint functors
  - pointwise kan extensions (preserved by representable functors)

- (co)limits
  - as (left)right local kan extensions along the unique functor to the terminal category
  - (sum)product-(co)equalizer (co)limits
  - pointwise (as kan extensions)

- $T - (co)algebras$ (for an endofunctor $T$)
  - we use proof functional extensionality
  - we use proof irrelevance in many cases (mostly for proof of equality of arrows)

- This implementation can be found at:
  https://bitbucket.org/amintimany/categories/
This implementation uses some features of Coq 8.5, most notably:

- Primitive projections for records:
- Universe polymorphism: for relative smallness/largeness
Primitive projections for records:
- The $\eta$ rule for records: two instances of a record type are \textit{definitionally} equal if all their respective projections are equal.
- E.g., for $\{x : A; y : A\}$ and $f \ u = \{x := y \ u; y := x \ u\}$, we have $f \ (f \ u) \equiv u$. 
Primitive projections for records:

- The η rule for records: two instances of a record type are \textit{definitionally} equal if all their respective projections are.
  
- E.g., for \(|x : A; y : A|\) and \(f \ u = |x := y u; y := x u|\), we have \(f \ (f \ u) \equiv u\).
  
- This provides \textit{definitional} equalities, e.g.: (similar to Coq/HoTT implementation)
  
  - For Categories: \((C^\text{op})^\text{op} \equiv C\)
  
  - For Functors: \((F^\text{op})^\text{op} \equiv F\)
  
  - For Natural Transformations: \((N^\text{op})^\text{op} \equiv N\)
- Universe polymorphism: for relative smallness/largeness
Universe polymorphism: for relative smallness/largeness

```coq
Class Category : Type@{\max(i+1, j+1)} :=
{  
  Obj : Type@{i}
  Hom : Obj → Obj → Type@{j}
  id : forall a : Obj, Hom a a
  compose : forall a b c, (f : Hom a b) (g : Hom c d) : Hom a c
}
```

Category is universe polymorphic
For each pair of levels \((m, n)\), Category@{m, n} is a copy at level \((m, n)\)

Universe levels in definitions and theorems are inferred by Coq and never appear
in the source code

For each definition, theorem, etc., we get some constraints on universe levels
The definition, theorem, etc. only works for those copies that satisfy the side
- Universe polymorphism: for relative smallness/largeness

```
Class Category : Type@{\max(i+1, j+1)} :=
{  
  Obj : Type@{i} 
  Hom : Obj \to Obj \to Type@{j} 
  id : \forall a : Obj, Hom a a 
  compose : \forall a b c, (f : Hom a b) (g : Hom c d) : Hom a c
  
}
```

- Category is universe polymorphic
Universe polymorphism: for relative smallness/largeness

```coq
Class Category : Type@{max(i+1, j+1)} :=
{
    Obj : Type@{i}
    Hom : Obj → Obj → Type@{j}
    id : forall a : Obj, Hom a a
    compose : forall a b c, (f : Hom a b) (g : Hom c d) : Hom a c
}
```

Category is universe polymorphic

For each pair of levels \((m, n)\), \(\text{Category}@{m, n}\) is a copy at level \((m, n)\)
Universe polymorphism: for relative smallness/largeness

\textbf{Class} \texttt{Category} : \texttt{Type@\{max(i+1, j+1)\}} :=
{
    \texttt{Obj} : \texttt{Type@\{i\}}
    \texttt{Hom} : \texttt{Obj \rightarrow Obj \rightarrow Type@\{j\}}
    \texttt{id} : \texttt{forall a : Obj, Hom a a}
    \texttt{compose} : \texttt{forall a b c, (f : Hom a b) (g : Hom c d) : Hom a c}
}

Category is universe polymorphic

For each pair of levels \((m, n)\), \texttt{Category@\{m, n\}} is a copy at level \((m, n)\)

Universe levels in definitions and theorems are inferred by Coq and never appear in the source code
Universe polymorphism: for relative smallness/largeness

```coq
Class Category : Type@{max(i+1, j+1)} :=
{
  Obj : Type@{i}
  Hom : Obj → Obj → Type@{j}
  id : forall a : Obj, Hom a a
  compose : forall a b c, (f : Hom a b) (g : Hom c d) : Hom a c
  ...
}
```

- Category is universe polymorphic
- For each pair of levels \((m, n)\), \(\text{Category@\{m, n\}}\) is a copy at level \((m, n)\)
- Universe levels in definitions and theorems are inferred by Coq and never appear in the source code
- For each definition, theorem, etc., we get some constraints on universe levels
- The definition, theorem, etc. only works for those copies that satisfy the side constraint
This notion of smallness/largeness using universe levels works so well that we can define $\textbf{Cat}$:

- **Instance** $\textbf{Cat} : \text{Category} := \{\text{Obj} := \text{Category}; \text{Hom} := \text{Functor}; \ldots\}$
This notion of smallness/largeness using universe levels works so well that we can define \textbf{Cat}:

\textbf{Instance Cat} : Category := \{Obj := Category; Hom := Functor; \ldots\}

Or prove the following:

\textbf{Theorem Complete_Preorder} (C : Category) (CC : Complete C):

\texttt{forall x y : (Obj C), Hom x y' \simeq ((Arrow C) \to Hom x y)
This notion of smallness/largeness using universe levels works so well that we can define **Cat**:

**Instance** **Cat** : **Category** := {**Obj** := **Category**; **Hom** := **Functor**; ...}

Or prove the following:

**Theorem** Complete_Preorder (**C** : **Category**) (**CC** : Complete **C**)
\[
\forall x\ y \ (\text{**Obj** } \text{**C**}) \ \text{Hom } x\ y' \cong ((\text{Arrow } \text{**C**}) \to \text{Hom } x\ y)
\]

This theorem results in a contradiction as soon as there are objects \(a\) and \(b\) in \(C\) such that \(|\text{hom}(a, b)| \geq 2\)
This notion of smallness/largeness using universe levels works so well that we can define \textbf{Cat}:

\textbf{Instance} \textbf{Cat} : \textbf{Category} := \{\text{Obj} := \text{Category}; \text{Hom} := \text{Functor}; \ldots\}

Or prove the following:

\textbf{Theorem} \textbf{Complete_Preorder} (\textbf{C} : \textbf{Category}) (\textbf{CC} : \textbf{Complete} \textbf{C}) :
\textbf{forall} x y : (\text{Obj} \textbf{C}), \text{Hom} x y' \simeq ((\text{Arrow} \textbf{C}) \to \text{Hom} x y)

This theorem results in a contradiction as soon as there are objects \(a\) and \(b\) in \(C\) such that \(|\text{hom}(a, b)| \geq 2\)

In fact, this theorem holds only for small categories
This notion of smallness/largeness using universe levels works so well that we can define \textbf{Cat}:

- **Instance** \textbf{Cat} : \textbf{Category} := \{\text{Obj} := \text{Category}; \text{Hom} := \text{Functor}; \ldots\}

- Or prove the following:

  \textbf{Theorem} \text{Complete_Preorder} (C : \text{Category}) (CC : \text{Complete } C) : 
  \forall x \ y : (\text{Obj } C), \text{Hom } x \ y' \simeq ((\text{Arrow } C) \to \text{Hom } x \ y)

This theorem results in a contradiction as soon as there are objects \(a\) and \(b\) in \(C\) such that \(|\text{hom}(a,b)| \geq 2\)

In fact, this theorem holds only for small categories.

This can be seen in universe constraints of this theorem.
This notion of smallness/largeness using universe levels works so well that we can define \textbf{Cat}:

\textbf{Instance} \textbf{Cat} : \textbf{Category} := \{\text{Obj} := \text{Category}; \text{Hom} := \text{Functor}; \ldots\} \\

Or prove the following:

\textbf{Theorem} \text{Complete_Preorder} \ (C : \text{Category}) \ (CC : \text{Complete} \ C) : \forall x y : \text{Obj} C, \text{Hom} x y \simeq ((\text{Arrow} C) \to \text{Hom} x y)

This theorem results in a contradiction as soon as there are objects \(a\) and \(b\) in \(C\) such that \(|\text{hom}(a, b)| \geq 2\)

In fact, this theorem holds only for small categories

This can be seen in universe constraints of this theorem

- For \(C : \text{Category}@\{k, 1\}\) we get the restriction that \(k \leq 1\)
- This is in contradiction with the fact that \(\text{Set} : \text{Category}@\{m, n\}\) with \(m > n\)
This notion of smallness/largeness using universe levels works so well that we can define \textbf{Cat}:

\textbf{Instance} \textbf{Cat} : \textbf{Category} := \{\text{Obj} := \text{Category}; \text{Hom} := \text{Functor}; \ldots\}

Or prove the following:

\textit{Theorem} \text{Complete_Preorder} (C : \text{Category}) (CC : \text{Complete C}) :
\hspace{1em} \forall \ x \ y : \text{(Obj C)}, \text{Hom} \ x \ y' \simeq ((\text{Arrow C}) \to \text{Hom} \ x \ y)

This theorem results in a contradiction as soon as there are objects \(a\) and \(b\) in \(C\) such that \(|\text{hom}(a, b)| \geq 2\)

In fact, this theorem holds only for small categories

This can be seen in universe constraints of this theorem

\begin{itemize}
  \item For \(C : \text{Category}\{k, 1\}\) we get the restriction that \(k \leq 1\)
  \item This is in contradiction with the fact that \(\text{Set} : \text{Category}\{m, n\}\) with \(m > n\)
\end{itemize}

\begin{verbatim}
Set@{m, n} :=
{|
  Obj := Type@{n} : Type@{m};
  Hom := fun A B ⇒ A → B : Obj → Obj → Type@{n}; ...
|} : Category@{m, n}
\end{verbatim}
Cat in Coq:

Instance **Cat** : Category@{i, j} := {Obj := Category@{k, l}; Hom := Functor; ...}
Cat in Coq:

```
Instance Cat : Category@{i, j} := {Obj := Category@{k, l}; Hom := Functor; ...}
```

But according to Coq’s universe polymorphism, if \( C : \text{Category}@{k, l} \) and \( C : \text{Category}@{k', l'} \), we must have \( k = k' \) and \( l = l' \).
Cat in Coq:

Instance Cat : Category@{i, j} := {Obj := Category@{k, l}; Hom := Functor; ...}

But according to Coq’s universe polymorphism, if C : Category@{k, l} and
C' : Category@{k', l'}, we must have k = k' and l = l'

This means Cat@{i, j, k, l} is not the category of all categories at level (k, l) or
lower but only at level (k, l)
\textbf{Cat in Coq:}

\begin{verbatim}
Instance \textbf{Cat} : Category\{@\{i, j\} := \{Obj := Category\{@\{k, l\}; Hom := Functor; ...\}
\end{verbatim}

- But according to Coq’s universe polymorphism, if \( C : \text{Category}\{@\{k, l\} \) and \( C : \text{Category}\{@\{k’, l’\} \), we must have \( k = k’ \) and \( l = l’ \)

- This means \( \text{Cat}\{@\{i, j, k, l\} \) is not the category of all categories at level \( (k, l) \) or lower but \textbf{only} at level \( (k, l) \)

- We can lift category:

\begin{verbatim}
lift (C : \text{Category}\{@\{k, l\}) : \text{Category}\{@\{k’, l’\} :=
{\{|
  Obj := Obj C;
  Hom := Hom C;

    ;
  ;
|}

for \( k < k’ \) and \( l < l’ \)
\end{verbatim}
Cat in Coq:

\[
\text{Instance } \textbf{Cat} : \text{Category} @ \{ i, j \} := \{ \text{Obj} := \text{Category} @ \{ k, l \}; \text{Hom} := \text{Functor}; \ldots \}
\]

But according to Coq’s universe polymorphism, if \( C : \text{Category} @ \{ k, l \} \) and \( C : \text{Category} @ \{ k', l' \} \), we must have \( k = k' \) and \( l = l' \)

This means \( \text{Cat} @ \{ i, j, k, l \} \) is not the category of all categories at level \( (k, l) \) or lower but only at level \( (k, l) \)

We can lift category:

\[
\text{lift } (C : \text{Category} @ \{ k, l \}) : \text{Category} @ \{ k', l' \} :=
\]

\[
\{ |
\text{Obj} := \text{Obj } C;
\text{Hom} := \text{Hom } C;
\}
\]

\[
\}
\]

for \( k < k' \) and \( l < l' \)

But

- We can’t prove or even specify (universe inconsistency)

\[
\text{forall } (C : \text{Category}), C = \text{lift } C
\]
Cat in Coq:

\[ \text{Instance } \textbf{Cat} : \text{Category}\{i, j\} := \{ \text{Obj} := \text{Category}\{k, l\}; \text{Hom} := \text{Functor}; \ldots \} \]

But according to Coq’s universe polymorphism, if \( C : \text{Category}\{k, l\} \) and \( C : \text{Category}\{k', l'\} \), we must have \( k = k' \) and \( l = l' \)

This means \( \text{Cat}\{i, j, k, l\} \) is not the category of all categories at level \((k, l)\) or lower but \textit{only} at level \((k, l)\)

We can lift category:

\[
\text{lift } (C : \text{Category}\{k, l\}) : \text{Category}\{k', l'\} := \\
\{ |
\text{Obj} := \text{Obj } C;
\text{Hom} := \text{Hom } C;
\text{...} \\
\}
\]

for \( k < k' \) and \( l < l' \)

But

- We can’t prove or even specify (universe inconsistency)
  \[ \text{forall } (C : \text{Category}), C = \text{lift } C \]
- We can’t prove \textit{forall} \((C : \text{Category}), \text{JMeq } C \ (\text{lift } C)\)
**Cat in Coq:**

\[
\begin{align*}
\text{Instance } \textbf{Cat}: \text{Category}_{\{i, j\}} := \{\text{Obj} := \text{Category}_{\{k, l\}}; \text{Hom} := \text{Functor}; \ldots\} \\
\end{align*}
\]

But according to Coq’s universe polymorphism, if \( C : \text{Category}_{\{k, l\}} \) and \( C' : \text{Category}_{\{k', l'\}} \), we must have \( k = k' \) and \( l = l' \).

This means \( \text{Cat}_{\{i, j, k, l\}} \) is not the category of all categories at level \((k, l)\) or lower but only at level \((k, l)\).

We can lift category:

\[
\begin{align*}
lift (C : \text{Category}_{\{k, l\}}) : \text{Category}_{\{k', l'\}} := \\
\{\|
\text{Obj} := \text{Obj} \ C; \\
\text{Hom} := \text{Hom} \ C; \\
\vdots \\
|\}
\end{align*}
\]

for \( k < k' \) and \( l < l' \).

But

- We can’t prove or even specify (universe inconsistency) \( \forall C : \text{Category}, C = \text{lift} \ C \)
- We can’t prove \( \forall C : \text{Category}, \text{JMeq} \ C (\text{lift} \ C) \)
- The equality \( \forall C : \text{Category}, \text{lift} \ C = \text{lift} (\text{lift} \ C) \) is not definitional
If we show that $\text{Cat}^{\{i, j, k, l\}}$ has exponentials, we get the constraints that $j = k = l$
If we show that $\text{Cat}^{\{i, j, k, 1\}}$ has exponentials, we get the constraints that $j = k = l$

Therefore, no copy of $\text{Set}$ is in a copy of $\text{Cat}$ in which we have exponentials
If we show that $\text{Cat}^{i, j, k, l}$ has exponentials, we get the constraints that $j = k = l$

Therefore, no copy of $\text{Set}$ is in a copy of $\text{Cat}$ in which we have exponentials

That means we can’t define Yoneda embedding as exponential transpose (currying) of the hom functor
If we show that $\textbf{Cat}^{\{i, j, k, l\}}$ has exponentials, we get the constraints that $j = k = l$

Therefore, no copy of $\textbf{Set}$ is in a copy of $\textbf{Cat}$ in which we have exponentials

That means we can’t define Yoneda embedding as exponential transpose (currying) of the hom functor

Defining Yoneda separately, it still can only be applied in a category $\mathcal{C}: \text{Category}^{\{i, j\}}$ if $i = j$. 
If we show that $\text{Cat}\{i, j, k, l\}$ has exponentials, we get the constraints that $j = k = l$

Therefore, no copy of $\text{Set}$ is in a copy of $\text{Cat}$ in which we have exponentials

That means we can’t define Yoneda embedding as exponential transpose (currying) of the hom functor

Defining Yoneda separately, it still can only be applied in a category $\mathcal{C} : \text{Category}\{i, j\}$ if $i = j$.

We can use Yoneda to prove that in any cartesian closed category:

$$(a^b)^c \simeq a^{b \times c}$$

but this lemma can’t be applied to $\text{Cat}$ or $\text{Set}$
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism).

Assume for $F, F'$: $C \to D$, we have $F \dashv G$ and $F' \dashv G$, i.e.,

$$\text{hom}_D(F, -) \cong \text{hom}_C(-, G)$$

Thus we have:

$$\text{hom}_D(F, -) \cong \text{hom}_D(F', -)$$

but for $H, H'$: $C \times C' \to D$, $H \cong H'$ iff $\text{curry}(H) \cong \text{curry}(H')$

1. For $f : a \times b \to c$ we have $\text{curry}(f) : a \to c^b$.

But according to axioms of exponentials we have

$$\text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D)$$

Which means:

$$F \circ Y_D \cong F' \circ Y_D$$

This immediately gives $F \cong F'$ as $Y_D$ (the Yoneda embedding for $D$) is an embedding.

But, we can't use the general fact above, as it involves both exponentials and $\text{Set}$ (through $\text{hom}$) in $\text{Cat}$ – we have proven a separate instance of this fact for $\text{Cat}_1$ for $f : a \times b \to c$ we have $\text{curry}(f) : a \to c^b$. 

---

1 For $f : a \times b \to c$ we have $\text{curry}(f) : a \to c^b$. 

---

Amin Timany  Bart Jacobs  Category Theory in Coq 8.5
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism)
Assume for $F, F' : C \to D : G$, we have $F \dashv G$ and $F' \dashv G$, i.e.,
\[
\hom_D(F, -) \simeq \hom_C(-, G) \text{ and } \hom_D(F', -) \simeq \hom_C(-, G)
\]
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism).
Assume for $F, F' : C \to D : G$, we have $F \dashv G$ and $F' \dashv G$, i.e.,

$$hom_D(F, -) \simeq hom_C(-, G) \text{ and } hom_D(F', -) \simeq hom_C(-, G)$$

Thus we have:

$$hom_D(F, -) \simeq hom_D(F', -)$$

\footnote{for $f : a \times b \to c$ we have $curry(f) : a \to c^b$}
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism). Assume for $F, F' : C \to D : G$, we have $F \dashv G$ and $F' \dashv G$, i.e.,

$$\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \quad \text{and} \quad \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)$$

Thus we have:

$$\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)$$

but for $H, H' : C \times C' \to D$, $H \simeq H'$ iff $\text{curry}(H) \simeq \text{curry}(H')$\(^1\)

---

\(^1\)for $f : a \times b \to c$ we have $\text{curry}(f) : a \to c^b$
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism)
Assume for \( F, F' : C \to D : G \), we have \( F \dashv G \) and \( F' \dashv G \), i.e.,
\[
\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \quad \text{and} \quad \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)
\]
Thus we have:
\[
\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)
\]
but for \( H, H' : C \times C' \to D \), \( H \simeq H' \) \iff \( \text{curry}(H) \simeq \text{curry}(H') \)
so, we can assume:
\[
\text{curry}(\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -))
\]

\[1\text{for } f : a \times b \to c \text{ we have } \text{curry}(f) : a \to c^b\]
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism).
Assume for $F, F' : C \to D : G$, we have $F \dashv G$ and $F' \dashv G$, i.e.,

$$\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \text{ and } \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)$$

Thus we have:

$$\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)$$

but for $H, H' : C \times C' \to D$, $H \simeq H'$ iff $\text{curry}(H) \simeq \text{curry}(H')$

so, we can assume:

$$\text{curry}(\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -))$$

But according to axioms of exponentials we have

$$\text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D)$$

\[1\] for $f : a \times b \to c$ we have $\text{curry}(f) : a \to c^b$
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism)
Assume for $F, F' : C \to D : G$, we have $F \dashv G$ and $F' \dashv G$, i.e.,

\[ \text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \text{ and } \text{hom}_D(F', -) \simeq \text{hom}_C(-, G) \]

Thus we have:

\[ \text{hom}_D(F, -) \simeq \text{hom}_D(F', -) \]

but for $H, H' : C \times C' \to D$, $H \simeq H' \text{ iff curry}(H) \simeq \text{curry}(H')^1$

so, we can assume:

\[ \text{curry} (\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -)) \]

But according to axioms of exponentials we have

\[ \text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D) \]

Which means:

\[ F \circ Y_D \simeq F' \circ Y_D \]

\[ ^1 \text{for } f : a \times b \to c \text{ we have } \text{curry}(f) : a \to c^b \]
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism). Assume for \( F, F' : C \to D : G \), we have \( F \dashv G \) and \( F' \dashv G \), i.e.,

\[
\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \text{ and } \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)
\]

Thus we have:

\[
\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)
\]

but for \( H, H' : C \times C' \to D \), \( H \simeq H' \) iff \( \text{curry}(H) \simeq \text{curry}(H') \)

so, we can assume:

\[
\text{curry}(\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -))
\]

But according to axioms of exponentials we have

\[
\text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D)
\]

Which means:

\[
F \circ Y_D \simeq F' \circ Y_D
\]

This immediately gives \( F \simeq F' \) as \( Y_D \) (the Yoneda embedding for \( D \)) is an embedding.

---

\(^1\text{for } f : a \times b \to c \text{ we have } \text{curry}(f) : a \to c^b\)
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism). Assume for \( F, F' : C \to D : G \), we have \( F \dashv G \) and \( F' \dashv G \), i.e.,

\[
\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \quad \text{and} \quad \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)
\]

Thus we have:

\[
\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)
\]

but for \( H, H' : C \times C' \to D \), \( H \simeq H' \) iff \( \text{curry}(H) \simeq \text{curry}(H') \)

so, we can assume:

\[
\text{curry}(\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -))
\]

But according to axioms of exponentials we have

\[
\text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D)
\]

Which means:

\[
F \circ Y_D \simeq F' \circ Y_D
\]

This immediately gives \( F \simeq F' \) as \( Y_D \) (the Yoneda embedding for \( D \)) is an embedding.

But, we can’t use the general fact above, as it involves both exponentials and \( \textbf{Set} \) (through \( \text{hom} \)) in \( \textbf{Cat} \)

\(^1\text{for } f : a \times b \to c \text{ we have } \text{curry}(f) : a \to c^b\)
Consider our proof of uniqueness of adjoint functors (up to natural isomorphism)
Assume for \( F, F' : C \to D : G \), we have \( F \dashv G \) and \( F' \dashv G \), i.e.,
\[
\text{hom}_D(F, -) \simeq \text{hom}_C(-, G) \quad \text{and} \quad \text{hom}_D(F', -) \simeq \text{hom}_C(-, G)
\]
Thus we have:
\[
\text{hom}_D(F, -) \simeq \text{hom}_D(F', -)
\]
but for \( H, H' : C \times C' \to D \), \( H \simeq H' \) iff \( \text{curry}(H) \simeq \text{curry}(H') \)^1
so, we can assume:
\[
\text{curry}(\text{hom}_D(F, -)) \simeq \text{curry}(\text{hom}_D(F', -))
\]
But according to axioms of exponentials we have
\[
\text{curry}(\text{hom}_D(F, -)) = F \circ \text{curry}(\text{hom}_D)
\]
Which means:
\[
F \circ Y_D \simeq F' \circ Y_D
\]
This immediately gives \( F \simeq F' \) as \( Y_D \) (the Yoneda embedding for \( D \)) is an embedding

But, we can’t use the general fact above, as it involves both exponentials and \( \text{Set} \) (through \( \text{hom} \)) in \( \text{cat} \) – we have proven a separate instance of this fact for \( \text{Cat} \)

---

^1 for \( f : a \times b \to c \) we have \( \text{curry}(f) : a \to c^b \)
Another issue that we faced is that \( \text{Set} \) seems to have a special place:

- If we show that \( \text{Set} : \text{Category}@\{i, j\} \) has \( \text{unit} : \text{Set} \) as the terminal object, we get the restriction \( j = \text{Set} \)
Another issue that we faced is that \textit{Set} seems to have a special place:

- If we show that \textit{Set} : \textit{Category}\{i, j\} has unit : \textit{Set} as the terminal object, we get the restriction \( j = \text{Set} \).
- The problem occurs when we want to show that Prop is the subobject classifier for \textit{Set}. As then we need a monic arrow:

\[ tr : \text{unit} \rightarrow \text{Prop} \]

and \( \text{unit} \rightarrow \text{Prop} \) is not a term of type \textit{Set}. 

We therefore postulate existence of a universe polymorphic singleton type:

\begin{verbatim}
Parameter UNIT : Type.
Parameter TT : UNIT.
Axiom UNIT_SINGLETON : ∀ x y : UNIT, x = y.
\end{verbatim}
Another issue that we faced is that $\text{Set}$ seems to have a special place:

- If we show that $\text{Set} : \text{Category}@\{i, j\}$ has $\text{unit} : \text{Set}$ as the terminal object, we get the restriction $j = \text{Set}$
- The problem occurs when we want to show that $\text{Prop}$ is the subobject classifier for $\text{Set}$. As then we need a monic arrow:

$$tr : \text{unit} \to \text{Prop}$$

and $\text{unit} \to \text{Prop}$ is not a term of type $\text{Set}$
- This can be solved by defining a singleton inductive type at a level strictly higher than $\text{Set}$
Another issue that we faced is that $\textsf{Set}$ seems to have a special place:

- If we show that $\textsf{Set} : \textsf{Category@\{i, j\}}$ has $\textsf{unit} : \textsf{Set}$ as the terminal object, we get the restriction $j = \textsf{Set}$
- The problem occurs when we want to show that $\textsf{Prop}$ is the subobject classifier for $\textsf{Set}$. As then we need a monic arrow:

$$tr : \textsf{unit} \rightarrow \textsf{Prop}$$

and $\textsf{unit} \rightarrow \textsf{Prop}$ is not a term of type $\textsf{Set}$

- This can be solved by defining a singleton inductive type at a level strictly higher than $\textsf{Set}$
- But, that would cause a problem for the part where we show that type $\textsf{nat} : \textsf{Set}$ of the library of Coq is the initial algebra for $T(X) = 1 + X$ in category $\textsf{Set}$
Another issue that we faced is that \texttt{Set} seems to have a special place:

- If we show that \texttt{Set} \texttt{: Category@\{i, j\}} has \texttt{unit} : \texttt{Set} as the terminal object, we get the restriction \( j = \texttt{Set} \)
- The problem occurs when we want to show that \texttt{Prop} is the subobject classifier for \texttt{Set}. As then we need a monic arrow:

\[
    tr : \texttt{unit} \to \texttt{Prop}
\]

and \texttt{unit} \to \texttt{Prop} is not a term of type \texttt{Set}

- This can be solved by defining a singleton inductive type at a level \textit{strictly} higher than \texttt{Set}
- But, that would cause a problem for the part where we show that type \texttt{nat} : \texttt{Set} of the library of Coq is the initial algebra for \( T(X) = 1 + X \) in category \texttt{Set}
- We therefore postulate existence of a universe polymorphic singleton type:

\begin{verbatim}
Parameter UNIT : Type.
Parameter TT : UNIT.
Axiom UNIT_SINGLETON : forall x y : UNIT, x = y.
\end{verbatim}
Conclusion:

- We presented an implementation of category theory covering some of the basic category theory.
Conclusion:

- We presented an implementation of category theory covering some of the basic category theory.
- We use features of Coq 8.5: primitive projections and universe polymorphism.

Universe polymorphism to represent smallness/largeness. This works well to a degree that we don’t need to mention any universe levels and can prove things like: $\text{Cat}$ and $\text{Complete_Preorder}$. It also has shortcomings: e.g., can’t use Yoneda in $\text{Cat}$ and $\text{Set}$. 
Conclusion:

- We presented an implementation of category theory covering some of the basic category theory.
- We use features of Coq 8.5: primitive projections and universe polymorphism.
- Universe polymorphism to represent smallness/largeness.
Conclusion:

- We presented an implementation of category theory covering some of the basic category theory.
- We use features of Coq 8.5: primitive projections and universe polymorphism.
- Universe polymorphism to represent smallness/largeness.
  - This works well to a degree that we don’t need to mention any universe levels and can prove things like: \text{Cat} and \text{Complete_Preorder}.
Conclusion:

- We presented an implementation of category theory covering some of the basic category theory.
- We use features of Coq 8.5: primitive projections and universe polymorphism.
- Universe polymorphism to represent smallness/largeness.
  - This works well to a degree that we don’t need to mention any universe levels and can prove things like: `Cat` and `Complete_Preorder`.
  - It also has shortcomings: e.g., can’t use Yoneda in `Cat` and `Set`.