Type theory in Lean - 6

Riccardo Brasca

Université Paris Cité

November 18th 2023

Equality

Let's analyze one last example of an inductive proposition, the equality $a = b$.

Equality

Let's analyze one last example of an inductive proposition, the equality $a = b$.

In type theory this is a precisely defined notion, without anything special.

Equality

Let's analyze one last example of an inductive proposition, the equality $a = b$.

In type theory this is a precisely defined notion, without anything special. Here is the definition

```
inductive Eq {A : Sort u} : A → A → Prop
| refl (a : A) : Eq a a

infix:50 " = "  => Eq
```

Equality

Let's analyze one last example of an inductive proposition, the equality $a = b$.

In type theory this is a precisely defined notion, without anything special. Here is the definition

```
inductive Eq {A : Sort u} : A → A → Prop
| refl (a : A) : Eq a a

infix:50 " = "  => Eq
```

In particular, $a = b$ is just notation for the proposition $\mathrm{Eq}\ a\ b$.

Equality

Let's analyze one last example of an inductive proposition, the equality $a = b$.

In type theory this is a precisely defined notion, without anything special. Here is the definition

```
inductive Eq {A : Sort u} : A → A → Prop
| refl (a : A) : Eq a a

infix:50 " = "  => Eq
```

In particular, $a = b$ is just notation for the proposition $\mathrm{Eq}\ a\ b$. There is not need to specify A here, it is guessed looking at the type of a and b, that must be of the same type.

Remark

- *It is an inductive family of propositions rather than a single proposition.*

Remark

- *It is an inductive family of propositions rather than a single proposition.*
- $a = b$ *only makes sense if* a *and* b *are terms of the same type.*

Remark

- *It is an inductive family of propositions rather than a single proposition.*
- $a = b$ *only makes sense if* a *and* b *are terms of the same type.*

We call the relation $a = b$ *propositional equality*.

Remark

- *It is an inductive family of propositions rather than a single proposition.*
- $a = b$ *only makes sense if* a *and* b *are terms of the same type.*

We call the relation $a = b$ *propositional equality*. Let's have a look at the usual rules.

Remark

- *It is an inductive family of propositions rather than a single proposition.*
- *$a = b$ only makes sense if a and b are terms of the same type.*

We call the relation $a = b$ *propositional equality*. Let's have a look at the usual rules.

- Formation rule: if $(A : \mathrm{Sort}\ u)$ and $(a\ b : A)$, we have a proposition $\mathrm{Eq}\ a\ b$, that we denote $a = b$.

Remark

- *It is an inductive family of propositions rather than a single proposition.*
- *$a = b$ only makes sense if a and b are terms of the same type.*

We call the relation $a = b$ *propositional equality*. Let's have a look at the usual rules.

- Formation rule: if $(A : \mathrm{Sort}\ u)$ and $(a\ b : A)$, we have a proposition $\mathrm{Eq}\ a\ b$, that we denote $a = b$.
- Constructors: there is only one constructor, called refl. Given $(a : A)$,

  $$(\mathrm{refl}\ a \ : a = a)$$

  In particular $\mathrm{refl}\ a$ is a proof that $a = a$.

Remark

If $a \equiv b$*, then* $(\mathrm{refl}\ a\ : a = b)$ *is accepted by Lean, since there is no difference between a and b. In particular, definitional equality implies propositional equality.*

Remark

If $a \equiv b$, then $(\mathrm{refl}\ a\ : a = b)$ is accepted by Lean, since there is no difference between a and b. In particular, definitional equality implies propositional equality.

- Eliminator. It is the most subtle aspect of the notion of equality. Here is the most general form.

Remark

If $a \equiv b$, then $(\mathrm{refl}\ a\ : a = b)$ is accepted by Lean, since there is no difference between a and b. In particular, definitional equality implies propositional equality.

- Eliminator. It is the most subtle aspect of the notion of equality. Here is the most general form.

```
#check @Eq.rec

{A : Sort u} → {a : A} →
{motive : (b : A) → Eq a b → Sort v} →
motive a (_ : Eq a a) → {b : A} →
(t : Eq a b) → motive b t
```

To better understand it, let's first of all fix $(A : \mathrm{Sort}\ u)$ and $(a : A)$ and let's consider `@Eq.rec A a`.

To better understand it, let's first of all fix $(A : \mathrm{Sort}\ u)$ and $(a : A)$ and let's consider `@Eq.rec A a`.

```
#check @Eq.rec A a

{motive : (b : A) → Eq a b → Sort v} →
motive a (_ : Eq a a) → {b : A} →
(t : Eq a b) → motive b t
```

To better understand it, let's first of all fix $(A : \mathrm{Sort}\ u)$ and $(a : A)$ and let's consider `@Eq.rec A a`.

```
#check @Eq.rec A a

{motive : (b : A) → Eq a b → Sort v} →
motive a (_ : Eq a a) → {b : A} →
(t : Eq a b) → motive b t
```

The next thing we want to simplify is the motive. It is a dependent function

$$(b : A) \to \mathrm{Eq}\ a\ b \to \mathrm{Sort}\ v$$

To better understand it, let's first of all fix $(A : \mathrm{Sort}\ u)$ and $(a : A)$ and let's consider `@Eq.rec A a`.

```
#check @Eq.rec A a

{motive : (b : A) → Eq a b → Sort v} →
motive a (_ : Eq a a) → {b : A} →
(t : Eq a b) → motive b t
```

The next thing we want to simplify is the motive. It is a dependent function

$$(b : A) \to \mathrm{Eq}\ a\ b \to \mathrm{Sort}\ v$$

Let's first of all consider the case of propositions, so $v = 0$ and $\mathrm{Sort}\ v = \mathrm{Prop}$.

To specify the motive we need a proposition $P\ b$ (depending on b), defined given h, a proof that $a = b$. For example, we can have a function

$$(P : A \to \mathrm{Prop}),$$

so a proposition $P\ b$ for all $(b : A)$ (regardless we can prove $a = b$).

To specify the motive we need a proposition $P\ b$ (depending on b), defined given h, a proof that $a = b$. For example, we can have a function

$$(P : A \to \mathrm{Prop}),$$

so a proposition $P\ b$ for all $(b : A)$ (regardless we can prove $a = b$). We can take for the motive the function

$$\mathrm{fun}\ b\ h \mapsto f\ b$$

To specify the motive we need a proposition $P\ b$ (depending on b), defined given h, a proof that $a = b$. For example, we can have a function

$$(P : A \to \mathrm{Prop}),$$

so a proposition $P\ b$ for all $(b : A)$ (regardless we can prove $a = b$). We can take for the motive the function

$$\mathrm{fun}\ b\ h \mapsto f\ b$$

We now have

```
#check @Eq.rec A a (fun b h ↦ f b)

P a → ∀ {b : A}, a = b → P b
```

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

This special case of the eliminator is called the *substitution principle*.

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

This special case of the eliminator is called the *substitution principle*.

Remark

Note that we don't need $P\ b$ to be defined for all $(b : A)$, but only if we already know that $a = b$.

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

This special case of the eliminator is called the *substitution principle*.

Remark

Note that we don't need $P\ b$ to be defined for all $(b : A)$, but only if we already know that $a = b$. This can look like we only have one single proposition, $P\ a$, but remember that we are defining $a = b$.

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

This special case of the eliminator is called the *substitution principle*.

Remark

Note that we don't need $P\ b$ to be defined for all $(b : A)$, but only if we already know that $a = b$. This can look like we only have one single proposition, $P\ a$, but remember that we are defining $a = b$. For example, replace $a = b$ by "a and b are friends".

So the eliminator just say that if we can prove $P\ a$ and we have $(b : A)$ such that $a = b$, then we have a proof of $P\ b$.

This special case of the eliminator is called the *substitution principle*.

Remark

Note that we don't need $P\ b$ to be defined for all $(b : A)$, but only if we already know that $a = b$. This can look like we only have one single proposition, $P\ a$, but remember that we are defining $a = b$. For example, replace $a = b$ by "a and b are friends". Then the eliminator says the following. Suppose we have a proposition $P\ b$ for all b friends of a. If $P\ a$ holds and b is a friend of a, then $P\ b$ holds.

Reflexivity

We now prove that equality is an equivalence relation.

Reflexivity

We now prove that equality is an equivalence relation.

Let's start with reflexivity.

Reflexivity

We now prove that equality is an equivalence relation.

Let's start with reflexivity. Let $(A : \mathrm{Sort}\ u)$ and let $(a : A)$.

Reflexivity

We now prove that equality is an equivalence relation.

Let's start with reflexivity. Let $(A : \mathrm{Sort}\ u)$ and let $(a : A)$. Then

$$(\mathrm{refl}\ a\ : a = a)$$

so $=$ is a reflexive relation.

Reflexivity

We now prove that equality is an equivalence relation.

Let's start with reflexivity. Let $(A : \mathrm{Sort}\ u)$ and let $(a : A)$. Then

$$(\mathrm{refl}\ a\ : a = a)$$

so $=$ is a reflexive relation.

Another way of thinking about $=$ is that it is the *smallest* reflexive relation (we will prove this in the examples).

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$.

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive.

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto x = a$$

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto x = a$$

Note that, as above, this function is defined on all $(x : A)$, not only when $a = x$ (the fact that $=$ appears also in the definition of P is irrelevant).

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto x = a$$

Note that, as above, this function is defined on all $(x : A)$, not only when $a = x$ (the fact that $=$ appears also in the definition of P is irrelevant).

We need to prove $P\ b$.

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto x = a$$

Note that, as above, this function is defined on all $(x : A)$, not only when $a = x$ (the fact that $=$ appears also in the definition of P is irrelevant).

We need to prove $P\ b$. The eliminator says exactly that if $P\ a$ holds and $a = b$, then $P\ b$ holds.

Symmetry

To prove that $=$ is a symmetric relation, let $(a\ b : A)$ and let $(h : a = b)$. We need to prove $b = a$.

We want to use the eliminator, so first of all we need to find the motive. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto x = a$$

Note that, as above, this function is defined on all $(x : A)$, not only when $a = x$ (the fact that $=$ appears also in the definition of P is irrelevant).

We need to prove $P\ b$. The eliminator says exactly that if $P\ a$ holds and $a = b$, then $P\ b$ holds. But $P\ a$ holds by reflexivity and $a = b$ is our assumption, so we are done.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

As above first of all we need the motive, to apply the eliminator.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

As above first of all we need the motive, to apply the eliminator. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto a = x$$

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

As above first of all we need the motive, to apply the eliminator. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto a = x$$

We need to prove $P\ c$.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

As above first of all we need the motive, to apply the eliminator. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto a = x$$

We need to prove $P\ c$. The eliminator says that it is enough to prove $P\ b$ and that $b = c$.

Transitivity

To prove that $=$ is a transitive relation, let $(a\ b\ c : A)$ and let $(hab : a = b)$ and $(hbc : b = c)$. We need to prove $a = c$.

As above first of all we need the motive, to apply the eliminator. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto a = x$$

We need to prove $P\ c$. The eliminator says that it is enough to prove $P\ b$ and that $b = c$.

We have $(hab : P\ b)$ and $(hbc : b = c)$, so we are done.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

To prove it we proceed as above.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

To prove it we proceed as above. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto f\ a = f\ x$$

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

To prove it we proceed as above. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto f\ a = f\ x$$

We need to prove $P\ b$.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

To prove it we proceed as above. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto f\ a = f\ x$$

We need to prove $P\ b$. The eliminator says that it is enough to prove $P\ a$ and that $a = b$.

Substitution

Let $(B : \mathrm{Sort}\ v)$ be a type and let $(f : A \to B)$ be a function. If $(a\ b : A)$ with $a = b$, then

$$f\ a = f\ b$$

This result is called the *substitution principle*.

To prove it we proceed as above. Let's consider the function $(P : A \to \mathrm{Prop})$ given by

$$\mathrm{fun}\ (x : A) \mapsto f\ a = f\ x$$

We need to prove $P\ b$. The eliminator says that it is enough to prove $P\ a$ and that $a = b$. But $P\ a$ holds by reflexivity (applied to the term $f\ a$), and $a = b$ is our assumption.

Corollary

In particular, if $(f : A \to \mathrm{Prop})$*, we have that, if* $a = b$*, then* $f\ a \iff f\ b$ *as expected.*

Corollary

In particular, if $(f : A \to \mathrm{Prop})$*, we have that, if* $a = b$*, then* $f\ a \iff f\ b$ *as expected.*

Remark

Let $(B : A \to \mathrm{Sort}\ v)$ *be a function and let*

$$\left(f : \prod_{(a:A)} B\ a\right)$$

be a dependent function.

Corollary

In particular, if $(f : A \to \mathrm{Prop})$*, we have that, if* $a = b$*, then* $f\ a \iff f\ b$ *as expected.*

Remark

Let $(B : A \to \mathrm{Sort}\ v)$ *be a function and let*

$$\left(f : \prod_{(a:A)} B\ a\right)$$

be a dependent function. If $(a\ b : A)$ *are terms such that* $a = b$*, can we deduce that* $f\ a = f\ b$*?*

Corollary

In particular, if $(f : A \to \mathrm{Prop})$, *we have that, if* $a = b$, *then* $f\ a \iff f\ b$ *as expected.*

Remark

Let $(B : A \to \mathrm{Sort}\ v)$ *be a function and let*

$$\left(f : \prod_{(a:A)} B\ a\right)$$

be a dependent function. If $(a\ b : A)$ *are terms such that* $a = b$, *can we deduce that* $f\ a = f\ b$*? This does not even make sense! Indeed* $(f\ a : B\ a)$ *and* $(f\ b : B\ b)$ *have not the same type.*

Corollary

In particular, if $(f : A \to \mathrm{Prop})$*, we have that, if* $a = b$*, then* $f\ a \iff f\ b$ *as expected.*

Remark

Let $(B : A \to \mathrm{Sort}\ v)$ *be a function and let*

$$\left(f : \prod_{(a:A)} B\ a\right)$$

be a dependent function. If $(a\ b : A)$ *are terms such that* $a = b$*, can we deduce that* $f\ a = f\ b$*? This does not even make sense! Indeed* $(f\ a : B\ a)$ *and* $(f\ b : B\ b)$ *have not the same type.*
There exists a generalization of equality to take into account this situation (called heterogenous equality), but it is less interesting.

Remark

- *Note that $=$ is a syntactic subsingleton, so the eliminator allows to define functions to any type.*

Remark

- *Note that $=$ is a syntactic subsingleton, so the eliminator allows to define functions to any type.*
- *When using that $a = b$, the idea is that if we need to prove/construct something for b it is enough to do it for a.*

Remark

- *Note that* $=$ *is a syntactic subsingleton, so the eliminator allows to define functions to any type.*
- *When using that* $a = b$*, the idea is that if we need to prove/construct something for* b *it is enough to do it for* a*. But beware that this "something" must be well defined.*

Remark

- *Note that* $=$ *is a syntactic subsingleton, so the eliminator allows to define functions to any type.*
- *When using that* $a = b$*, the idea is that if we need to prove/construct something for* b *it is enough to do it for* a*. But beware that this "something" must be well defined. More precisely, finding the motive is often the main point of the proof.*

Remark

- *Note that* $=$ *is a syntactic subsingleton, so the eliminator allows to define functions to any type.*
- *When using that* $a = b$*, the idea is that if we need to prove/construct something for* b *it is enough to do it for* a*. But beware that this "something" must be well defined. More precisely, finding the motive is often the main point of the proof. It should be well defined for all* x *such that* $a = x$*.*

Uniqueness of refl

If $(h\ h' : a = b)$ are two proofs that $a = b$, does it follow that $h = h'$? Yes, because by proof irrelevance $h \equiv h'$ and definitional equality implies propositional equality.

Uniqueness of refl

If $(h\ h' : a = b)$ are two proofs that $a = b$, does it follow that $h = h'$? Yes, because by proof irrelevance $h \equiv h'$ and definitional equality implies propositional equality.

It is interesting to try to prove it without using proof irrelevance.

Uniqueness of refl

If $(h\ h' : a = b)$ are two proofs that $a = b$, does it follow that $h = h'$? Yes, because by proof irrelevance $h \equiv h'$ and definitional equality implies propositional equality.

It is interesting to try to prove it without using proof irrelevance. Of course this is not technically possible in Lean, but let's have a look.

Uniqueness of refl

If $(h\ h' : a = b)$ are two proofs that $a = b$, does it follow that $h = h'$? Yes, because by proof irrelevance $h \equiv h'$ and definitional equality implies propositional equality.

It is interesting to try to prove it without using proof irrelevance. Of course this is not technically possible in Lean, but let's have a look.
We consider the motive M given by

$$\mathrm{fun}\ (x : A)\ (e : a = x) \mapsto \forall\, (e' : a = x), e = e'$$

Uniqueness of refl

If $(h\ h' : a = b)$ are two proofs that $a = b$, does it follow that $h = h'$? Yes, because by proof irrelevance $h \equiv h'$ and definitional equality implies propositional equality.

It is interesting to try to prove it without using proof irrelevance. Of course this is not technically possible in Lean, but let's have a look.
We consider the motive M given by

$$\mathrm{fun}\ (x : A)\ (e : a = x) \mapsto \forall\, (e' : a = x), e = e'$$

In particular, $M\ b\ h$ is the statement

$$\forall\, (h' : a = b), h = h'$$

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof.

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

The eliminator allows precisely to build such a term, that is a dependent function.

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

The eliminator allows precisely to build such a term, that is a dependent function.

It says that to build t it is enough to do so in the cases b is a and h is $\mathrm{Eq.refl}\ a$.

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

The eliminator allows precisely to build such a term, that is a dependent function.

It says that to build t it is enough to do so in the cases b is a and h is $\mathrm{Eq.refl}\ a$. To build $M\ a\ (\mathrm{Eq.refl}\ a)$, that is a dependent function, we can of course use lambda abstraction.

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

The eliminator allows precisely to build such a term, that is a dependent function.

It says that to build t it is enough to do so in the cases b is a and h is $\mathrm{Eq.refl}\ a$. To build $M\ a\ (\mathrm{Eq.refl}\ a)$, that is a dependent function, we can of course use lambda abstraction.

In practice the eliminator says it is enough to prove the following

$$\forall\, (e' : a = a), \mathrm{Eq.refl}\ a = e'$$

It follows that if we are able to construct a term $(t : M\ b\ h)$ then $t\ h'$ finishes the proof. Note that in this case the term we want to define, $t\ h'$, only makes sense in the presence of $(h : a = b)$.

The eliminator allows precisely to build such a term, that is a dependent function.

It says that to build t it is enough to do so in the cases b is a and h is $\mathrm{Eq.refl}\ a$. To build $M\ a\ (\mathrm{Eq.refl}\ a)$, that is a dependent function, we can of course use lambda abstraction.

In practice the eliminator says it is enough to prove the following

$$\forall\,(e' : a = a), \mathrm{Eq.refl}\ a = e'$$

One can think to use again the eliminator.

One can think to use again the eliminator. If we play a similar game as above, considering the motive N given by

$$\text{fun } (x : A)\ (f : a = x) \mapsto \text{Eq.refl } a = e'$$

One can think to use again the eliminator. If we play a similar game as above, considering the motive N given by

$$\mathrm{fun}\ (x : A)\ (f : a = x) \mapsto \mathrm{Eq.refl}\ a = e'$$

we obtain a new goal that is identical to the old one, so we didn't progress.

One can think to use again the eliminator. If we play a similar game as above, considering the motive N given by

$$\mathrm{fun}\ (x : A)\ (f : a = x) \mapsto \mathrm{Eq.refl}\ a = e'$$

we obtain a new goal that is identical to the old one, so we didn't progress. The problem is that x and f don't appear after the $\mapsto$ symbol, so the eliminator does not help.

One can think to use again the eliminator. If we play a similar game as above, considering the motive N given by

$$\mathrm{fun}\ (x : A)\ (f : a = x) \mapsto \mathrm{Eq.refl}\ a = e'$$

we obtain a new goal that is identical to the old one, so we didn't progress. The problem is that x and f don't appear after the $\mapsto$ symbol, so the eliminator does not help.

You can try, but you soon realize that there is no meaningful way to generalize $\mathrm{Eq.refl}\ a = e'$ to use the eliminator (more precisely one cannot find a useful motive).

One can think to use again the eliminator. If we play a similar game as above, considering the motive N given by

$$\text{fun}\ (x : A)\ (f : a = x) \mapsto \text{Eq.refl}\ a = e'$$

we obtain a new goal that is identical to the old one, so we didn't progress. The problem is that x and f don't appear after the $\mapsto$ symbol, so the eliminator does not help.

You can try, but you soon realize that there is no meaningful way to generalize $\text{Eq.refl}\ a = e'$ to use the eliminator (more precisely one cannot find a useful motive).

This problem is indeed undecidable in proof relevant type theory.

Sets

We now move on to define sets and their basic properties.

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$.

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$. The idea is that if P is provable then $P = \mathrm{True}$, otherwise $P = \mathrm{False}$.

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$. The idea is that if P is provable then $P = \mathrm{True}$, otherwise $P = \mathrm{False}$.

If $(A : \mathrm{Type}\ u)$ is any type, we define $\mathrm{Set}\ A$ as

$$A \to \mathrm{Prop}$$

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$. The idea is that if P is provable then $P = \mathrm{True}$, otherwise $P = \mathrm{False}$.

If $(A : \mathrm{Type}\ u)$ is any type, we define $\mathrm{Set}\ A$ as

$$A \to \mathrm{Prop}$$

In particular, $(\mathrm{Set}\ A : \mathrm{Type}\ u)$ is a new type. Its terms will be called *sets of elements of type A*.

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$. The idea is that if P is provable then $P = \mathrm{True}$, otherwise $P = \mathrm{False}$.

If $(A : \mathrm{Type}\ u)$ is any type, we define $\mathrm{Set}\ A$ as

$$A \to \mathrm{Prop}$$

In particular, $(\mathrm{Set}\ A : \mathrm{Type}\ u)$ is a new type. Its terms will be called *sets of elements of type A*.

Remark

Note that, in contrast to ZF, Set *is not at all a primitive notion.*

Sets

We now move on to define sets and their basic properties.

We will see that, thanks to the additional axioms used in mathlib, if $(P : \mathrm{Prop})$, then $P = \mathrm{True}$ or $P = \mathrm{False}$. The idea is that if P is provable then $P = \mathrm{True}$, otherwise $P = \mathrm{False}$.

If $(A : \mathrm{Type}\ u)$ is any type, we define $\mathrm{Set}\ A$ as

$$A \to \mathrm{Prop}$$

In particular, $(\mathrm{Set}\ A : \mathrm{Type}\ u)$ is a new type. Its terms will be called *sets of elements of type A*.

Remark

Note that, in contrast to ZF, Set *is not at all a primitive notion. Moreover, we need to specify a type before speaking about sets.*

We can now define the usual operations on sets. We fix $(A : \mathrm{Type}\ u)$.

We can now define the usual operations on sets. We fix $(A : \mathrm{Type}\ u)$.

Definition

Let $(S\ T : \mathrm{Set}\ A)$. Remember that this means that

$$(S\ T : A \to \mathrm{Prop})$$

We can now define the usual operations on sets. We fix $(A : \mathrm{Type}\ u)$.

Definition

Let $(S\ T : \mathrm{Set}\ A)$. Remember that this means that

$$(S\ T : A \to \mathrm{Prop})$$

We define $S \cup T$ as the set

$$\mathrm{fun}\ a \mapsto S\ a \vee T\ a$$

We can now define the usual operations on sets. We fix $(A : \mathrm{Type}\ u)$.

Definition

Let $(S\ T : \mathrm{Set}\ A)$. Remember that this means that

$$(S\ T : A \to \mathrm{Prop})$$

We define $S \cup T$ as the set

$$\mathrm{fun}\ a \mapsto S\ a \vee T\ a$$

Similarly, we define $S \cap T$ as the set

$$\mathrm{fun}\ a \mapsto S\ a \wedge T\ a$$

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Remark

- *There is not only one "empty set". Indeed there is the "empty set of type* $\mathrm{Set}\ A$*" for all* A*.*

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Remark

- *There is not only one "empty set". Indeed there is the "empty set of type* $\mathrm{Set}\ A$*" for all* A*. Note that* $(\emptyset : \mathrm{Set}\ A)$ *and* $(\emptyset : \mathrm{Set}\ B)$ *don't have the same type, so they cannot be equal (they cannot be compared).*

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Remark

- *There is not only one "empty set". Indeed there is the "empty set of type* $\mathrm{Set}\ A$*" for all A. Note that* $(\emptyset : \mathrm{Set}\ A)$ *and* $(\emptyset : \mathrm{Set}\ B)$ *don't have the same type, so they cannot be equal (they cannot be compared).*
- *A is not a term of type* $\mathrm{Set}\ A$*.*

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Remark

- *There is not only one "empty set". Indeed there is the "empty set of type* $\mathrm{Set}\ A$*" for all* A*. Note that* $(\emptyset : \mathrm{Set}\ A)$ *and* $(\emptyset : \mathrm{Set}\ B)$ *don't have the same type, so they cannot be equal (they cannot be compared).*
- A *is not a term of type* $\mathrm{Set}\ A$*.*

Definition

We define $(\mathrm{univ} : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{True}$$

Definition

We define $(\emptyset : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{False}$$

Remark

- *There is not only one "empty set". Indeed there is the "empty set of type* $\mathrm{Set}\ A$*" for all* A*. Note that* $(\emptyset : \mathrm{Set}\ A)$ *and* $(\emptyset : \mathrm{Set}\ B)$ *don't have the same type, so they cannot be equal (they cannot be compared).*
- A *is not a term of type* $\mathrm{Set}\ A$*.*

Definition

We define $(\mathrm{univ} : \mathrm{Set}\ A)$ as

$$\mathrm{fun}\ a \mapsto \mathrm{True}$$

It is the "biggest" set of elements of type A.

Definition

If $(S : \mathrm{Set}\ A)$ and $(a : A)$, we define $a \in S$ as

$$S\ a$$

Definition

If $(S : \mathrm{Set}\ A)$ and $(a : A)$, we define $a \in S$ as

$$S\ a$$

In particular $a \in S$ is the same as the fact that $S\ a$ holds.

Definition

If $(S : \mathrm{Set}\ A)$ and $(a : A)$, we define $a \in S$ as

$$S\ a$$

In particular $a \in S$ is the same as the fact that $S\ a$ holds.

Definition

If $(S\ T : \mathrm{Set}\ A)$, we define $S \subseteq T$ as

$$\forall (a : A),\ S\ a \to T\ a$$

Definition

If $(S : \mathrm{Set}\ A)$ and $(a : A)$, we define $a \in S$ as

$$S\ a$$

In particular $a \in S$ is the same as the fact that $S\ a$ holds.

Definition

If $(S\ T : \mathrm{Set}\ A)$, we define $S \subseteq T$ as

$$\forall (a : A),\ S\ a \to T\ a$$

This implies immediately that

$$S \subseteq T \iff \forall (a : A),\ a \in S \to a \in T$$

Definition

If $(S : \mathrm{Set}\ A)$, we define S^c, the complement of S as

$$\mathrm{fun}\ a \mapsto \neg(S\ a)$$

Definition

If $(S : \mathrm{Set}\ A)$, we define S^c, the complement of S as

$$\mathrm{fun}\ a \mapsto \neg(S\ a)$$

Definition

If $(S\ T : \mathrm{Set}\ A)$ are sets, we define $S \setminus T$, the set theoretic difference of S and T by

$$\mathrm{fun}\ a \mapsto S\ a \wedge \neg(T\ a)$$

One can easily prove basic facts about these operations, for example

- for all $(S : \mathrm{Set}\ A)$, we have $S \subseteq S$.

One can easily prove basic facts about these operations, for example

- for all $(S : \mathrm{Set}\ A)$, we have $S \subseteq S$.
- $\subseteq$ is a transitive relation.
- ...

Remark

If $(S : \mathrm{Set}\ A)$, then S is of course a term, but it is not a type. In particular, we cannot have terms $(t : S)$ of type S (in Lean $(t : S)$ is accepted, but the elaborator does some work, transforming S into a type).

Extensionality

Using the properties of equality, it is immediate to show that if $a \in S$ and $S = T$, then $a \in T$.

Extensionality

Using the properties of equality, it is immediate to show that if $a \in S$ and $S = T$, then $a \in T$. In particular

$$S = T \Rightarrow (\forall \, (a : A), a \in S \iff a \in T)$$

Extensionality

Using the properties of equality, it is immediate to show that if $a \in S$ and $S = T$, then $a \in T$. In particular

$$S = T \Rightarrow (\forall\ (a : A),\ a \in S \iff a \in T)$$

Does the converse hold?

Extensionality

Using the properties of equality, it is immediate to show that if $a \in S$ and $S = T$, then $a \in T$. In particular

$$S = T \Rightarrow (\forall\ (a : A), a \in S \iff a \in T)$$

Does the converse hold? Unraveling the definitions, this is the same as asking if S and T (that are functions) are equal supposing they are pointwise equal.

Extensionality

Using the properties of equality, it is immediate to show that if $a \in S$ and $S = T$, then $a \in T$. In particular

$$S = T \Rightarrow (\forall\ (a : A),\ a \in S \iff a \in T)$$

Does the converse hold? Unraveling the definitions, this is the same as asking if S and T (that are functions) are equal supposing they are pointwise equal.

This question (and its generalization to arbitrary functions) is called *extensionality*, and it will be a consequence of the additional axioms used in mathlib.

One can now prove all the basic results in set theory

One can now prove all the basic results in set theory, often without using extensionality.

One can now prove all the basic results in set theory, often without using extensionality.

We prove in the examples Cantor's theorem (without extensionality!).

Theorem (Cantor)

Let $(A : \mathrm{Type}\ u)$ *and let* $(f : A \to \mathrm{Set}\ A)$*. Then*

$$\neg(\mathrm{Surjective}\ f)$$

One can now prove all the basic results in set theory, often without using extensionality.

We prove in the examples Cantor's theorem (without extensionality!).

Theorem (Cantor)

Let $(A : \mathrm{Type}\ u)$ *and let* $(f : A \to \mathrm{Set}\ A)$*. Then*

$$\neg(\mathrm{Surjective}\ f)$$

Here, if $(f : A \to B)$, then $\mathrm{Surjective}\ f$ is *defined* by

$$\forall (b : B),\ \exists (a : A),\ f\ a = b$$