Programming with Higher Inductive Types

Henning Basold, Herman Geuvers, Niels van der Weide

December 20, 2016

How to define Integers

```
data Pos =
    One : Pos
  | S : Pos → Pos
data ℤ =
    Minus : Pos → ℤ
  | Zero : ℤ
  | Plus : Pos → ℤ
```

How to define Integers

A more logical definition of $\mathbb{Z}$ would be

```
data ℤ =
    Z : ℤ
  | S : ℤ → ℤ
  | P : ℤ → ℤ
```

and we require that S and P are inverses.

How to define Integers

A more logical definition of $\mathbb{Z}$ would be

```
data ℤ =
    Z : ℤ
  | S : ℤ → ℤ
  | P : ℤ → ℤ
```

and we require that S and P are inverses.
However, inductive types should be 'freely generated'. We can't allow extra equations.

Our Goal: Higher Inductive Types

Higher inductive types allow the programmer to define data types with extra equations.

Topics

- What's ‘equality’?
- What are Higher Inductive Types (HITs), and what can we do with them?
- In the end: how can we implement this in Coq? (Coq doesn't have HITs)

Equality by rewriting (**Definitional Equality**)

Functional languages rewrite terms.

```
data nat =
    Z : nat
  | S : nat → nat

plus : nat → nat → nat
plus Z m = m
plus (S n) m = S (plus n m)
```

We rewrite ‘plus (S Z) (S Z)’ to ‘S (S Z)’.

Equality as a proposition (**Propositional Equality**)

Using Curry-Howard we can define equality as a type.

```
data Eq (A : Type) : A → A → Type =
    refl : (a : A) → Eq A a a
```

We denote the type ‘Eq A a b’ by ‘a = b’.
Note: we can also talk about *equalities between equalities* via the type ‘Eq (Eq A a b) p q’. These are called **higher equalities**.

Comparison

Definitional equality is stronger, but propositional is more flexible.
We will *mostly* use **propositional** equality.

Examples of Higher Inductive Types

```
data ℕ/2ℕ =
    Z : ℕ/2ℕ
  | S : ℕ/2ℕ → ℕ/2ℕ
  | mod : Z = S(S Z)
```

Examples of Higher Inductive Types

```
data ℕ/2ℕ =
    Z : ℕ/2ℕ
  | S : ℕ/2ℕ → ℕ/2ℕ
  | mod : Z = S(S Z)
```

Note: if we have $f : A \to B$ and $p : x = y$ (with $x, y : A$), then we have $\mathrm{ap}(f, p) : f\,x = f\,y$.

Examples of Higher Inductive Types

```
data ℕ/2ℕ =
    Z : ℕ/2ℕ
  | S : ℕ/2ℕ → ℕ/2ℕ
  | mod : Z = S(S Z)
```

Note: if we have $f : A \to B$ and $p : x = y$ (with $x, y : A$), then we have $\mathrm{ap}(f, p) : f\,x = f\,y$. This gives

$$\mathrm{ap}(\mathrm{S}, \mathrm{mod}) : S\,Z = S(S(S\,Z)),$$

$$\mathrm{ap}(\mathrm{S}, \mathrm{ap}(\mathrm{S}, \mathrm{mod}) : S(S\,Z) = S(S(S(S\,Z)))),$$

and so on.

Examples of Higher Inductive Types

```
data ℤ =
    Z : ℤ
  | S : ℤ → ℤ
  | P : ℤ → ℤ
  | inv1 : (x : ℤ) → P(S x) = x
  | inv2 : (x : ℤ) → S(P x) = x
```

What about higher equalities?

We have $P : \mathbb{Z} \to \mathbb{Z}$ and

$$\mathsf{inv}\,2(S(PZ)) : S(P\,Z) = S(P(S(P\,Z))),$$

What about higher equalities?

We have $P : \mathbb{Z} \to \mathbb{Z}$ and

$$\mathsf{inv}\,2(S(PZ)) : S(P\,Z) = S(P(S(P\,Z))),$$

so we get

$$\mathsf{ap}(P, \mathsf{inv}\,2(S(PZ))) : P(S(P\,Z)) = P(S(P(S(P\,Z))))$$

What about higher equalities?

We have $P : \mathbb{Z} \to \mathbb{Z}$ and

$$\mathrm{inv}\,2(S(PZ)) : S(P\,Z) = S(P(S(P\,Z))),$$

so we get

$$\mathrm{ap}(P, \mathrm{inv}\,2(S(PZ))) : P(S(P\,Z)) = P(S(P(S(P\,Z))))$$

We also have

$$\mathrm{inv}\,1(P(S(P\,Z))) : P(S(P\,Z)) = P(S(P(S(P\,Z)))).$$

Are these equal?

Short Intermezzo: Hedberg's Theorem

Theorem
If we give an inhabitant of $A + (A \to \bot)$ for a type A, then all inhabitants of $x = y$ for $x, y : A$ are equal.

Short Intermezzo: Hedberg's Theorem

Theorem

If we give an inhabitant of $A + (A \to \bot)$ for a type A, then all inhabitants of $x = y$ for $x, y : A$ are equal.
Briefly, if A has decidable equality, then all proofs of equality in A are equal.

Short Intermezzo: Hedberg's Theorem

Theorem

If we give an inhabitant of $A + (A \to \bot)$ for a type A, then all inhabitants of $x = y$ for $x, y : A$ are equal.
Briefly, if A has decidable equality, then all proofs of equality in A are equal.
Equivalently, if two equalities in a type are unequal, then that type does not have decidable equality.

How to Program with HITs?

How to map $\mathbb{N}/2\mathbb{N}$ to some type A? What about $\mathbb{Z}$?

Programming with $\mathbb{N}/2\mathbb{N}$

To make $\mathbb{N}/2\mathbb{N} \to A$, we need to give

- $z : A$ which is the image of Z;
- $s : A \to A$ which is the image of S;

Programming with $\mathbb{N}/2\mathbb{N}$

To make $\mathbb{N}/2\mathbb{N} \to A$, we need to give

- $z : A$ which is the image of Z;
- $s : A \to A$ which is the image of S;
- an equality (a proof obligation)

$$m : z = s(s\,z).$$

Programming with $\mathbb{N}/2\mathbb{N}$

Seeing $\mathbb{N}/2\mathbb{N}$ as booleans, we can negate it.

- Choose $A = \mathbb{N}/2\mathbb{N}$;
- For z we pick $S\,Z$;
- For s we pick S;
- The proof obligation is: $S\,Z = S(S(S\,Z))$. We give

$$\mathsf{ap}(S, \mathsf{mod})$$

Programming with $\mathbb{Z}$

To make $\mathbb{Z} \to A$, we need to give

- $z : A$ which is the image of Z;
- $s : A \to A$ and $p : A \to A$ for S and P respectively;

Programming with $\mathbb{Z}$

To make $\mathbb{Z} \to A$, we need to give

- $z : A$ which is the image of Z;
- $s : A \to A$ and $p : A \to A$ for S and P respectively;
- equalities (proof obligations)

$$i_1 : (a : A) \to a = p(s\,a),$$

$$i_2 : (a : A) \to a = s(p\,a).$$

$\mathbb{Z}$ does not have decidable equality!

In Homotopy Type Theory $\mathbb{Z}$ does not have decidable equality.
For the proof we assume we have

- A type C;
- A point $b : C$;
- An equality $l : b = b$ such that there is no equality between l and $\mathrm{refl}\, b$.

(We can prove that there is such a type assuming Voevodsky's Univalence Axiom)

The proof

We make $f : \mathbb{Z} \to C$.

- We send Z to b.
- We send S and P to the identity map;
- For inv1 we need to prove $b = b$ for which we take l;
- For inv2 we also need to prove $b = b$ which we prove by refl b.

The proof

We make $f : \mathbb{Z} \to C$.

- We send Z to b.
- We send S and P to the identity map;
- For inv1 we need to prove $b = b$ for which we take l;
- For inv2 we also need to prove $b = b$ which we prove by refl b.

Now we have

$$\mathrm{ap}(f, \mathrm{ap}(P, \mathrm{inv}\,2(S(PZ)))) = \mathrm{refl}\, b,$$

$$\mathrm{ap}(f, \mathrm{inv}\,1(P(S(P\,Z)))) = l.$$

So, these paths are unequal, and thus $\mathbb{Z}$ does not have decidable equality.

How to do this in Coq?

Coq does not have HITs, but you can add axioms.

```
Module Export Ints.

Private Inductive Z : Type :=
| nul : Z
| succ : Z → Z
| pred : Z → Z.

Axiom inv1 : forall n : Z, n = pred(succ n).
Axiom inv2 : forall n : Z, n = succ(pred n).
```

How to do this in Coq?

The recursion principle is more complicated.

```
Fixpoint Z_rec
  (P : Type)
  (a : P)
  (s : P → P)
  (p : P → P)
  (i1 : forall (m : P), m = p(s m))
  (i2 : forall (m : P), m = s(p m))
  (x : Z)
  {struct x}
: P
:=
(match x return _ → _ → P with
| nul ⇒ fun _ ⇒ fun _ ⇒ a
| succ n ⇒ fun _ ⇒ fun _ ⇒ s ((Z_rec P a s p i1 i2) n)
| pred n ⇒ fun _ ⇒ fun _ ⇒ p ((Z_rec P a s p i1 i2) n)
end) i1 i2.
```

How to do this in Coq?

Computation rules for the equalities go as expected.

```
Axiom Z_rec_beta_inv1 :
forall
  (P : Type)
  (a : P)
  (s : P → P)
  (p : P → P)
  (i1 : forall (m : P), m = p(s m))
  (i2 : forall (m : P), m = s(p m))
  (n : Z)
, ap (Z_rec P a s p i1 i2) (inv1 n) = i1 (Z_rec P a s p i1 i2 n).

end Ints.
```