

Overview of the Remainder of the Reading Group

Mutual Inductive Types

Inductive-Recursive Types

Inductive-Inductive Types

Higher Inductive Types

Initial Algebra Semantics

Mutual Inductive Types: Main Idea

- ▶ Define two inductive types A and B simultaneously
- ▶ The constructors of A may refer to the type B , and the constructors of B may refer to the type A

Example: even and odd numbers

Mutual Inductive Types: Example

Inductive $\text{isEven} : \text{nat} \rightarrow \text{Type} :=$
| Zeven : $\text{isEven } 0$
| Sodd : forall (n: nat)
 (p: $\text{isOdd } n$),
 $\text{isEven } (S n)$

with

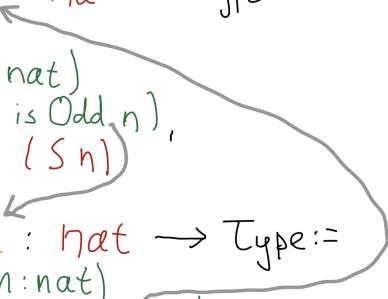
$\text{isOdd} : \text{nat} \rightarrow \text{Type} :=$
| Seven : forall (n: nat)
 (p: $\text{isEven } n$),
 $\text{isOdd } (S n)$.

Mutual Inductive Types: Example

Inductive $\text{isEven} : \text{nat} \rightarrow \text{Type} :=$
| Zeven : $\text{isEven } 0$
| Sodd : forall (n:nat)
 (p: isOdd n),
 $\text{isEven } (S n)$

with

$\text{isOdd} : \text{nat} \rightarrow \text{Type} :=$
| Seven : forall (n:nat)
 (p: isEven n),
 $\text{isOdd } (S n)$.



Inductive-Recursive Types: Main Idea

- ▶ Define an inductive type A together with a recursive function $f : A \rightarrow B$ for some type B
- ▶ The constructors of A may refer to the function f

Example: list of which each element is different

Inductive-Recursive Types: Example

Inductive $Dlist$ ($A: Type$): $Type :=$
| nil : $Dlist A$
| $cons$: forall ($a:A$)
 ($as: Dlist A$),
 $Dlist A$

Inductive-Recursive Types: Example

Inductive $Dlist$ ($A: Type$): $Type :=$

| nil : $Dlist A$

| cons : forall ($a:A$)
 ($as: Dlist A$),

fresh as a
→ $Dlist A$

with

fresh : $Dlist A \rightarrow A \rightarrow Type$

fresh nil a = true

fresh (cons x xs p) a = $x \neq a \wedge$ fresh xs a

Inductive-Recursive Types: Example

Inductive $Dlist$ ($A: Type$): $Type :=$

| nil : $Dlist A$

| cons : forall ($a:A$)
($as: Dlist A$),

fresh as a
 $\rightarrow Dlist A$

with

fresh : $Dlist A \rightarrow A \rightarrow Type$

fresh nil a = true

fresh (cons x xs p) a = $x \neq a \wedge$ fresh xs a

Inductive-Inductive Types: Main Idea

- ▶ Define an inductive type A together with an inductive family B on A
- ▶ The constructors of A may refer to B , and the constructors of B may refer to A

Example: sorted lists

Inductive-Inductive Types: Example

Inductive $Slist$: Type :=
| nil : $Slist$
| cons : forall (n : nat)
 (ns : $Slist$),

$Slist$

Inductive-Inductive Types: Example

Inductive $Slist$: Type :=

| nil : $Slist$

| cons : forall (n : nat)
 (ns : $Slist$),

leAll n ns

→ $Slist$

with

leAll : nat → $Slist$ → Type :=

| lnil : forall (n : nat),

leAll n nil

| lcons : forall (n : nat)

(m : nat)

(ms : $Slist$)

(p : $n \leq m$)

(q : leAll m ms),

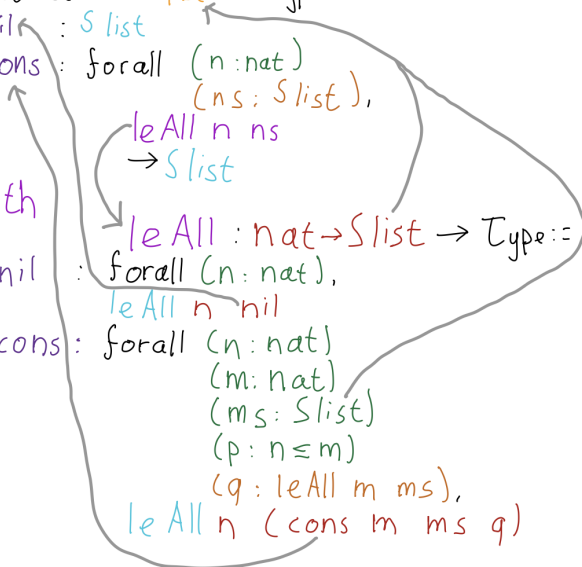
leAll n (cons m ms q)

Inductive-Inductive Types: Example

Inductive $Slist$: Type :=
| nil : $Slist$
| cons : forall (n : nat)
 (ns : $Slist$),
 leAll n ns
 → $Slist$

with

| leAll : nat → $Slist$ → Type :=
| lnil : forall (n : nat),
 leAll n nil
| lcons : forall (n : nat)
 (m : nat)
 (ms : $Slist$)
 (p : n ≤ m)
 (q : leAll m ms),
 leAll n (cons m ms q)



Higher Inductive Types: Main Idea

- ▶ Define an inductive type A by specifying constructors for its inhabitants and for equalities of this type
- ▶ The constructors of the equalities may refer to the constructors for the inhabitants

Example: finite sets

Higher Inductive Types: Example

Inductive FinSet ($A: \text{Type}$) : $\text{Type} :=$
| empty : $\text{FinSet } A$
| add : forall ($a:A$)
 ($as: \text{FinSet } A$),
 $\text{FinSet } A$

However,

add 1 (add 2 nil)

add 2 (add 1 nil)

Higher Inductive Types: Example

Inductive FinSet ($A: \text{Type}$): $\text{Type} :=$

| empty : $\text{FinSet } A$

| add : forall ($a:A$)
 ($as: \text{FinSet } A$),

$\text{FinSet } A$

| add-add : forall ($a:A$)
 ($as: \text{FinSet } A$),
 add a (add a as)

| swap : forall ($a:A$)
 ($b:A$)
 ($as: \text{FinSet } A$),
 add a (add b as)
 ||
 add b (add a as)

Higher Inductive Types: Example

Inductive FinSet (A: Type) : Type :=

| empty : FinSet A

| add : forall (a:A)
 (as: FinSet A),

 FinSet A

| add-add : forall (a:A)
 (as: FinSet A),

 add a (add a as)

 ||

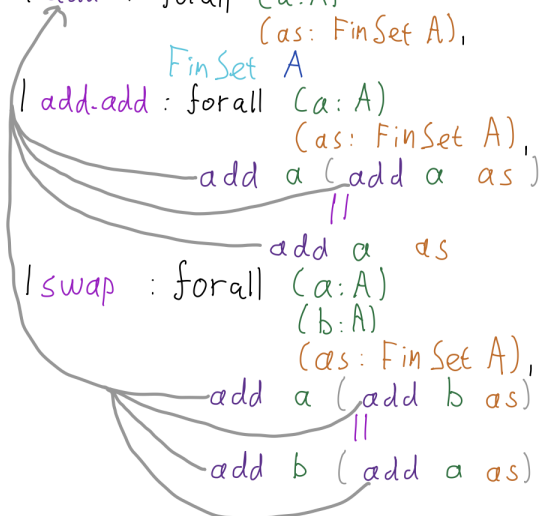
 add a as

| swap : forall (a:A)
 (b:A)
 (as: FinSet A),

 add a (add b as)

 ||

 add b (add a as)



Overview

Induction-Recursion

Inductive type A
together with
recursive function $f: A \rightarrow B$
for some type B

Induction-Induction

Inductive type A
together with
inductive family
 $B: A \rightarrow \text{Type}$

Higher Inductive Type

Specify a type by
its constructors and
equality constructors

Initial Algebra Semantics: Main Idea

- ▶ Two ways to view types: **type theoretic** and **category theoretic**
- ▶ **Type theoretic**: based on induction
- ▶ **Category theoretic**: recursion and uniqueness

How to specify a type?

Type theorists

1. **Formation rule**
→ how to construct the type
2. **Introduction rules**
→ how to construct inhabitants?
3. **Elimination rules**
→ how to construct **dependent** functions out of that type?
4. **Computation rules**
→ how to calculate using the elimination rule?

Category theorists

1. **Formation rule**
→ how to construct the type
2. **Introduction rules**
→ how to construct inhabitants?
3. **Recursion rules**
→ how to construct **nondependent** functions out of that type?
4. **Computation rules**
→ how to calculate using the recursion rule?
5. **Uniqueness rules**
→ how to prove that functions out of that type are equal?

Initial Algebra Semantics: Main Idea

- ▶ Advantage of the type theoretic view: it directly gives us **proof principles**
- ▶ Advantage of the category theoretic view: it is simpler to give semantics for this presentation
- ▶ **Initial Algebra Semantics**: these two presentations of inductive types are equivalent