Deep Induction Rules for Inductive-Inductive Types

Patricia Johann and Ben Lenox

Appalachian State University

TFP'26, 28 January

Structural induction for ADTs

```
data List (a : Set) : Set where
  [] : List a
  _::_ : a → List a → List a
```

Structural induction for ADTs

```
data List (a : Set) : Set where
  [] :  List a
  _::_ :  a → List a → List a

{a : Set} → (P : List a → Set) →
P [] →
((x : a) → (xs : List a) → P xs → P (x :: xs)) →
(xs : List a) → P xs
```

Deep induction for ADTs

```
data List (a : Set) : Set where
  [] : List a
  _::_ : a → List a → List a

List^ : {a : Set} → (a → Set) → List a → Set
List^ Q [] = ⊤
List^ Q (x :: xs) = Q x × List^ Q xs
```

Deep induction for ADTs

```
data List (a : Set) : Set where
  [] : List a
  _::_ : a → List a → List a

List^ : {a : Set} → (a → Set) → List a → Set
List^ Q [] = ⊤
List^ Q (x :: xs) = Q x × List^ Q xs

{a : Set} → (Q : a → Set) → (P : List a → Set) →
  P [] →
  ((x : a) → (xs : List a) → Q x → P xs → P (x :: xs)) →
  (xs : List a) → List^ Q xs → P xs
```

Structural induction for deep ADTs

```
data Forest (a : Set) : Set where
  fempty :  Forest a
  fnode :   a → List (Forest a) → Forest a
```

Structural induction for deep ADTs

```
data Forest (a : Set) : Set where
  fempty :  Forest a
  fnode :   a → List (Forest a) → Forest a

{a : Set} → (P : Forest a → Set) →
P fempty →
((x : a) → (xss : List (Forest a)) → P (fnode x xss)) →
(xs : Forest a) → P xs
```

Structural induction for deep ADTs

```
data Forest (a : Set) : Set where
  fempty :  Forest a
  fnode :   a → List (Forest a) → Forest a

{a : Set} → (P : Forest a → Set) →
            P fempty →
            ((x : a) → (xss : List (Forest a)) → P (fnode x xss)) →
            (xs : Forest a) → P xs
```

Unfortunately, this rule is neither what we expect nor is it terribly useful...

Deep induction for deep ADTs

```
data Forest (a : Set) : Set where
  fempty :  Forest a
  fnode  :  a → List (Forest a) → Forest a

Forest^ : {a : Set} → (a → Set) → Forest a → Set
Forest^ Q fempty  =  ⊤
Forest^ Q (fnode x xss)  =  Q x × List^ (Forest^ Q) xss
```

Deep induction for deep ADTs

```
data Forest (a : Set) : Set where
  fempty :  Forest a
  fnode  :  a → List (Forest a) → Forest a

Forest^ : {a : Set} → (a → Set) → Forest a → Set
Forest^ Q fempty  =  ⊤
Forest^ Q (fnode x xss)  =  Q x × List^ (Forest^ Q) xss

{a : Set} → (Q : a → Set) → (P : Forest a → Set) →
  P fempty →
  ((x : a) → (xss : List (Forest a)) → Q x → List^ P xss → P (fnode x xss)) →
  (xs : Forest a) → Forest^ Q xs → P xs
```

The state of the art in deep induction

- **Type-indexed types**
  - **ADTs — e.g., lists, trees, forests**

[Johann and Polonsky 2020]

The state of the art in deep induction

- **Type-indexed types**
  - **ADTs — e.g., lists, trees, forests**
    [Johann and Polonsky 2020]
  - **nested types — e.g., perfect trees, bushes**
    [Johann and Polonsky 2020]

The state of the art in deep induction

- **Type-indexed types**
  - **ADTs — e.g., lists, trees, forests**
    [Johann and Polonsky 2020]
  - **nested types — e.g., perfect trees, bushes**
    [Johann and Polonsky 2020]
  - **GADTs — e.g., sequences, typed lambda terms**
    [Johann and Ghiorzi 2022]

The state of the art in deep induction

- **Type-indexed types**
  - **ADTs — e.g., lists, trees, forests**
    [Johann and Polonsky 2020]
  - **nested types — e.g., perfect trees, bushes**
    [Johann and Polonsky 2020]
  - **GADTs — e.g., sequences, typed lambda terms**
    [Johann and Ghiorzi 2022]
- **Term-indexed types**
  - **inductive families — e.g., vectors, Fin- and vector-indexed sequences**
    [Johann and Morehouse 2025]

The state of the art in deep induction

- **Type-indexed types**
  - **ADTs — e.g., lists, trees, forests**
    [Johann and Polonsky 2020]
  - **nested types — e.g., perfect trees, bushes**
    [Johann and Polonsky 2020]
  - **GADTs — e.g., sequences, typed lambda terms**
    [Johann and Ghiorzi 2022]
- **Term-indexed types**
  - **inductive families — e.g., vectors, Fin- and vector-indexed sequences**
    [Johann and Morehouse 2025]
  - **inductive-inductive types — e.g., contexts and types, sorted lists**
    [Johann and Lenox 2026]

Structural induction for IFs

```
data Nat : Set where
  zero : Nat
  suc :  Nat → Nat

data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_ :  a → Vec a m → Vec a (suc m)
```

Structural induction for IFs

```
data Nat : Set where
  zero : Nat
  suc :  Nat → Nat

data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_ :  a → Vec a m → Vec a (suc m)

{a : Set} → {n : Nat} → (P : {n : Nat} → Vec a n → Set) →
  P [ ] →
  ({m : Nat} → (x : a) → (xs : Vec a m) → P xs → P (x :: xs)) →
  (xs : Vec a n) → P xs
```

Deep induction for IFs

```
data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_  :  a → Vec a m → Vec a (suc m)
```

```
Vec^ : {a : Set} → {n : Nat} → (a → Set) → (Nat → Set) → Vec a n → Set
Vec^ {n = zero} Q_a Q_n [ ]  =  Q_n zero
Vec^ {n = suc m} Q_a Q_n (x :: xs)  =  Q_a x × Vec^ Q_a Q_n xs × (Q_n m → Q_n (suc m))
```

Deep induction for IFs

```
data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_  :  a → Vec a n → Vec a (suc n)
```

```
Vec^ : {a : Set} → {n : Nat} → (a → Set) → (Nat → Set) → Vec a n → Set
Vec^ {n = zero} Qa Qn [ ]  =  Qn zero
Vec^ {n = suc m} Qa Qn (x :: xs)  =  Qa x × Vec^ Qa Qn xs × (Qn m → Qn (suc m))

{a : Set} → {n : Nat} → (Qa : a → Set) → (Qn : Nat → Set) →
                                  (P : {n : Nat} → Vec a n → Set) →
```

Deep induction for IFs

```
data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_  :  a → Vec a n → Vec a (suc n)
```

```
Vec^ : {a : Set} → {n : Nat} → (a → Set) → (Nat → Set) → Vec a n → Set
Vec^ {n = zero} Q_a Q_n [ ]  =  Q_n zero
Vec^ {n = suc m} Q_a Q_n (x :: xs)  =  Q_a x × Vec^ Q_a Q_n xs × (Q_n m → Q_n (suc m))
```

```
{a : Set} → {n : Nat} → (Q_a : a → Set) → (Q_n : Nat → Set) →
                                  (P : {n : Nat} → Vec a n → Set) →
  (Q_n zero → P [ ]) →
```

Deep induction for IFs

```
data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_  :  a → Vec a n → Vec a (suc n)
```

```
Vec^ : {a : Set} → {n : Nat} → (a → Set) → (Nat → Set) → Vec a n → Set
Vec^ {n = zero} Q_a Q_n [ ]  =  Q_n zero
Vec^ {n = suc m} Q_a Q_n (x :: xs)  =  Q_a x × Vec^ Q_a Q_n xs × (Q_n m → Q_n (suc m))
```

```
{a : Set} → {n : Nat} → (Q_a : a → Set) → (Q_n : Nat → Set) →
                                  (P : {n : Nat} → Vec a n → Set) →
  (Q_n zero → P [ ]) →
  ({m : Nat} → (x : a) → (xs : Vec a m) → Q_a x → P xs → Q_n (suc m) → P (x :: xs)) →
```

Deep induction for IFs

```
data Vec (a : Set) : Nat → Set where
  [ ] :  Vec a zero
  _::_  :  a → Vec a n → Vec a (suc n)
```

```
Vec^ : {a : Set} → {n : Nat} → (a → Set) → (Nat → Set) → Vec a n → Set
Vec^ {n = zero} Q_a Q_n [ ]  =  Q_n zero
Vec^ {n = suc m} Q_a Q_n (x :: xs)  =  Q_a x × Vec^ Q_a Q_n xs × (Q_n m → Q_n (suc m))
```

```
{a : Set} → {n : Nat} → (Q_a : a → Set) → (Q_n : Nat → Set) →
                              (P : {n : Nat} → Vec a n → Set) →
  (Q_n zero → P [ ]) →
  ({m : Nat} → (x : a) → (xs : Vec a m) → Q_a x → P xs → Q_n (suc m) → P (x :: xs)) →
  (xs : Vec a n) → Vec^ Q_a Q_n xs → P xs
```

Inductive-inductive types (IITs)

```
record Ordered (a : Set) : Set where
  field
    _≥_ : a → a → Set
```

Inductive-inductive types (IITs)

```
record Ordered (a : Set) : Set where
  field
    _≥_ : a → a → Set

mutual
  data SList (a : Set) {{orda : Ordered a}} : Set where
    snil : SList a
    scons : {x : a} → {xs : SList a} → x ≥L xs → SList a
```

Inductive-inductive types (IITs)

```
record Ordered (a : Set) : Set where
  field
    _≥_ : a → a → Set

mutual
  data SList (a : Set) {{orda : Ordered a}} : Set where
    snil : SList a
    scons : {x : a} → {xs : SList a} → x ≥L xs → SList a

  data _≥L_ {a : Set} {{orda : Ordered a}} (x : a) : SList a → Set where
    triv : x ≥L snil
    extn : {y : a} → {ys : SList a} → (y≥ys : y ≥L ys) → x ≥ y →
           x ≥L ys → x ≥L (scons y≥ys)
```

Liftings for IITs

mutual

$$\mathsf{SList}^{\wedge} : \{\mathsf{a} : \mathsf{Set}\} \to \{\{\mathsf{orda} : \mathsf{Ordered\ a}\}\} \to (\mathsf{a} \to \mathsf{Set}) \to \mathsf{SList\ a} \to \mathsf{Set}$$

$$\mathsf{SList}^{\wedge}\ \mathsf{Q}\ \mathsf{snil} = \top$$

$$\mathsf{SList}^{\wedge}\ \mathsf{Q}\ (\mathsf{scons}\ \{\mathsf{x}\}\ \{\mathsf{xs}\}\ \mathsf{x}{\geq}\mathsf{xs}) = \mathsf{Q\ x}\ \times\ \mathsf{SList}^{\wedge}\ \mathsf{Q\ xs}\ \times\ {\geq_{\mathsf{L}}^{\wedge}}\ \mathsf{Q\ x}{\geq}\mathsf{xs}$$

Liftings for IITs

```
mutual

  SList^ : {a : Set} → {{orda : Ordered a}} → (a → Set) → SList a → Set
  SList^ Q snil = ⊤
  SList^ Q (scons {x} {xs} x≥xs) = Q x × SList^ Q xs × ≥L^ Q x≥xs

≥L^ : {a : Set} → {{orda : Ordered a}} →
                  (a → Set) → {x : a} → {xs : SList a} → x ≥L xs → Set
```

Liftings for IITs

```
mutual

  SList^ : {a : Set} → {{orda : Ordered a}} → (a → Set) → SList a → Set
  SList^ Q snil = ⊤
  SList^ Q (scons {x} {xs} x≥xs) = Q x × SList^ Q xs × ≥L^ Q x≥xs

  ≥L^ : {a : Set} → {{orda : Ordered a}} →
                (a → Set) → {x : a} → {xs : SList a} → x ≥L xs → Set
  ≥L^ Q {x} triv = Q x × SList^ Q snil
```

Liftings for IITs

```
mutual

  SList^ : {a : Set} → {{orda : Ordered a}} → (a → Set) → SList a → Set
  SList^ Q snil = ⊤
  SList^ Q (scons {x} {xs} x≥xs) = Q x × SList^ Q xs × ≥L^ Q x≥xs

  ≥L^ : {a : Set} → {{orda : Ordered a}} →
                (a → Set) → {x : a} → {xs : SList a} → x ≥L xs → Set
  ≥L^ Q {x} triv = Q x × SList^ Q snil
  ≥L^ Q (extn {x} {y} {ys} y≥ys x≥y x≥ys) = Q x × Q y × SList^ Q ys × ≥L^ Q y≥ys ×
                ≥L^ Q x≥ys × (SList^Q ys → SList^Q (scons y≥ys))
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥: {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥: {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
    (hnil : PI snil) →
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥ : {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
    (hnil : PI snil) →
    (hcons : {x : a} → {xs : SList a} → (x≥xs : x ≥L xs) → Q x →
             (PIxs : PI xs) → P≥ x≥xs PIxs → PI (scons x≥xs)) →
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥ : {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
    (hnil : PI snil) →
    (hcons : {x : a} → {xs : SList a} → (x≥xs : x ≥L xs) → Q x →
             (PIxs : PI xs) → P≥ x≥xs PIxs → PI (scons x≥xs)) →
    (htriv : {x : a} → Q x → P≥ {x} triv hnil) →
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥ : {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
    (hnil : PI snil) →
    (hcons : {x : a} → {xs : SList a} → (x≥xs : x ≥L xs) → Q x →
          (PIxs : PI xs) → P≥ x≥xs PIxs → PI (scons x≥xs)) →
    (htriv : {x : a} → Q x → P≥ {x} triv hnil) →
    (hextn : {x y : a} → {ys : SList a} → (y≥ys : y ≥L ys) → (x≥y : x ≥ y) → (x≥ys : x ≥L ys) →
          Q x → (Qy : Q y) → (PIys1 : PI ys) → (P≥yys : P≥ y≥ys PIys1) → (PIys2 : PI ys) →
          (P≥xys : P≥ x ≥ ys PIys2) → P≥ (extn y≥ys x≥y x≥ys) (hcons y≥ys Qy PIys1 P≥yys)) →
```

Deep induction for IITs

```
mutual
  SListDInd : {a : Set} → {{orda : Ordered a}} → (Q : a → Set) →
    (PI : SList a → Set) →
    (P≥: {x : a} → {xs : SList a} → x ≥L xs → PI xs → Set) →
    (hnil : PI snil) →
    (hcons : {x : a} → {xs : SList a} → (x≥xs : x ≥L xs) → Q x →
          (PIxs : PI xs) → P≥ x≥xs PIxs → PI (scons x≥xs)) →
    (htriv : {x : a} → Q x → P≥ {x} triv hnil) →
    (hextn : {x y : a} → {ys : SList a} → (y≥ys : y ≥L ys) → (x≥y : x ≥ y) → (x≥ys : x ≥L ys) →
          Q x → (Qy : Q y) → (PIys1 : PI ys) → (P≥yys : P≥ y≥ys PIys1) → (PIys2 : PI ys) →
          (P≥xys : P≥ x ≥ ys PIys2) → P≥ (extn y≥ys x≥y x≥ys) (hcons y≥ys Qy PIys1 P≥yys)) →
    (xs : SList a) → SList^ Q xs → PI xs
```

Deep induction for IITs (cont.)

$$
\begin{aligned}
&\geq_{\mathsf{L}}\mathsf{DInd} : \{a : \mathsf{Set}\} \to \{\{\mathsf{orda} : \mathsf{Ordered}\, a\}\} \to (Q : a \to \mathsf{Set}) \to \\
&\quad (\mathsf{PI} : \mathsf{SList}\, a \to \mathsf{Set}) \to \\
&\quad (\mathsf{P}\geq : \{x : a\} \to \{xs : \mathsf{SList}\, a\} \to x \geq_{\mathsf{L}} xs \to \mathsf{PI}\, xs \to \mathsf{Set}) \to \\
&\quad (\mathsf{hnil} : \mathsf{PI}\, \mathsf{snil}) \to \\
&\quad (\mathsf{hcons} : \{x : a\} \to \{xs : \mathsf{SList}\, a\} \to (x{\geq}xs : x \geq_{\mathsf{L}} xs) \to Q\, x \to \\
&\qquad (\mathsf{PIxs} : \mathsf{PI}\, xs) \to \mathsf{P}\geq\ x{\geq}xs\ \mathsf{PIxs} \to \mathsf{PI}\, (\mathsf{scons}\, x{\geq}xs)) \to \\
&\quad (\mathsf{htriv} : \{x : a\} \to Q\, x \to \mathsf{P}\geq\ \{x\}\, \mathsf{triv}\, \mathsf{hnil}) \to \\
&\quad (\mathsf{hextn} : \{x\, y : a\} \to \{ys : \mathsf{SList}\, a\} \to (y{\geq}ys : y \geq_{\mathsf{L}} ys) \to (x{\geq}y : x \geq y) \to (x{\geq}ys : x \geq_{\mathsf{L}} ys) \to \\
&\qquad Q\, x \to (Qy : Q\, y) \to (\mathsf{PIys1} : \mathsf{PI}\, ys) \to (\mathsf{P}{\geq}yys : \mathsf{P}\geq\ y{\geq}ys\ \mathsf{PIys1}) \to (\mathsf{PIys2} : \mathsf{PI}\, ys) \to \\
&\qquad (\mathsf{P}{\geq}xys : \mathsf{P}\geq\ x \geq ys\ \mathsf{PIys2}) \to \mathsf{P}\geq\ (\mathsf{extn}\, y{\geq}ys\, x{\geq}y\, x{\geq}ys)\, (\mathsf{hcons}\, y{\geq}ys\, Qy\, \mathsf{PIys1}\, \mathsf{P}{\geq}yys)) \to
\end{aligned}
$$

Deep induction for IITs (cont.)

$$
\begin{aligned}
&\geq_{\mathsf{L}}\mathsf{DInd} : \{a : \mathsf{Set}\} \to \{\{\mathsf{orda} : \mathsf{Ordered}\, a\}\} \to (Q : a \to \mathsf{Set}) \to \\
&\quad (\mathsf{PI} : \mathsf{SList}\, a \to \mathsf{Set}) \to \\
&\quad (P{\geq} : \{x : a\} \to \{xs : \mathsf{SList}\, a\} \to x \geq_{\mathsf{L}} xs \to \mathsf{PI}\, xs \to \mathsf{Set}) \to \\
&\quad (\mathsf{hnil} : \mathsf{PI}\, \mathsf{snil}) \to \\
&\quad (\mathsf{hcons} : \{x : a\} \to \{xs : \mathsf{SList}\, a\} \to (x{\geq}xs : x \geq_{\mathsf{L}} xs) \to Q\, x \to \\
&\qquad\quad (\mathsf{PIxs} : \mathsf{PI}\, xs) \to P{\geq}\ x{\geq}xs\ \mathsf{PIxs} \to \mathsf{PI}\, (\mathsf{scons}\, x{\geq}xs)) \to \\
&\quad (\mathsf{htriv} : \{x : a\} \to Q\, x \to P{\geq}\ \{x\}\ \mathsf{triv}\ \mathsf{hnil}) \to \\
&\quad (\mathsf{hextn} : \{x\, y : a\} \to \{ys : \mathsf{SList}\, a\} \to (y{\geq}ys : y \geq_{\mathsf{L}} ys) \to (x{\geq}y : x \geq y) \to (x{\geq}ys : x \geq_{\mathsf{L}} ys) \to \\
&\qquad\quad Q\, x \to (Qy : Q\, y) \to (\mathsf{PIys1} : \mathsf{PI}\, ys) \to (P{\geq}yys : P{\geq}\ y{\geq}ys\ \mathsf{PIys1}) \to (\mathsf{PIys2} : \mathsf{PI}\, ys) \to \\
&\qquad\quad (P{\geq}xys : P{\geq}\ x \geq ys\ \mathsf{PIys2}) \to P{\geq}\ (\mathsf{extn}\ y{\geq}ys\ x{\geq}y\ x{\geq}ys)\ (\mathsf{hcons}\ y{\geq}ys\ Qy\ \mathsf{PIys1}\ P{\geq}yys)) \to \\
&\quad \{x : a\} \to \{xs : \mathsf{SList}\, a\} \to (x{\geq}xs : x \geq_{\mathsf{L}} xs) \to (Q x{\geq}xs : \geq_{\mathsf{L}}^{\wedge} Q\, x\, xs\, x{\geq}xs) \to \\
&\quad P{\geq}\ x{\geq}xs\ (\mathsf{SListDInd}\ Q\ \mathsf{PI}\ P{\geq}\ \mathsf{hnil}\ \mathsf{hcons}\ \mathsf{htriv}\ \mathsf{hextn}\ xs\ (\mathsf{SprojIndex}\ x{\geq}xs\ Qx{\geq}xs))
\end{aligned}
$$

Method Modifications for IITs

- **We do not include a standalone check that the term index of the element constructed using `c` satisfies its predicate.**

Method Modifications for IITs

- **We do not include a standalone check that the term index of the element constructed using `c` satisfies its predicate.**
- **We do not check that recursive arguments to `c` satisfy their predicates if they are term indices of recursive arguments to `c`.**

Method Modifications for IITs

- **We do not include a standalone check that the term index of the element constructed using `c` satisfies its predicate.**
- **We do not check that recursive arguments to `c` satisfy their predicates if they are term indices of recursive arguments to `c`.**
- **We *do* check that each recursive argument to `c` *and its term index* satisfies its (possibly lifted) predicate, and the proof for the recursive argument uses the proof for its term index.**

Method Modifications for IITs

- **We do not include a standalone check that the term index of the element constructed using `c` satisfies its predicate.**
- **We do not check that recursive arguments to `c` satisfy their predicates if they are term indices of recursive arguments to `c`.**
- **We *do* check that each recursive argument to `c` *and its term index* satisfies its (possibly lifted) predicate, and the proof for the recursive argument uses the proof for its term index.**
- **We use the induction hypothesis for the outermost constructor of the term index of the term `c` constructs, together with the other arguments to `c`'s induction hypothesis, to construct the conclusion of the induction hypothesis for `c`.**

Method Modifications for IITs

- **We do not include a standalone check that the term index of the element constructed using `c` satisfies its predicate.**
- **We do not check that recursive arguments to `c` satisfy their predicates if they are term indices of recursive arguments to `c`.**
- **We *do* check that each recursive argument to `c` *and its term index* satisfies its (possibly lifted) predicate, and the proof for the recursive argument uses the proof for its term index.**
- **We use the induction hypothesis for the outermost constructor of the term index of the term `c` constructs, together with the other arguments to `c`'s induction hypothesis, to construct the conclusion of the induction hypothesis for `c`.**
- **We use the deep induction rule for the indexing type, together with the proof projected out from the given element's lifting, to construct the proof for the given element's term index in the final result of the indexed type's deep induction rule.**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**
- **Our methodology for IITs specializes to those given earlier for IFs, GADTs, nested types, and ADTs — each of which subsumes those that came before it.**

ADTs $\hookrightarrow$ nested types $\hookrightarrow$ GADTs $\hookrightarrow$ IFs $\hookrightarrow$ IITs

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**
- **Our methodology for IITs specializes to those given earlier for IFs, GADTs, nested types, and ADTs — each of which subsumes those that came before it.**

ADTs $\hookrightarrow$ nested types $\hookrightarrow$ GADTs $\hookrightarrow$ IFs $\hookrightarrow$ IITs

- **Deep induction specializes to structural induction, but is especially useful for deep data types (particularly truly nested types which are deep over themselves).**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**
- **Our methodology for IITs specializes to those given earlier for IFs, GADTs, nested types, and ADTs — each of which subsumes those that came before it.**

ADTs $\hookrightarrow$ nested types $\hookrightarrow$ GADTs $\hookrightarrow$ IFs $\hookrightarrow$ IITs

- **Deep induction specializes to structural induction, but is especially useful for deep data types (particularly truly nested types which are deep over themselves).**
- **Deep induction for sorted lists can prove, e.g., that if $>$ is the built-in ordering on Nat, then mapping a $>$- and evenness-preserving function over a list of even natural numbers in strictly decreasing order results in another such list.**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**
- **Our methodology for IITs specializes to those given earlier for IFs, GADTs, nested types, and ADTs — each of which subsumes those that came before it.**

ADTs $\hookrightarrow$ nested types $\hookrightarrow$ GADTs $\hookrightarrow$ IFs $\hookrightarrow$ IITs

- **Deep induction specializes to structural induction, but is especially useful for deep data types (particularly truly nested types which are deep over themselves).**
- **Deep induction for sorted lists can prove, e.g., that if $>$ is the built-in ordering on Nat, then mapping a $>$- and evenness-preserving function over a list of even natural numbers in strictly decreasing order results in another such list.**
- **There are variations on IFs and IITs that require more advanced techniques for constructing their liftings and deep induction rules.**

Conclusion

- **Deep induction inducts over *all* of the structure in a data type, applying custom predicates to its layers of "internal" data.**
- **Our methodology for constructing deep induction rules requires no data-type-specific techniques.**
- **Our methodology for IITs specializes to those given earlier for IFs, GADTs, nested types, and ADTs — each of which subsumes those that came before it.**

ADTs $\hookrightarrow$ nested types $\hookrightarrow$ GADTs $\hookrightarrow$ IFs $\hookrightarrow$ IITs

- **Deep induction specializes to structural induction, but is especially useful for deep data types (particularly truly nested types which are deep over themselves).**
- **Deep induction for sorted lists can prove, e.g., that if $>$ is the built-in ordering on Nat, then mapping a $>$- and evenness-preserving function over a list of even natural numbers in strictly decreasing order results in another such list.**
- **There are variations on IFs and IITs that require more advanced techniques for constructing their liftings and deep induction rules.**
- **See the paper and its code for examples and more details...**

Thank You!