Deep Induction Rules for Advanced Data Types

Patricia Johann

Appalachian State University

PA@ASL May 2025

Structural induction for lists

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

Structural induction for lists

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

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

Deep induction for lists

$$\mathsf{List}^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to \mathsf{List\,a} \to \mathsf{Set}$$

$$\mathsf{List}^{\wedge}\,\mathsf{Q}\,[\,] \;=\; \top$$

$$\mathsf{List}^{\wedge}\,\mathsf{Q}\,(\mathsf{x} :: \mathsf{xs}) \;=\; \mathsf{Q\,x} \times \mathsf{List}^{\wedge}\,\mathsf{Q\,xs}$$

Deep induction for lists

$$\mathsf{List}^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to \mathsf{List\,a} \to \mathsf{Set}$$

$$\mathsf{List}^{\wedge}\,\mathsf{Q}\,[\,] \;=\; \top$$

$$\mathsf{List}^{\wedge}\,\mathsf{Q}\,(\mathsf{x} :: \mathsf{xs}) \;=\; \mathsf{Q\,x} \times \mathsf{List}^{\wedge}\,\mathsf{Q\,xs}$$

$$\begin{aligned}
&(\mathsf{P} : \mathsf{List\,a} \to \mathsf{Set}) \to (\mathsf{Q} : \mathsf{a} \to \mathsf{Set}) \to \\
&\quad \mathsf{P}\,[\,] \to ((\mathsf{x} : \mathsf{a}) \to (\mathsf{xs} : \mathsf{List\,a}) \to \mathsf{Q\,x} \to \mathsf{P\,xs} \to \mathsf{P}\,(\mathsf{x} :: \mathsf{xs})) \to \\
&\quad (\mathsf{xs} : \mathsf{List\,a}) \to \mathsf{List}^{\wedge}\,\mathsf{Q\,xs} \to \mathsf{P\,xs}
\end{aligned}$$

Structural induction for perfect trees

```
data PTree (a : Set) : Set where
  pleaf  : a → PTree a
  pnode  : PTree (a × a) → PTree a
```

Structural induction for perfect trees

```
data PTree (a : Set) : Set where
  pleaf  : a → PTree a
  pnode  : PTree (a × a) → PTree a

(P : {a : Set} → PTree a → Set) →
  ({a : Set} → (x : a) → P (pleaf x)) →
  ({a : Set} → (xs : PTree (a × a)) → P xs → P (pnode xs)) →
  (xs : PTree a) → P xs
```

Deep induction for perfect trees

$$\times^{\wedge} : (\mathbf{a} \to \mathbf{Set}) \to (\mathbf{b} \to \mathbf{Set}) \to \mathbf{a} \times \mathbf{b} \to \mathbf{Set}$$

$$\times^{\wedge}\, \mathbf{Q_a}\, \mathbf{Q_b}\, (\mathbf{x}, \mathbf{y}) = \mathbf{Q_a}\, \mathbf{x} \times \mathbf{Q_b}\, \mathbf{y}$$

Deep induction for perfect trees

$$\times^{\wedge} : (\mathbf{a} \to \mathbf{Set}) \to (\mathbf{b} \to \mathbf{Set}) \to \mathbf{a} \times \mathbf{b} \to \mathbf{Set}$$
$$\times^{\wedge}\, \mathbf{Q_a}\, \mathbf{Q_b}\, (\mathbf{x}, \mathbf{y}) = \mathbf{Q_a}\, \mathbf{x} \times \mathbf{Q_b}\, \mathbf{y}$$

$$\mathbf{PTree}^{\wedge} : (\mathbf{a} \to \mathbf{Set}) \to \mathbf{PTree}\, \mathbf{a} \to \mathbf{Set}$$
$$\mathbf{PTree}^{\wedge}\, \mathbf{Q}\, (\mathbf{pleaf}\, \mathbf{x}) = \mathbf{Q}\, \mathbf{x}$$
$$\mathbf{PTree}^{\wedge}\, \mathbf{Q}\, (\mathbf{pnode}\, \mathbf{xs}) = \mathbf{PTree}^{\wedge}\, (\times^{\wedge}\, \mathbf{Q}\, \mathbf{Q})\, \mathbf{xs}$$

Deep induction for perfect trees

$$\times^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to (\mathsf{b} \to \mathsf{Set}) \to \mathsf{a} \times \mathsf{b} \to \mathsf{Set}$$
$$\times^{\wedge}\, \mathsf{Q_a}\, \mathsf{Q_b}\, (\mathsf{x}, \mathsf{y}) = \mathsf{Q_a}\, \mathsf{x} \times \mathsf{Q_b}\, \mathsf{y}$$

$$\mathsf{PTree}^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to \mathsf{PTree}\, \mathsf{a} \to \mathsf{Set}$$
$$\mathsf{PTree}^{\wedge}\, \mathsf{Q}\, (\mathsf{pleaf}\, \mathsf{x}) = \mathsf{Q}\, \mathsf{x}$$
$$\mathsf{PTree}^{\wedge}\, \mathsf{Q}\, (\mathsf{pnode}\, \mathsf{xs}) = \mathsf{PTree}^{\wedge}\, (\times^{\wedge}\, \mathsf{Q}\, \mathsf{Q})\, \mathsf{xs}$$

$$(\mathsf{P} : \{\mathsf{a} : \mathsf{Set}\} \to (\mathsf{a} \to \mathsf{Set}) \to \mathsf{PTree}\, \mathsf{a} \to \mathsf{Set}) \to$$
$$(\{\mathsf{a} : \mathsf{Set}\} \to (\mathsf{Q} : \mathsf{a} \to \mathsf{Set}) \to (\mathsf{x} : \mathsf{a}) \to \mathsf{Q}\, \mathsf{x} \to \mathsf{P}\, \mathsf{Q}\, (\mathsf{pleaf}\, \mathsf{x})) \to$$
$$(\{\mathsf{a} : \mathsf{Set}\} \to (\mathsf{Q} : \mathsf{a} \to \mathsf{Set}) \to (\mathsf{xs} : \mathsf{PTree}\, (\mathsf{a} \times \mathsf{a})) \to \mathsf{P}\, (\times^{\wedge}\, \mathsf{Q}\, \mathsf{Q})\, \mathsf{xs} \to \mathsf{P}\, \mathsf{Q}\, (\mathsf{pnode}\, \mathsf{xs})) \to$$
$$(\mathsf{Q} : \mathsf{a} \to \mathsf{Set}) \to (\mathsf{xs} : \mathsf{PTree}\, \mathsf{a}) \to \mathsf{PTree}^{\wedge}\, \mathsf{Q}\, \mathsf{xs} \to \mathsf{P}\, \mathsf{Q}\, \mathsf{xs}$$

GADTs and their Henry Ford encodings

```
data Seq : Set → Set where
  inj :  a → Seq a
  pair :  Seq b → Seq c → Seq (b × c)
```

GADTs and their Henry Ford encodings

```
data Seq : Set → Set where
  inj :  a → Seq a
  pair :  Seq b → Seq c → Seq (b × c)
```

```
data Seq (a : Set) : Set  where
  inj :  a → Seq a
  pair :  (b × c) ≡ a → Seq b → Seq c → Seq a
```

GADTs and their Henry Ford encodings

```
data Seq : Set → Set where
  inj :  a → Seq a
  pair :  Seq b → Seq c → Seq (b × c)

data Seq (a : Set) : Set  where
  inj :  a → Seq a
  pair :  (b × c) ≡ a → Seq b → Seq c → Seq a

data _≡_ (x : a) : a → Set where
  refl : x ≡ x
```

GADTs and their Henry Ford encodings

```
data Seq : Set → Set where
  inj :  a → Seq a
  pair :  Seq b → Seq c → Seq (b × c)

data Seq (a : Set) : Set  where
  inj :  a → Seq a
  pair :  (b × c) ≡ a → Seq b → Seq c → Seq a

data _≡_ (x : a) : a → Set where
  refl : x ≡ x

≡^ : (a → Set) → (b → Set) → a ≡ b → Set
≡^  Q Q′ refl  =  (x : a) → Q x ≡ Q′ x
```

Deep induction for GADTs

$$\mathsf{Seq}^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to \mathsf{Seq\,a} \to \mathsf{Set}$$

$$\mathsf{Seq}^{\wedge}\ \mathsf{Q_a}\ (\mathsf{inj\,x}) = \mathsf{Q_a\,x}$$

$$\mathsf{Seq}^{\wedge}\ \mathsf{Q_a}\ (\mathsf{pair\,p\,s_b\,s_c}) = \exists[\mathsf{Q_b}]\exists[\mathsf{Q_c}]\ \equiv^{\wedge}(\times^{\wedge}\ \mathsf{Q_b}\ \mathsf{Q_c})\ \mathsf{Q_a\,p} \times \mathsf{Seq}^{\wedge}\ \mathsf{Q_b\,s_b} \times \mathsf{Seq}^{\wedge}\ \mathsf{Q_c\,s_c}$$

The suggestive notation $\exists[\mathsf{x}]\,\mathsf{F\,x}$ is syntactic sugar for the type of dependent pairs (x, b) with $\mathsf{x} : \mathsf{a}$, $\mathsf{b} : \mathsf{F\,x}$, and $\mathsf{F} : \mathsf{a} \to \mathsf{Set}$

Deep induction for GADTs

$$\mathsf{Seq}^{\wedge} : (\mathsf{a} \to \mathsf{Set}) \to \mathsf{Seq\,a} \to \mathsf{Set}$$

$$\mathsf{Seq}^{\wedge}\ \mathsf{Q_a}\ (\mathsf{inj\,x}) = \mathsf{Q_a\,x}$$

$$\mathsf{Seq}^{\wedge}\ \mathsf{Q_a}\ (\mathsf{pair\,p\,s_b\,s_c}) = \exists[\mathsf{Q_b}]\exists[\mathsf{Q_c}]\ \equiv^{\wedge} (\times^{\wedge}\ \mathsf{Q_b}\ \mathsf{Q_c})\ \mathsf{Q_a\,p} \times \mathsf{Seq}^{\wedge}\ \mathsf{Q_b\,s_b} \times \mathsf{Seq}^{\wedge}\ \mathsf{Q_c\,s_c}$$

The suggestive notation $\exists[\mathsf{x}]\,\mathsf{F\,x}$ is syntactic sugar for the type of dependent pairs (x, b) with $\mathsf{x} : \mathsf{a}$, $\mathsf{b} : \mathsf{F\,x}$, and $\mathsf{F} : \mathsf{a} \to \mathsf{Set}$

$$\begin{aligned}
&(\mathsf{P} : \{\mathsf{a} : \mathsf{Set}\} \to (\mathsf{a} \to \mathsf{Set}) \to \mathsf{Seq\,a} \to \mathsf{Set}) \to \\
&\quad (\{\mathsf{a} : \mathsf{Set}\} \to (\mathsf{x} : \mathsf{a}) \to (\mathsf{Q_a} : \mathsf{a} \to \mathsf{Set}) \to \mathsf{Q_a\,x} \to \mathsf{P\ Q_a\ (inj\,x)}) \to \\
&\quad (\{\mathsf{a\,b\,c} : \mathsf{Set}\} \to (\mathsf{p} : (\mathsf{b} \times \mathsf{c}) \equiv \mathsf{a}) \to (\mathsf{s_b} : \mathsf{Seq\,b}) \to (\mathsf{s_c} : \mathsf{Seq\,c}) \to (\mathsf{Q_a} : \mathsf{a} \to \mathsf{Set}) \to \\
&\qquad (\mathsf{Q_b} : \mathsf{b} \to \mathsf{Set}) \to (\mathsf{Q_c} : \mathsf{c} \to \mathsf{Set}) \to \mathsf{P\ Q_b\ s_b} \to \mathsf{P\ Q_c\ s_c} \to \mathsf{P\ Q_a\ (pair\,p\,s_b\,s_c)}) \to \\
&\quad (\mathsf{Q_a} : \mathsf{a} \to \mathsf{Set}) \to (\mathsf{s} : \mathsf{Seq\,a}) \to \mathsf{Seq}^{\wedge}\ \mathsf{Q_a\,s} \to \mathsf{P\ Q_a\,s}
\end{aligned}$$

Liftings for inductive families

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

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

Liftings for inductive families

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

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

Vec^ : (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 inductive families

$$
\begin{aligned}
&(Q_a : a \to \mathsf{Set}) \to (Q_n : \mathsf{Nat} \to \mathsf{Set}) \to (P : \{n : \mathsf{Nat}\} \to \mathsf{Vec}\ a\ n \to \mathsf{Set}) \to \\
&\quad (Q_n\ \mathsf{zero} \to P\ [\,]) \to \\
&\quad (\{n : \mathsf{Nat}\} \to (x : a) \to (xs : \mathsf{Vec}\ a\ n) \to Q_a\ x \to P\ xs \to Q_n\ (\mathsf{suc}\ n) \to P\ (x :: xs)) \to \\
&\quad (xs : \mathsf{Vec}\ a\ n) \to \mathsf{Vec}^{\wedge}\ Q_a\ Q_n\ xs \to P\ xs
\end{aligned}
$$

Conclusion

- **Check type indices satisfy their predicates parameterizing P**
- **Check term indices' satisfy their predicates parameterizing P provided term indices of recursive subdata do**

Conclusion

- **Check type indices satisfy their predicates parameterizing P**
- **Check term indices' satisfy their predicates parameterizing P provided term indices of recursive subdata do**
- **The techniques scales: Fancier examples are linked off my web page.**

If

```
data VSeq (a : Set) {d : Set} : Vec d n → Set where
  vinj : a → (xs : Vec d n) → VSeq a xs
  vpair : (b × c) ≡ a → {ys : Vec d m} → {zs : Vec d n} →
            VSeq b ys → VSeq c zs → VSeq a (ys ++ zs)
```

then, for any **a : Set**, every **xs : Vec Nat n**, and every **s : VSeq a xs**, if every subterm of **s** constructed using **vinj** is indexed by a vector of even length all of whose elements are odd, then **s** itself is indexed by such a vector.

Conclusion

- **Check type indices satisfy their predicates parameterizing P**
- **Check term indices' satisfy their predicates parameterizing P provided term indices of recursive subdata do**
- **The techniques scales: Fancier examples are linked off my web page.**

  If

  $$\begin{aligned}&\mathsf{data\ VSeq\ (a : Set)\ \{d : Set\}\ :\ Vec\ d\ n \to Set\ where}\\&\quad \mathsf{vinj\ :\ a \to (xs : Vec\ d\ n) \to VSeq\ a\ xs}\\&\quad \mathsf{vpair\ :\ (b \times c) \equiv a \to \{ys : Vec\ d\ m\} \to \{zs : Vec\ d\ n\} \to}\\&\qquad \mathsf{VSeq\ b\ ys \to VSeq\ c\ zs \to VSeq\ a\ (ys +\!\!+ zs)}\end{aligned}$$

  then, for any a : Set, every xs : Vec Nat n, and every s : VSeq a xs, if every subterm of s constructed using vinj is indexed by a vector of even length all of whose elements are odd, then s itself is indexed by such a vector.

- **Even scales to inductive-inductive types!**

Thank You!