Generic Derivation of Induction for Impredicative Encodings in Cedille

Denis Firsov and Aaron Stump

Department of Computer Science
The University of Iowa

January 9, 2018

Outline

1. Motivation
2. Type theory
3. Induction for natural numbers
4. Induction generically

Motivation I

- It is **possible** to encode inductive datatypes in pure type theory.

  Nat = $\forall$ X : $\star$. (X $\to$ X) $\to$ X $\to$ X.

- It is **impossible** to derive induction principle in the second-order dependent type theory (Geuvers, 2001).
- As a consequence, most languages come with built-in infrastructure for defining inductive datatypes (Agda, Coq, Idris, etc.).

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

- Is it possible to extend CC with some **typing constructs** so that the induction becomes provable?

Motivation II

The Calculus of Dependent Lambda Eliminations (CDLE).

- CDLE is a pure type theory proposed by Aaron Stump (JFP, 2017).
- It adds three typing constructs to the Curry-style Calculus of Constructions:
  1. dependent intersection types,
  2. implicit products,
  3. a primitive heterogeneous equality.
- Cedille is an implementation of CDLE type theory (in Agda!).

Extension: Dependent intersection types

- Formation

$$\frac{\Gamma \vdash T : \star \quad \Gamma, x : T \vdash T' : \star}{\Gamma \vdash \iota\, x : T.\, T' : \star}$$

- Introduction

$$\frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : [t_1/x]T' \quad \Gamma \vdash p : t_1 \simeq t_2}{\Gamma \vdash [t_1, t_2\{p\}] : \iota\, x : T.\, T'}$$

- Elimination

$$\frac{\Gamma \vdash t : \iota\, x : T.\, T'}{\Gamma \vdash t.1 : T} \text{ first view} \qquad \frac{\Gamma \vdash t : \iota\, x : T.\, T'}{\Gamma \vdash t.2 : [t.1/x]T'} \text{ second view}$$

- Erasure

$$\begin{array}{lcl} |[t_1, t_2\{p\}]| & = & |t_1| \\ |t.1| & = & |t| \\ |t.2| & = & |t| \end{array}$$

Extension: Implicit products

- Formation

$$\frac{\Gamma, x : T' \vdash T : \star}{\Gamma \vdash \forall\, x : T'.\ T : \star}$$

- Introduction

$$\frac{\Gamma, x : T' \vdash t : T \quad x \notin FV(|t|)}{\Gamma \vdash \Lambda\, x : T'.\ t : \forall\, x : T'.\ T}$$

- Elimination

$$\frac{\Gamma \vdash t : \forall\, x : T'.\ T \quad \Gamma \vdash t' : T'}{\Gamma \vdash t\ \ -t' : [t'/x]\,T}$$

- Erasure

$$\begin{array}{lcl} |\Lambda\, x : T.\ t| & = & |t| \\ |t\ \ -t'| & = & |t| \end{array}$$

Extension: Equality

- Formation rule

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash t' : T'}{\Gamma \vdash t \simeq t' : \star}$$

- Introduction

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \beta : t \simeq t}$$

- Elimination

$$\frac{\Gamma \vdash t' : t_1 \simeq t_2 \quad \Gamma \vdash t : [t_1/x]T}{\Gamma \vdash \rho\ t' \ -\ t : [t_2/x]T}$$

- Erasure

$$\begin{array}{lcl} |\beta| & = & \lambda x. x \\ |\rho\ t\ -\ t'| & = & |t'| \end{array}$$

Definition of natural numbers

- Define Church-style natural numbers

  ```
  cNat ◀ ⋆ = ∀ X : ⋆. (X → X) → X → X.
  cZ ◀ cNat = Λ X. λ s. λ z. z.
  cS ◀ cNat → cNat = λ n. Λ X. λ s. λ z. s (n X s z).
  ```
- Define inductivity predicate for cNat:

  ```
  cNatInductive ◀ cNat → ⋆ = λ x : cNat.
  ∀ Q : cNat → ⋆.
  (∀ x : cNat. Q x → Q (cS x)) → Q cZ → Q x.
  ```
- Define the "true" type of natural numbers as dependent intersection of cNat and predicate cNatInductive.

  ```
  Nat ◀ ⋆ = ι x : cNat. cNatInductive x.
  ```
- Define constructors for Nat

  ```
  Z ◀ Nat = [ cZ, Λ X. λ s. λ z. z { β } ].
  S ◀ Nat → Nat = λ n. [ cS n.1,
   Λ P. λ s. λ z. s -n.1 (n.2 P s z) { β } ].
  ```

Induction for natural numbers

- If `n : Nat` then `n.1` is `cNat` and `n.2 : cNatInductive n.1`. Moreover, `n ≃ n.1`.
- The goal is to prove that every "true" natural `Nat` is inductive:
  `NatInductive ◀ Nat → ⋆ = λ x : Nat. ∀ Q : Nat → ⋆.`
  `(∀ x : Nat. Q x → Q (S x)) → Q Z → Q x.`
- Define the following predicate combinator
  `Lift ◀ (Nat → ⋆) → cNat → ⋆ = λ Q : Nat → ⋆.`
  `λ x : cNat. Σ x' : Nat. (x ≃ x'.1 × Q x')`
- Since `x ≃ x.1` then for any predicate `Q` on `Nat`
  `equiv ◀ Π n : Nat. Q n ⇔ Lift Q n.1`

1. Let `n` be natural, `Q` predicate on `Nat`, `s` and `z` be step and base cases.
2. Use `equiv` to get step `s'` and base `b'` cases for `Lift Q` from `s` and `z`.
3. Since, `n.1` is inductive then we use `n.2 (Lift Q) s' z'` to derive `Lift Q n.1`.
4. Finally, get `Q n` from `Lift Q n.1`.

Mendler-style inductive datatypes I

- Categorically, inductive datatypes are modelled as initial F-algebras.
- Mendler-style F-algebra is a pair of object (*carrier*) X and a natural transformation $\mathcal{C}(-, X) \to \mathcal{C}(F\ -, X)$.
- In Cedille, object is a type and a natural transformation is a polymorphic function:

```
AlgM ◀ ⋆ → ⋆ = λ X : ⋆.
 ∀ R : ⋆. (R → X) → F R → X.
```

- The object of initial Mendler-style F-algebra is a least fixed point of `F`:

```
FixM ◀ ⋆ = ∀ X : ⋆. AlgM X → X.
```

- There is a homomorphism from the carrier of initial algebra to the carrier of any other algebra:

```
foldM ◀ ∀ X : ⋆. AlgM X → FixM → X = <..>
```

- Define the arrow of initial Mendler-style F-algebra:

```
inM ◀ AlgM FixM = λ c. λ v. λ alg.
 alg (foldM alg) (fmap c v).
```

Mendler-style inductive datatypes II

- Goal is to define an inductive subset of `FixM` as an intersection type.
- The value `x : FixM` and the proof that `x` is inductive must be equal:

```
FixM ◀ ⋆ = ∀ X : ⋆. AlgM X → X.

IsIndFixM ◀ FixM → ⋆ = λ x : FixM.
 ∀ Q : FixM → ⋆.  PrfAlgM FixM Q inM → Q x.
```

- Proof algebra

```
AlgM ◀ ⋆ → ⋆ = λ X : ⋆.
 ∀ R : ⋆. (R → X) → F R → X.


PrfAlgM ◀ Π X : ⋆. (X → ⋆) → AlgM X → ⋆
 = λ X : ⋆. λ Q : X → ⋆. λ alg : AlgM X.
   ∀ R : ⋆.
   ∀ cast : R → X. ∀ _ : ∀ r : R. cast r ≃ r.
   (Π r : R. Q (cast r)) →
   Π fr : F R.  Q (alg cast fr).
```

Mendler-style inductive datatypes III

- Inductive subset of `FixM` is then

  ```
  FixIndM ◀ ⋆ = ι x : FixM. IsIndFixM x.
  ```

- We implement the initial Mendler-style F-algebra

  ```
  inFixIndM ◀ AlgM FixIndM = <..>
  ```

- Induction principle

  ```
  inductionM ◀ ∀ Q : FixIndM → ⋆.
   PrfAlgM FixIndM Q inFixIndM →
   Π x : FixIndM. Q x = <..>
  ```

Properties I

- Naturality of Mendler-style algebras

```
Natural ◀ Π X : ⋆. AlgM X → ⋆ =
 λ X : ⋆. λ algM : AlgM X.
 ∀ R : ⋆. ∀ f : R → X. ∀ fr : F R.
 algM f fr ≃ algM (λ x. x) (fmap f fr).
```

- Assuming naturality of Mendler-style F-algebras we prove
  - Universality
  - Reflection
  - Cancellation
  - Fusion

Lambek's lemma

- To start with we convert the initial Mendler-style F-algebra to the Church-style F-algebra:

```
inFixIndM' ◀ F FixIndM → FixIndM
 = inFixIndM (λ x. x).
```

- The categorical model of inductive types gives the exact recipe on how to implement the inverse of `inFixIndM'`, namely:

```
outFixIndM ◀ FixIndM → F FixIndM
 = fold (fmap inFixIndM).
```

- We show that it is a pre-inverse and post-inverse:

```
inoutM ◀ Π x : FixIndM.
 inFixIndM' (outFixIndM x) ≃ x = <..>

outinM ◀ Π x : F FixIndM.
 outFixIndM (inFixIndM' x) ≃ x = <..>
```

Discussion

- Church-style encoding is based on conventional F-algebras:

  $\texttt{AlgC} \blacktriangleleft \star \to \star = \lambda\ \texttt{X} : \star.\ \texttt{F X} \to \texttt{X}.$

- Church-style encoding satisfies the same set of properties without naturality assumptions.
- Derived rule of induction allows to prove the isomorphism of Church and Mendler-style encodings.
- Surprising observation is that derivation of induction for Mendler-style encodings uses only the first functor law.
- The consequence is that we can take fixed points and prove induction for positive schemes which are not functors:

  $\texttt{F} \blacktriangleleft \star \to \star = \lambda\ \texttt{X} : \star.\ \Sigma\ \texttt{x1} : \texttt{X}.\ \Sigma\ \texttt{x2} : \texttt{X}.\ \texttt{x1} \neq \texttt{x2}.$

  $\texttt{mapId} \blacktriangleleft \forall\ \texttt{X Y} : \star.\ \texttt{Id X Y} \to \texttt{F X} \to \texttt{F Y}$

Ongoing and Future work

- Proof reuse (by Larry Diehl).
- Bestiary of lambda-encodings (by Richard Blair).
- Type inference algorithm for Cedille (by Chris Jenkins).
- Constant time predecessor for linear space lambda-encodings.
- Generic course-of-value datatypes.
- (Small) Induction-recursion.

Thank you for your attention!