Higher Inductive Types in Programming

Henning Basold, Herman Geuvers, Niels van der Weide

May 10, 2017

"A canonical type A is defined by prescribing how a canonical object of type A is formed as well as how two equal canonical objects of type A are formed. There is no limitation on this prescription except that the relation of equality which it defines between canonical objects of type A must be reflexive, symmetric and transitive. If the rules for forming canonical objects as well as equal canonical objects of a certain type are called the introduction rules for that type, we may thus say with Gentzen(1934) that a canonical type (proposition) is defined by its introduction rules."

"*A canonical type A is defined by prescribing how a canonical object of type A is formed as well as how two equal canonical objects of type A are formed. There is no limitation on this prescription except that the relation of equality which it defines between canonical objects of type A must be reflexive, symmetric and transitive. If the rules for forming canonical objects as well as equal canonical objects of a certain type are called the introduction rules for that type, we may thus say with Gentzen(1934) that a canonical type (proposition) is defined by its introduction rules.*"
Martin-Löf, Per. "Constructive mathematics and computer programming." *Studies in Logic and the Foundations of Mathematics* 104 (1982): 153-175.

"*A canonical type A is defined by prescribing how a canonical object of type A is formed as well as how two equal canonical objects of type A are formed. There is no limitation on this prescription except that the relation of equality which it defines between canonical objects of type A must be reflexive, symmetric and transitive. If the rules for forming canonical objects as well as equal canonical objects of a certain type are called the introduction rules for that type, we may thus say with Gentzen(1934) that a canonical type (proposition) is defined by its introduction rules.*"
Martin-Löf, Per. "Constructive mathematics and computer programming." *Studies in Logic and the Foundations of Mathematics* 104 (1982): 153-175.

Higher Inductive Types

Higher inductive type (HIT): generated by inductive point constructors and path constructors.
Canonical types in Martin-Löf's sense corresponds with higher inductive types in HoTT.

Goal

Define HITs formally and illustrate the definition with examples.

Related Work

- Running Circles Around (In) Your Proof Assistant; or, Quotients that Compute (Licata)
- Higher Inductive Types in Programming (Basold, Geuvers, Van der Weide)
- Type Theory in Type Theory with Quotient Inductive Types (Altenkirch, Kaposi)
- Higher Inductive Types in the Groupoid Model (Dybjer, Moeneclaey)
- The HoTT Library: A Formalization of Homotopy Type Theory in Coq (Bauer, Gross, Lumsdaine, Shulman, Sozeau, Spitters)

Syntax of HITs

For a higher inductive type, we want to add equations like

$$\prod x : A, t = r$$

With t and r 'canonical terms'.

Syntax of HITs

For a higher inductive type, we want to add equations like

$$\prod x : A, t = r$$

With t and r 'canonical terms'.
This means the scheme looks something like

Inductive $T\ (B_1 : \text{Type}) \ldots (B_\ell : \text{Type}) :=$
$|\ \ c_1 : H_1[T\ B_1 \cdots B_\ell] \to T\ B_1 \cdots B_\ell$
$\ldots$
$|\ \ c_k : H_k[T\ B_1 \cdots B_\ell] \to T\ B_1 \cdots B_\ell$
$|\ \ p_1 : \prod(x : A_1[T\ B_1 \cdots B_\ell]), t_1 = r_1$
$\ldots$
$|\ \ p_n : \prod(x : A_n[T\ B_1 \cdots B_\ell]), t_n = r_n$

Constructor Terms

We start with:

- We have context Γ;
- We have $c_i : H_i(T) \to T$ (given by inductive type);
- We have a parameter $x : A[T]$ with A polynomial functor.

Building Constructor Terms

$$\frac{\Gamma \vdash t : B \qquad T \text{ does not occur in } B}{x : A \Vdash t : B} \qquad \frac{}{x : A \Vdash x : A}$$

Building Constructor Terms

$$\frac{\Gamma \vdash t : B \qquad T \text{ does not occur in } B}{x : A \Vdash\ t : B} \qquad \frac{}{x : A \Vdash\ x : A}$$

$$\frac{j \in \{1, 2\} \qquad x : A \Vdash\ r : G_1 \times G_2}{x : A \Vdash\ \pi_j\, r : G_j}$$

$$\frac{j = \{1, 2\} \qquad x : A \Vdash\ r_j : G_j}{x : A \Vdash\ (r_1, r_2) : G_1 \times G_2}$$

Building Constructor Terms

$$\frac{\Gamma \vdash t : B \qquad T \text{ does not occur in } B}{x : A \Vdash t : B} \qquad \frac{}{x : A \Vdash x : A}$$

$$\frac{j \in \{1,2\} \qquad x : A \Vdash r : G_1 \times G_2}{x : A \Vdash \pi_j\, r : G_j}$$

$$\frac{j = \{1,2\} \qquad x : A \Vdash r_j : G_j}{x : A \Vdash (r_1, r_2) : G_1 \times G_2}$$

$$\frac{j \in \{1,2\} \qquad x : A \Vdash r : G_j}{x : A \Vdash \mathrm{in}_j\, r : G_1 + G_2}$$

Building Constructor Terms

$$\frac{\Gamma \vdash t : B \qquad T \text{ does not occur in } B}{x : A \Vdash\ t : B} \qquad \frac{}{x : A \Vdash\ x : A}$$

$$\frac{j \in \{1,2\} \qquad x : A \Vdash\ r : G_1 \times G_2}{x : A \Vdash\ \pi_j\, r : G_j}$$

$$\frac{j = \{1,2\} \qquad x : A \Vdash\ r_j : G_j}{x : A \Vdash\ (r_1, r_2) : G_1 \times G_2}$$

$$\frac{j \in \{1,2\} \qquad x : A \Vdash\ r : G_j}{x : A \Vdash\ \mathrm{in}_j\, r : G_1 + G_2}$$

$$\frac{x : A \Vdash\ r : H_i[T]}{x : A \Vdash\ c_i\, r : T}$$

The Scheme

$$
\begin{array}{l}
\texttt{Inductive}\ T\ (B_1 : \mathrm{Type}) \ldots (B_\ell : \mathrm{Type}) := \\
\mid\ c_1 : H_1[T\ B_1 \cdots B_\ell] \to T\ B_1 \cdots B_\ell \\
\ldots \\
\mid\ c_k : H_k[T\ B_1 \cdots B_\ell] \to T\ B_1 \cdots B_\ell \\
\mid\ p_1 : \prod(x : A_1[T\ B_1 \cdots B_\ell]), t_1 = r_1 \\
\ldots \\
\mid\ p_n : \prod(x : A_n[T\ B_1 \cdots B_\ell]), t_n = r_n
\end{array}
$$

Here we have

- H_i and A_j are polynomials;
- t_j and r_j are constructor terms over $c_1, \ldots, c_k$ with $x : A_j \Vdash\ t_j, r_j : T$.

Note: all HITs in this talk are finitary. Also, only 1-HITs.

Introduction Rules

$$\frac{\Gamma \vdash B_1 : \textsc{Type} \qquad \cdots \qquad \Gamma \vdash B_\ell : \textsc{Type}}{\Gamma \vdash T\ B_1 \cdots B_\ell : \textsc{Type}}$$

$$\frac{\vdash \Gamma \quad \textsc{Ctx}}{\Gamma \vdash c_i : H_i[T] \to T}$$

$$\frac{\vdash \Gamma \quad \textsc{Ctx}}{\Gamma \vdash p_j : \prod(x : A_j[T]) \to t_j = r_j}$$

Lifting Constructor Terms

To lift a constructor term $x : A[T] \Vdash r : G[T]$, we need:

- Constructors $c_i : H_i[X] \to X$;
- A type family $Y : T \to \textsc{Type}$;
- Terms $\Gamma \vdash f_i : \prod(x : H_i[T]), \overline{H}_i(Y)\, x \to Y(c_i\, x)$.

Lifting Constructor Terms

To lift a constructor term $x : A[T] \Vdash r : G[T]$, we need:

- Constructors $c_i : H_i[X] \to X$;
- A type family $Y : T \to \mathrm{Type}$;
- Terms $\Gamma \vdash f_i : \prod(x : H_i[T]), \overline{H}_i(Y)\, x \to Y(c_i\, x)$.

Then we define

$$\Gamma, x : A[T], h_x : \overline{A}(Y)\, x \vdash \widehat{r} : \overline{G}(Y)\, r$$

Lifting Constructor Terms

To lift a constructor term $x : A[T] \Vdash r : G[T]$, we need:

- Constructors $c_i\colon H_i[X] \to X$;
- A type family $Y\colon T \to \textsc{Type}$;
- Terms $\Gamma \vdash f_i : \prod(x : H_i[T]), \overline{H}_i(Y)\, x \to Y(c_i\, x)$.

Then we define

$$\Gamma, x : A[T], h_x : \overline{A}(Y)\, x \vdash \widehat{r} : \overline{G}(Y)\, r$$

by induction as follows

$$\widehat{t} := t \qquad \widehat{x} := h_x \qquad \widehat{c_i\, r} := f_i\, r\, \widehat{r}$$

$$\widehat{\pi_j\, r} := \pi_j\, \widehat{r} \qquad \widehat{(r_1, r_2)} := (\widehat{r_1}, \widehat{r_2}) \qquad \widehat{\mathrm{in}_j\, r} := \widehat{r}$$

Elimination Rule

$$\frac{\begin{array}{c} Y\colon T \to \textsc{Type} \\ \Gamma \vdash f_i : \prod(x : H_i[T]), \overline{H}_i(Y)\, x \to Y\,(c_i\, x) \\ \Gamma \vdash q_j : \prod(x : A_j[T])(h_x : \overline{A}_j(Y)\, x), \widehat{t_j} =^{Y}_{(p_j\, x)} \widehat{r_j} \end{array}}{\Gamma \vdash T\mathrm{ind}(f_1, \ldots, f_k, q_1, \ldots, q_n) : \prod(x : T), Y\, x}$$

Note that $\widehat{t_j}$ and $\widehat{r_j}$ depend on all the f_i.

Elimination Rule

$$\frac{\begin{array}{c} Y : T \to \textsc{Type} \\ \Gamma \vdash f_i : \prod(x : H_i[T]), \overline{H}_i(Y)\, x \to Y\,(c_i\, x) \\ \Gamma \vdash q_j : \prod(x : A_j[T])(h_x : \overline{A}_j(Y)\, x), \widehat{t_j} =^{Y}_{(p_j\, x)} \widehat{r_j} \end{array}}{\Gamma \vdash T\mathrm{ind}(f_1, \ldots, f_k, q_1, \ldots, q_n) : \prod(x : T), Y\, x}$$

Note that $\widehat{t_j}$ and $\widehat{r_j}$ depend on all the f_i.

Computation Rules

$$T\text{ind}\,(c_i\ t) = f_i\ t\ (\overline{H}_i(T\text{ind})\ t),$$
$$\text{apD}\ T\text{ind}\ p_j\ a = q_j\ a\ (\overline{A}_j(T\text{ind})\ a).$$

HITs in Proof Assistants

How to program HITs in Coq/Agda?
Idea: add paths as axioms/postulates.

The Interval (Naively)

For the interval

Inductive $I^1 :=$
$| \; z : I^1$
$| \; o : I^1$
$| \; s : z = o$

we add the code

```
data I : Set where
    z : I
    o  : I
postulate
    seg : z == o
```

The Interval (Naively)

Problem: we can do too much.

$\texttt{f} : \texttt{I} \to \texttt{Nat}$
$\texttt{f z} = 0$
$\texttt{f o} = 1$

Then we have ap f seg : $0 = 1$.
This should not be possible.

The Interval (Correctly)

Solution: restrict access.

```
private
    data I' : Set where
        Zero : I'
        One  : I'

I : Set
I = I'

z : I
z = Zero

o : I
o = One
```

HITs in Proof Assistants: Elimination Rule

How to get the right elimination rule?

Elimination Rule (Naively)

We can try to postulate it.

```
postulate
    I-rec : {C : Set} → (a b : C) → (p : a == b)
        → I → C
```

Elimination Rule (Naively)

We can try to postulate it.

```
postulate
    I−rec : {C : Set} → (a b : C) → (p : a == b)
        → I → C
```

Problem: computation rules for points hold *propositionally*.

Elimination Rule (Better)

Define it as a function.

```
I−rec : {C : Set} → (a b : C) → (p : a == b)
    → I → C
I−rec a b _ Zero = a
I−rec a b _ One = b
```

Now computation rules for points are definitional.
This is how Licata did it.

HITs in Proof Assistants: Elimination Rule (Better)

Problem: define

```
Inductive C : Set :=
| N : C
| S : C
| E : N = S
| W : N = S
```

Define

$$f = I\text{ind}\ N\ S\ E$$

$$g = I\text{ind}\ N\ S\ W$$

HITs in Proof Assistants: Elimination Rule (Better)

Problem: define

```
Inductive C : Set :=
| N : C
| S : C
| E : N = S
| W : N = S
```

Define

$$f = I\text{ind } N\ S\ E$$

$$g = I\text{ind } N\ S\ W$$

Then

$$f = g$$

by refl!

HITs in Proof Assistants: Elimination Rule (Even Better)

In Coq (workaround in Agda is more annoying).

```
Definition I−rec (C : Type) (a, b : C) (p : a = b) : I → C :=
:= fun x ⇒
(match x return _ → C with
    | zero ⇒ fun _ ⇒ a
    | one ⇒ fun _ ⇒ b
) p.
```

This is solution in the HoTT library in Coq.

HITs in Proof Assistants: Computation Rules for Paths

Postulating them works fine.

```
postulate
    βseg : {C : Set} → (a b : C) → (p : a == b)
        → ap (I-rec a b p) seg == p
```

Now computation rules for points are definitional.

Some Examples of HITs

- Integers modulo n
- Finite Sets (free lattice)
- Lists (free monoid)
- Integers
- Expressions with $+$ and natural numbers
- Combinatory logic (K and S)
- Type Theory

Integers as a HIT

Let's define the integers.

```
Inductive ℤ? :=
| 0 : ℤ?
| S : ℤ? → ℤ?
| P : ℤ? → ℤ?
| i1 : ∏(x : ℤ?), S(P x) = x
| i2 : ∏(x : ℤ?), P(S x) = x
```

Integers as a HIT

Let's define the integers.

```
Inductive ℤ? :=
| 0 : ℤ?
| S : ℤ? → ℤ?
| P : ℤ? → ℤ?
| i₁ : ∏(x : ℤ?), S(P x) = x
| i₂ : ∏(x : ℤ?), P(S x) = x
```

Is this right?

Integers as a HIT

Let's define the integers.

`Inductive` $\mathbb{Z}? :=$
$| \ 0 : \ \mathbb{Z}?$
$| \ S : \mathbb{Z}? \to \mathbb{Z}?$
$| \ P : \mathbb{Z}? \to \mathbb{Z}?$
$| \ i_1 : \prod(x : \mathbb{Z}?), S(P\,x) = x$
$| \ i_2 : \prod(x : \mathbb{Z}?), P(S\,x) = x$

Is this right? No!

Integers as a HIT

Theorem

Equality of $\mathbb{Z}?$ *is not decidable.*

Sketch of proof:

- Hedberg: if equality of a type T is decidable, then T is a set.
- Sufficient: $\mathbb{Z}?$ is not a set.

$\mathbb{Z}$? is not a set

Consider:

$$i_2\,0 : S(P\,Z) = 0,$$

$$\text{ap } P\,(i_2\,0) : P(S(P\,Z)) = P\,0,$$

$\mathbb{Z}$? is not a set

Consider:

$$i_2\,0 : S(P\,Z) = 0,$$

$$\text{ap }\ P\,(i_2\,0) : P(S(P\,Z)) = P\,0,$$

$$i_1(P\,Z) : P(S(P\,Z)) = P\,0.$$

Claim: ap $P\,(i_2\,0) : P(S(P\,Z)) = P\,0$ and $i_1(P\,Z) : P(S(P\,Z)) = P\,0$ are not equal (assuming univalence).

Brief Intermezzo: the Circle

Recall the circle.

```
Inductive S^1 :=
| b : S^1
| l : b = b
```

Then with univalence: l and refl are unequal.

$\mathbb{Z}$? is not a set

Define $\mathbb{Z} \to S^1$ as follows

- 0 goes to b.
- P and S go to identity.
- $i_1\, x$ goes to l.
- $i_2\, x$ goes to refl.

Then $i_1(P\, Z)$ is mapped to l, but ap $P\, (i_2\, 0)$ to refl.

Rule of Thumb

Truncate if you don't need higher structure.

```
Inductive ℤ :=
| 0 : ℤ
| S : ℤ → ℤ
| P : ℤ → ℤ
| i₁ : ∏(x : ℤ), S(P x) = x
| i₂ : ∏(x : ℤ), P(S x) = x
| t : ∏(x, y : ℤ)(p, q : x = y), p = q
```

Integers Modulo 2

Example in similar spirit.

```
Inductive ℕ/2ℕ :=
| 0 : ℕ/2ℕ
| S : ℕ/2ℕ → ℕ/2ℕ
| m : ∏(n : ℕ/2ℕ), S(S n) = n
| t : ∏(x, y : ℕ/2ℕ)(p, q : x = y), p = q
```

Finite sets as free lattice, lists as free monoid.

Interesting: can we generalize this definition to arbitrary n?

Expressions with $+$ and $\mathbb{N}$ as a HIT

Let's define the expressions.

```
Inductive Exp:=
| val : ℕ → Exp
| plus : Exp → Exp → Exp
| eval : ∏(n, m : ℕ), plus(val n)(val m) = val(n + m)
```

Examples in a similar spirit: type theory in type theory, combinatory logic.

Normalization of Expressions

With this definition we can define

$$\mathsf{norm} : \prod(e : \mathsf{Exp}) \sum(n : \mathbb{N}), ||e = \mathsf{val}\ n||$$

where

$\texttt{Inductive}\ ||A|| :=$
$|\ \ \iota : A \to ||A||$
$|\ \ t : \prod(x, y : ||A||), x = y$

Semantics of Expressions

With this definition we can define

$$\mathsf{sem}_{S^1} : \mathsf{Exp} \to b = b$$

sending val n to l^n and plus to path concatenation.

Questions

- Can we make a good library for integers modulo 2? And integers?
- Can we define Scott's graph model, and show it is a model of combinatory logic using HITs?
- Simple imperative languages as a HIT?