Towards a theory of higher inductive types

Thorsten Altenkirch[1] Paolo Capriotti[1] *Gabe Dijkstra*[1]
Fredrik Nordvall Forsberg[2]

[1]University of Nottingham

[2]University of Strathclyde

May 20th, 2015

Goal

Our is goal is to:

- Define a general class of higher inductive types
- Akin to W-types
- Building upon Shulman and Lumsdaine's semantics

Higher inductive types versus ordinary inductive types

Ordinary inductive types:

$$
\begin{aligned}
&\underline{\mathit{data}}\ T : \mathit{Type}\ \underline{\mathit{where}} \\
&\quad c_0 : F_0\ T \to T \\
&\quad\ \vdots \\
&\quad c_k : F_k\ T \to T
\end{aligned}
$$

where every $F_i : \mathit{Type} \to \mathit{Type}$ is a (strictly positive) functor.

Higher inductive types versus ordinary inductive types

Ordinary inductive types:

$$\underline{data}\ T : Type\ \underline{where}$$
$$c : F_0\ T + \ldots + F_k\ T \to T$$

where every $F_i : Type \to Type$ is a (strictly positive) functor.

Higher inductive types versus ordinary inductive types

Ordinary inductive types:

$$\underline{data}\ T : Type\ \underline{where}$$
$$c : F\ T \to T$$

where $F : Type \to Type$ is a (strictly positive) functor.

Higher inductive types versus ordinary inductive types

Higher inductive types, e.g. the circle:

$\underline{data}\ S^1 : Type\ \underline{where}$
$\quad base : S^1$
$\quad loop : base =_{S^1} base$

- Dependencies on previous constructors
- *Higher* constructors: target of constructors not always T, but can also be an iterated path space of T.

Single functor $Type \to Type$ no longer suffices

Higher inductive types versus ordinary inductive types

Higher inductive types, e.g. propositional truncation:

$$\begin{array}{l} \underline{\mathit{data}} ||A|| : \mathit{Type}\ \underline{\mathit{where}} \\ \quad [\_] : A \to ||A|| \\ \quad \mathit{trunc} : (x\ y : ||A||) \to x = y \end{array}$$

- Dependencies on previous constructors
- *Higher* constructors: target of constructors not always T, but can also be an iterated path space of T.

Single functor $\mathit{Type} \to \mathit{Type}$ no longer suffices

General framework

Constructors are *dependent dialgebras*:

$$
\begin{array}{l}
\underline{data}\ T : Type\ \underline{where} \\
\quad c_0 : (x : F_0\ T) \rightarrow G_0\ (T, x) \\
\quad c_1 : (x : F_1\ (T, c_0)) \rightarrow G_1\ ((T, c_0), x) \\
\quad \vdots \\
\quad c_k : (x : F_k\ (T, c_0, \ldots, c_{k-1})) \rightarrow G_k\ ((T, c_0, \ldots, c_{k-1}), x)
\end{array}
$$

We will call:

- Every F_i an *argument* functor
- Every G_i a *target* functor

General framework – example: *interval*

The interval type:

$$\begin{aligned}&\underline{data}\ I : Type\ \underline{where}\\&\quad zero : I\\&\quad one\ \ : I\\&\quad seg\ \ : zero = one\end{aligned}$$

Argument functors:

$$\begin{array}{llll}F_0\ X & :\equiv 1 & (F_0 : Type & \to Type)\\ F_1\ (X, z) & :\equiv 1 & (F_1 : (F_0, G_0)\text{-alg} & \to Type)\\ F_2\ (X, z, o) & :\equiv 1 & (F_2 : (F_1, G_1)\text{-alg} & \to Type)\end{array}$$

Target functors:

$$\begin{array}{llll}G_0\ (X\qquad\ \ , x) & :\equiv X & (G_0 : \int_{Type} & F_0 \to Type)\\ G_1\ ((X, z)\quad\ , x) & :\equiv X & (G_1 : \int_{(F_0, G_1)\text{-alg}} & F_1 \to Type)\\ G_2\ ((X, z, o), x) & :\equiv (z = o) & (G_2 : \int_{(F_1, G_1)\text{-alg}} & F_2 \to Type)\end{array}$$

General framework – example: *interval*

The interval type:

$$\begin{array}{l} \underline{\mathit{data}}\ I : \mathit{Type}\ \underline{\mathit{where}} \\ \quad \mathit{zero} : I \\ \quad \mathit{one}\ \ : I \\ \quad \mathit{seg}\ \ : \mathit{zero} = \mathit{one} \end{array}$$

Category of elements:
objects:
$(X : \mathit{Type}) \times F_0\, X$
morphisms $(X, x) \to (Y, y)$:
$(f : X \to Y) \times (F_0\, f\, x = y)$

Argument functors:

$$\begin{array}{llll} F_0\, X & :\equiv 1 & (F_0 : \mathit{Type} & \to \mathit{Type}) \\ F_1\, (X, z) & :\equiv 1 & (F_1 : (F_0, G_0)\text{-alg} & \to \mathit{Type}) \\ F_2\, (X, z, o) & :\equiv 1 & (F_2 : (F_1, G_1)\text{-alg} & \to \mathit{Type}) \end{array}$$

Target functors:

$$\begin{array}{llll} G_0\, (X \qquad , x) & :\equiv X & (G_0 : \int_{\mathit{Type}} & F_0 \to \mathit{Type}) \\ G_1\, ((X, z) \quad , x) & :\equiv X & (G_1 : \int_{(F_0, G_1)\text{-alg}} & F_1 \to \mathit{Type}) \\ G_2\, ((X, z, o), x) & :\equiv (z = o) & (G_2 : \int_{(F_1, G_1)\text{-alg}} & F_2 \to \mathit{Type}) \end{array}$$

General framework

Constructors are *dependent dialgebras*:

$$c : (x : F\ X) \to G\ (X, x)$$

where

- $\mathbb{C} : \mathit{Cat}$
- $F : \mathbb{C} \to \mathit{Type}$ (*argument* functor)
- $G : \int_{\mathbb{C}} F \to \mathit{Type}$ (*target* functor)

General framework – 0-constructors

0-constructors are of the form:

$$c : (x : F\ X) \to U\ X$$

where

- $\mathbb{C} : \mathit{Cat}$ with a forgetful functor $U : \mathbb{C} \to \mathit{Type}$
- $F : \mathbb{C} \to \mathit{Type}$
- $G : \int_{\mathbb{C}} F \to \mathit{Type}$

  $$G\ (X, x) \ :\equiv\ U\ X$$

General framework – 1-constructors

1-constructors are of the form:

$$c : (x : F\ X) \to (l_0\ X\ x = r_0\ X\ x)$$

where

- $C : \mathit{Cat}$ with a forgetful functor $U : \mathbb{C} \to \mathit{Type}$
- $F : \mathbb{C} \to \mathit{Type}$
- $G : \int_{\mathbb{C}} F \to \mathit{Type}$
- $l_0\ r_0 : F \to U$

$$G\ (X, x)\ :\equiv\ (l_0\ X\ x = r_0\ X\ x)$$

We call this G functor Eq_0

General framework – 2-constructors

For 2-constructors:

$$c : (x : F\ X) \to (l_1\ X\ x = r_1\ X\ x)$$

where

- $l_0\ r_0 : F \to U$ (with $Eq_0\ (X, x)\ :\equiv\ (l_0\ X\ x = r_0\ X\ x)$)
- $l_1\ r_1 : 1 \to Eq_0$ (with $Eq_1\ (X, x)\ :\equiv\ (l_1\ X\ x = r_1\ X\ x)$)

General framework – 3-constructors

For 3-constructors:

$$c : (x : F\ X) \to (l_2\ X\ x = r_2\ X\ x)$$

where

- $l_0\ r_0 : F \to U$ (with $Eq_0\ (X, x)\ :\equiv\ (l_0\ X\ x = r_0\ X\ x)$)
- $l_1\ r_1 : 1 \to Eq_0$ (with $Eq_1\ (X, x)\ :\equiv\ (l_1\ X\ x = r_1\ X\ x)$)
- $l_2\ r_2 : 1 \to Eq_1$ (with $Eq_2\ (X, x)\ :\equiv\ (l_2\ X\ x = r_2\ X\ x)$)

General framework – $(n+1)$-constructors

For $(n+1)$-constructors:

$$c : (x : F\ X) \to (l_n\ X\ x = r_n\ X\ x)$$

where

- $l_0\ r_0 : F \to U$ (with $Eq_0\ (X, x)\ :\equiv\ (l_0\ X\ x = r_0\ X\ x)$)
- $l_1\ r_1 : 1 \to Eq_0$ (with $Eq_1\ (X, x)\ :\equiv\ (l_1\ X\ x = r_1\ X\ x)$)
- $l_2\ r_2 : 1 \to Eq_1$ (with $Eq_2\ (X, x)\ :\equiv\ (l_2\ X\ x = r_2\ X\ x)$)

  $\vdots$ $\vdots$

- $l_n\ r_n : 1 \to Eq_{n-1}$ (with $Eq_n\ (X, x)\ :\equiv\ (l_n\ X\ x = r_n\ X\ x)$)

Strict positivity – ordinary inductive types

We can't allow any argument functor: it has to be strictly positive:

$$\underline{data}\ Term : Type\ \underline{where}$$
$$\quad lam : (Term \to Term) \to Term$$

Strict positivity – higher inductive types

We can't allow any argument functor: it has to be strictly positive:

$$
\begin{array}{ll}
\underline{\textit{data}}\ \textit{InitialField} : \textit{Type}\ \underline{\textit{where}} & \\
\quad 0 & : \textit{InitialField} \\
\quad 1 & : \textit{InitialField} \\
\quad \_+\_ & : \textit{InitialField} \to \textit{InitialField} \to \textit{InitialField} \\
\quad \_*\_ & : \textit{InitialField} \to \textit{InitialField} \to \textit{InitialField} \\
\quad \vdots & \\
\quad \_^{-1} & : (x : \textit{InitialField}) \to (x = 0 \to \bot) \to \textit{InitialField} \\
\quad \vdots &
\end{array}
$$

InitialField does not exist: $\_^{-1}$ is not strictly positive

Type-containers

Strictly positive functors $Type \to Type$: containers

- Shapes $S : Type$
- Positions $T : S \to Type$

$$\llbracket S \lhd P \rrbracket_0 : Type \to Type$$
$$\llbracket S \lhd P \rrbracket_0\, X \;:\equiv\; (s : S) \times (P\, s \to X)$$

$$\llbracket S \lhd P \rrbracket_1 : (X \to Y) \to \llbracket S \lhd P \rrbracket_0\, X \to \llbracket S \lhd P \rrbracket_0\, Y$$
$$\llbracket S \lhd P \rrbracket_1\, f\, (s, t) \;:\equiv\; (s, f \circ t)$$

Example:

$$\begin{aligned} const_A\, X &= \llbracket A \lhd (\lambda a.0) \rrbracket_0\, X \\ &= A \times (0 \to X) \\ &= A \end{aligned}$$

$\mathbb{C}$-containers

Strictly positive functors $\mathbb{C} \to \mathit{Type}$: $\mathbb{C}$-containers (or *familially representable*)

- Shapes $S : \mathit{Type}$
- Positions $T : S \to |\mathbb{C}|$

$$
\begin{aligned}
&[\![S \lhd P]\!]_0 : \mathbb{C} \to \mathit{Type} \\
&[\![S \lhd P]\!]_0\, X :\equiv (s : S) \times \mathbb{C}\,(P\,s, X)
\end{aligned}
$$

$$
\begin{aligned}
&[\![S \lhd P]\!]_1 : \mathbb{C}\,(X, Y) \to [\![S \lhd P]\!]_0\, X \to [\![S \lhd P]\!]_0\, Y \\
&[\![S \lhd P]\!]_1\, f\,(s, t) :\equiv (s, f \circ t)
\end{aligned}
$$

Example (assuming $0 : |\mathbb{C}|$ is initial):

$$
\begin{aligned}
\mathit{const}_A\, X &= [\![A \lhd (\lambda a.0)]\!]_0\, X \\
&= A \times \mathbb{C}\,(0, X) \\
&= A
\end{aligned}
$$

$\mathbb{C}$-container morphisms

- Data for higher constructors requires *natural transformations*
- Natural transformations between containers: *container morphisms*:

For containers $S \lhd P$ and $T \lhd Q$, container morphisms are:

$$\begin{aligned} & (f : S \to T) \\ \times\ & (g : (a : S) \to \mathbb{C}\,(Q\,(f\ a), P\ a)) \end{aligned}$$

Container morphisms are *complete*:

- Each container morphism gives rise to a natural transformation and vice versa

Expressivity of containers

Data for constructors can be given using containers and container morphisms:

- Argument functors are given as containers
- Forgetful functors $U_i : (F_i, G_i)\text{-alg} \to \mathit{Type}$ can be given as containers if there exist $L_i \dashv U_i$
- Data for Eq_n functors are given as container morphisms
- Eq_n functors can be given as containers if we have $(n+1)$-HITs

Simplified approach to 1-HITs

In practice, constructor arguments rarely seem to refer to previous constructors.

We can identify a class of 1-HITs where we have:

- 0-constructors which do not depend on other constructors
- 1-constructors which may only depend on 0-constructors *in the targets*
- No dependencies between the 1-constructors

Examples: circle, suspension, truncation

Non-example: hub-spokes version of the torus

Simplified approach to 1-HITs

If we have:

- $F_0 : \mathit{Type} \to \mathit{Type}$ with $U_0 : (F_0, G_0)\text{-alg} \to \mathit{Type}$
- $F_1 : (F_0, G_0)\text{-alg} \to \mathit{Type}$ such that $F_1 = F_1' \circ U_0$
- $\ldots$ where $F_1' : \mathit{Type} \to \mathit{Type}$

then

$$F_1 \to U_0 \ \simeq \ F_1' \to F_0^*$$

where F_0^* is the free monad of F_0.

This approach has been fully formalised in Agda.

Coherence

- We can use $\mathbb{C}$-containers to internalise the theory
- We still need to be able to talk about the categories of dependent dialgebras

Coherence – Category of dependent dialgebras

Given $\mathbb{C}$ a category with functors $F : \mathbb{C} \to \mathit{Type}$ and $G : \int_{\mathbb{C}} F \to \mathit{Type}$, we can consider the category of dialgebras (F, G)-alg:

- objects:

$$\begin{array}{ll} & (X : |\mathbb{C}|) \\ \times & (\theta : (x : F\ X) \to G\ (X, x)) \end{array}$$

- morphisms $(X, \theta) \to (Y, \rho)$:

$$\begin{array}{ll} & (f : \mathbb{C}\ (X, Y)) \\ \times & (\mathit{comm} : (x : F\ X) \to G\ (f, \mathit{refl})\ (\theta\ x) = \rho\ (F\ f\ x)) \end{array}$$

Coherence

- We can use $\mathbb{C}$-containers to internalise the theory
- We still need to be able to talk about the categories of dependent dialgebras

Design choices:

- Define the categories with *strict* equality and possibly lose some expressivity (e.g. the torus)
- Work with appropriately defined $(\infty, 1)$-categories and deal with the coherence issues, that increase with the number of constructors

Conclusions

- Higher inductive types are sequences of dependent dialgebras
- $\mathbb{C}$-containers allow us to formalise the data needed to define constructors
- A simplified approach to 1-HITs has been successfully formalised
- Coherence problems increase with the number of constructors
- We can work in a type theory with strict equality and avoid coherence problems but we lose some expressivity