Universes of data types in constructive type theory

Lecture 3: Inductive families, and inductive-recursive definitions

Fredrik Nordvall Forsberg

University of Strathclyde
https://fredriknf.com/pc22/

Proof and Computation 2022 Autumn School, Fischbachau

Inductive families

We have seen how to describe inductive types such as $\mathbb{N}$ and lists using a universe of data type codes.

What about predicates such as $\mathsf{Even} : \mathbb{N} \to \mathsf{Type}$?

Formation

$$\frac{n : \mathbb{N}}{\mathsf{Even}\ n\ \mathsf{type}}$$

Introduction

$$\frac{}{\mathsf{ez} : \mathsf{Even}\ 0} \qquad \frac{n : \mathbb{N} \qquad e : \mathsf{Even}\ n}{\mathsf{ess}\ n\ p : \mathsf{Even}\ (\mathsf{suc}\ (\mathsf{suc}\ n))}$$

Inductive families

We have seen how to describe inductive types such as $\mathbb{N}$ and lists using a universe of data type codes.

What about predicates such as $\mathsf{Even} : \mathbb{N} \to \mathsf{Type}$?

Formation

$$\frac{n : \mathbb{N}}{\mathsf{Even}\; n\ \mathsf{type}}$$

Introduction

$$\frac{}{\mathsf{ez} : \mathsf{Even}\, 0} \qquad \frac{n : \mathbb{N} \qquad e : \mathsf{Even}\; n}{\mathsf{ess}\; n\; p : \mathsf{Even}\, (\mathsf{suc}\, (\mathsf{suc}\; n))}$$

Remark: Not the case that each Even n is self-contained inductive definition — pedantic difference between family of inductive types and inductive family of types. (Dybjer [1994])

Another example: canonical finite types

Inductive family Fin : $\mathbb{N} \to$ Type, where Fin n has exactly n elements.

Given by two constructors

$$\mathsf{fz} : (n : \mathbb{N}) \to \mathsf{Fin}\,(\mathsf{suc}\; n)$$
$$\mathsf{fs} : (n : \mathbb{N}) \to \mathsf{Fin}\; n \to \mathsf{Fin}\,(\mathsf{suc}\; n)$$

Fin 0	Fin 1	Fin 2	Fin 3	Fin 4	Fin 5	Fin 6
	fz	fz	fz	fz	fz	fz
		fs fz	fs fz	fs fz	fs fz	fs fz
			fs fs fz	fs fs fz	fs fs fz	fs fs fz
				fs fs fs fz	fs fs fs fz	fs fs fs fz
					fs fs fs fs fz	fs fs fs fs fz
						fs fs fs fs fs fz

Describing inductive families

Let us describe inductive families $I \to \text{Type}$.

Compared to describing inductive types, not much changes — all data still contained in types of constructors.

- We need to record the index of the constructed element.
- We need to record the index of inductive arguments.

Describing inductive families

Let us describe inductive families $I \to \text{Type}$.

Compared to describing inductive types, not much changes — all data still contained in types of constructors.

- We need to record the index of the constructed element.
- We need to record the index of inductive arguments.

Formation

$$\frac{}{\text{SP type}}$$

Introduction

$$\frac{}{\text{done} : \text{SP}}$$

$$\frac{A : \text{Type} \qquad \gamma : A \to \text{SP}}{\text{nonind}\, A\, \gamma : \text{SP}} \qquad \frac{A : \text{Type} \qquad \gamma : \text{SP}}{\text{ind}\, A\, \gamma : \text{SP}}$$

Describing inductive families

Let us describe inductive families $I \to \mathsf{Type}$.

Compared to describing inductive types, not much changes — all data still contained in types of constructors.

- We need to record the index of the constructed element.
- We need to record the index of inductive arguments.

Formation

$$\frac{I \ \text{type}}{\mathsf{SPF}\, I \ \text{type}}$$

Introduction

$$\frac{}{\mathsf{done} : \mathsf{SPF}\, I}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SPF}\, I}{\mathsf{nonind}\, A\, \gamma : \mathsf{SPF}\, I} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SPF}\, I}{\mathsf{ind}\, A\, \gamma : \mathsf{SPF}\, I}$$

Describing inductive families

Let us describe inductive families $I \to \mathsf{Type}$.

Compared to describing inductive types, not much changes — all data still contained in types of constructors.

- We need to record the index of the constructed element.
- We need to record the index of inductive arguments.

Formation

$$\frac{I \ \mathsf{type}}{\mathsf{SPF}\, I \ \mathsf{type}}$$

Introduction

$$\frac{i : I}{\mathsf{done} : \mathsf{SPF}\, I}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SPF}\, I}{\mathsf{nonind}\, A\, \gamma : \mathsf{SPF}\, I} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SPF}\, I}{\mathsf{ind}\, A\, \gamma : \mathsf{SPF}\, I}$$

Describing inductive families

Let us describe inductive families $I \to \mathsf{Type}$.

Compared to describing inductive types, not much changes — all data still contained in types of constructors.

- We need to record the index of the constructed element.
- We need to record the index of inductive arguments.

Formation

$$\frac{I \text{ type}}{\mathsf{SPF}\, I \text{ type}}$$

Introduction

$$\frac{i : I}{\mathsf{done} : \mathsf{SPF}\, I}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SPF}\, I}{\mathsf{nonind}\, A\, \gamma : \mathsf{SPF}\, I} \qquad \frac{A : \mathsf{Type} \qquad f : A \to I \qquad \gamma : \mathsf{SPF}\, I}{\mathsf{ind}\, A\, \gamma : \mathsf{SPF}\, I}$$

Arg and D_γ

The type of the decoding changes to

$$\mathsf{Arg} : \mathsf{SPF}\, I \to (I \to \mathsf{Type}) \to (I \to \mathsf{Type})$$

Arg and D_γ

The type of the decoding changes to

$$\mathrm{Arg} : \mathrm{SPF}\, I \to (I \to \mathrm{Type}) \to (I \to \mathrm{Type})$$

$$\begin{aligned}
\mathit{Arg}\,(\mathrm{done}\, i)\, X\, j &\equiv (i =_I j) \\
\mathit{Arg}\,(\mathrm{nonind}\, A\, \gamma)\, X\, j &\equiv (a : A) \times (\mathrm{Arg}\,(\gamma a)\, X\, j) \\
\mathit{Arg}\,(\mathrm{ind}\, A\, f\, \gamma)\, X\, j &\equiv ((a : A) \to X\,(f\ a)) \times (\mathit{Arg}\,\gamma\, X\, j)
\end{aligned}$$

Arg and D_γ

The type of the decoding changes to

$$\text{Arg} : \text{SPF}\, I \to (I \to \text{Type}) \to (I \to \text{Type})$$

$$
\begin{aligned}
Arg\,(\text{done}\; i)\, X\, j &\equiv (i =_I j) \\
Arg\,(\text{nonind}\; A\, \gamma)\, X\, j &\equiv (a : A) \times (\text{Arg}\,(\gamma a)\, X\, j) \\
Arg\,(\text{ind}\; A\, f\, \gamma)\, X\, j &\equiv ((a : A) \to X\,(f\, a)) \times (Arg\, \gamma\, X\, j)
\end{aligned}
$$

Arg and D_γ

The type of the decoding changes to

$$\text{Arg} : \text{SPF}\, I \to (I \to \text{Type}) \to (I \to \text{Type})$$

$$\begin{aligned} Arg\,(\text{done}\; i)\, X\, j &\equiv (i =_I j) \\ Arg\,(\text{nonind}\; A\, \gamma)\, X\, j &\equiv (a : A) \times (\text{Arg}\,(\gamma a)\, X\, j) \\ Arg\,(\text{ind}\; A\, f\, \gamma)\, X\, j &\equiv ((a : A) \to X\,(f\; a)) \times (Arg\, \gamma\, X\, j) \end{aligned}$$

Generic rules:

Formation

$$\frac{\gamma : \text{SPF}\, I}{D_\gamma : I \to \text{Type}}$$

Introduction

$$\frac{i : I \qquad x : \text{Arg}\, \gamma\, D_\gamma\; i}{\text{intro}_\gamma\, x : D_\gamma\; i}$$

Example: code for $\mathsf{Fin} : \mathbb{N} \to \mathsf{Type}$

Recall Fin was given by constructors

$$\mathsf{fz} : (n : \mathbb{N}) \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$
$$\mathsf{fs} : (n : \mathbb{N}) \to \mathsf{Fin}\,n \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$

Example: code for Fin : $\mathbb{N} \to$ Type

Recall Fin was given by constructors

$$\mathsf{fz} : (n : \mathbb{N}) \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$
$$\mathsf{fs} : (n : \mathbb{N}) \to \mathsf{Fin}\,n \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$

Again we can combine two codes into one with

$$\gamma +_{\mathsf{SPF}} \psi :\equiv \mathsf{nonind}(\mathsf{Bool}, \lambda x\,\mathsf{if}\,x\,\mathsf{then}\,\gamma\,\mathsf{else}\,\psi)$$

Example: code for Fin : $\mathbb{N} \to$ Type

Recall Fin was given by constructors

$$\mathsf{fz} : (n : \mathbb{N}) \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$
$$\mathsf{fs} : (n : \mathbb{N}) \to \mathsf{Fin}\,n \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$

Again we can combine two codes into one with

$$\gamma +_{\mathsf{SPF}} \psi :\equiv \mathsf{nonind}(\mathsf{Bool}, \lambda x\, \mathsf{if}\ x\ \mathsf{then}\ \gamma\ \mathsf{else}\ \psi)$$

and give a code for Fin as

$$\gamma_{\mathsf{Fin}} :\equiv (\mathsf{nonind}\,\mathbb{N}\,(\lambda\, n.\mathsf{done}\,(\mathsf{suc}\,n))) +_{\mathsf{SPF}}$$
$$(\mathsf{nonind}\,\mathbb{N}\,(\lambda\, n.\mathsf{ind}\,\mathbf{1}\,(\lambda\_.n)\,\mathsf{done}\,(\mathsf{suc}\,n)))$$

Example: code for Fin : $\mathbb{N} \to$ Type

Recall Fin was given by constructors

$$\mathsf{fz} : (n : \mathbb{N}) \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$
$$\mathsf{fs} : (n : \mathbb{N}) \to \mathsf{Fin}\,n \to \mathsf{Fin}\,(\mathsf{suc}\,n)$$

Again we can combine two codes into one with

$$\gamma +_{\mathsf{SPF}} \psi :\equiv \mathsf{nonind}(\mathsf{Bool}, \lambda x\,\mathsf{if}\,x\,\mathsf{then}\,\gamma\,\mathsf{else}\,\psi)$$

and give a code for Fin as

$$\gamma_{\mathsf{Fin}} :\equiv (\mathsf{nonind}\,\mathbb{N}\,(\lambda\,n.\mathsf{done}\,(\mathsf{suc}\,n))) +_{\mathsf{SPF}} (\mathsf{nonind}\,\mathbb{N}\,(\lambda\,n.\mathsf{ind}\,\mathbf{1}\,(\lambda\_.n)\,\mathsf{done}\,(\mathsf{suc}\,n)))$$

Remark: We can equivalently "factor out" the common code 'nonind $\mathbb{N}$':

$$\gamma'_{\mathsf{Fin}} :\equiv \mathsf{nonind}\,\mathbb{N}\,(\lambda\,n.\big((\mathsf{done}\,(\mathsf{suc}\,n)) +_{\mathsf{SPF}} \mathsf{ind}\,\mathbf{1}\,(\lambda\_.n)\,\mathsf{done}\,(\mathsf{suc}\,n)\big))$$

which is a description divirging from "a finite list of constructors".

Induction versus recursion

Note that because $\mathbb{N}$ is an inductive type, we can also define $\mathsf{Fin} : \mathbb{N} \to \mathsf{Type}$ by *recursion* (using large elimination).

$$\begin{aligned} \mathsf{Fin}\,0 &:\equiv \mathbf{0} \\ \mathsf{Fin}\,(\mathsf{suc}\,n) &:\equiv \mathbf{1} + \mathsf{Fin}\,n \end{aligned}$$

Induction versus recursion

Note that because $\mathbb{N}$ is an inductive type, we can also define $\mathsf{Fin} : \mathbb{N} \to \mathsf{Type}$ by *recursion* (using large elimination).

$$\begin{aligned} \mathsf{Fin}\,0 &:\equiv \mathbf{0} \\ \mathsf{Fin}\,(\mathsf{suc}\,n) &:\equiv \mathbf{1} + \mathsf{Fin}\,n \end{aligned}$$

Inductive definition: given by constructors.

Recursive definition: function defined on all constructors.

Induction versus recursion

Note that because $\mathbb{N}$ is an inductive type, we can also define $\mathsf{Fin} : \mathbb{N} \to \mathsf{Type}$ by *recursion* (using large elimination).

$$\begin{aligned} \mathsf{Fin}\,0 &:\equiv \mathbf{0} \\ \mathsf{Fin}\,(\mathsf{suc}\,n) &:\equiv \mathbf{1} + \mathsf{Fin}\,n \end{aligned}$$

Inductive definition: given by constructors.

Recursive definition: function defined on all constructors.

It can make sense to define $D : A \to \mathsf{Type}$ inductively even if A is not an inductive type.

Soundness

Compared to inductive types, inductive families do not really add any proof-theoretical strength to type theory.

This is also reflected in the naive model construction, which basically stays the same.

Denormalised finite types

A finite type is isomorphic to Fin n for some n, but might have more structure, e.g.

$$(d : Weekday) \to Hours[d] \times Minutes[d] \times Seconds[d]$$

As an exercise, can we describe finite types with their structure intact?

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\text{FinType type}}$$

Introduction

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\mathsf{FinType\ type}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\mathsf{FinType\ type}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}}$$

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\mathsf{FinType\ type}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\mathsf{FinType\ type}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : ??? \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : ??? \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

An attempt to describe finite types with popular operations

Formation

$$\frac{}{\mathsf{FinType\ type}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\ ?? \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\ ?? \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType\ type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType\ type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\ ?? \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\ ?? \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType}\ \mathsf{type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\,a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\,a) \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType\ type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}} \qquad \mathsf{size}\,(\mathsf{fin}\ n) \equiv n$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\, a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\, a) \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType}\ \mathsf{type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}} \qquad \mathsf{size}\,(\mathsf{fin}\ n) \equiv n$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \mathsf{size}\,(a \times_f b) \equiv \mathsf{size}\ a \cdot \mathsf{size}\ b$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType}\ \mathsf{type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}} \qquad \mathsf{size}\,(\mathsf{fin}\ n) \equiv n$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \mathsf{size}\,(a \times_f b) \equiv \mathsf{size}\ a \cdot \mathsf{size}\ b$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}} \qquad \mathsf{size}\,(a +_f b) \equiv \mathsf{size}\ a + \mathsf{size}\ b$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}}$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}}$$

The size of a finite type

We need to simultaneously compute the *size* of finite types!

Formation

$$\frac{}{\mathsf{FinType\ type}} \qquad \frac{}{\mathsf{size} : \mathsf{FinType} \to \mathbb{N}}$$

Introduction

$$\frac{n : \mathbb{N}}{\mathsf{fin}\ n : \mathsf{FinType}} \qquad \mathsf{size}\,(\mathsf{fin}\ n) \equiv n$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a \times_f b : \mathsf{FinType}} \qquad \mathsf{size}\,(a \times_f b) \equiv \mathsf{size}\ a \cdot \mathsf{size}\ b$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{FinType}}{a +_f b : \mathsf{FinType}} \qquad \mathsf{size}\,(a +_f b) \equiv \mathsf{size}\ a + \mathsf{size}\ b$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}} \qquad \mathsf{size}\,(\Sigma_a b) \equiv \mathsf{sum}\,(\mathsf{size}\ a)(\mathsf{size} \circ b)$$

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\ a) \to \mathsf{FinType}}{\Pi_a b : \mathsf{FinType}} \qquad \mathsf{size}\,(\Pi_a b) \equiv \mathsf{prod}\,(\mathsf{size}\ a)(\mathsf{size} \circ b)$$

What happened?

We defined FinType inductively, and at “the same time” we defined size : FinType $\to \mathbb{N}$ recursively.

What happened?

We defined FinType inductively, and at "the same time" we defined size : FinType $\to \mathbb{N}$ recursively.

This is what Dybjer [2000] calls an *inductive-recursive definition*.

What happened?

We defined FinType inductively, and at “the same time” we defined size : FinType $\to \mathbb{N}$ recursively.

This is what Dybjer [2000] calls an *inductive-recursive definition*.

Typical use case: construct data and its interpretation at the same time.

What happened?

We defined FinType inductively, and at "the same time" we defined size : FinType $\to \mathbb{N}$ recursively.

This is what Dybjer [2000] calls an *inductive-recursive definition*.

Typical use case: construct data and its interpretation at the same time.

For example: a universe and its decoding.

$$\frac{a : U \qquad b : T(a) \to U}{\sigma\, a\, b : U} \qquad T\,(\sigma\, a\, b) \equiv (x :\, T\, a) \times \big(T\,(b\, x)\big)$$

$$\vdots$$

Induction-recursion allows you to construct your own bespoke universes of types.

Acting on families

Inductive families had a fixed index set I; they are initial algebras of functors $(I \to \mathsf{Type}) \to (I \to \mathsf{Type})$.

Inductive-recursive definitions on the other hand also generate the index set, which is not fixed; they are initial algebras of functors $\mathsf{Fam}\, D \to \mathsf{Fam}\, D$ for some (possibly large) type D.

$$\mathsf{Fam}\, D :\equiv (I : \mathsf{Type}) \times (I \to D)$$

Acting on families

Inductive families had a fixed index set I; they are initial algebras of functors $(I \to \mathsf{Type}) \to (I \to \mathsf{Type})$.

Inductive-recursive definitions on the other hand also generate the index set, which is not fixed; they are initial algebras of functors $\mathsf{Fam}\, D \to \mathsf{Fam}\, D$ for some (possibly large) type D.

$$\mathsf{Fam}\, D :\equiv (I : \mathsf{Type}) \times (I \to D)$$

Lemma: When D is small, there is an equivalence

$$\mathsf{powfam} : (D \to \mathsf{Type}) \cong \mathsf{Fam}\, D$$

with

$$\mathsf{powfam}\, P \equiv ((d : D) \times P[d], \mathsf{fst})$$
$$\mathsf{fampow}\, (A, Q) \equiv \lambda d.(a : A) \times \big((Q\, a) =_D d\big)$$

(simple version of Grothendieck construction).

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Formation

$$\frac{}{\mathsf{SP}\ \mathsf{type}}$$

Introduction

$$\frac{}{\mathsf{done} : \mathsf{SP}}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SP}}{\mathsf{nonind}\, A\, \gamma : \mathsf{SP}} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SP}}{\mathsf{ind}\, A\, \gamma : \mathsf{SP}}$$

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Formation

$$\frac{D \text{ type}}{\mathsf{IR}\, D \text{ type}}$$

Introduction

$$\frac{}{\mathsf{done} : \mathsf{SP}}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SP}}{\mathsf{nonind}\, A\, \gamma : \mathsf{SP}} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SP}}{\mathsf{ind}\, A\, \gamma : \mathsf{SP}}$$

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Formation

$$\frac{D \text{ type}}{\mathsf{IR}\, D \text{ type}}$$

Introduction

$$\frac{d : D}{\mathsf{done} : \mathsf{IR}\, D}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{SP}}{\mathsf{nonind}\, A\, \gamma : \mathsf{SP}} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SP}}{\mathsf{ind}\, A\, \gamma : \mathsf{SP}}$$

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Formation

$$\frac{D \text{ type}}{\mathsf{IR}\, D \text{ type}}$$

Introduction

$$\frac{d : D}{\mathsf{done} : \mathsf{IR}\, D}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{IR}\, D}{\mathsf{nonind}\, A\, \gamma : \mathsf{IR}\, D} \qquad \frac{A : \mathsf{Type} \qquad \gamma : \mathsf{SP}}{\mathsf{ind}\, A\, \gamma : \mathsf{SP}}$$

Describing inductive-recursive definitions

Following Dybjer and Setzer [1999, 2003], we again start from the universe SP.

- We need to record what the decoding of the constructed element is.
- Later arguments may depend on *the decoding* of inductive arguments.

Formation

$$\frac{D \text{ type}}{\mathsf{IR}\, D \text{ type}}$$

Introduction

$$\frac{d : D}{\mathsf{done} : \mathsf{IR}\, D}$$

$$\frac{A : \mathsf{Type} \qquad \gamma : A \to \mathsf{IR}\, D}{\mathsf{nonind}\, A\, \gamma : \mathsf{IR}\, D} \qquad \frac{A : \mathsf{Type} \qquad \gamma : (A \to D) \to \mathsf{IR}\, D}{\mathsf{ind}\, A\, \gamma : \mathsf{IR}\, D}$$

Arg and D_γ

The type of the decoding changes to

$$\mathsf{Arg} : \mathsf{IR}\, D \to \mathsf{Fam}\, D \to \mathsf{Fam}\, D$$

Arg and D_γ

The type of the decoding changes to

$$\mathsf{Arg} : \mathsf{IR}\, D \to \mathsf{Fam}\, D \to \mathsf{Fam}\, D$$

$$\begin{aligned}
Arg\,(\mathsf{done}\, d)\,(U, T) &\equiv (\mathbf{1}, \lambda_.d) \\
Arg\,(\mathsf{nonind}\, A\, \gamma)\,(U, T) &\equiv \Sigma_{a:A}(\mathsf{Arg}\,(\gamma a)\,(U, T)) \\
Arg\,(\mathsf{ind}\, A\, \gamma)\,(U, T) &\equiv \Sigma_{g:A \to U}(Arg\,(\gamma\,(T \circ g))\,(U, T))
\end{aligned}$$

Arg and D_γ

The type of the decoding changes to

$$\text{Arg} : \text{IR}\, D \to \text{Fam}\, D \to \text{Fam}\, D$$

$$
\begin{aligned}
Arg\,(\text{done}\, d)\,(U, T) &\equiv (\mathbf{1}, \lambda\_.d) \\
Arg\,(\text{nonind}\, A\, \gamma)\,(U, T) &\equiv \Sigma_{a:A}(\text{Arg}\,(\gamma a)\,(U, T)) \\
Arg\,(\text{ind}\, A\, \gamma)\,(U, T) &\equiv \Sigma_{g:A \to U}(Arg\,(\gamma\,(T \circ g))\,(U, T))
\end{aligned}
$$

Generic rules:

Formation

$$\frac{\gamma : \text{IR}\, D}{U_\gamma \text{ type}} \qquad T_\gamma : U_\gamma \to D$$

Introduction

$$\frac{x : \text{fst}(\text{Arg}\,\gamma\,(U_\gamma, T_\gamma))}{\text{intro}_\gamma\, x : D_\gamma\, i} \qquad T_\gamma\,(\text{intro}_\gamma\, x) \equiv \text{snd}(\text{Arg}\,\gamma\,(U_\gamma, T_\gamma))\, x$$

Soundness of inductive-recursive definitions

Again the rules can be justified using a set-theoretical model.

Soundness of inductive-recursive definitions

Again the rules can be justified using a set-theoretical model.

However this time it is much harder to prove that least fixed points exist — uses large cardinal assumption that a Mahlo cardinal exists.

(**Def:** An Inaccessible cardinal M is Mahlo is every normal function $M \to M$ has an inaccessible fixed point.)

Soundness of inductive-recursive definitions

Again the rules can be justified using a set-theoretical model.

However this time it is much harder to prove that least fixed points exist — uses large cardinal assumption that a Mahlo cardinal exists.

(**Def:** An Inaccessible cardinal M is Mahlo is every normal function $M \to M$ has an inaccessible fixed point.)

Using a Mahlo cardinal makes some sense, because “Mahlo universes” can be constructed using induction-recursion.

Reducing *small* induction-recursion to inductive families

The situation re proof-theoretic strength is completely different if D is a small set; all the power of induction-recursion lies in the ability to represent large things as small things.

Reducing *small* induction-recursion to inductive families

The situation re proof-theoretic strength is completely different if D is a small set; all the power of induction-recursion lies in the ability to represent large things as small things.

To be precise: Small induction-recursion can be reduced to inductive families, which do not add proof-theoretical strength beyond inductive types [Hancock, McBride, Ghani, Malatesta and Altenkirch, 2013].

Reducing *small* induction-recursion to inductive families

The situation re proof-theoretic strength is completely different if D is a small set; all the power of induction-recursion lies in the ability to represent large things as small things.

To be precise: Small induction-recursion can be reduced to inductive families, which do not add proof-theoretical strength beyond inductive types [Hancock, McBride, Ghani, Malatesta and Altenkirch, 2013].

To be even more precise: for small D, we can define

$$\mathsf{translate} : \mathsf{IR}\, D \to \mathsf{SPF}\, D$$

such that

$$\begin{array}{ccc} \mathsf{Fam}\,D & \xrightarrow{\mathsf{Arg}_{\mathsf{IR}}\,\gamma} & \mathsf{Fam}\,D \\ \mathsf{powfam}\uparrow\;\downarrow\mathsf{fampow} & & \mathsf{powfam}\uparrow\;\downarrow\mathsf{fampow} \\ (D \to \mathsf{Type}) & \xrightarrow[\mathsf{Arg}_{\mathsf{SPF}}\,(\mathsf{translate}\,\gamma)]{} & (D \to \mathsf{Type}) \end{array}$$

Idea of translation

Main idea: Make all inductive arguments display their decoding in their index.

Quantify over new non-inductive arguments to represent these indices ($\rightsquigarrow$ smallness assumption).

Idea of translation

Main idea: Make all inductive arguments display their decoding in their index.

Quantify over new non-inductive arguments to represent these indices ($\rightsquigarrow$ smallness assumption).

For example,

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\; a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}} \qquad \mathsf{size}\,(\Sigma_a b) \equiv \mathsf{sum}\,(\mathsf{size}\; a)(\mathsf{size} \circ b)$$

becomes

$$\frac{n : \mathbb{N} \quad a : \mathsf{FinType}'\, n \quad m : \mathsf{Fin}\, n \to \mathbb{N} \quad b : (x : \mathsf{Fin}\, n) \to \mathsf{FinType}'\,(m\, x)}{\sigma\; n\; a\; m\; b : \mathsf{FinType}'\,(\mathsf{sum}\; n\; m)}$$

Idea of translation

Main idea: Make all inductive arguments display their decoding in their index.

Quantify over new non-inductive arguments to represent these indices ($\rightsquigarrow$ smallness assumption).

For example,

$$\frac{a : \mathsf{FinType} \qquad b : \mathsf{Fin}\,(\mathsf{size}\,a) \to \mathsf{FinType}}{\Sigma_a b : \mathsf{FinType}} \qquad \mathsf{size}\,(\Sigma_a b) \equiv \mathsf{sum}\,(\mathsf{size}\,a)(\mathsf{size} \circ b)$$

becomes

$$\frac{n : \mathbb{N} \quad a : \mathsf{FinType}'\,n \quad m : \mathsf{Fin}\,n \to \mathbb{N} \quad b : (x : \mathsf{Fin}\,n) \to \mathsf{FinType}'\,(m\,x)}{\sigma\,n\,a\,m\,b : \mathsf{FinType}'\,(\mathsf{sum}\,n\,m)}$$

Using powfam, we then define $[\![\mathsf{FinType}]\!] \equiv (n : \mathbb{N}) \times \mathsf{FinType}'\,n$ and $[\![\mathsf{size}]\!] = \mathsf{fst}$.

Summary

Variations on the universe SP of data type descriptions can also describe inductive families and inductive-recursive definitions.

The latter increases the strength of the theory immensely.

However small inductive-recursive definitions can be reduced to mere inductive families.

This reduction can be carried out internally by translating codes for IR into codes for SPF and proving that their meaning is preserved.

Many topics not covered, e.g.:

- Higher inductive types
- Inductive-inductive definitions (cf. recent work by Kovács and Kaposi)
- Models of induction-recursion in *constructive* set theories
- Coinductive definitions