Inductive-Recursive Definitions and Generic Programming

Anton Setzer
Swansea University

1. What is Generic Programming?
2. Universes and Generic Programming
3. Inductive-Recursive Definitions

Notations

- Set = “small types”.
- $$(x : A, y : B) \to C$$

  is type of functions,
  - mapping $x : A$, $y : B$ to C,
  - where B might depend on x,
  - C might depend on x, y.
  - Sometimes written as $\Pi x : A.\Pi y : B.C$.
- $(x : A) \times B$ = dependent product (B might depend on A).
  - Sometimes written as $\Sigma x : A.B$.

1. What is Generic Programming?

- Two different notions:
  - In **object-oriented programming**: higher type polymorphism.
    - E.g. to form $\mathrm{List}(A)$ depending on $A : \mathrm{Set}$.
  - In **functional programming**:
    - Definition of functions by induction on the buildup of types.
    - **Notion used here**.
  - Notion of **Generative Programming**
    - Idea of automatic generation of programs.
    - “Never write the same code twice”.

2. Universes

- Universes in type theory = general framework for generic programming.
- **Universe** given by
  - a **set of codes** for sets

$$\mathrm{U} : \mathrm{Set}$$

  - a **decoding function**

$$\mathrm{T} : \mathrm{U} \to \mathrm{Set}$$

Examples

1.

$$\mathrm{U} = \mathbb{N} \ , \qquad \mathrm{T}(n) := \mathbb{N}^n$$

2.

$$\begin{aligned}
\mathrm{data\ U} \quad & = \quad \widehat{\mathbb{N}} \ \mid \ (\widehat{\rightarrow})\ (a, b : \mathrm{U}) \ \mid \ (\widehat{\times})\ (a, b : \mathrm{U}) \ , \\
\mathrm{T}\ \widehat{\mathbb{N}} \quad & = \quad \mathbb{N} \\
\mathrm{T}\ (a\ \widehat{\rightarrow}\ b) \quad & = \quad \mathrm{T}\ a \rightarrow \mathrm{T}\ b \\
\mathrm{T}\ (a\ \widehat{\times}\ b) \quad & = \quad \mathrm{T}\ a \times \mathrm{T}\ b
\end{aligned}$$

Universes and Generic Programm.

- Universes allow to define generic functions
  - Assume universes $(\mathrm{U}_0, \mathrm{T}_0)$, $(\mathrm{U}_1, \mathrm{T}_1)$ and $A : \mathrm{Set}$.
  - A **generic function** can be given as a function

$$f \;:\; \mathrm{U}_0 \to A$$

or

$$f \;:\; \mathrm{U}_0 \to \mathrm{U}_1$$

or

$$f \;:\; (u : \mathrm{U}_0) \to \mathrm{T}_0\ u \to ((u' : \mathrm{U}_1) \times \mathrm{T}_1\ u')$$

or

$$\ldots$$

- If U_0 is inductively defined, f can be defined by induction on U_0.

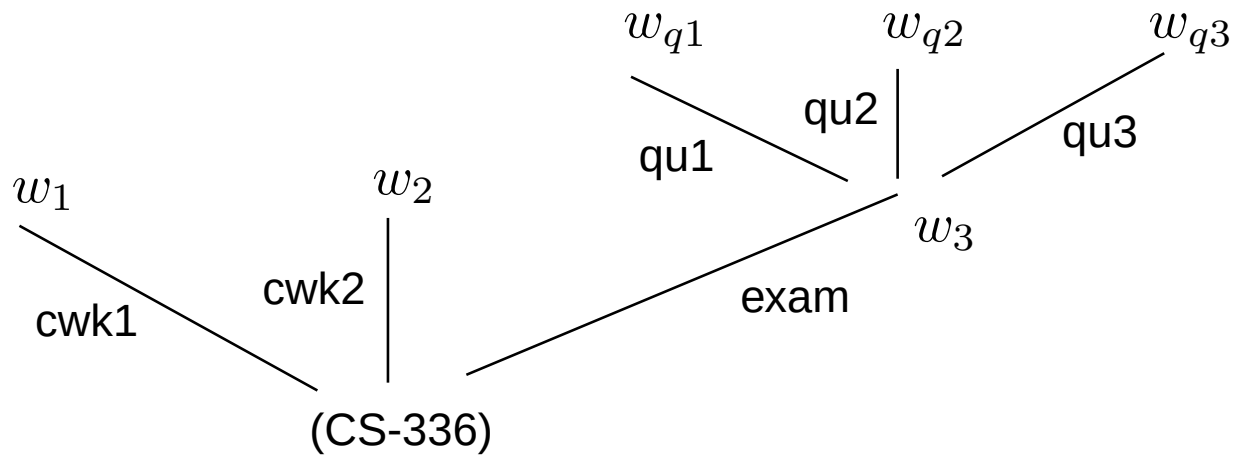
Example Module Structure

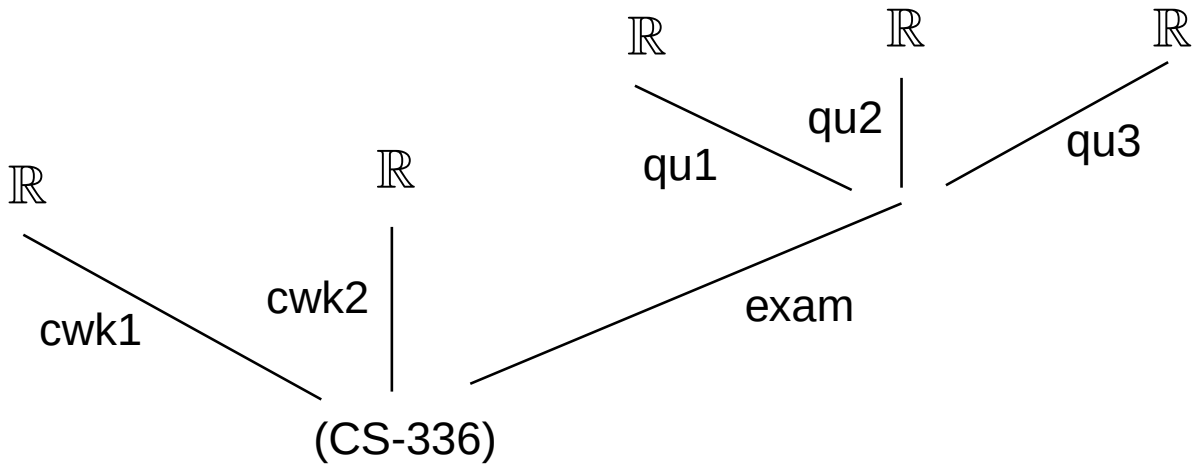
$$\text{ModStruct} \quad = \quad \text{data ms } (l : \text{List } ((weight : \mathbb{R}) \times \text{ModStruct}))$$

$$\text{Marks} : \text{ModStruct} \to \text{Set}$$
$$\text{Marks } (\text{ms } [\,]) \quad = \quad \mathbb{R}$$
$$\text{Marks } (\text{ms } [\langle w_0, l_0 \rangle, \ldots, \langle w_n, l_n \rangle]) \quad = \quad \text{Marks } l_0 \times \cdots \times \text{Marks } l_n$$

$$\text{result} : (s : \text{ModStruct}, \text{Marks } s) \to \mathbb{R}$$
$$\text{result } (\text{ms } [\,])\ r \quad = \quad r$$
$$\text{result } (\text{ms } [\langle w_0, l_0 \rangle, \ldots, \langle w_n, l_n \rangle])\ \langle m_0, \ldots, m_n \rangle \quad = \quad \frac{\text{result } l_0\ m_0 * w_0 + \cdots + \text{result } l_n\ m_n * w_n}{w_0 + \cdots w_n}$$

Module Structure





Example Module Structure

- In order to define a show function, one needs to modify

$$\mathrm{ModStruct} = \mathrm{data}\,\mathrm{ms}\;(name : \mathrm{String})\\ (l : \mathrm{List}\;((weight : \mathbb{R}) \times \mathrm{ModStruct}))\;\;.$$

- $\Rightarrow$ we don't have "the generic data structure"
  instead there are lots of generic data structures

Gen. Program. on Ind. Data Types

- We want to carry out generic programming on the set of **inductively defined data types**.
  - E.g. the **show** function, **equality** on data types, the **zipper** etc.
- Inductively defined sets are **initial algebras for strictly positive functors**.
- Use of **universes of operators**

$$\mathrm{U} : \mathrm{Set} \qquad \mathrm{F} : \mathrm{U} \rightarrow \mathrm{Set} \rightarrow \mathrm{Set}$$

Generic Haskell

- One version of **Generic Haskell** corresponds to the following universe:
  (here $\mathrm{U}_0, \mathrm{T}_0 : \mathrm{U}_0 \to \mathrm{Set}$ is a universe of basic sets):

$$
\begin{aligned}
\mathrm{data\ U} &= \mathrm{id} \mid \mathrm{K}\ (a : \mathrm{U}_0) \mid \mathrm{U} \mathbin{\widehat{+}} \mathrm{U} \mid \mathrm{U} \mathbin{\widehat{\times}} \mathrm{U} \\
&\quad \mid \mathrm{is\_of}\ (s : \mathrm{String})\ (u : \mathrm{U}) \\
\mathrm{F}_{\mathrm{id}} &= \lambda X.X \\
\mathrm{F}_{\mathrm{K}\ a} &= \lambda X.\mathrm{T}_0\ a \\
\mathrm{F}_{a \mathbin{\widehat{+}} b} &= \lambda X.\mathrm{F}_a\ X + \mathrm{F}_b\ X \\
\mathrm{F}_{(a \mathbin{\widehat{\times}} b)} &= \lambda X.\mathrm{F}_a\ X \times \mathrm{F}_b\ X \\
\mathrm{F}_{\mathrm{is\_of}\ s\ u} &= \lambda X.\mathrm{F}_u\ X
\end{aligned}
$$

- Then define for $f : \mathrm{U}$
  $\mathrm{A}_f = \mathrm{data\ intro}_f\ (x : \mathrm{F}_f\ \mathrm{A}_f)$

Generic Haskell

- Examples in generic Haskell (e.g. **zipper**) can now easily be tranformed into generic functions in dependent type theory.
  - Amounts to defining generic functions on the universe U of universe operators.

Dependent Inductive Definitions

- **Dependent inductive definitions** can be defined (referring to a basic universe of sets $\mathrm{U}_0, \mathrm{T}_0$) by defining

$$\mathrm{U} : \mathrm{Set} \quad \mathrm{F} : \mathrm{U} \to \mathrm{Set} \to \mathrm{Set}$$

with the following elements:

- $\iota : \mathrm{U}, \quad \mathrm{F}_\iota\ X = \{*\}$
  - Let $f := \iota$. Then

$$\mathrm{intro}_f : \{*\} \to \mathrm{A}_f$$

Corresponds to the constructor having essentially **no argument**.

Dependent Inductive Definitions

- $\sigma : (a : \mathrm{U}_0, g : \mathrm{T}_0\ a \to \mathrm{U}) \to \mathrm{U}$

  $\mathrm{F}_{\sigma\ a\ g}\ X = (x : \mathrm{T}_0\ a) \times \mathrm{F}_{g\ x}\ X$

  - Let $f := \sigma\ a\ g$. Then

$$\mathrm{intro}_f : ((x : \mathrm{T}_0\ a) \times (\mathrm{F}_{g\ x}\ \mathrm{A}_f)) \to \mathrm{A}_f$$

Corresponds to the constructor having **one non-inductive argument**, and **later arguments depending on it**.

Dependent Inductive Definitions

- $\delta : (a : \mathrm{U}_0, u : \mathrm{U}) \to \mathrm{U}$

  $\mathrm{F}_{\delta\ a\ u}\ X = (\mathrm{T}_0\ a \to X) \times \mathrm{F}_u\ X$

  - Let $f := \delta\ a\ u$. Then

    $$\mathrm{intro}_f : ((\mathrm{T}_0\ a \to \mathrm{A}_f) \times \mathrm{F}_u\ \mathrm{A}_f) \to \mathrm{A}_f$$

    Corresponds to the constructor having one **inductive argument**, and **later arguments given by** u.

- Variants for this have been used by Benke/Dybjer/Jansson e.g. for defining decidable equality on finitary data types and showing in type theory that this is an equivalence relation.

3. Inductive-Recursive Definitions

- General theory of universes

$$\mathrm{U} : \mathrm{Set} \qquad \mathrm{T} : \mathrm{U} \to D$$

for some type D.

- Example Universe closed under $\times$ has constructor

$$(\widehat{\times}) : (a : \mathrm{U}, b : \mathrm{T}\ a \to \mathrm{U}) \to \mathrm{U}$$
$$\mathrm{T}\ ((\widehat{\times})\ a\ b) = (x : \mathrm{T}\ a) \times \mathrm{T}\ (b\ x)$$

- Amounts to having **initial algebras for endo functors**

$$F : \mathrm{Fam}_D \to \mathrm{Fam}_D$$

where

$$\mathrm{Fam}_D := (X : \mathrm{Set}) \times (X \to D)$$

Generic Program. on Ind.-rec. Def.

- Now one can define a universe of endo functors

$$\mathrm{OP}_D : \mathrm{Type} \qquad \mathrm{F}_D : \mathrm{OP}_D \to \mathrm{Fam}_D \to \mathrm{Fam}_D$$

and define **inductive-recursively** the universes given by OP_D

$$\begin{aligned} \mathrm{U}_D &: \mathrm{OP}_D \to \mathrm{Set} \\ \mathrm{T}_D &: (\gamma : \mathrm{OP}_D, u : \mathrm{U}_{D,\gamma}) \to D \end{aligned}$$

s.t. OP_D contains codes for all ind-rec. definitions.

- Type theory with relatively few rules, which is highly expressive.
- Allows generic programming on inductive-recursive definitions.

Conclusion

- Dependent type theory and universes provide and excellent **framework for generic programming**.
  - **Not much difference to ordinary programming**.
- There is **not only one generic data structure**, but there are many which can be custom built.
- Closed inductive-recursive definitions provides a **highly expressive dependent type theory**.
- Richness of inductive-recursive definitions is still to be discovered by general computer science.
- Literature: Articles with **P. Dybjer** on **inductive-recursive definitions** available from

http://www.cs.swan.ac.uk/∼setzer