What is the problem with Induction-Recursion?

Or: Hank's latest obsession

Thorsten Altenkirch

Functional Programming Laboratory,
School of Computer Science,
University of Nottingham

to Peter Hancock
at his 60th birthday seminar

December 19, 2011

An inductive definition

Rose trees:

$$\begin{array}{l} \textbf{data}\ R : Set\ \textbf{where} \\ \quad leaf : R \\ \quad node : (n : \mathbb{N})\ (f : Fin\ n \to R) \to R \end{array}$$

We can represent R as a functor.

$$\begin{array}{l} F : Set \to Set \\ F\ X = \top \uplus \Sigma\ \mathbb{N}\ (\lambda\ n \to Fin\ n \to X) \end{array}$$

T is the initial algebra of F.

An inductive recursive definition

A universe closed under $\mathbb{N}$ and Π:

$$\textbf{data}\ U : \mathit{Set}$$
$$\mathit{El} : U \to \mathit{Set}$$

$$\textbf{data}\ U\ \textbf{where}$$
$$\quad \mathit{nat} : U$$
$$\quad \pi : (a : U) \to (\mathit{El}\ a \to U) \to U$$
$$\mathit{El}\ \mathit{nat} = \mathbb{N}$$
$$\mathit{El}\ (\pi\ a\ b) = (x : \mathit{El}\ a) \to \mathit{El}\ (b\ x)$$

We also have an initial algebra semantics here.

The category of Families

We define the category of families.
Objects are given as:

$$\begin{array}{l} \textit{record Fam}\ (D : \textit{Set}_1) : \textit{Set}_1\ \textbf{where} \\ \quad U : \textit{Set} \\ \quad T : U \rightarrow D \end{array}$$

and morphisms as:

$$\begin{array}{l} \textit{record Fam}\rightarrow ((U, T)\ (U', T') : \textit{Fam}\ D) : \textit{Set}_1\ \textbf{where} \\ \quad f : U \rightarrow U' \\ \quad \Delta : (x : U) \rightarrow T\ x \equiv T'\ (f\ x) \end{array}$$

Note that this not equivalent to Set/D because D is large!

An Endofunctor on *Fam Set*

Our inductive recursive definition corresponds to an endofunctor on *Fam Set*:

$$
\begin{aligned}
&F_U : \mathit{Fam\ Set} \to \mathit{Set}\\
&F_U\ (U, T) = \top \uplus \Sigma\ U\ (\lambda\ x \to T\ x \to U)\\
&F_T : (UT : \mathit{Fam\ Set}) \to F_U\ UT \to \mathit{Set}\\
&F_T\ (U, T)\ (\mathit{inj}_1\ \mathit{tt}) = \mathbb{N}\\
&F_T\ (U, T)\ (\mathit{inj}_2\ (a, b)) = (x : T\ a) \to T\ (b\ x)\\
&F : \mathit{Fam\ Set} \to \mathit{Fam\ Set}\\
&F\ UT = \mathit{record}\ \{\\
&\quad U = F_U\ UT;\\
&\quad T = F_T\ UT\,\}
\end{aligned}
$$

(U, El) is the initial algebra of F.

Representing inductive definitions

Not every functor defines a data type.

We are only interested in strictly positve inductive definitions.

We can codify inductive definitions as follows:

$$\begin{array}{l}\textbf{data}\ \mathit{ID} : \mathit{Set}_1\ \textbf{where}\\ \quad \iota : \mathit{ID}\\ \quad \sigma : (S : \mathit{Set}) \to (\phi : S \to \mathit{ID}) \to \mathit{ID}\\ \quad \delta : (P : \mathit{Set}) \to (\phi : \mathit{ID}) \to \mathit{ID}\end{array}$$

Each code gives rise to an endofunctor:

$$\begin{array}{ll}[\![\_]\!] : \mathit{ID} \to \mathit{Set} \to \mathit{Set} & \\ [\![\,\iota\,]\!] & X = \top\\ [\![\,\sigma\ S\ \phi\,]\!] & X = \Sigma\ S\ (\lambda\ s \to [\![\phi\ s\,]\!]\ X)\\ [\![\,\delta\ P\ \phi\,]\!] & X = (P \to X) \times [\![\phi\,]\!]\ X\end{array}$$

$$\begin{array}{l}R : \mathit{ID}\\ R = \sigma\ \mathit{Bool}\ (\lambda\ b \to \textbf{if}\ b\ \textbf{then}\ \iota\\ \qquad\qquad\qquad\qquad \textbf{else}\ \sigma\ \mathbb{N}\ (\lambda\ n \to \delta\ (\mathit{Fin}\ n)\ \iota))\end{array}$$

Representing inductive recursive definitions

Following Dybjer/Setzer:

$$
\begin{aligned}
&\textbf{data}\ \mathit{IR}\ (D : \mathit{Set}_1) : \mathit{Set}_1\ \textbf{where}\\
&\quad \iota : D \to \mathit{IR}\ D\\
&\quad \sigma : (S : \mathit{Set}) \to (\phi : S \to \mathit{IR}\ D) \to \mathit{IR}\ D\\
&\quad \delta : (P : \mathit{Set}) \to (\phi : (P \to D) \to \mathit{IR}\ D) \to \mathit{IR}\ D
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{UEl} : \mathit{IR}\ \mathit{Set}\\
&\mathit{UEl} = \sigma\ \mathit{Bool}\ (\lambda\ b \to \textbf{if}\ b\ \textbf{then}\ \iota\ \mathbb{N}\\
&\qquad\qquad\qquad \textbf{else}\ \delta\ \top\ (\lambda\ a \to \delta\ (a\ tt)\\
&\qquad\qquad\qquad\qquad (\lambda\ b \to \iota\ ((x : a\ tt) \to b\ x)
\end{aligned}
$$

Semantics

$$
\begin{aligned}
&[\![\_]\!]_U : \forall\{D\} \to IR\ D \to Fam\ D \to Set\\
&[\![\, \iota\ \_ \,]\!]_U \quad (U, T) = \top\\
&[\![\, \sigma\ S\ \phi \,]\!]_U\ (U, T) = \Sigma\ S\ (\lambda\ s \to [\![\phi\ s\,]\!]_U\ (U, T))\\
&[\![\, \delta\ P\ \phi \,]\!]_U\ (U, T) =\\
&\quad \Sigma\ (P \to U)\ (\lambda\ us \to [\![\phi\ (\lambda\ p \to T\ (us\ p))\,]\!]_U\ (U, T))
\end{aligned}
$$

$$
\begin{aligned}
&[\![\_]\!]_T : \forall\ \{D\} \to (\phi : IR\ D)\ (UT : Fam\ D) \to [\![\phi\,]\!]_U\ UT \to D\\
&[\![\, \iota\ d \,]\!]_T \quad (U, T)\ \_ \qquad = d\\
&[\![\, \sigma\ S\ \phi \,]\!]_T\ (U, T)\ (s, x) \ = [\![\phi\ s\,]\!]_T\ (U, T)\ x\\
&[\![\, \delta\ P\ \phi \,]\!]_T\ (U, T)\ (us, x) = [\![\phi\ (\lambda\ p \to T\ (us\ p))\,]\!]_T\ (U, T)\ x
\end{aligned}
$$

$$
\begin{aligned}
&[\![\_]\!] : \forall\ \{D\} \to IR\ D \to Fam\ D \to Fam\ D\\
&[\![\, \phi \,]\!]\ (U, T) = ([\![\phi\,]\!]_U\ (U, T)), ([\![\phi\,]\!]_T\ (U, T))
\end{aligned}
$$

So far so good

- So far we have been able to develop inductive-recursive definitions in analogy to inductive definitions.
- Both give rise to an initial algebra semantics.
- Both can be codified using Dybjer-Setzer codes.

Container

We can compute a normal form for inductive definitions:

$$\begin{array}{l}
\textit{record Cont} : \textit{Set}_1 \ \textbf{where} \\
\quad \textit{constructor} \ _ \lhd _ \\
\quad \textit{field} \\
\qquad S : \textit{Set} \\
\qquad P : S \to \textit{Set}
\end{array}$$

$$\begin{array}{l}
[\![_]\!] : \textit{Cont} \to \textit{Set} \to \textit{Set} \\
[\![S \lhd P]\!] \, A = \Sigma \, S \, (\lambda \, s \to P \, s \to A)
\end{array}$$

Container can be coerced into *ID*:

$$\begin{array}{l}
\textit{emb} : \textit{Cont} \to \textit{ID} \\
\textit{emb} \, (S \lhd P) = \sigma \, S \, (\lambda \, s \to \delta \, (P \, s) \, \iota)
\end{array}$$

Container normal form

Any inductive definition can be normalized to a container:

$$\iota_C : \mathit{Cont}$$
$$\iota_C = \top \lhd \lambda \_ \to \bot$$
$$\sigma_C : (S : \mathit{Set}) \to (S \to \mathit{Cont}) \to \mathit{Cont}$$
$$\sigma_C\ S\ F = \Sigma\ S\ (\lambda\ s \to \mathit{Cont}.S\ (F\ s))$$
$$\qquad \lhd \lambda\ s' \to \mathit{Cont}.P\ (F\ (\mathit{proj}_1\ s'))\ (\mathit{proj}_2\ s')$$
$$\delta_C : (P : \mathit{Set}) \to \mathit{Cont} \to \mathit{Cont}$$
$$\delta_C\ P\ (S \lhd Q) = S \lhd (\lambda\ s \to P \uplus (Q\ s))$$
$$\mathit{cnf} : \mathit{ID} \to \mathit{Cont}$$
$$\mathit{cnf}\ \iota = \iota_C$$
$$\mathit{cnf}\ (\sigma\ S\ \phi) = \sigma_C\ S\ (\lambda\ s \to \mathit{cnf}\ (\phi\ s))$$
$$\mathit{cnf}\ (\delta\ P\ \phi) = \delta_C\ P\ (\mathit{cnf}\ \phi)$$

Applications of containers

Using containers to represent inductive definitions we can

1. Derive a semantically complete, small representation of morphisms
2. Show that inductive definitions are closed under composition (giving rise to a 2-category)

Container morphisms

We can calculate the representation using Yoneda:

$$
\begin{array}{l}
\textit{record ContM } ((S, P)\ (T, Q) : \textit{Cont}) : \textit{Set } \textbf{where} \\
\quad \textit{field} \\
\qquad f : S \to T \\
\qquad r : (s : S) \to Q\ (f\ s) \to P\ s
\end{array}
$$

Horizontal composition

$$
\begin{aligned}
&I : Cont \\
&I = \top \lhd (\lambda \_ \to \top) \\
&\_ \circ \_ : Cont \to Cont \to Cont \\
&(S \lhd P) \circ (T \lhd Q) = (\Sigma\ S\ (\lambda\ s \to P\ s \to T)) \\
&\quad \lhd (\lambda\ sf \to \Sigma\ (P\ (proj_1\ sf))\ (\lambda\ p \to Q\ (proj_2\ sf\ p)))
\end{aligned}
$$

Containers for IR?

- We cannot computer a container normal form for IR since σ and δ do not commute.
- Can we still establish the same results as for inductive definitions?

1. a complete notion of morphisms
2. composition of IR definitions

Recursive definitions of morphisms

- Neil and Hank showed that IR morphisms can be calculated recursively.
- For illustration I show how this works for ID (without calculating the container normal form).

$$\_ \Rightarrow \_ : ID \to ID \to Set$$

$$\begin{aligned}
\iota \Rightarrow \iota \quad &= \top \\
\iota \Rightarrow \sigma\ S\ \phi \quad &= \Sigma\ S\ (\lambda\ s \to \iota \Rightarrow \phi\ s) \\
\iota \Rightarrow \delta\ P\ \phi \quad &= (P \to \bot) \times \iota \Rightarrow \phi \\
\sigma\ S\ \phi \Rightarrow \psi &= (s : S) \to \phi\ s \Rightarrow \psi \\
\delta\ P\ \phi \Rightarrow \psi &= \phi \Rightarrow (\psi \circ P+)
\end{aligned}$$

$$\_ \circ \_ + : ID \to Set \to ID$$

$$\begin{aligned}
\iota \circ P + \quad &= \iota \\
\sigma\ S\ \phi \circ P + &= \sigma\ S\ (\lambda\ s \to (\phi\ s) \circ P+) \\
\delta\ Q\ \phi \circ P + &= \sigma\ (Q \to Maybe\ P) \\
&\quad (\lambda\ f \to \delta\ (\Sigma\ Q\ (\lambda\ q \to f\ q \equiv nothing))\ (\phi \circ P+))
\end{aligned}$$

Recursive composition?

The question remains can we define horizontal composition recursively?
Again we only look at *ID* only (but do not exploit container normal form).

$$
\begin{aligned}
&\_ \times ID\_ : ID \to ID \to ID \\
&\iota \times ID\; \psi = \psi \\
&\sigma\; S\; \phi \times ID\; \psi = \sigma\; S\; (\lambda\; s \to \phi\; s \times ID\; \psi) \\
&\delta\; P\; \phi \times ID\; \psi = \delta\; P\; (\phi \times ID\; \psi)
\end{aligned}
$$

$$
\begin{aligned}
&\_ \circ\_ : ID \to ID \to ID \\
&\iota \circ \psi = \iota \\
&\sigma\; S\; \phi \circ \psi = \sigma\; S\; (\lambda\; s \to (\phi\; s \circ \psi)) \\
&\delta\; P\; \phi \circ \psi = (P \implies \psi) \times ID\; (\phi \circ \psi)
\end{aligned}
$$

But how to define $P \Rightarrow$?

Summary

- We don't have a normal form for IR codes.
- We can define a complete notion of morphisms by recursion.
- But it is not clear wether IR codes are closed under composition.