Higher Order Categorical Semantics

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science
February 17, 2020
Advisor: Bas Spitters

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

in English. . .

Resumé

in Danish...

Acknowledgments

Lasse Letager Hansen,
Aarhus, February 17, 2020.

Contents

Chapter 1

Introduction

motivate and explain the problem to be addressed

example of a citation: [**?**]

get your bibtex entries from `https://dblp.org/`

Chapter 2

M-types

2.1 Containers / Signatures

A Container (or Signature) is a pair $S = (A, B)$ of types $\vdash A : \mathcal{U}$ and $a : A \vdash B(a) : \mathcal{U}$. From a container we can define a polynomial functor, defined for objects (types) as

$$\begin{aligned} P_S &: \mathcal{U} \to \mathcal{U} \\ P(X) := P_S(X) &= \sum_{a:A} B(a) \to X \end{aligned} \tag{2.1}$$

and for a function $f : X \to Y$ as

$$\begin{aligned} Pf &: PX \to PY \\ Pf(a, g) &= (a, f \circ g) \end{aligned} \tag{2.2}$$

As an example lets look at type for streams over the type A, defined using the container $S = (A, \mathbf{1})$, applying the polynomial functor we get

$$P_S(X) = \sum_{a:A} \mathbf{1} \to X \tag{2.3}$$

since we are working in a Category with exponentials we get $\mathbf{1} \to X \equiv X^{\mathbf{1}} \equiv X$, furthermore $\mathbf{1}$ and X does not depend on A here, so this will be equivalent to the definition

$$P_S(X) = A \times X \tag{2.4}$$

Now we define the coalgebra for this functor with type

$$\mathtt{Coalg}_S = \sum_{C:\mathcal{U}} C \to PC \tag{2.5}$$

and morphisms

$$\begin{aligned} \_ \Rightarrow \_ &: \mathtt{Coalg}_S \to \mathtt{Coalg}_S \\ (C, \gamma) \Rightarrow (D, \delta) &= \sum_{f:C \to D} \delta \circ f = Pf \circ \gamma \end{aligned} \tag{2.6}$$

M-types can now be defined from a container S as the type $\mathtt{M}$ such that $(\mathtt{M}, \mathtt{out} : \mathtt{M} \to P_S\mathtt{M})$ fulfills the property

$$\mathtt{Final}_S := \sum_{(X,\rho):\mathtt{Coalg}_S} \prod_{(C,\gamma):\mathtt{Coalg}_S} \mathtt{isContr}((C,\gamma) \Rightarrow (X,\rho)) \tag{2.7}$$

that is $\prod_{(C,\gamma):\mathtt{Coalg}_S} \mathtt{isContr}((C,\gamma) \Rightarrow (\mathtt{M}, \mathtt{out}))$. We denote this construction of the type $\mathtt{M}$, as $\mathtt{M}(A, B)$ or $\mathtt{M}S$.

If we continue our example for streams this will give us the M-type, we can see that $P_S(\mathtt{M}) = A \times \mathtt{M}$, meaning we have the following diagram, where $\mathtt{out}$ is an isomorphism (because of the finality of

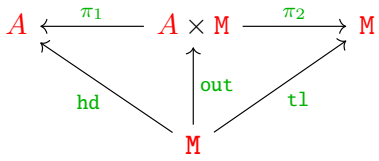

Figure 2.1: M-types of streams

the coalgebra), with inverse $\mathtt{in} : P_S\mathtt{M} \to \mathtt{M}$. We now have a semantic for the rules we would expect for streams, if we let $\mathtt{cons} = \mathtt{in}$ and $\mathtt{Stream}\ A = \mathtt{M}(A, \mathbf{1})$,

$$\frac{A : \mathcal{U} \quad s : \mathtt{Stream}\ A}{\mathtt{hd}\ s : A}\ \mathtt{E_{hd}} \tag{2.8}$$

$$\frac{A : \mathcal{U} \quad s : \mathtt{Stream}\ A}{\mathtt{tl}\ s : \mathtt{Stream}\ A}\ \mathtt{E_{tl}} \tag{2.9}$$

$$\frac{A : \mathcal{U} \quad x : A \quad xs : \mathtt{Stream}\ A}{\mathtt{cons}\ x\ xs : \mathtt{Stream}\ A}\ \mathtt{I_{cons}} \tag{2.10}$$

2.2 ITrees as M-types

2.2.1 Delay Monad

We want the following rules for ITrees

$$\frac{r : R}{\mathtt{Ret}\ r : \mathtt{itree}\ E\ R}\ \mathtt{I_{Ret}} \tag{2.11}$$

$$\frac{A : \mathcal{U} \quad a : E\ A \quad f : A \to \mathtt{itree}\ E\ R}{\mathtt{Vis}\ a\ f : \mathtt{itree}\ E\ R}\ \mathtt{I_{Vis}}. \tag{2.12}$$

Elimination rules

$$\frac{t : \mathtt{itree}\ E\ R}{\mathtt{Tau}\ t : \mathtt{itree}\ E\ R}\ \mathtt{E_{Tau}}. \tag{2.13}$$

check this statement

We start by looking at $\mathtt{itree}$s without the $\mathtt{Vis}$ constructor, this type is also know as the delay monad . We say this type is given by $S = (\mathbf{1} + R, \lambda\{\mathtt{inl}\ \_ \to \mathbf{1}\ ;\ \mathtt{inr}\ \_ \to \mathbf{0}\})$ equal to $\mathtt{M}S$, we then get the polynomial functor

$$P_S(X) = \sum_{x:\mathbf{1}+R} \lambda\{\mathtt{inl}\ \_ \to \mathbf{1}; \mathtt{inr}\ \_ \to \mathbf{0}\}\ x \to X \tag{2.14}$$

This type is equal to the type:

$$P_S(X) = X + R \times (\mathbf{0} \to X) \tag{2.15}$$

we know that $\mathbf{0} \to X \equiv \mathbf{1}$, so we can further reduce to

$$P_S(X) = X + R \tag{2.16}$$

meaning we get the following diagram. What this diagram says is that we can define the operations

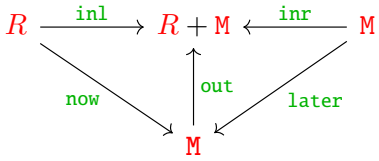

Figure 2.2: Delay monad

`now` and `later` using $\texttt{in} = \texttt{out}^{-1}$ together with the injections `inl` and `inr`.

(Later = Tau, Ret = Now)

2.2.2 Tree

Now lets look at the example where we remove the `Tau` constructor. We let

$$S = \left(R + \sum_{A:\mathcal{U}} E\ A\ ,\ \lambda\{\texttt{inl}\ \_ \to \mathbf{0}\ ;\ \texttt{inr}\ (A, e) \to A\} \right). \tag{2.17}$$

This will give us the polynomial functor:

$$P_S(X) = \sum_{x : R + \sum_{A:\mathcal{U}} E\ A} \lambda\{\texttt{inl}\ \_ \to \mathbf{0}\ ;\ \texttt{inr}\ (A, e) \to A\}\ x \to X \tag{2.18}$$

which simplifies to

$$P_S(X) = (R \times (\mathbf{0} \to X)) + (\sum_{A:\mathcal{U}} E\ A \times (A \to X)) \tag{2.19}$$

and further

$$P_S(X) = R + \sum_{A:\mathcal{U}} E\ A \times (A \to X) \tag{2.20}$$

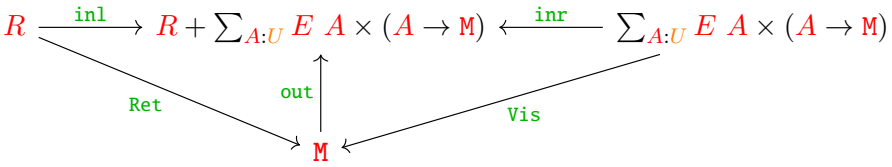

Figure 2.3: TODO: ???

Again we can define `Ret` and `Vis` using the `in` functor.

2.3 ITrees

Now we should have all the knowledge needed to make ITrees using $\mathtt{M}$-types. We define ITrees by the container:

$$S = \left(\mathbf{1} + R + \sum_{A:\mathcal{U}} (E\ A) \quad , \quad \lambda \{ \mathtt{inl}\ (\mathtt{inl}\ \_) \to \mathbf{1}\ ;\ \mathtt{inl}\ (\mathtt{inr}\ \_) \to \mathbf{0}\ ;\ \mathtt{inr}(A, \_) \to A \} \right) \tag{2.21}$$

Then the (reduced) polynomial functor becomes

$$P_S(X) = X + R + \sum_{A:\mathcal{U}} ((E\ A) \times (A \to X)) \tag{2.22}$$

Giving us the diagram

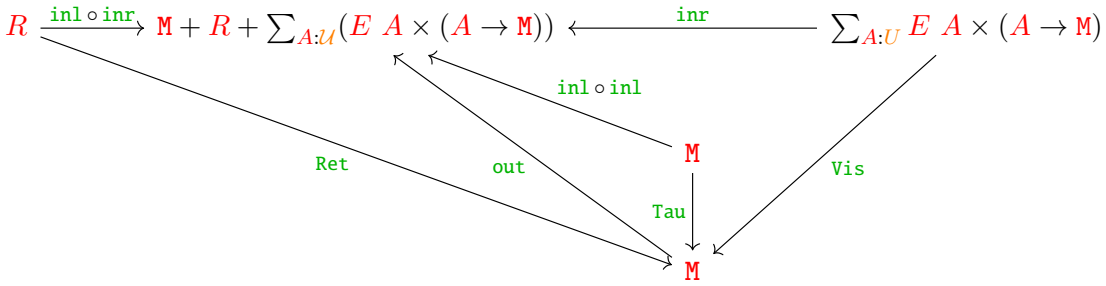

Figure 2.4: TODO: ???

Chapter 3

Conclusion

conclude on the problem statement from the introduction

Appendix A

The Technical Details