# The Partiality Monad

## Achim Jung Fest
## An Intersection of Neighbourhoods

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

jww Paolo Capriotti, Nils Anders Danielsson, Nicolai Kraus, Bernhard Reus

September 9, 2018

# Do we need partiality?

## The Totalitarian View

We do not need to talk about partial computations!
A non-terminating program has a bug. We need to talk about non-terminating programs as much as we need to talk about programs with syntax errors.
Hence, there is no need for domain theory either.

# The reformed totalitarian

- There are situations where we need partiality.
- For example if we want to implement (and partially verify) an interpreter for the type theory we are working in.
- What about the reals: all functions $f : \mathbb{R} \to \mathrm{Bool}$ are constant. What is the type of $\leq$ witnessing that this relation is semi-decidable.
- We may want to model and reason about implementations of partial languages.

# Partiality is an effect

- Effects in functional programming can be encapsulated as monads (following Moggi/Wadler).
- E.g.

  State $S$ is the type of states.

  $$M_{\mathrm{State}}\,A \equiv S \to S \times A$$

  Error $E$ is the type of errors.

  $$M_{\mathrm{Error}}\,A \equiv E + A$$

- The corresponding Kleisli category represents effectful computations.
- We can use these definitions to reason about effectful computations and execute them at compile time.
  See *Beauty in the Beast*, Haskell workshop 07, with W.Swierstra
- Partiality should just be another effect monad.
- In this sense Haskell is not pure.

# What is the partiality monad?

- There are different notions of partiality:

  Decidable partiality

  $$M_{\text{DecP}}\,A \equiv 1 + A$$

  Propositional partiality

  $$M_{\text{PropP}}\,A \equiv \Sigma P : \textbf{Prop}.P \to A$$

- But we are looking for a different kind of partiality here.
- We want to allow non-terminating computations and recursive programs.

# Capretta's monad

- In [*] Capretta describes a coinductive definition of a monad to capture general recursion.
- Using a destructor we can define $M_{\mathrm{Delay}}\,A$

$$\mathrm{next}\,A : M_{\mathrm{Delay}}\,A \to \{\mathrm{return} : A\} + \{\mathrm{step} : M_{\mathrm{Delay}}\,A\}$$

- Using copatterns we can define:

$$\begin{aligned}
&\mathrm{now} : A \to M_{\mathrm{Delay}}\,A \\
&\mathrm{next}\,(\mathrm{now}\,a) \equiv \mathrm{return}\,a \\
&\mathrm{later} : M_{\mathrm{Delay}}\,A \to M_{\mathrm{Delay}}\,A \\
&\mathrm{next}\,(\mathrm{later}\,d) \equiv \mathrm{step}\,d
\end{aligned}$$

- Exercise: define bind
  $\_ >\!\!>\!= \_ : M_{\mathrm{Delay}}\,A \to (A \to M_{\mathrm{Delay}}\,B) \to M_{\mathrm{Delay}}\,B$
- Equality on $M_{\mathrm{Delay}}\,A$ is strong bisimilarity.
  I.e. $M_{\mathrm{Delay}}\,A$ is the terminal coalgebra of $A + \_$.

[*] V. Capretta, *General Recursion via Coinductive Types*,LMCS 2005

# Too intensional

- $M_{\mathrm{Delay}}$ doesn't yet capture what we want.
- We can differentiate between a computation that terminates now or in one step or in two steps etc ...
- Capretta defines a notion of weak bisimilarity on $M_{\mathrm{Delay}}$:
  - First we inductively define $\_ \downarrow \_ : M_{\mathrm{Delay}}\, A \to A \to \mathbf{Prop}$ (terminates with):

$$\mathrm{next}\, d = \mathrm{return}\, a \to d \downarrow a$$
$$\mathrm{next}\, d = \mathrm{step}\, d' \to d' \downarrow a \to d \downarrow a$$

  - We define weak bisimilarity $\_ \approx \_ : M_{\mathrm{Delay}}\, A \to M_{\mathrm{Delay}}\, A \to \mathbf{Prop}$

$$d \approx d' \equiv \forall a : A. d \downarrow a \leftrightarrow d' \downarrow a$$

# Partiality as a quotient

- We can define $M_{\mathrm{Pq}} : \mathbf{Set} \to \mathbf{Set}$ using a quotient:
  $M_{\mathrm{Pq}}\,A :\equiv M_{\mathrm{Delay}}\,A/\approx$
- We understand quotients as inductively defined:

$$[\_] : M_{\mathrm{Delay}}\,A \to M_{\mathrm{Pq}}\,A$$
$$[\_]^{=} : d \approx d' \to [d] = [d']$$

  The first constructor constructs elements, the 2nd equalities of $M_{\mathrm{Pq}}\,A$.
- To define a function $f : M_{\mathrm{Pq}}\,A \to B$ we need

$$g : M_{\mathrm{Delay}}\,A \to B$$
$$h : d \approx d' \to g\,d = g\,d'$$

- Using pattern matching we can now define

$$f\,[d] :\equiv g\,d$$
$$f\,[p]^{=} :\equiv h\,p$$

  I am overloading notation and write $\mathrm{ap}\,f\,p$ (apply path) as $f\,p$ where $\mathrm{ap}\,f : x = y \to f\,x = f\,y$.

# Is this a monad?

We would like to show:

1. $M_{\mathrm{Pq}}$ is a monad.
2. $M_{\mathrm{Pq}}\,A$ is an $\omega$-CPO,

We (A.,Capretta, Uustalu) tried this in 2005 and failed...
The problem is that you need to commute quotients and coinductive (i.e. infinitary) definitions and you need instances of the axiom of choice to do this.
This is reminiscent of a similar problem with the Cauchy Reals: Without (countable) choice we cannot show that the Cauchy Reals are Cauchy complete.
This problem was adressed in HoTT by using a Higher Inductive Type to define the Cauchy Reals (HoTT book, chapter 11.3).
Can we do something similar here?

# Using countable choice

- In HoTT countable choice $(\mathrm{AC}^\omega)$can be expressed as

$$\Pi x : \mathbb{N}.||P\,x|| \to ||\Pi x : \mathbb{N}.P\,x||$$

  where $P : \mathbb{N} \to \mathbf{Prop}$ and $||A||$ is the propositional truncation of $A$.
- Chapman, Uustalu and Niccolò showed in 2015 that assuming $\mathrm{AC}^\omega$ one can show that $M_{\mathrm{Pq}}$ is a monad.
  *Quotienting the Delay Monad by Weak Bisimilarity* ICTAC 2015

# Defining $M_{\mathrm{P}}$ as a Higher Inductive Type

$$M_{\mathrm{P}}\ A : \mathbf{Set}$$
$$\sqsubseteq\ : M_{\mathrm{P}}\ A \to M_{\mathrm{P}}\ A \to \mathbf{Prop}$$

$$\bot : M_{\mathrm{P}}\ A$$
$$\eta : A \to M_{\mathrm{P}}\ A$$
$$\bigsqcup : \Pi_{f:\mathbb{N}\to M_{\mathrm{P}}\ A}(\Pi_{n:\mathbb{N}} f(n) \sqsubseteq f(n+1)) \to M_{\mathrm{P}}\ A$$

$$\frac{}{d \sqsubseteq d} \qquad \frac{}{\bot \sqsubseteq d} \qquad \frac{\bigsqcup(f,p) \sqsubseteq d}{\Pi_{n:\mathbb{N}} f(n) \sqsubseteq d} \qquad \frac{\Pi_{n:\mathbb{N}} f(n) \sqsubseteq d}{\bigsqcup(f,p) \sqsubseteq d}$$

$$\frac{d \sqsubseteq d' \qquad d' \sqsubseteq d}{d = d'}$$

A.,Danielsson,Kraus *Partiality, Revisited*, FOSSACS 2017

# This is a Quotient Inductive-Inductive Type (QIIT)

- We omit constructors for set and prop truncation.
- Since $M_{\mathrm{P}}\,A$ is set truncated, we call this a quotient inductive type (QIT) a special case of a HIT.
- Indeed, since $M_{\mathrm{P}}\,A$ and $\sqsubseteq$ are defined mutually it is a Quotient Inductive-Inductive Type (QIIT).
- The same applies to the definition of the Reals.

# The basic idea

- $M_{\mathrm{P}}\,A$ is the free $\omega$-CPO over $A$.
- Hence it is an $\omega$-CPO (1) and it is a monad (2).
- This is also reminiscent of the definition of the Cauchy Reals in the HoTT book which defines the Reals as the Cauchy completion of the rationals.
- We can show that assuming $\mathrm{AC}^{\omega}$ that the two definitions are equivalent $M_{\mathrm{P}}\,A = M_{\mathrm{Pq}}\,A$.
- The essence here is that QITs (and HITs) define elements and equality at the same time. This avoids many instance of AC.

# What next?

- We can now represent and reason about partial computations and general recursion in total Type Theory.
- This is an effect, at runtime we can just run the potentially non-terminating programs.
- Who says that Type Theory is not Turing complete?
- We can use QI(I)Ts to construct recursive types using $\omega$-colimits.
- With Frederik Forsberg, Ambrus Kaposi, Andras Kovac and Jakob von Raumer we are working on the theory of QIITs.
- Can we develop higher domain theory using higher directed type theory making the relation between recursive values and recursive types precise?