Stop thinking about bottoms when writing programs ...  

Thorsten Altenkirch  
University of Nottingham

Trouble with ⊥

\[(\ast) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\]
\[ 0 \ast n = 0 \]
\[ (m + 1) \ast n = m \ast n + n \]

\[x \ast y = y \ast x\ ?\]

No, because
\[0 \ast \bot = 0\]
\[\bot \ast 0 = \bot\]

Do we need \(\bot\) to be lazy?

\[\text{from} :: \mathbb{N} \to [\mathbb{N}]\]
\[\text{from} \ n = n : (\text{from} \ (n + 1))\]

• \(\text{from}\) is total, if we interpret lists as a terminal coalgebra.
\[[A] = \nu X.1 + A \times X\]

• Many useful algebraic properties do not hold.
• Correctness proofs get obliterated with reasoning about \(\bot\).
• Do we actually care about non-terminating programs?
• Programs are not natural phenomena.
• Programs are constructed!
data vs codata

\[ \text{data} \quad \text{SK} = S \mid K \mid \text{SK} : @ \text{SK} \]
\[ \text{nf} :: \text{SK} \rightarrow \text{SK} \]
\[ \text{nf} S = S \]
\[ \text{nf} K = K \]
\[ \text{nf} (t : @ u) = (\text{nf} t)@u(\text{nf} u) \]
\[ (@) :: \text{SK} \rightarrow \text{SK} \rightarrow \text{SK} \]
\[ K \quad @t = K : @ t \]
\[ K : @ t \quad @u = t \]
\[ S \quad @t = S : @ t \]
\[ S : @ t \quad @u = (S : @ t) : @ u \]
\[ ((S : @ t) : @ u)@v = (t@v)@u(\text{nf} v) \]

\[ \text{evenLength} :: [a] \rightarrow \text{Bool} \]
\[ \text{evenLength} [] = \text{True} \]
\[ \text{evenLength} (a : as) = \neg (\text{evenLength} n) \]

• evenLength is total, …
• if we interpret lists as initial algebra:
\[ [A] = \mu X.1 + A \times X \]
• Problem:
\[ \text{evenLength} \ (\text{from} \ 0) = \bot \]

Can we always avoid \( \bot \)?

data vs codata

• Finite lists
\[ \text{data} \quad [a] = [] | a : [a] \]

• Potentially infinite lists:
\[ \text{codata} \quad [a]^\omega = [] | a : [a]^\omega \]

• Better types
\[ \text{from} :: N \rightarrow [a]^\omega \]
\[ \text{evenLength} :: [a] \rightarrow \text{Bool} \]

• \( \text{evenLength} \ (\text{from} \ 0) \) doesn’t typecheck.

Computational Reals

• Define computational reals \( \mathbb{R} \) using Cauchy sequences.
• We cannot implement
\[ \text{pos} :: \mathbb{R} \rightarrow \text{Bool} \]

• Indeed, all total computable functions of type \( \mathbb{R} \rightarrow \text{Bool} \) are constant (Brouwer).
• However, there are perfectly reasonable partial implementations of \( \text{pos} \).
Monads...

- We need \( \perp \) for:
  - Interpreters.
  - Functions on \( \mathbb{R} \).
  - more examples?

Epigram

- Epigram is a dependently typed programming language...
- All Epigram programs are total (i.e. no \( \perp \)).
- It is not a programming language in Peter Mosses sense.
- because not all computable functions can be expressed.
- I am going to show how we can fix this...
- without making Epigram partial.

The Delay monad

\[
\text{codata } D \ a = \text{Now} \ a \ |
\text{Later} \ (D \ a)
\]

\[
\text{instance } \text{Monad } D \text{ where}
\]

\[
\text{return} \ = \text{Now}
\]

\[
\text{Now} \ a \ \bowtie \ k \ = \ k \ a
\]

\[
\text{Later} \ d \ \bowtie \ k \ = \text{Later} \ (d \ \bowtie \ k)
\]

\[
\perp \ = \text{Later} \ \perp
\]

\[
\text{data } ST \ s \ a = M \ (s \rightarrow (a, s))
\]

\[
\text{instance } \text{Monad } ST \ s \text{ where}
\]

\[
\text{return} \ a \ = \ M \ (\lambda s \rightarrow (a, s))
\]

\[
(ST \ f) \ \bowtie \ g \ = \ M \ (\lambda s \rightarrow \text{let} \ (a, s') = f \ s
\]

\[
M \ g' = g \ a
\]

\[
\text{in} \ g' \ s')
\]
Iteration with Delay

\[ \text{rep} :: (a \to D \text{ (Either } b a)) \to a \to D b \]
\[ \text{rep } k \ a = k \ a \triangleright\triangleright \lambda b a \to \]
\[ \text{case } b a \text{ of} \]
\[ \text{Left } b \to \text{Now } b \]
\[ \text{Right } a \to \text{Later } (\text{rep } k \ a) \]

Fixpoints with Delay

\[ \text{rec} :: ((a \to D b) \to (a \to D b)) \to a \to D b \]
\[ \text{rec } \phi \ a = \text{aux } (\lambda \_ \to \bot) \]
\[ \text{where } \text{aux} :: (a \to D b) \to D b \]
\[ \text{aux } k = \text{race } (k \ a) \text{ (Later } (\text{aux } (\phi \ k))) \]
\[ \text{race} :: (D a) \to (D a) \to (D a) \]
\[ \text{race } (\text{Now } a) = \text{Now } a \]
\[ \text{race } (\text{Later } \bot) (\text{Now } a) = \text{Now } a \]
\[ \text{race } (\text{Later } d) (\text{Later } d') = \text{Later } (\text{race } d \ d') \]

From Delay to Partial

- \( D \) is too intensional...
- We can observe how fast a function terminates.
- Hence \( \text{rec } f \neq f (\text{rec } f) \)
- We define
  \[ P \ a = D a / \simeq \]
  where \( \simeq \subseteq D a \times D a \) identifies values with different finite delay.
- We have to show that \( \triangleright\triangleright, \text{rep}, \text{rec} \) preserve \( \simeq \).
- We have \( \text{rec } f \simeq f (\text{rec } f) \)
  if \( f \) is \( \omega \)-continuous,
  however all definable \( f \) are.

Defining \( \simeq \)

- \( (\downarrow) \subseteq D a \times a \) is defined inductively.
  \[ \begin{array}{c}
  \text{Now } a \downarrow a \\
  \text{Later } d \downarrow a
  \end{array} \]

- \( (\subseteq) \subseteq D a \times D a \)
  \[ d \subseteq d' = \forall a. d \downarrow a \implies d' \downarrow a \]

- \( (\simeq) \subseteq D a \times D a \)
  \[ d \simeq d' = d \subseteq d' \land d' \subseteq d \]
Deja vu?

• Constructive Domain Theory!
• \( P a = a_\perp \)
• Note that constructively
  \[ a_\perp \neq a + \{ \perp \} \]
  because we cannot observe non-termination.
• \( P a \) and hence \( a \rightarrow P b \) are \( \omega \)-CPOs.
• \( \text{rec } f = \bigcup_{i \in \mathbb{N}} f^i \perp \) the code before constructs \( \sqcup \) in \( a \rightarrow P b \).

Conclusions and further work

• Using the partiality monad we can encapsulate partial programs in a total language.
• Partiality is an effect
• We can reason about partial programs at compile time using the definition of \( P a \).
• and we can execute non-terminating programs at run-time.
• In future Epigram could support partiality without giving up the advantages of having a total language for most programs.
• Still to do: recursive datatypes by a constructive implementation of the standard domain-theoretic construction.

Thank you

• Thanks to Conor McBride & the Epigram Team
  (James Chapman, Peter Morris, Wouter Swierstra)
  see www.e-pig.org for more information on Epigram.
• Acknowledgements to Tarmo Uustalu and Venanzio Capretta
  for joint work on a partial paper...
• Looking for my papers?
  Type “Thorsten” into google...