Typed Tagless Final Interpreters

Stuart Terrett

May 10, 2020

Preamble

```
{-# LANGUAGE GADTs #-}
module TTFI where
```

Typed

The object language is *typed*

Typed

The object language is *typed*

$$\textit{add}\ (\textit{int}\ 2)\ (\textit{bool}\ 3)$$

Typed

The object language is *typed*

$$\mathit{add}\ (\mathit{int}\ 2)\ (\mathit{bool}\ 3)$$

will not compile

tagless

The object language terms are not wrapped in a sum type with type tags

tagless

The object language terms are not wrapped in a sum type with type tags

$$
\begin{aligned}
&\textbf{data}\ \mathit{Term} = \\
&\quad \mathit{TInt}\ \mathit{Int} \\
&\quad \mid \mathit{TBool}\ \mathit{Bool} \\
&\quad \mid \mathit{TStr}\ \mathit{String}
\end{aligned}
$$

final

Terms of the object language are represented as expressions in the metalanguage

final

Terms of the object language are represented as expressions in the metalanguage

$$
\begin{aligned}
&\mathit{twoPlusTwo} :: (\mathit{AddSym}\ \mathit{repr}) \Rightarrow \mathit{repr}\ \mathit{Int} \\
&\mathit{twoPlusTwo} = \mathit{add}\ (\mathit{int}\ 2)\ (\mathit{int}\ 2)
\end{aligned}
$$

final

Terms of the object language are represented as expressions in the metalanguage

$$
\begin{aligned}
&\mathit{twoPlusTwo} :: (\mathit{AddSym}\ \mathit{repr}) \Rightarrow \mathit{repr}\ \mathit{Int}\\
&\mathit{twoPlusTwo} = \mathit{add}\ (\mathit{int}\ 2)\ (\mathit{int}\ 2)
\end{aligned}
$$

Not abstract syntax

interpreter

The syntax is a typeclass

$$
\begin{array}{l}
\textbf{class}\ \mathit{AddSym}\ \mathit{repr}\ \textbf{where} \\
\quad \mathit{int} :: \mathit{Int} \to \mathit{repr}\ \mathit{Int} \\
\quad \mathit{add} :: \mathit{repr}\ \mathit{Int} \to \mathit{repr}\ \mathit{Int} \to \mathit{repr}\ \mathit{Int}
\end{array}
$$

interpreter

The semantics are an instance of the typeclass

$$\textbf{newtype}\ R\ a = R\ \{ unR :: a \}$$

$$\begin{aligned}&\textbf{instance}\ \mathit{AddSym}\ R\ \textbf{where}\\ &\quad \mathit{int} = R\\ &\quad \mathit{add}\ (R\ x)\ (R\ y) = R\ \$\ x + y\end{aligned}$$

extensible

Adding interpretations

```
newtype S a = S { unS :: Int → String }
instance AddSym S where
  int = S ∘ const ∘ show
  add (S x) (S y) = S $ λc →
    "(" <> x c <> " + " <> y c <> ")"
```

extensible

Adding operations

$$\begin{array}{l}
\textbf{class}\ \mathit{MultSym}\ \mathit{repr}\ \textbf{where} \\
\quad \mathit{mul} :: \mathit{repr}\ \mathit{Int} \to \mathit{repr}\ \mathit{Int} \to \mathit{repr}\ \mathit{Int} \\
\textbf{instance}\ \mathit{MultSym}\ R\ \textbf{where} \\
\quad \mathit{mul}\ (R\ x)\ (R\ y) = R\ \$\ x * y \\
\textbf{instance}\ \mathit{MultSym}\ S\ \textbf{where} \\
\quad \mathit{mul}\ (S\ x)\ (S\ y) = S\ \$\ \lambda c \to \\
\qquad \texttt{"("} <> x\ c <> \texttt{" * "} <> y\ c <> \texttt{")"}
\end{array}$$

adding booleans

$$
\begin{array}{l}
\textbf{class } \mathit{BoolSym}\ \mathit{repr}\ \textbf{where} \\
\quad \mathit{bool} :: \mathit{Bool} \rightarrow \mathit{repr}\ \mathit{Bool} \\
\quad \mathit{lte} :: \mathit{repr}\ \mathit{Int} \rightarrow \mathit{repr}\ \mathit{Int} \rightarrow \mathit{repr}\ \mathit{Bool} \\
\quad \mathit{when} :: \mathit{repr}\ \mathit{Bool} \rightarrow \mathit{repr}\ a \rightarrow \mathit{repr}\ a \rightarrow \mathit{repr}\ a
\end{array}
$$

adding booleans

$$
\begin{array}{l}
\textbf{instance } \mathit{BoolSym}\ R\ \textbf{where} \\
\quad \mathit{bool} = R \\
\quad \mathit{lte}\ (R\ x)\ (R\ y) = R\ \$\ x \leqslant y \\
\quad \mathit{when}\ (R\ b)\ (R\ t)\ (R\ f) = R\ \$\ \textbf{if}\ b\ \textbf{then}\ t\ \textbf{else}\ f
\end{array}
$$

adding booleans

$$
\begin{array}{l}
\textbf{instance}\ \mathit{BoolSym}\ R\ \textbf{where} \\
\quad \mathit{bool} = R \\
\quad \mathit{lte}\ (R\ x)\ (R\ y) = R\ \$\ x \leqslant y \\
\quad \mathit{when}\ (R\ b)\ (R\ t)\ (R\ f) = R\ \$\ \textbf{if}\ b\ \textbf{then}\ t\ \textbf{else}\ f
\end{array}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{BoolSym}\ S\ \textbf{where} \\
\quad \mathit{bool} = S \circ \mathit{const} \circ \mathit{show} \\
\quad \mathit{lte}\ (S\ x)\ (S\ y) = S\ \$\ \lambda c \rightarrow \\
\qquad \texttt{"("} <> x\ c <> \texttt{" <= "} <> y\ c <> \texttt{")"} \\
\quad \mathit{when}\ (S\ b)\ (S\ t)\ (S\ f) = S\ \$\ \lambda c \rightarrow \\
\qquad \texttt{"if "} <> b\ c \\
\qquad <> \texttt{"\textbackslash nthen "} <> t\ c \\
\qquad <> \texttt{"\textbackslash n else"} <> f\ c
\end{array}
$$

adding lambda abstraction

$$
\begin{array}{l}
\textbf{class}\ \mathit{LamSym}\ \mathit{repr}\ \textbf{where} \\
\quad \mathit{lam} :: (\mathit{repr}\ a \to \mathit{repr}\ b) \to \mathit{repr}\ (a \to b) \\
\quad \mathit{app} :: \mathit{repr}\ (a \to b) \to \mathit{repr}\ a \to \mathit{repr}\ b
\end{array}
$$

adding lambda abstraction

$$
\begin{array}{l}
\textbf{instance}\ \mathit{LamSym}\ R\ \textbf{where} \\
\quad \mathit{lam}\ f = R\ \$\ \mathit{unR} \circ f \circ R \\
\quad \mathit{app}\ (R\ f)\ (R\ a) = R\ \$\ f\ a
\end{array}
$$

adding lambda abstraction

```haskell
instance LamSym R where
  lam f = R $ unR ∘ f ∘ R
  app (R f) (R a) = R $ f a
```

```haskell
instance LamSym S where
  lam f = S $ λc →
    let
      x = "x" <> show c
    in
      "\\" <> x <> ".\n"
       <> (unS ∘ f ∘ S $ const x) (c + 1)
  app (S f) (S a) = S $ λc →
    "(" <> f c <> " " <> a c <> ")"
```

adding recursion!!

$$
\begin{array}{l}
\textbf{class}\ \mathit{FixSym}\ \mathit{repr}\ \textbf{where} \\
\quad \mathit{fix} :: (\mathit{repr}\ a \to \mathit{repr}\ a) \to \mathit{repr}\ a
\end{array}
$$

adding recursion!!

```
instance FixSym R where
  fix f = R $ fx (unR ∘ f ∘ R)
    where fx g = g (fx g)
```

adding recursion!!

```
instance FixSym R where
  fix f = R $ fx (unR ∘ f ∘ R)
    where fx g = g (fx g)
```

```
instance FixSym S where
  fix f = S $ λc →
    let
      self = "self" <> show c
    in
      "(fix " <> self <> ".\n"
      <> (unS ∘ f ∘ S $ const self) (c + 1) <> ")"
```

simply typed lambda calculus with integer and boolean literals

$$
\begin{aligned}
&\textbf{class}\ (\mathit{AddSym}\ \mathit{repr} \\
&\quad , \mathit{MultSym}\ \mathit{repr} \\
&\quad , \mathit{BoolSym}\ \mathit{repr} \\
&\quad , \mathit{LamSym}\ \mathit{repr} \\
&\quad , \mathit{FixSym}\ \mathit{repr} \\
&\quad) \Rightarrow \mathit{Symantics}\ \mathit{repr}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Symantics}\ R \\
&\textbf{instance}\ \mathit{Symantics}\ S
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{eval} :: R\ a \to a \\
&\mathit{eval} = \mathit{unR}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{pprint} :: S\ a \to \mathit{String} \\
&\mathit{pprint}\ e = \mathit{unS}\ e\ 0
\end{aligned}
$$

factorial

$$
\begin{aligned}
&\mathit{factorial} :: (\mathit{Symantics}\ \mathit{repr}) \Rightarrow \mathit{repr}\ (\mathit{Int} \to \mathit{Int})\\
&\mathit{factorial} = \mathit{fix}\ (\lambda \mathit{self} \to \mathit{lam}\ (\lambda n \to\\
&\quad \mathit{when}\ (\mathit{lte}\ n\ (\mathit{int}\ 0))\\
&\qquad (\mathit{int}\ 1)\\
&\qquad (\mathit{mul}\ n\ (\mathit{self}\ \text{`}\mathit{app}\text{`}\ (\mathit{add}\ n\ (\mathit{int}\ (-1)))))))
\end{aligned}
$$

(hint: now's a good time to `stack ghci src/TTFI.hs`)

contrast: initial encoding

$$
\begin{array}{l}
\textbf{data}\ \mathit{SymI}\ h\ a\ \textbf{where} \\
\quad \mathit{INT} :: \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \\
\quad \mathit{Add} :: \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \\
\quad \mathit{Mul} :: \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \\
\quad \mathit{BOOL} :: \mathit{Bool} \to \mathit{SymI}\ h\ \mathit{Bool} \\
\quad \mathit{Lte} :: \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Int} \to \mathit{SymI}\ h\ \mathit{Bool} \\
\quad \mathit{When} :: \mathit{SymI}\ h\ \mathit{Bool} \to \mathit{SymI}\ h\ a \to \mathit{SymI}\ h\ a \to \mathit{SymI}\ h\ a \\
\quad \mathit{Var} :: h\ a \to \mathit{SymI}\ h\ a \\
\quad \mathit{Lam} :: (\mathit{SymI}\ h\ a \to \mathit{SymI}\ h\ b) \to \mathit{SymI}\ h\ (a \to b) \\
\quad \mathit{App} :: \mathit{SymI}\ h\ (a \to b) \to \mathit{SymI}\ h\ a \to \mathit{SymI}\ h\ b \\
\quad \mathit{Fix} :: (\mathit{SymI}\ h\ a \to \mathit{SymI}\ h\ a) \to \mathit{SymI}\ h\ a
\end{array}
$$

`h` is the evaluation context (eg `R` or `S` etc)

initial interpreter

Here we evaluate in `R`

$$
\begin{aligned}
&\mathit{evalI} :: \mathit{SymI}\ R\ a \to a\\
&\mathit{evalI}\ (\mathit{INT}\ i) = i\\
&\mathit{evalI}\ (\mathit{Add}\ x\ y) = (\mathit{evalI}\ x) + (\mathit{evalI}\ y)\\
&\mathit{evalI}\ (\mathit{Mul}\ x\ y) = (\mathit{evalI}\ x) * (\mathit{evalI}\ y)\\
&\mathit{evalI}\ (\mathit{BOOL}\ b) = b\\
&\mathit{evalI}\ (\mathit{Lte}\ x\ y) = (\mathit{evalI}\ x) \leqslant (\mathit{evalI}\ y)\\
&\mathit{evalI}\ (\mathit{When}\ b\ t\ f) = \mathbf{if}\ (\mathit{evalI}\ b)\ \mathbf{then}\ (\mathit{evalI}\ t)\ \mathbf{else}\ (\mathit{evalI}\ f)\\
&\mathit{evalI}\ (\mathit{Var}\ x) = \mathit{unR}\ x\\
&\mathit{evalI}\ (\mathit{Lam}\ f) = \mathit{evalI} \circ f \circ \mathit{Var} \circ R\\
&\mathit{evalI}\ (\mathit{App}\ f\ a) = (\mathit{evalI}\ f)\ (\mathit{evalI}\ a)\\
&\mathit{evalI}\ (\mathit{Fix}\ f) = \mathit{evalI}\ (\mathit{fx}\ f)\\
&\quad \mathbf{where}\ \mathit{fx}\ g = g\ (\mathit{fx}\ g)
\end{aligned}
$$

expression problem

Adding interpreters to `SymI` is straightforward

$$pprintI :: Sym\ S\ a \to String$$
$$pprintT = \ldots$$

expression problem

Adding interpreters to `SymI` is straightforward

$$pprintI :: Sym\ S\ a \to String$$
$$pprintT = \ldots$$

But adding new operations isn't possible

expression problem

Adding interpreters to `SymI` is straightforward

$$
\begin{aligned}
&pprintI :: Sym\ S\ a \rightarrow String \\
&pprintT = \ldots
\end{aligned}
$$

But adding new operations isn't possible

Because of the contravariant `SymI` in `Lam`, we can't use the coproduct-of-functors representation, because `SymI` is not representable as the fixpoint-of-a-functor

bijection

We can show the final and initial embeddings are equivalent by establishing a bijection

final to initial

instance $AddSym\ (SymI\ \ h)$ **where**
  $int = INT$
  $add = Add$
instance $MultSym\ (SymI\ \ h)$ **where**
  $mul = Mul$
instance $BoolSym\ (SymI\ \ h)$ **where**
  $bool = BOOL$
  $lte = Lte$
  $when = When$
instance $LamSym\ (SymI\ \ h)$ **where**
  $lam = Lam$
  $app = App$
instance $FixSym\ (SymI\ \ h)$ **where**
  $fix = Fix$

instance $Symantics\ (SymI\ \ h)$

$fToI :: SymI\ \ h\ a \rightarrow SymI\ \ h\ a$
$fToI = id$

initial to final

$$
\begin{aligned}
&iToF :: (Symantics\ repr) \Rightarrow SymI\ repr\ a \rightarrow repr\ a\\
&iToF\ (INT\ x) = int\ x\\
&iToF\ (Add\ x\ y) = add\ (iToF\ x)\ (iToF\ y)\\
&iToF\ (Mul\ x\ y) = mul\ (iToF\ x)\ (iToF\ y)\\
&iToF\ (BOOL\ b) = bool\ b\\
&iToF\ (Lte\ x\ y) = lte\ (iToF\ x)\ (iToF\ y)\\
&iToF\ (When\ b\ t\ f) = when\ (iToF\ b)\ (iToF\ t)\ (iToF\ f)\\
&iToF\ (Var\ a) = a\\
&iToF\ (Lam\ f) = lam\ (\lambda x \rightarrow iToF\ (f\ (Var\ x)))\\
&iToF\ (App\ f\ a) = app\ (iToF\ f)\ (iToF\ a)\\
&iToF\ (Fix\ f) = fix\ (\lambda self \rightarrow iToF\ (f\ (Var\ self)))
\end{aligned}
$$