Applicative Functors in Isabelle/HOL

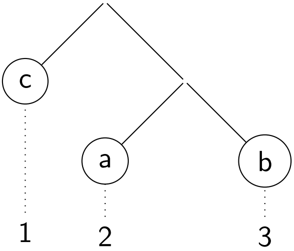
Joshua Schneider
joshuas@student.ethz.ch

December 3, 2015

Outline

Example: Tree Labels

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```



Inspired by G. Hutton and D. Fulger, "Reasoning About Effects: Seeing the Wood Through the Trees." in *Proceedings of the Symposium on Trends in Functional Programming*, (Nijmegen, The Netherlands, 2008).

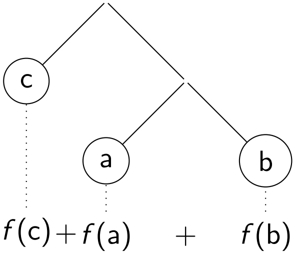
Composing Stateful Computations

Standard solution: state monad

```
fresh = do
  x <- get
  put (x + 1)
  return x

numberTree (Leaf _) = do
  x <- fresh
  return (Leaf x)
numberTree (Node l r) = do
  l' <- numberTree l
  r' <- numberTree r
  return (Node l' r')
```

Short Circuit Evaluation



f is partial! `f :: a -> Maybe Int`

```
evalTree (Leaf x) = f x
evalTree (Node l r) = do
  l' <- evalTree l
  r' <- evalTree r
  return (l' + r')
```

Cue Applicative Functors

```
class Functor f => Applicative f where
  pure  :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b

numberTree (Leaf _) = pure Leaf <*> fresh
numberTree (Node l r) =
  pure Node <*> numberTree l <*> numberTree r

evalTree (Leaf x) = f x
evalTree (Node l r) =
  pure (+) <*> evalTree l <*> evalTree r
```

The Laws

To do: Applicative functor laws
To do: Notation pure, $\diamond$

C. McBride and R. Paterson, "Applicative Programming with Effects." *Journal of Functional Programming, 18* (1). 2008, 1–13.

Lifting Terms and Equations

Lift the term f $a + b$:

$$
\begin{array}{ccccccccc}
(+) & & (& \mathrm{f} & & a) & & b \\
\mathrm{pure}\,(+) & \diamond & (& \mathrm{pure}\,\mathrm{f} & \diamond & a) & \diamond & b
\end{array}
$$

To do: Idiomatic term

Lift an equation: Addition is commutative

$$x + y = y + x$$

$$\mathrm{pure}\,(+) \diamond x \diamond y = \mathrm{pure}\,(+) \diamond y \diamond x$$

Hinze's Lemmas (1)

Lemma (Normal Form)

Let e be an idiomatic term with variables $x_1, \ldots, x_n$. There exists f such that

$$e = \mathsf{pure}\, f \diamond x_1 \diamond \ldots \diamond x_n.$$

Can lift equations

1. where both sides have the same list of variables, and
2. no variable is repeated.

R. Hinze, "Lifting Operators and Laws." 2010. Retrieved June 6, 2015, `http://www.cs.ox.ac.uk/ralf.hinze/Lifting.pdf`

Hinze's Lemmas (2)

Lemma (Lifting Lemma)

To do.

A Proof Method for Isabelle

Project Goal

Implement a proof method for Isabelle/HOL which lifts equations to applicative functors.

$$\text{base equation} \Longrightarrow \text{lifted equation}$$

Proof method: User interface for goal state transformation

$$\text{base equation} \longleftarrow \text{lifted equation}$$

Overview of Operation

$$e_1 = e_2$$

1. Transform into *canonical forms*

$$\Longleftarrow \quad \text{pure}\, f \diamond x_1 \diamond \ldots \diamond x_n = \text{pure}\, g \diamond x_1 \diamond \ldots x_n$$

2.

$$\Longleftarrow \quad f = g$$

3.

$$\Longleftarrow \quad \forall y_1 \ldots y_n.\ f y_1 \ldots y_n = g y_1 \ldots y_n$$

Combinatory Logic

- Eliminate variables from terms, introduce combinator constants
- BCKW system is equivalent to lambda calculus

$$\begin{aligned}\mathbf{B}gfx &= g(fx)\\ \mathbf{C}fxy &= fyx\\ \mathbf{K}xy &= x\\ \mathbf{W}fx &= fxx\end{aligned}$$

- If not all combinators are available, not all terms can be represented.

Bracket Abstraction

Turn lambda terms into combinator representation

$$
\begin{aligned}
& \lambda xy.\, x(fy) \\
= \quad & \lambda x.\, [y](x(fy)) \\
= \quad & \lambda x.\, \mathbf{B}x[y](fy) \\
= \quad & \lambda x.\, \mathbf{B}xf \\
= \quad & [x](\mathbf{B}xf) \\
= \quad & \mathbf{C}[x](\mathbf{B}x)f \\
= \quad & \mathbf{CB}f
\end{aligned}
$$

$$\mathbf{CB}fxy = x(fy)$$

Fancier Idioms

Some idioms satisfy additional laws, one or more of

$$\text{pure}\,\mathbf{C} \diamond f \diamond x \diamond y = f \diamond y \diamond x \tag{c}$$

$$\text{pure}\,\mathbf{K} \diamond x \diamond y = x \tag{k}$$

$$\text{pure}\,\mathbf{W} \diamond f \diamond x = f \diamond x \diamond x \tag{w}$$

Examples

- sum type $\alpha + \beta$: (w)
- environment functor $\beta \Rightarrow \alpha$: (c), (k), (w)

Interchange law?

$$\text{pure}\,(\lambda yg.\, gy) \diamond \text{pure}\,x \diamond f = f \diamond \text{pure}\,x$$

Bracket Abstraction Revisited

Assume an applicative functors satisfies (c).

$$
\begin{array}{rlrl}
 & \lambda xy.\, x(fy) & & x \diamond (\mathrm{pure}\, f \diamond y) \\
= & \lambda x.\, [y](x(fy)) & = & [\![y]\!](x \diamond (\mathrm{pure}\, f \diamond y)) \diamond y \\
= & \lambda x.\, \mathbf{B}x[y](fy) & = & (\mathbf{B} \diamond x \diamond [\![y]\!](\mathrm{pure}\, f \diamond y)) \diamond y \\
= & \lambda x.\, \mathbf{B}xf & = & (\mathrm{pure}\, \mathbf{B} \diamond x \diamond \mathrm{pure}\, f) \diamond y \\
= & [x](\mathbf{B}xf) & = & [\![x]\!](\mathrm{pure}\, \mathbf{B} \diamond x \diamond \mathrm{pure}\, f) \diamond x \diamond y \\
= & \mathbf{C}[x](\mathbf{B}x)f & = & (\mathrm{pure}\, \mathbf{C} \diamond [\![x]\!](\mathrm{pure}\, \mathbf{B} \diamond x) \diamond \mathrm{pure}\, f) \diamond x \diamond y \\
= & \mathbf{CB}f & = & \mathrm{pure}\, \mathbf{C} \diamond \mathrm{pure}\, \mathbf{B} \diamond \mathrm{pure}\, f \diamond x \diamond y \\
 & & = & \mathrm{pure}\, (\mathbf{CB}f) \diamond y \diamond x \\
 & & = & \underbrace{\mathrm{pure}\, (\lambda xy.\, x(fy)) \diamond y \diamond x}_{\text{a canonical form}}
\end{array}
$$

What are the Variables?

- Remember that both canonical forms need the same variable lists:
$$\mathsf{pure}\, f \diamond x_1 \diamond \ldots \diamond x_n = \mathsf{pure}\, g \diamond x_1 \diamond \ldots x_n$$
- Must be able to represent terms with available combinators
- Instantiation:
$$\begin{aligned} & \forall xy.\ \mathsf{pure}\, f \diamond x \diamond y = \ldots \\ \Longrightarrow\ & \forall z.\quad \mathsf{pure}\, f \diamond z \diamond z = \ldots \end{aligned}$$
What if we want to prove the latter, but can only represent the former?
- Algorithm depends on available combinators, partially a heuristic

Instances of Applicative

To do.

To do