Applicative と等価な typeclass を作って Agda で検証してみました

さとうまさひろ (てぴか)
@jx7548469
@tepika_math

自己紹介

- さとうまさひろ (てぴか) と申します
- (多元) 数理学科出身です
  - 数学が好きです
- 趣味はこんなのです↓
  - 音楽とダンスが好きです
    - 音楽ライブに時々出演しています
  - 最近車を買って、山とか海とか行ってます

早速なんですが

最近 Haskell 勉強してます !!

Haskell を勉強してる理由…

Haskell を勉強してる理由…

- やっぱかっこいいから

Haskell を勉強してる理由…

- やっぱかっこいいから
- 数学科出身だから

Haskell を勉強してる理由…

- やっぱかっこいいから
- 数学科出身だから

しかし…

Haskell を勉強してる理由…

- やっぱかっこいいから
- 数学科出身だから

しかし…
Haskell はやっぱり難しい !!

Haskell を勉強してる理由…

- やっぱかっこいいから
- 数学科出身だから

しかし…
Haskell はやっぱり難しい !!
特に型クラス !!!

Haskell を勉強してる理由…

- やっぱかっこいいから
- 数学科出身だから

しかし…
Haskell はやっぱり難しい !!
特に型クラス !!!
Monadって Haskell の挫折ポイントだと思うが、
個人的には Applicative も分かりにくかった

Applicative

Applicative の定義の一部抜粋:

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

Applicative

Applicative の定義の一部抜粋:

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

Applicative 則:

```
-- Identity
pure id <*> v = v
-- Homomorphism
pure f <*> pure x = pure (f x)
-- Interchange
u <*> pure y = pure ($ y) <*> u
-- Composition
pure (.) <*> u <*> v <*> w = u <*> (v <*> w)
```

もう一つの Applicative

liftA2 を使った Applicative の定義の抜粋:

```
class Functor f => Applicative f where
  pure :: a -> f a
  liftA2 :: (a -> b -> c) -> f a -> f b -> f c
```

もう一つの Applicative

liftA2 を使った Applicative の定義の抜粋:

```
class Functor f => Applicative f where
  pure :: a -> f a
  liftA2 :: (a -> b -> c) -> f a -> f b -> f c
```

Applicative 則 (liftA2 Ver.):

???

Applicative 即 (LiftA2 Ver.) の周辺調査

- Stack Overflow の記事 [Stack Overflow: 290b176332]
  - 記事タイトル↓
    - “What are the applicative functor laws in terms of pure and liftA2?”
  - liftA2 Ver. の Applicative 則は記載してあったが, 元の Applicative 則と同値だという証明ができない or 見当たらない…
- [Abel, A. 2024] の preprint
  - 論文タイトル:
    - Equivalence of Applicative Functors and Multifunctors
  - liftA2 ではなく MultiFunctor

    ```
    liftAn :: (a1 -> a2 -> ... -> a_n)
                  -> f a1 -> f a2 -> ... -> f a_n
    ```

    で扱っているので、純粋に liftA2 自体の条件を知ることができない

そこで、自分で Applicative 則 (LiftA2 Ver) を作っちゃいました !!

もう一つの Applicative

Applicative 則 (liftA2 Ver.):

```
-- Identity
liftA2 id pure = id
-- Homomorphism
liftA2 f (pure a) (pure b) = pure (f a b)
-- Homomorphism-Right
liftA2 f u (pure b) = (\a -> f a b) <$> u
-- LiftA2-Fmap-Composition-Left
liftA2 f (g <$> u) v = liftA2 (\a b -> f (g a) b) u v
-- Composition-Left
liftA2 f (liftA2 g u v) w
  = liftA3 (\a b d -> f (g a b) d) u v w
-- Composition-Right
liftA2 f w (liftA2 g u v)
  = liftA3 (\c a b -> f c (g a b) w u v
```

where

```
liftA3 :: (a -> b -> c -> d) -> f a -> f b -> f c -> f d
liftA3 f u v w = liftA2 id (liftA2 f u v) w
```

等価性の証明の大雑把な方針

(i) <*>から liftA2 へ変換する関数 ap-to-liftA2 を定義

(ii) liftA2 から<*>へ変換する関数 liftA2-to-ap を定義

(iii) liftA2-to-ap と ap-to-liftA2 の合成関数が恒等関数になることを Applicative Law を使い証明

(iv) ap-to-liftA2 と liftA2-to-ap の合成関数が恒等関数になることを (liftA2 Ver. の)Applicative Law を使い証明

詳しい証明はこちら↓

`https://github.com/k27c8ff627uxz/haskell-type-class`

等価性の証明の大雑把な方針

- Applicative(<*> Ver.) から Applicative(liftA2 Ver.) への関数 ap-to-liftA2

```
ap-to-liftA2 ::
  (f (a -> b) -> f a -> f b) ->
  (a -> b -> c) -> f a -> f b -> f c
ap-to-liftA2 (<*>) f m1 m2 =
  (pure f <*> m1) <*> m2
```

- Applicative(liftA2 Ver.) から Applicative(<*> Ver.) への関数 liftA2-to-ap

```
liftA2-to-ap ::
  ((a -> b -> c) -> f a -> f b -> f c) ->
  f (a -> b) -> f a -> f b
liftA2-to-ap liftA2 f m =
  liftA2 id f m
```

ふとした疑問:

ふとした疑問:
Applicative を LiftA2 形式にしたら少し分かりやすくなったけど…
もう少し分かりやすくならないかな？

ふとした疑問:
Applicative を LiftA2 形式にしたら少し分かりやすくなったけど…
もう少し分かりやすくならないかな？
→ MyApplicative クラスなるものを作りました !!

MyApplicative の定義

```haskell
class Functor f => MyApplicative f where
  pure :: a -> f a
  fpair :: f a -> f b -> f (a, b)
```

MyApplicative の定義

```haskell
class Functor f => MyApplicative f where
  pure :: a -> f a
  fpair :: f a -> f b -> f (a, b)
```

MyApplicative 則

```haskell
-- Homomorphism
f <$> pure a = pure (f a)
-- Homomorphism-Left
fpair (pure a) v = (\b -> (a, b)) <$> v
-- Homomorphism-Right
fpair u (pure b) = (\a -> (a, b)) <$> u
-- Associate
(\((a, b), c) -> (a, (b, c)) <$> (fpair (fpair u v) w)
    = fpair u (fpair v w)
-- FPair-Fmap
(\(x, y) -> (f x, f y)) <$> (fpair u v)
    = fpair (f <$> u) (g <$> v)
```

互いの変換規則

- Applicative(<*> Ver.) から MyApplicative(fpair Ver.) への変換

```
ap-to-fpair ::
  (f (a -> b) -> f a -> f b) ->
  f a -> f b -> f (a, b)
ap-to-fpair <*> m1 m2 =
  (pure (\a -> \b -> (a, b)) <*> m1) <*> m2
```

- MyApplicative(fpair Ver.) から Applicative(<*> Ver.) への変換

```
fpair-to-ap ::
  (f a -> f b -> f (a, b)) ->
  f (a -> b) -> f a -> f b
fpair-to-ap fpair u m1 =
  (\r a -> r a) <$> (fpair u m1)
```

詳しい証明はこちら↓

https://github.com/k27c8ff627uxz/haskell-type-class

互いの変換規則

- Applicative(LiftA2 Ver.) から MyApplicative(fpair Ver.) への変換

```
lftA2-to-fpair ::
  ((a -> b -> c) -> f a -> f b -> f c) ->
  f a -> f b -> f (a, b)
liftA2-to-fpair liftA2 m1 m2 =
  liftA2 (\a b -> (a, b)) m1 m2
```

- MyApplicative(fpair Ver.) から Applicative(LiftA2 Ver.) への変換

```
fpair-to-liftA2 ::
  (f a -> f b -> f (a, b)) ->
  (a -> b -> c) -> f a -> f b -> f c
fpair-to-liftA2 fpair f m1 m2 =
  uncurry f <$> fpair m1 m2
```

詳しい証明はこちら↓

https://github.com/k27c8ff627uxz/haskell-type-class

わかったこと

わかったこと

- Applicative を変形することで、自分にとって分かりやすい形になったと思う

今回作った Agda のソース

- https://github.com/k27c8ff627uxz/haskell-type-class

Reference

[Stack Overflow: 290b176332]

https://stackoverflow.com/questions/29017633/what-are-the-applicative-functor-laws-in-terms-of-pure-and-lifta2

[Abel, A. 2024] Abel, Andreas. "Equivalence of Applicative Functors and Multifunctors." arXiv preprint arXiv:2401.14286 (2024).