アプリカティブ則を満たすことの証明

以下にアプリカティブファンクターが満たす法則を示す。

- `pure id <*> v = v`
- `pure f <*> x = fmap f x`
- `pure (.) <*> u <*> v <*> w = u <*> (v <*> w)`
- `pure f <*> pure x = pure (f x)`
- `u <*> pure y = pure ($ y) <*> u`

この中でも、 `pure f <*> x = fmap f x` が重要である。アプリカティブのインスタンスがこれを満たすことを証明する。

> 数式的な意味での等号を`...`で表現する。

`Maybe` 型コンストラクタ

`Maybe` のApplicativeインスタンスは以下である。

```
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    (Just f) <*> something = fmap f something
```

したがって、 `pure = Just` から

```
pure f <*> x ... Just f <*> x
             ... (Just f) <*> x
             ... fmap f x
```

を得る。■

リスト型コンストラクタ `[]`

`[]` のApplicativeインスタンスは以下である。

```
instance Applicative [] where
    pure x = [x]
    fs <*> xs = [f x | f <- fs, x <- xs]
```

したがって、 `pure f = [f]` から

```
pure f <*> x ... [f] * x
             ... [f' x' | f' <- [f], x' <- x]
```

```
              ... [f x' | x' <- x]
              ... fmap f x
```

を得る。■

IO アクション

IO のApplicativeインスタンスは以下である。

```
instance Applicative IO where
    pure = return
    a <*> b = do
        f <- a
        x <- b
        return (f x)
```

したがって

```
pure f <*> x ... return f <*> x
             ... do
                    f' <- (return f)
                    x' <- x
                    return (f' x')
             ... fmap f x
```

を得る。■