Modify Program by Monad

115763K 氏名 比嘉健太 指導教員：河野真治

1 研究目的と提案手法

プログラムを変更するとプログラムの実行結果も変化する。その際、変更前のプログラムでは正しく実行できていた内容が変更によって損なわれる場合がある。プログラムの変更時に正しい実行結果が損なわれていないか自動で検知することを目的とする。

本研究では、自動で実行結果の変更を検知するために、プログラムの変更を Monad を利用して行なう。Monad を利用したプログラム変更の際、変更前のプログラムを保存しながら変更する。変更前のプログラムは保存されているので変更後も実行することができる。両方の実行結果を比較することにより実行結果が損なわれていないか検知する。

また、プログラムの変更を Monad によって定義することによって、プログラムを変更することの意味や性質などを解析したい。

2 Haskell における Monad

Monad とはプログラムの型に対するメタ計算に対応するもの [1] である。

プログラムのデータは型に所属しており、関数は型に所属する値を受けとって値を返すものとして考える。その際、任意の型 A を内包する型 T を定義することができる。型 A を内包した型 T を T A と記述する。

ここで、型 T A の値と型 A から型 T B に変換する関数 f をとり、型 T B の値を返す関数 join を定義することができる。関数 join では関数 f で行なわれる計算と join で行なわれる計算がある。計算の区別のために後者のことをメタ計算と呼ぶ。メタ計算は関数 f の内容に関わらず、型 T の定義により決定する。このことにより、型 T に対する join を定義することで型 T に対するメタ計算を定義することが可能となる。型 T A の値に対し、任意の関数を適用する際に join を用いることで、関数適用時に型 T のメタ計算を毎回行なうことができる。このように、型 T に対してメタ計算を対応させたものが Monad である。

型 A の値を型 T A の値に変更する関数と関数 join は Monad 則 [2](図 3) を満たす必要がある。プログラミング言語 Haskell においては型 A の値を型 T A の値に変更する関数名は return であり、join は中置関数 >>= である。これらの関数 2 つが Monad 則を満たす時、メタ計算と計算が正しく分離して行なれることが保証される。

3 プログラムの変更を表現する Monad

プログラムの変更を Monad で表現するために、まず型変数を持つ型 Diff を定義した (図 1)。

```
data Diff a = Diff [String] a [String] a
```

図 1: 型変数を持つ型 Diff の定義 (Haskell)

型変数 a を持つデータ型 Diff は、データコンストラクタ Diff に対して型 a を持つ変数と文字列のリストを 2 セット渡すことで構成される。このデータコンストラクタ Diff が型 A の変数を型 Diff A に変換する T に相当している。変数を 2 つ持つことができるため、片方を変更前のプログラムの計算用に、片方を変更後のプログラムの計算用に利用する。文字列のリストは変更前のプログラムが正しく保存されているか確認するために利用するものである。このリストはメタ計算にのみ利用される。

次に Diff を Monad として定義する。Haskell では Monad は型に対する型クラスとして表現される。型クラスとは特定の型 A が特定の型クラス X に属するために必要な関数群である。その関数群が定義された型 A を型クラス X のインスタンスであると呼ぶ。ある型を型クラス Monad のインスタンスにする際には関数 return と >>= を定義する必要がある。

型 Diff に対して関数 return と >>= を定義する (図 2)。

```
mu (Diff lx (Diff llx x _ _)
         ly (Diff _ _ lly y)) =
    Diff (lx ++ llx) x (ly ++ lly) y

instance Monad Diff where
    return x = Diff [] x [] x
    d >>= f  = mu $ fmap f d
```

図 2: Diff を Monad のインスタンスとして定義する (Haskell)

return はデータコンストラクタ Diff によって型 A を型 Diff A へと変換する関数である。>>= では、メタ計算として型 Diff が持っている 2 つ変数に対する計算を行なう。片方の変数はプログラム変更前の計算を行ない、もう片方は変更後の計算を行なうようにする。変更前の計算も行なうことで、変更前のプログラムの保存とする。

また、型 Diff が Monad 則 (図 3) を満たしていることは

定理証明支援系プログラミング言語 Agda によって証明済みである。

```
return a >>= k  =  k a
m >>= return    =  m
m >>= (\x -> k x >>= h)  =  (m >>= k) >>= h
```

図 3: Haskell における Monad 則

4 Diff Monad を用いたプログラムの実行

Diff Monad を用いた Haskell のプログラムを示す (図 4)。

```
generator :: Int -> Diff [Int]
generator x = let intList = [1..x] in
                 returnD intList

primeFilter :: [Int] -> Diff [Int]
primeFilter xs = let primeList    = filter isPrime xs
                     refactorList = filter even xs    in
                 returnDD primeList refactorList

count :: [Int] -> Diff Int
count xs = let primeCount = length xs in
           returnD primeCount

primeCount :: Int -> Diff Int
primeCount x = generator x >>= primeFilter >>= count
```

図 4: Diff Monad を用いたプログラムの例

このプログラムは整数 n を取り、1 から n までの整数の中から素数の個数を調べるプログラムである。1 から n までの整数の個数を調べる primeCount 関数は 3 つの関数からなる

- 1 から n までの整数を返す関数 generator
- 整数のリストから素数であるもののみを残したリストを返す関数 primeFilter
- リストの中に存在する要素の個数を返す関数 count

このプログラムの primeFilter 関数が返す Diff Monad を変更する。本来ならば素数である整数のみを返す計算だったが、変更により偶数である整数のみを返すようにした。図 4 のプログラムを実行した例が図 5 である。

```
*Main> primeCount 10
Diff ["[1,2,3,4,5,6,7,8,9,10]","[2,3,5,7]","4"] 4
     ["[1,2,3,4,5,6,7,8,9,10]","[2,4,6,8,10]","5"] 5
```

図 5: Diff Monad を用いたプログラムの実行例

変更前のプログラムの実行順序が上側の実行結果である。

- 1 から 10 までのリストを作成し
- 素数のみを残すために 2,3,5,7 が残り
- その個数を数えるために 4 となる

変更後のプログラムの実行順序が下側の実行結果である。

- 1 から 10 までのリストを作成し
- 偶数のみを残すために 2,4,6,8,10 が残り
- その個数を数えるために 5 となる

変更前の実行結果を保存しながら、プログラムが変更された後の新しい実行結果が得られた。この実行結果を比較することにより、変更前のプログラムの実行結果が損なわれていないか検知する。

今回は検証のために変更前と変更後の両方のプログラムを実行した。しかし、必ず両方実行しなくてはならない訳では無い。Haskell には遅延評価の機構が備わっており、値は必要とされるまで計算が実行されない。そのため、変更後のプログラムの実行結果のみを表示する場合などは変更前の計算は行なわれない。遅延評価と Diff Monad を組み合わせることで、必要の無い計算は増やさずに変更前のプログラムを保存できる。

5 まとめと課題

Diff Monad を定義することにより、変更前のプログラムを保存しつつ変更後のプログラムとして実行することが可能となった。

今後の課題としては大きく 2 つある。変更の個数の拡張と Monad によってプログラムを変更することの意味を調べることである。現状の Diff Monad はプログラムの変更を 1 つまでしか持つことができない。変更を無限回行なえる Diff を定義することで、プログラムに対する変更を Monad による変更のみで表したい。

また、プログラムに対する変更が Monad によって表される場合、圏論の視点からどのような意味が捉えられるか調査したい。Monad は圏論から導入された概念であり、プログラム側の Monad と圏論側の Monad は対応している。Diff Monad を圏論の観点で捉えることにより、Diff Monad の性質などを圏論側から導出できないかといった狙いがある。

ひいては、プログラムを作ることや変更することは理論的にどのような意味を持つのかを探っていきたい。

参考文献

[1] Eugenio Moggi, Notion of Computation and Monads (1991)

[2] Monad - HaskellWiki http://www.haskell.org/haskellwiki/Monad