C#における Algebraic Effects の実装の試み

野口 龍　松本 倫子　吉田 紀彦

関数型言語において計算的効果を扱うための手法である Algebraic Effects の処理のモジュール化と再利用性に着目して，オブジェクト指向言語へ Algebraic Effects を適用する研究がなされている．しかし，まだその例は少なく，また，その手法も様々である．本研究では Algebraic Effects のオブジェクト指向言語への幅広い適用性を検討するため，C# においてライブラリとして Algebraic Effects の実装を試み，例外処理などにおける Algebraic Effects の有用性を検討した．本発表では実装したライブラリの概要について報告する．

1　はじめに

オブジェクト指向言語は主にクラスをベースにモジュール化を行うプログラミングパラダイムである．しかし，例外処理をはじめ従来のクラスではモジュール化が難しい処理が存在する．

Algebraic Effects は関数型言語において例外処理や IO などの計算的効果を取り扱う手法として，モナドに代わるものとして注目されている．Algebraic Effects では例外処理を始めとした計算的効果をシグネチャとハンドラに分けて取り扱うが，これがモジュール化と再利用性を高めている．

Algebraic Effects が持つ柔軟なモジュール化と再利用性に着目して，Algebraic Effects をオブジェクト指向言語に取り入れる試みが行われている．しかし，例は少なく，またその実装方法も様々である．

本研究ではオブジェクト指向言語全般への幅広い適用性を検証する一助となるべく C# に焦点を当ててライブラリとして Algebraic Effects を部分的に実装し，それを用いて例外処理などの計算的効果をモジュール化することを試みた．本発表では本ライブラリの概要について報告する．

An Attempt to Implement Algebraic Effects in C#
Ryu Noguchi, Noriko Matsumoto, Yoshihiko Yoshida, 埼玉大学, Saitama University.

2　関連研究

Algebraic Effects は計算的効果をシグネチャとハンドラに分けて記述する．Algebraic Effects のライブラリを開発する際にはこの 2 要素に加えハンドラ内部で用いられる限定継続について考える必要がある．特に限定継続はオブジェクト指向言語とはあまり馴染みがない概念であるため実装の障壁となる要素である．

Eff[1] は Algebraic Effect の研究用に開発された関数型言語で OCaml で実装されている．

JEff[2] は Algebraic Effect をオブジェクト指向言語に取り入れるために考案されたオブジェクト指向言語で実験的な実装がある．

Lua での実装である eff.lua[3] では限定継続を Lua に備わっているコルーチンで実装している．コルーチンによる限定継続の実装はハンドラ内での限定継続呼び出しが一回限り（ワンショット）であるという制約があり，それにより非決定計算を記述できないなどの欠点があるが，例外処理や IO などでは依然有用である．また，限定継続にコルーチンを利用したものとして，Lua よりもコルーチンの呼び出しに関する制約が強い JavaScript においても実装が見られる [4].

Java での実装 [5] は多相型を基本とした静的型付け言語の利点を損なわない型設計を行っている．シグネ

チャをインターフェースとして表現し，ハンドラをシグネチャを継承したクラスで表現する．ただし限定継続は予め印をつけた関数をコンパイルして得られたJVM のバイトコードを後処理して実現しておりライブラリ単体として提供することは難しい．その代わりワンショットの制約はなく非決定計算などより多くの計算的効果を扱うことができる．

特殊なものとしてオブジェクト指向言語と関数型言語の両方の性質を持つ Scala にて実装された事例 [6] があるが，限定継続がモナドで実現されており Java や C# のようなモナドが提供されていない言語で同じようなアプローチをするのは難しい．

ほかにも C 言語で longjump を応用した実装が存在する [7]．

3 C# での実装の方針

関連研究を踏まえて，本研究ではこれまで実装が試みられていない C# を取り上げた． C# を特に取り上げた理由として，Unity [8] を初めとしたプロダクトで採用されている実績があり，実装した際の有用度が高い言語であることが挙げられる．

また，本研究ではユーザビリティを考えライブラリ単体で Algebraic Effects を実装することを目標とした． そのため C# と類似の特徴を持つ Java での先行研究の実装を移植することはできず，他の手法を用いて実装する必要がある．本研究では C# に備わっているコルーチンに着目し，限定継続の実装に Lua での実装で用いられていた手法を応用した． しかし，C# のコルーチンは Lua のものとは異なる箇所があり，Algebraic Effects を実装する障壁となる点がある．

1 つめは処理の移譲を行う yield が値を返さないことである．yield は計算的効果を発生させるために用いる perform を実装するために利用するが，C# の yield では戻り値のある計算的効果を扱うことができない．これは復帰可能な例外処理や IO を取り扱う際に障害となる．

2 つ目は C# のコルーチンがスタックをまたいで yield を行うことができない stackless コルーチンであることである．これはコルーチン内部で呼び出した関数内での計算的効果の利用が困難であることを意味する．Lua のコルーチンは stackfull コルーチンであるため，このような問題がない．なお， JavaScript のコルーチンも stackless コルーチンであるが，yield が値を返せるため引数で計算的効果を発生させることができ，C# のコルーチンとは自由度が異なる．

最後は静的型付け言語特有の型の記述の煩雑さである．C# では特別な戻り値の型である場合のみそのメソッドをコルーチンとみなす．この型は継承による別名がつけられないので略記ができずメソッドのシグネチャの記述が煩雑になる問題がある．

これらの問題点を受け入れ，ナイーブに実装する方針も考えられるが，本研究では限定継続の実装に関するコルーチンの利用方法を改変することによりこれらの問題を低減することを試みた．この改変の詳細及びそれによる悪影響については 4 節にて詳しく述べる．

計算的効果のシグネチャとハンドラの記述の実装の方針については，クラス型オブジェクト指向言語としての特徴を活かすべく，本研究では Lua のものではなく Java での実装で用いられたインターフェースによるシグネチャ記述とクラス拡張によるハンドラ記述の手法を応用する．ただし，後述するように perform 操作に相当するインターフェースとハンドラ内部のロジックと型設計は独自に行う必要があった．

4 ライブラリの実装

以上の方針を踏まえて具体的な実装方法について述べる．はじめに限定継続の実装について述べる．前述の通り C# のコルーチンには限定継続を実現する際にユーザビリティを大きく低下させる問題がある．本ライブラリではこの問題を低減させるためにコルーチンの呼び出しと被呼び出しの関係を入れ替えた．つまり，計算的効果のハンドラがコルーチンとして計算的効果を利用しているメイン処理を呼び出していたのに対し，本ライブラリではメイン処理からコルーチンとして計算的効果を呼び出している．これにより計算的効果を単なるメソッドとして扱うことができ戻り値を扱ったり深いスタックからの呼び出しを行うことができた．しかし，この方法では限定継続の破棄によるスタックの破棄ができない．そこで本ライブラリで

はスタックの破棄を例外をスローすることにより実現した．また，関係入れ替えたため前述の問題が今度はハンドラ内の処理に現れる．しかし，ハンドラの記述は本処理の記述より少ないため問題が少ないと見られる．

次にシグネチャとハンドラの実装について述べる．前述のようにシグネチャとハンドラは静的型を割り当てる必要があり，Java での実装を参考にした．ただし，限定継続の実装方法が異なるため一部の型は独自に用意した．

5 ライブラリの適用例

以下は本ライブラリによって実装した例外処理である．このように try-catch の catch 節の内容をクラスに閉じ込め再利用することができる．

```
// 計算的効果のシグネチャ
interface IException<TRes>
{
  // 計算的効果はコルーチン
  IEnumerator<Effect<TRes>>
    Throw(string message);
}

// ハンドラ
class MyExceptionHandler : IException<Void>,
     Handler<int,int?>
{
  public int? Pure(int pure)
  {
    return pure;
  }

  public IEnumerator<Effect<Void>> Throw(string
       message)
  {
    Console.WriteLine(message);
    // 継続の破棄
    // 続く処理を中断する
    yield return Return<int?>(null);
  }
}

// 計算的効果を扱うメソッド
int SafeDivide
  (int a, int b,
  IException<Void> exc, Performer p)
{
  if (b == 0) {
    p.Perform(exc("divide␣by␣zero"));
  }
  return a / b;
}

// エントリポイント
int Main(string[] args)
{
  // Handle メソッド
  // 処理とハンドラを渡して
  // 計算的効果を扱う
  var result = EffectNet.Handle<int,int,
       MyExceptionHandler>
    (new MyExceptionHandler(),
      (exc, p) => SafeDivide(2, 0, exc, p));
}
```

6 おわりに

本研究では Algebraic Effects のオブジェクト指向言語における広い適用性と有用性を検討するため，C# において Algebraic Effects を実装することを試みた．結果，限定継続がワンショットであること，継続の破棄が標準の例外機構に依存していることなど不完全な箇所があるが，Algebraic Effects を C# のライブラリとして部分的に実装することができ，例外処理などにおいてその有用性を検討することができた．

今後の研究として，今回導入した限定継続の実装手法の改良や形式的記述，そして異なるアプローチを視野に入れた実装手法の検討を行いたい．

参 考 文 献

[1] Eff. https://www.eff-lang.org.

[2] Pablo Inostroza and Tijs van der Storm. JEff: Objects for Effect. *Onward! 2018 Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, November 2018.

[3] eff.lua. https://github.com/Nymphium/eff.lua.

[4] eff.js. https://github.com/MakeNowJust/eff.js.

[5] Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. Effect Handlers for the Masses. *Proceedings of the ACM on Programming Languages*, November 2018.

[6] Jonathan Immanuel Brachthäuser and Philipp Schuster. Effekt: Extensible Algebraic Effects in Scala. *Proceedings of 8th ACM SIGPLAN International Scala Symposium*, October 2017.

[7] Daan Leijen. Implementing Algebraic Effects in C "Monads for Free in C". *Microsoft Research Technical Report*, 2017.

[8] Unity. https://unity.com.