Неалгебраические эффекты как способ контроля эффектов

Используемые технологии

- Язык реализации: Haskell
- Парсинг кода: parsec
- Backend:

  В первых версиях использовалась компиляция в LLVM (Low-level virtual machine) с помощью llvm-hs и последующая линковка с RTS

  В последующих версиях от LLVM пришлось отказаться в пользу транспиляции в Haskell (используется haskell-src-exts) из-за большой сложности компиляции анонимных функций

- RTS: реализована в виде кода на Haskell
- Source code: https://github.com/ilyasm0919/Lang33

Контроль эффектов

- Упрощает отслеживание возможностей кода без его полного анализа
- Предоставляет статические гарантии, проверяемые компилятором
- Уменьшает связность кода, увеличивает модульность
- Сильно упрощает тестирование
- Упрощает разработку, развертывание, …

Системы эффектов

Способ контроля эффектов, основанный на приписывании функциям не только их возвращаемых значений, но и выполняемых ими эффектов

- Алгебраические

  Поддерживают только эффекты “не сильно влияющие на поток выполнения”

- Неалгебраические

  Поддерживают все эффекты, включая сильно меняющие поток выполнения (например, call/cc или select)

Примеры программ

Hello world:

```
main() {
    print("Hello, world!\n")
}
```

Примеры программ

Echo:

```
main() {
    while 1 {
        print(getLine())
        print("\n")
    }
}
```

Примеры программ

Factorial:

```
factorial(n) {
    if sub(0, n) { Fail("Negative number") null }
    res = 1
    i = n
    while i {
        res = mul(res, n)
        i = sub(i, 1)
    }
    res
}
```

Примеры программ

Factorial:

Использование встроенного эффекта *Failable*

```
factorial(n) {
    if sub(0, n) { Fail("Negative number") null }
    res = 1
    i = n
    while i {
        res = mul(res, n)
        i = sub(i, 1)
    }
    res
}
```

Примеры программ

Factorial (продолжение):

```
main() {
    factorial(parseNum(getLine@())) with handler h {
        pure(x) {
            print(numToStr(x))
        }

        Fail(msg, cont) {
            print(msg)
        }
    }
}
```

Примеры программ

Factorial (продолжение):

```
main() {
    factorial(parseNum(getLine@())) with handler h {
        pure(x) {
            print(numToStr(x))
        }

        Fail(msg, cont) {
            print(msg)
        }
    }
}
```

Запуск функции

Обработка случая отсутствия эффектов

Обработка эффекта *Fail*

Немного о типах

- Строгая статическая типизация
- Вывод типов по Хиндли-Милнеру (доработанному для эффектов)
- Функции высших порядков

Типы с эффектами

```
factorial : fun(Num) [Failable<><><>|e]Num
factorial(n) {
    if sub(0, n) { Fail("Negative number") null }
    res = 1
    i = n
    while i {
        res = mul(res, n)
        i = sub(i, 1)
    }
    res
}
```

Типы с эффектами

Функция использует эффект *Failable* Но может быть использована вместе с другими эффектами

```
factorial : fun(Num) [Failable<><><><>|e]Num
factorial(n) {
    if sub(0, n) { Fail("Negative number") null }
    res = 1
    i = n
    while i {
        res = mul(res, n)
        i = sub(i, 1)
    }
    res
}
```

Альтернативы

- Монады

  Значение каждого эффекта постоянно

  Примеры языков: Haskell, ML, OCaml

  А также: F#, Scala, Scheme, Clojure

Альтернативы

- Free-монады

  Недостаточно выразительны

  Примеры: Haskell

- Алгебраические эффекты

  По сути Free-монады, но заложенные в языке

  Примеры: Koka, Eff, Unison

Альтернативы

- Freer-монады

  Это и есть неалгебраические эффекты. Только не заложенные в языке

  Примеры: Haskell

- Неалгебраические эффекты

  Примеры: отсутствуют

Еще один пример эффектов

Объявление эффекта:

```
effect Logger<><><> {
    Log(Str) : Unit
}
```

Еще один пример эффектов

Использование эффекта:

```
sum(a, b) {
    Log(setSubstring@("a = ", 4, 0, numToStr(a)))
    Log(setSubstring@("b = ", 4, 0, numToStr(b)))
    Log(setSubstring@("a + b = ", 8, 0, numToStr(add(a, b))))
}
```

Еще один пример эффектов

Создание обработчика эффекта:

```
printLog() {
    handler h {
        Log(msg, cont) {
            print@(setSubstring("Log: \n", 5, 0, msg))
            cont(null) with h
        }
    }
}
```

Еще один пример эффектов

Запуск:

```
testLogs() {
    print@("Print log\n")
    sum(5, 7) with printLog()
}
```

```
Print log
Log: a = 5
Log: b = 7
Log: a + b = 12
```

Еще один пример эффектов

Создание другого обработчика эффекта:

```
ignoreLog() {
    handler h {
        Log(msg, cont) {
            cont(null) with h
        }
    }
}
```

Еще один пример эффектов

Создание еще одного обработчика эффекта:

```
printLogThenIgnore() {
    handler h {
        Log(msg, cont) {
            print@(setSubstring("\n", 0, 0, msg))
            cont(null) with ignoreLog()
        }
    }
}
```

Еще один пример эффектов

Запуск:

```
testLogs() {
    print@("Print log\n")
    sum(5, 7) with printLog()

    print@("\nIgnore log\n")
    sum(5, 7) with ignoreLog()

    print@("\nPrint log then ignore\n")
    sum(5, 7) with printLogThenIgnore()
}
```

```
Print log
Log: a = 5
Log: b = 7
Log: a + b = 12

Ignore log

Print log then ignore
a = 5
```

Еще один пример эффектов

И даже такой обработчик эффекта:

```
printEvenLog() {
    handler h1 {
        Log(msg, cont) {
            print@(setSubstring("\n", 0, 0, msg))
            cont(null) with handler h2 {
                Log(msg, cont) {
                    cont(null) with h1
                }
            }
        }
    }
}
```

Еще один пример эффектов

Запуск:

```
testLogs() {
    print@("Print log\n")
    sum(5, 7) with printLog()

    print@("\nIgnore log\n")
    sum(5, 7) with ignoreLog()

    print@("\nPrint log then ignore\n")
    sum(5, 7) with printLogThenIgnore()

    print@("\nPrint even log\n")
    sum(5, 7) with printEvenLog()
}
```

```
Print log
Log: a = 5
Log: b = 7
Log: a + b = 12

Ignore log

Print log then ignore
a = 5

Print even log
a = 5
a + b = 12
```