Построение синтаксических анализаторов на основе алгебраических эффектов

Георгий Лукьянов
georgiy.lukjanov@gmail.com

Артём Пеленицын
apel@sfedu.ru

Южный Федеральный Университет
Институт Математики, Механики и Компьютерных Наук имени И. И. Воровича

4 Апреля 2017

План

Вычислительные эффекты

```
add : Int -> Int -> Int
```

```
addSt : State (Int, Int) Int
```

```
addIO : IO Int
```

- Контролируемые
  - State
  - Reader
  - Exception
  - ...
- Неконтролируемые
  - IO

План

Язык программирования Frank

Последняя версия представлена на POPL'17 в статье Do be do be do от Sam Lindley, Conor McBride, Craig McLaughlin

Особенности

- **Строгая** стратегия вычислений
- Алгебраические типы данных
- Разделение **типов-значений** и **типов-вычислений**
- **Интерфейсы** — сигнатуры эффектов
- **Операции** — обработчики эффектов
- Обычные функции являются тривиальными операторами, обрабатывающими пустой множество эффектов
- Полиморфизм эффектов основанный на понятии **охватывающего поля эффектов** (англ. ambient ability)

Натуральные числа

```
data Nat = zero | suc Nat
```

Рекурсивные функции

```
eqNat : Nat -> Nat -> Bool
eqNat zero zero   = true
eqNat (suc x) (suc y) = eqNat x y
eqNat zero _      = false
eqNat _  zero     = false
```

Вычисляющая условная операция

```
iffi : Bool -> X -> X -> X
iffi true  t f = t
iffi false t f = f
```

Стандартная условная операция

```
if : Bool -> {X} -> {X} -> X
if true  t f = t!
if false t f = f!
```

Интерфейсы эффектов

Эффект изменяемого состояния

```
interface State S = get : S
                  | put : S -> Unit
```

Эффект потенциально ошибочных вычислений

```
interface Error = fail : Zero
```

Обработчики эффектов

Обработчик эффекта State

```
state : S -> <State S>X -> X
state _ x                = x
state s <get -> k>       = state s (k s)
state _ <put s -> k>     = state s (k unit)
```

Обработчик эффекта Error

```
catch : <Error>X -> Maybe X
catch x = just x
catch <fail -> _> = nothing
```

План

Парсер как комбинация эффектов

Обработка комбинации эффектов

```
parse : {[Error, State (List Char)] X} ->
        (List Char) -> Maybe X
parse p str = catch (state str p!)
```

Парсер как комбинация эффектов

Обработка комбинации эффектов

```
parse : {[Error, State (List Char)] X} ->
        (List Char) -> Maybe X
parse p str = catch (state str p!)
```

Базовые парсеры

```
item : [Error, State (List Char)] Char
item! = on get! { nil        -> fail
                | (x :: xs) -> put xs; x}

sat : {Char -> [Error, State (List Char)] Bool} ->
      [Error, State (List Char)] Char
sat p = on item! {c -> if (p c) {c} {fail}}
```

Парсер как комбинация эффектов

Парсер для заданного символа

```
char : Char -> [Error, State (List Char)] Char
char c = sat {x -> eqChar x c}
```

Парсер для заданной строки

```
string : (List Char) ->
         [Error, State (List Char)] (List Char)
string str = map char str
```

Парсер как комбинация эффектов

Комбинатор детерминированного выбора

```
choose : {[Error, State (List Char)] X} ->
         {[Error, State (List Char)] X} ->
          [Error, State (List Char)] X
choose p1 p2 =
  on (parse p1 get!) { (right _)  -> p1!
                     | (left  _)  ->  on (parse p2 get!)
                                      { (right _)   -> p2!
                                      | (left err)  -> fail
                                      }
                     }
```

Парсер как комбинация эффектов

Комбинаторы последовательности

```
many : {[Error, State (List Char)]X} ->
        [Error, State (List Char)](List X)
many p = choose {some p} {nil}

some : {[Error, State (List Char)] X} ->
        [Error, State (List Char)](List X)
some p = p! :: many p
```

Интерфейс эффекта

```
interface Parser X Y =
    fail : Y
  | sat : {Char -> Bool} -> Char
  | choose : {[Parser X Y] Y} -> {[Parser X Y] Y} -> Y
  | many : {[Parser X Y] Y} -> List Y
```

Обработчик эффекта

```
runParser : (List Char) -> <Parser X Y>Y ->
            Pair (Maybe Y) (List Char)
runParser xs r = pair (just r) xs
runParser xs <fail -> _> = pair nothing xs
runParser nil <sat p -> k> = pair nothing nil
runParser (x::xs) <sat p -> k> =
  if (p x) {runParser xs (k x)} {pair nothing (x::xs)}
runParser xs <choose p1 p2 -> k> =
  on (runParser xs p1!)
    { (pair (just _) _) -> runParser xs p1!
    | (pair nothing _) -> on (runParser xs p2!)
      { (pair (just _) _) -> runParser xs p2!
      | (pair nothing _) -> pair nothing xs
      }
    }
```

Обработчик эффекта (продолжение)

```
runParser : (List Char) -> <Parser X Y>Y ->
            Pair (Maybe Y) (List Char)
runParser xs <many p -> k> =
  on (runParser xs p!)
     { (pair (just ok) rest) ->
         cons (pair (just ok) nil)
              (runParser rest (many p; k nil))
     | (pair nothing rest) -> runParser xs (k nil)
     }
```

Проблема гомогенности

```
runParser : (List Char) -> <Parser X Y>Y ->
            Pair (Maybe Y) (List Char)
```

Проблема гомогенности

```
runParser : (List Char) -> <Parser X Y>Y ->
            Pair (Maybe Y) (List Char)
```

$< Parser\ Char\ Char > \neq < Parser\ Char\ (List\ Char) >$

Проблема гомогенности

```
runParser : (List Char) -> <Parser X Y>Y ->
            Pair (Maybe Y) (List Char)
```

$< \mathit{Parser\ Char\ Char} > \neq < \mathit{Parser\ Char\ (List\ Char)} >$

Гетерогенный список с помощью экзистенциального типа (Haskell)

```
data Obj = forall a. (Show a) => Obj a

xs :: [Obj]
xs = [Obj 1, Obj "foo", Obj 'c']
```

Интерфейс эффекта

```
interface Parser =
    fail : forall Y . Y
  | sat : {Char -> Bool} -> Char
  | choose : forall Y . {[Parser] Y} -> {[Parser] Y} -> Y
  | many : forall Y . {[Parser] Y} -> List Y
```

План

Результаты

- Построение парсеры на основе комбинации эффектов
- Исследование возможности описания парсера как монолитного эффекта

References

- Do be do be do. Sam Lindley, Conor McBride, and Craig McLaughlin. POPL 2017.
- A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. J. Log. Algebr. Meth. Program., 84(1):108–123, 2015. URL http://dx.doi.org/10.1016/j.jlamp.2014.02.001.
- P. B. Levy. Call-By-Push-Value: A Functional/Imperative Synthesis, volume 2 of Semantics Structures in Computation. Springer, 2004.
- Monadic Parser Combinators // *Graham Hutton*, *Erik Meijer* – Department of Computer Science, University of Nottingham, 1996
- Статья и эти слайды `https://github.com/geo2a/plc-mmcs-2017-frank-paper`
- Frankoparsec — experimental parser combinators library implemented in Frank `https://github.com/geo2a/frankoparsec`