Функциональный парсер легковесного языка разметки Markdown на основе комбинирования монад и моноидального представления исходного текста

Г. А. Лукьянов, ПМИ, группа 4.1
Научный руководитель: асс. каф. ИВЭ А. М. Пеленицын

20.06.2015

Институт математики, механики и компьютерных наук ЮФУ

Поставленные задачи

1. Разработка двух библиотек **монадических парсеров**, с применением различных технологий **комбинирования вычислительных эффектов**. Обе библиотеки должны использовать **моноидальное** представление исходного текста.
2. **Сравнение подходов** к комбинированию вычислительных эффектов, выявление их преимуществ и недостатков.
3. Разработка **транслятора Markdown** с LaTeX-вставками в HTML и LaTeX.

Использовались результаты работ:

MParsers96 Monadic Parser Combinators // *Graham Hutton*, *Erik Meijer* – Department of Computer Science, University of Nottingham, 1996

Monoids13 Adding Structure to Monoids // *Mario Blaževic* – Stilo International plc – **Haskell Symposium 2013**

ExtEff13 Extensible Effects An Alternative to Monad Transformers // *Oleg Kiselyov*, *Amr Sabry*, *Cameron Swords* – Indiana University, USA – **Haskell Symposium 2013**

Исходный тип для парсера [MParsers96]

Тип Parser

```
newtype Parser a = Parser {
        parse :: String → Maybe (a, String)}
```

Экземпляр класса типов Monad

```
instance Monad Parser where
        return t = Parser $ λs → Just (t, s)
        m >>= k = Parser $ λs →
                do (u, v) ← parse m s
                   (x, y) ← parse (k u) v
                   return (x, y)
```

Строковые типы в Haskell

String

Псевдоним для списка символов

ByteString

Наиболее низкоуровневый тип

Text

Тип для работы с Unicode-текстом

Строковые типы как моноиды [Monoids]

Полиморфный по входу базовый парсер

```
item :: TextualMonoid t ⇒ Parser t Char
item = Parser f
       where f inp = splitCharacterPrefix inp
```

Фукнция, отделяющая префикс

```
splitCharacterPrefix :: TextualMonoid t ⇒
                        t → Maybe (Char, t)
```

Парсер как стек монад

Обновленный тип Parser

```
newtype Parser t a =
        Parser (StateT (ParserState t)
                       (Either (ErrorReport t)) a)
```

Extensible Effects [ExtEff13]

Тип Eff

type *Eff r a* = *Free* (*Union r*) *a*

Пример статического набора эффектов

Eff (Reader Int :> Reader Bool :> Void) a

Парсеры на Extensible Effects [ExtEff13]

Парсер-предикат и его эффекты

```
sat :: (Member Fail r
       ,Member (State String) r
       ) ⇒ (Char → Bool) → Eff r Char
sat p = do x ← item
           if p x then return x else die
```

Запуск парсера

```
parse p inp = run ∘ runFail ∘ runState inp $ p
```

Парсеры на Extensible Effects [ExtEff13]

Парсер для слов и его эффекты

```
word :: (Member Fail r
        , Member (State String) r
        , Member (Choose) r) ⇒ Eff r String
word = some letter
```

Запуск парсера

```
parseWithChoose p inp =
  run ∘ runChoice ∘ runFail ∘ runState inp $ p
```

Язык Markdown

```
# Заголовок первого уровня

Параграфы отделяются пустой строкой

Второй параграф. *Наклонный,*
**Полужирный,** и `Моноширинный.`

Неупорядоченный список:

* Пункт
* Другой
* Ещё один

> Блочные цитаты
> записываются так.
>
> Могут содержать,
> несколько параграфов.

## Возможны блочные вставки LaTeX:

$$e^{i \pi} + 1 = 0$$
```

Заголовок первого уровня

Параграфы отделяются пустой строкой

Второй параграф. *Наклонный,* **Полужирный,** и `Моноширинный.`

Неупорядоченный список:

- Пункт
- Другой
- Ещё один

> Блочные цитаты записываются так.
>
> Могут содержать, несколько параграфов.

Возможны блочные вставки LaTeX:

$$e^{i\pi} + 1 = 0$$

AST для Markdown в Haskell

Документ

$$\textsf{type}\ \mathit{Document} = [\mathit{Block}]$$

Блок

$$\begin{aligned}\textsf{data}\ \mathit{Block} = {} & \mathit{Blank} \\ & \mid \mathit{Header}\ (\mathit{Int}, \mathit{Line}) \\ & \mid \mathit{Paragraph}\ [\mathit{Line}] \\ & \mid \mathit{UnorderedList}\ [\mathit{Line}] \\ & \mid \mathit{BlockQuote}\ [\mathit{Line}]\end{aligned}$$

AST для Markdown в Haskell

Строка

$$\textbf{data}\ \mathit{Line} = \mathit{Empty} \mid \mathit{NonEmpty}\ [\mathit{Inline}]$$

Элементы строки

$$\begin{aligned}\textbf{data}\ \mathit{Inline} = {} & \mathit{Plain\ String} \\ \mid {} & \mathit{Bold\ String} \\ \mid {} & \mathit{Italic\ String} \\ \mid {} & \mathit{Monospace\ String}\end{aligned}$$

Конструирование AST

Парсер для документа

```
doc :: TM.TextualMonoid t ⇒ Parser t Document
doc = many block
      where block = blank <|> header <|> paragraph
                    <|> unorderdList <|> blockquote
                    <|> blockMath
```

Пример парсера для заголовка

```
header :: TM.TextualMonoid t ⇒ Parser t Block
header = do
         hashes ← token $ some $ char '#'
         text   ← nonEmptyLine
         return $ Header (length hashes, text)
```

Результаты

1. Разработана библиотека монадических парсеров, полиморфных по входным данным. Акцент сделан на подробность сообщений об ошибках.
2. Разработан транслятор подмножества Markdown с LaTeX-вставками в HTML.
3. Начата разработка библиотеки парсеров, основанных на Extensible Effects.
4. Исходный код доступен в Git-репозиториях:
   `https://github.com/geo2a/markdown_monparsing`
   `https://github.com/geo2a/ext-effects-parsers`