Структурирование вычислений с эффектами

Г. А. Лукьянов

Научный руководитель: к.т.н., с.н.с, доц.каф. ИВЭ А. Н. Литвиненко

Южный Федеральный Университет
Институт математики, механики и компьютерных наук имени И. И. Воровича

21 Июня 2017

Постановка задачи

Цель

Сравнительный анализ трёх реализаций *систем контроля вычислительных эффектов*: преобразователи монад в Haskell, расширяемые эффекты в Haskell, алгебраические эффекты во Frank.

Задачи

1. *Реализация парсер-комбинаторных библиотек* с использованием трёх систем контроля эффектов.
2. *Сравнение производительности* и *определение качественных отличий* полученных реализаций парсер-комбинаторных библиотек.
3. *Дизайн и реализация клиент-серверного приложения* для учёта студенческой активности с использованием различных подходов к контролю над эффектами.

План

Вычислительные эффекты

Функция с побочным эффектом в языке без системы эффектов

```
int plus(int x, int y) {
  print("Неструктурированный побочный эффект.");
  return x + y;
}
```

Чистая функция в Haskell

```
plus :: Int -> Int -> Int
plut x y = x + y
```

Функция с побочным эффектом в Haskell

```
plusIO :: Int -> Int -> IO Int
plutIO x y = do
  print "Структурированный побочный эффект."
  return (x + y)
```

Подходы к структурированию эффектов и их реализации

Монадический подход

- Монады и преобразователи монад в Haskell

Алгебраические эффекты

- Расширяемые эффекты в Haskell
- Язык программирования Frank

План

Структура эффектов парсера

- Изменяемое состояние входного потока
- Потенциальная невозможность или неоднозначность разбора

Парсер как монадический стек

```haskell
newtype Parser a =
  Parser (StateT String (Either Error a)

parse :: Parser a -> String -> Either Error (a, String)
parse (Parser p) s = runStateT p s
```

Парсер на основе расширяемых эффектов

```
type Parsable r = (Member Fail r, Member (State String) r)

type Parser r a = Parsable r => Eff r a

parse :: Eff (Fail :> State String :> Void) a ->
         String -> (String, Maybe a)
parse p inp = run . runState inp . runFail $ p
```

Frank: парсер как комбинация эффектов

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

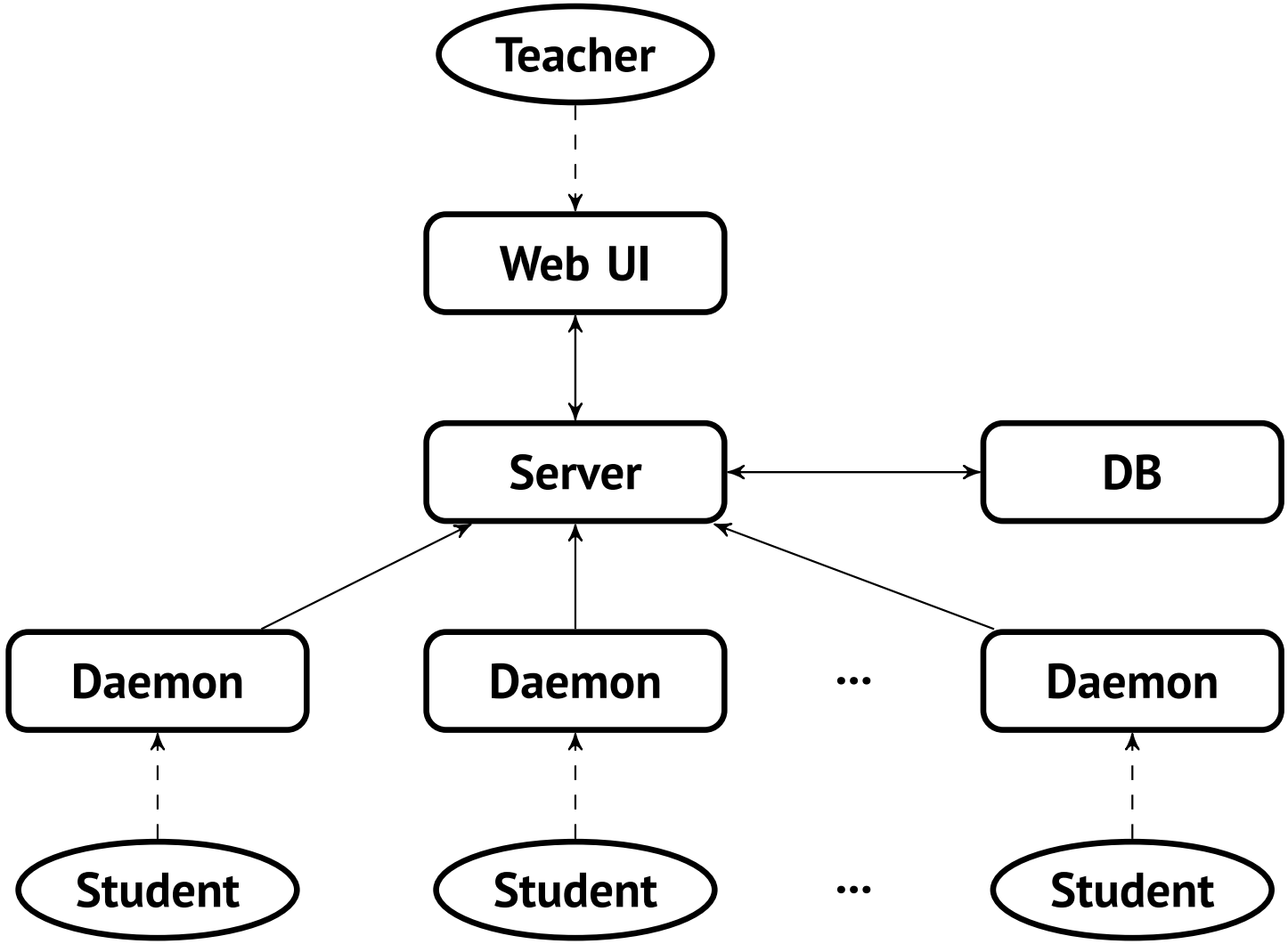
Frank: парсер как монолитный эффект

Возможный интерфейс эффекта **Parser**

```
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

Архитектура “Student’s Big Brother”



Особенности реализации “Student’s Big Brother”

- Спецификация HTTP API на уровне типов
- Единые типы в серверном и клиентском коде
- Генерация кода с использованием механизмов обобщённого программирования
- Управление побочными эффектами сервера с помощью преобразователей монад
- Применение расширяемых эффектов в реализации агента сбора данных (daemon)

Результаты

- Разработаны *библиотеки комбинаторов парсеров* на основе **преобразователей монад** и **расширяемых эффектов**.
- **Выделены отличия** преобразователей монад и расширяемых эффектов.
- Произведено **сравнение производительности** разработанных библиотек с Pandoc.
- Разработана **серия прототипов библиотек парсеров** для Frank и выявлены *ограничения выразительности*.
- Разработана и апробирована **система контроля производительности студентов** на лабораторных работах по программированию. В реализации использованы методы контроля побочных эффектов.