

Лекция 14: что дальше?

Функциональное программирование на Haskell

Алексей Романов

6 марта 2023 г.

МИЭТ

Нерассмотренные части Haskell: стандартные библиотеки

- В самом начале упоминалось, что тип `String` очень плох для практического использования. Альтернативы в пакетах `text` и `bytestring`.
- Коллекции:
 - `containers`: `Data.Map`, `Data.Set`, `Data.Sequence`.
 - `unordered-containers`: `Data.Hash{Map/Set}`.
 - Массивы: `array` и `vector`.
- Комбинаторы парсеров: `megaparsec`.
- Сборка проектов и управление зависимостями: `cabal` и `stack`.
- Линзы (`lens`): обобщение геттеров и сеттеров.
- Альтернативы `show` для больших структур: `pretty-show` и `prettyprinter`.

Нерассмотренные части Haskell: язык

- Семейства типов.
- Типы высших рангов с `forall` внутри типа.
 - Важный пример: монада `ST`.
- Экзистенциальные типы (выраженные всё равно через `forall`).
- Синонимы образцов.
- **Template Haskell**: генерация кода во время компиляции.
 - Могли видеть использование в документации QuickCheck/Hedgehog.
- Исключения.
- Foreign Function Interface для интеграции с C.

Подходы к эффектам

- Многие монады (и аппликативные функторы) можно рассматривать как побочные эффекты.
 - **IO**: понятно.
 - **State**: чтение и запись в память.
 - **[]**: недетерминированность
 - ...

Подходы к эффектам

- Многие монады (и аппликативные функторы) можно рассматривать как побочные эффекты.
 - **IO**: понятно.
 - **State**: чтение и запись в память.
 - **[]**: недетерминированность
 - ...
- Как их комбинировать?
- Композиция монад не обязательно монада.

Подходы к эффектам

- Многие монады (и аппликативные функторы) можно рассматривать как побочные эффекты.
 - **IO**: понятно.
 - **State**: чтение и запись в память.
 - **[]**: недетерминированность
 - ...
- Как их комбинировать?
- Композиция монад не обязательно монада.
- Трансформеры монад: **transformers** и **mtl**
- Алгебраические эффекты
- Свободные (free) и «более свободные» (freer) монады
- Небольшой обзор: **Monad transformers, free monads, mtl, laws and a new approach**

Функциональные структуры данных

- Функциональные структуры данных очень отличаются от привычных!
- Классическая книга: Okasaki, Purely functional data structures
- Список «после Окасаки» (2010). И ещё немного.

Функциональные структуры данных

- Функциональные структуры данных очень отличаются от привычных!
- Классическая книга: Okasaki, Purely functional data structures
- **Список «после Окасаки» (2010). И ещё немного.**
- Функциональные структуры, позволяющие доступ к своим старым состояниям, полезны и в императивных языках.
 - А изменяемые — в Haskell через **IO** или **ST**.

- **Линейные типы:** значения, которые могут быть использованы только один раз.
 - Меньше сборки мусора.
 - Можно использовать изменяемые структуры данных.
 - Безопасное изменение состояния типа (например, тип файла может включать, открыт ли он).

Будущее Haskell

- **Линейные типы:** значения, которые могут быть использованы только один раз.
 - Меньше сборки мусора.
 - Можно использовать изменяемые структуры данных.
 - Безопасное изменение состояния типа (например, тип файла может включать, открыт ли он).
- **Зависимые типы:** типы, параметризованные значениями.
 - Стандартный пример: вектор с размером в типе.
 - Библиотека **singletons** позволяет эмулировать многие применения зависимых типов.

Будущее Haskell

- **Линейные типы**: значения, которые могут быть использованы только один раз.
 - Меньше сборки мусора.
 - Можно использовать изменяемые структуры данных.
 - Безопасное изменение состояния типа (например, тип файла может включать, открыт ли он).
- **Зависимые типы**: типы, параметризованные значениями.
 - Стандартный пример: вектор с размером в типе.
 - Библиотека **singletons** позволяет эмулировать многие применения зависимых типов.
- Не совсем будущее: уточнённые типы (refinement types) в Liquid Haskell.
 - **LiquidHaskell: знакомство** и **Liquid Haskell Tutorial**

Альтернативы Haskell

- OCaml: похож на Haskell без ленивости и (пока) классов типов.
- Erlang: динамический функциональный язык, ориентированный на многозадачность.
- Варианты Lisp (можно посмотреть Racket).

Альтернативы Haskell

- OCaml: похож на Haskell без ленивости и (пока) классов типов.
- Erlang: динамический функциональный язык, ориентированный на многозадачность.
- Варианты Lisp (можно посмотреть Racket).
- Для JVM: Scala, Clojure.
- Для .Net: F# (почти тот же OCaml).

Альтернативы Haskell

- OCaml: похож на Haskell без ленивости и (пока) классов типов.
- Erlang: динамический функциональный язык, ориентированный на многозадачность.
- Варианты Lisp (можно посмотреть Racket).
- Для JVM: Scala, Clojure.
- Для .Net: F# (почти тот же OCaml).
- ФП без сборщика мусора: Rust.
 - Мощная и интересная система типов, описывающая время жизни значений.
 - Очень быстрое развитие.
 - Реальная альтернатива C++ для низкоуровневого программирования.

Альтернативы Haskell

- OCaml: похож на Haskell без ленивости и (пока) классов типов.
- Erlang: динамический функциональный язык, ориентированный на многозадачность.
- Варианты Lisp (можно посмотреть Racket).
- Для JVM: Scala, Clojure.
- Для .Net: F# (почти тот же OCaml).
- ФП без сборщика мусора: Rust.
 - Мощная и интересная система типов, описывающая время жизни значений.
 - Очень быстрое развитие.
 - Реальная альтернатива C++ для низкоуровневого программирования.
- Языки с зависимыми типами: Agda, Idris, Coq.

Альтернативы Haskell

- OCaml: похож на Haskell без ленивости и (пока) классов типов.
- Erlang: динамический функциональный язык, ориентированный на многозадачность.
- Варианты Lisp (можно посмотреть Racket).
- Для JVM: Scala, Clojure.
- Для .Net: F# (почти тот же OCaml).
- ФП без сборщика мусора: Rust.
 - Мощная и интересная система типов, описывающая время жизни значений.
 - Очень быстрое развитие.
 - Реальная альтернатива C++ для низкоуровневого программирования.
- Языки с зависимыми типами: Agda, Idris, Coq.
- Языки спецификаций: TLA+, Alloy.

- Лямбда-исчисление: то, с чего всё началось.
 - И множество его обобщений.

Области математики, связанные с ФП

- Лямбда-исчисление: то, с чего всё началось.
 - И множество его обобщений.
- Математическая логика в целом
 - Логики высших порядков.
 - Изоморфизм Карри-Ховарда.
 - Формализация математики.
 - И автоматический поиск доказательств.
 - Agda, Idris и Coq сюда же!

Области математики, связанные с ФП

- Лямбда-исчисление: то, с чего всё началось.
 - И множество его обобщений.
- Математическая логика в целом
 - Логики высших порядков.
 - Изоморфизм Карри-Ховарда.
 - Формализация математики.
 - И автоматический поиск доказательств.
 - Agda, Idris и Coq сюда же!
- Теория категорий: источник понятий функтора, монады, естественного преобразования.

Дополнительное чтение

- [What I Wish I Knew When Learning Haskell](#)
- [An opinionated guide to Haskell in 2018](#)
- [Real World OCaml: Functional Programming for the Masses](#) (книга, доступна бесплатно)
- [Много ссылок по Agda](#)
- [Введение в Idris](#)
- [The Rust Programming Language](#) (книга, доступна бесплатно)
- [Category Theory for Programmers](#) (книга, доступна бесплатно)
- [The Great Theorem Prover Showdown](#)
- [Use of Formal Methods at Amazon Web Services](#)
- [How to write a 21st Century Proof](#)