

Содержание

| | |
|---|-----------|
| Введение | 19 |
| Краткая биография автора | 21 |
| О пользовании книгой | 22 |
| Состав и структура представления информации | 24 |
| Благодарности | 26 |
| Контактная информация | 26 |
| | |
| 1 Основы функционального программирования | 27 |
| <i>В этой главе рассматриваются основополагающие принципы функционального программирования как отдельного направления в математической науке и технологии создания программного обеспечения. Приводится история развития функционального программирования, описываются предпосылки его развития. В главе рассматривается больше теоретического материала, нежели практического, поэтому пока изложение ведется без рассмотрения синтаксиса языка Haskell. Однако для приведения примеров используется именно этот язык наряду с математическими формулами. Изложение знаний о языке Haskell начнется с главы 2.</i> | |
| 1.1 История функционального программирования | 28 |
| <i>Краткая история развития теории функционального программирования в мире и в России. Разработка функциональных языков для подтверждения теоретических выкладок. Проблемы, с которыми столкнулись исследователи при разработке функциональных языков программирования. Современное состояние теории функционального программирования. Стандарт Haskell 98 как результат унификации и стандартизации процессов развития функционального программирования.</i> | |
| <i>Предпосылки создания функционального программирования</i> | <i>34</i> |
| <i>История языка Haskell</i> | <i>38</i> |

| | |
|---|----|
| Заключительные слова | 42 |
| 1.2 Основные свойства функциональных языков | 44 |
| <i>Описание основных свойств функциональных языков программирования. Краткость и простота, строгая типизация, модульность, функциональные значения и объекты, чистота и отложенные вычисления. Понимание свойств функциональных языков на примере языка Haskell.</i> | |
| Краткость и простота | 44 |
| Строгая типизация | 48 |
| Модульность | 50 |
| Функции — это значения и объекты вычисления | 51 |
| Чистота (отсутствие побочных эффектов и детерминированность) | 53 |
| Отложенные (ленивые) вычисления | 55 |
| 1.3 Типовые задачи, решаемые методами ФП | 58 |
| <i>Краткое описание семи типовых задач, которые решаются методами функционального программирования. Получение остаточной процедуры. Построение математического описания функций. Определение формальной семантики языка программирования. Описание динамических структур данных. Автоматическое построение «значительной» части программы по описанию структур данных, которые обрабатываются создаваемой программой. Доказательство наличия некоторого свойства программы. Эквивалентная трансформация программ.</i> | |
| Получение остаточной процедуры | 60 |
| Построение математического описания функций | 62 |
| Определение формальной семантики языка программирования | 64 |
| Описание динамических структур данных | 64 |
| Автоматическое построение функций по описанию структур данных | 66 |
| Доказательство наличия некоторого свойства программы | 67 |
| Эквивалентная трансформация программ | 68 |
| 1.4 Конструирование функций | 69 |
| <i>Описание метода конструирования функций, предложенного Ч. Хоаром (синтаксически ориентированное конструирование). Метаязык для конструирования функций. Примеры определения типов и функций для обработки этих типов.</i> | |
| Декартово произведение | 70 |
| Размеченное объединение | 71 |

| | |
|---|------------|
| Примеры определения типов данных | 72 |
| 1.5 Доказательство свойств функций | 86 |
| <i>Задача доказательства свойств функций. Описание процесса доказательства свойств функций в зависимости от типа области определения функций. Примеры доказательства свойств функций.</i> | |
| Область определения D — линейно-упорядоченное множество | 88 |
| Множество D определяется как индуктивный класс | 89 |
| Рассмотрение некоторых примеров доказательства свойств функций | 91 |
| Вопросы для самоконтроля | 95 |
| Задачи для самостоятельного решения | 97 |
| 2 Базовые принципы языка Haskell | 99 |
| <i>Глава посвящена введению в основные положения языка Haskell, приводится описание синтаксиса для решения основных задач по созданию отдельных функций и законченных модулей. Рассматриваются базовые объекты «список» и «функция» для изучения в рамках функционального программирования, а также их реализация на языке Haskell.</i> | |
| 2.1 Списки — основа функциональных языков | 100 |
| <i>Понятие списка в функциональном программировании. Списки как основная структура для работы с функциональными языками. Базисные операции для работы со списками. Списки и списочные структуры. Программная реализация списков в функциональных языках. Списки в языке Haskell. Генераторы списков и математические последовательности. Бесконечные списки и другие структуры данных. Кортежи.</i> | |
| Проекция списков в язык Haskell | 101 |
| Несколько слов о программной реализации | 104 |
| Примеры | 106 |
| Определители списков и математические последовательности | 110 |
| Кортежи | 117 |
| 2.2 Списки — основа функциональных языков | 118 |
| <i>Функция — основной объект изучения функционального программирования. Соглашения по именованию объектов в языке Haskell. Описание и определение функций на языке Haskell. Образцы и клозы. Передача параметров и возвращение значений функциями. Инфиксный способ записи функций. Функция как объект</i> | |

для передачи в другие функции. Программа на языке Haskell — функция, описывающая процесс вычисления.

| | |
|--|------------|
| Соглашения по именованию | 118 |
| Общий вид определения функции | 119 |
| Образцы и клозы | 119 |
| Вызовы функций | 125 |
| Использование λ -исчисления | 126 |
| Инфиксный способ записи функций | 127 |
| Несколько слов о функциях высшего порядка | 131 |
| 2.3 Списки — основа функциональных языков | 131 |
| <i>Структуры и типы данных. Типы функций. Каррированные и некаррированные функции. Язык Haskell и его механизмы для организации каррированных и некаррированных функций. Описание типов функций на языке Haskell. Частичное применение. Ленивые (отложенные) вычисления на языке Haskell.</i> | |
| Структуры данных и их типы | 132 |
| Синонимы типов | 136 |
| Типы функций в функциональных языках | 137 |
| Полиморфные типы | 140 |
| 2.4 Списки — основа функциональных языков | 142 |
| <i>Отражающие выражения и конструкции. Локальные переменные для оптимизации кода на функциональном языке и на языке Haskell. Использование накапливающего параметра (аккумулятора) для оптимизации процесса вычислений. Принципы построения определений функций с накапливающим параметром. Головная и хвостовая рекурсии.</i> | |
| Охрана | 143 |
| Ветвление алгоритма | 145 |
| Локальные переменные | 146 |
| Двумерный синтаксис | 149 |
| Накапливающий параметр — аккумулятор | 150 |
| Принципы построения определений с накапливающим параметром | 152 |
| 2.5 Списки — основа функциональных языков | 153 |
| <i>Модули как способы структуризации и организации программ на языке Haskell. Импорт и экспорт данных при помощи модулей. Сокращение данных. Абстрактные типы данных и интерфейсы. Иные аспекты использования модулей.</i> | |

| | |
|---|------------|
| <i>Абстрактные типы данных</i> | 156 |
| <i>Другие аспекты использования модулей</i> | 157 |
| <i>Литературный код</i> | 158 |
| Вопросы для самоконтроля | 160 |
| Задачи для самостоятельного решения | 161 |
| 3 Классы и их экземпляры | 164 |
| <i>Эта глава посвящена рассмотрению симбиоза парадигм функционального и объектно-ориентированного программирования. Большинство современных функциональных языков поддерживают механизмы и методы, разработанные в рамках объектно-ориентированного программирования, в том числе и такие базовые концепты, как «наследование», «инкапсуляция» и «полиморфизм». Не обошел своим вниманием этот аспект и язык Haskell, в котором имеются достаточные средства для программирования в объектно-ориентированном стиле.</i> | |
| 3.1 Параметрический полиморфизм данных | 165 |
| <i>Понятие класса и его реализации в языке Haskell. Чистый (параметрический) полиморфизм на языке Haskell. Примеры параметрического полиморфизма в императивных и функциональных языках, а также в языке Haskell.</i> | |
| 3.2 Классы в языке Haskell как способ абстракции действительности | 170 |
| <i>Расширенное описание понятия класса в языке Haskell. Класс как высшая абстракция данных и методов для их обработки. Методы класса — шаблоны функций для реализации обработки данных. Минимальное описание методов класса и связь методов.</i> | |
| <i>Модель типизации Хиндли-Милнера</i> | 171 |
| <i>Определение классов</i> | 176 |
| 3.3 Наследование и реализация | 180 |
| <i>Наследование классов и наследование методов. Экземпляры классов — реализация интерфейсов, предоставляемых реализуемым классом. Реализация методов для обработки данных. Класс — шаблон типа, реализация класса — тип данных.</i> | |
| <i>Наследование</i> | 180 |
| <i>Реализация</i> | 182 |
| <i>Реализация для существующих типов</i> | 186 |
| <i>Сорта типов</i> | 187 |
| <i>Дополнительные возможности при определении типов данных</i> | 189 |

| | | |
|-----|--|------------|
| 3.4 | Стандартные классы языка Haskell | 192 |
| | <i>Краткое описание всех стандартных классов, разработанных для облегчения программирования на языке Haskell. Дерево наследования стандартных классов. Типичные способы использования стандартных классов языка Haskell. Реализация стандартных классов — типы в языке Haskell.</i> | |
| | Класс <i>Bounded</i> | 192 |
| | Класс <i>Enum</i> | 193 |
| | Класс <i>Eq</i> | 195 |
| | Класс <i>Floating</i> | 195 |
| | Класс <i>Fractional</i> | 196 |
| | Класс <i>Functor</i> | 197 |
| | Класс <i>Integral</i> | 198 |
| | Класс <i>Ix</i> | 199 |
| | Класс <i>Monad</i> | 199 |
| | Класс <i>Num</i> | 200 |
| | Класс <i>Ord</i> | 201 |
| | Класс <i>Read</i> | 202 |
| | Класс <i>Real</i> | 203 |
| | Класс <i>RealFloat</i> | 203 |
| | Класс <i>RealFrac</i> | 204 |
| | Класс <i>Show</i> | 206 |
| 3.5 | Сравнение с другими языками программирования | 207 |
| | <i>Более или менее полное сравнение понятий «класс» и «реализация класса» в языке Haskell с объектно-ориентированными языками программирования (на примере языков C++ и Java, а также некоторых других языков). Глобальные отличия понятия «класс» в функциональных и объектно-ориентированных языках.</i> | |
| | Окончательные замечания | 209 |
| | Вопросы для самоконтроля | 210 |
| | Задачи для самостоятельного решения | 211 |
| 4 | Монады — последовательное выполнение действий в функциональной парадигме | 213 |
| | <i>Глава описывает такое незаурядное понятие, введенное в функциональной парадигме программирования, как «монада». Монады, основанные на математической теории категорий, позволяют внедрить в функциональный подход опре-</i> | |

деленные структуры для выполнения императивных действий, как, например, операции ввода/вывода, обработка исключений, хранение состояний в процессе вычислений, и многие другие действия, связанные с побочными эффектами. Вместе с тем монады позволяют обернуть императивные действия в функциональную оболочку, спрятав все от «императивного мира» внутри монады.

| | | |
|-----|--|------------|
| 4.1 | Монада как тип-контейнер | 214 |
| | <i>Описание монады как типа-контейнера. Использование монад в функциональных языках. Свойства монадических типов. Операции связывания с передачей и без передачи результата выполнения операции на предыдущем шаге. Правила построения монад.</i> | |
| | <i>Определение понятия «монада»</i> | <i>215</i> |
| | <i>Нотация <code>do</code></i> | <i>218</i> |
| | <i>Правила построения монад</i> | <i>220</i> |
| 4.2 | Последовательное выполнение действий | 222 |
| | <i>Действие — элемент функциональной парадигмы. Императивный код внутри функционального. Выполнение действий и возвращение результата. Сокращенный способ записи последовательности действий. Списки действий. Программирование при помощи действий.</i> | |
| | <i>Класс <code>Computations</code></i> | <i>224</i> |
| | <i>Монада <code>State</code></i> | <i>227</i> |
| 4.3 | Операции ввода/вывода в языке Haskell | 239 |
| | <i>Более или менее полное описание базовых операций ввода/вывода в языке Haskell. Монада <code>IO</code>. Обработка исключений. Использование файлов, каналов и обработчиков. Нарушение теоретических принципов функционального программирования в монаде <code>IO</code>.</i> | |
| | <i>Действия ввода/вывода</i> | <i>240</i> |
| | <i>Программирование при помощи действий</i> | <i>244</i> |
| | <i>Обработка исключений</i> | <i>245</i> |
| | <i>Файлы и потоки</i> | <i>247</i> |
| | <i>Окончательные замечания</i> | <i>251</i> |
| 4.4 | Стандартные монады языка Haskell | 252 |
| | <i>Подробное описание монадических типов в стандартной библиотеке языка Haskell. Назначение и применимость монадических типов. Примеры использования стандартных монадических типов (кроме списков и монады <code>IO</code>). Модуль <code>Monad</code>. Монады <code>Glasgow Haskell Compiler</code>.</i> | |

| | |
|---|------------|
| Модуль <i>Monad</i> | 253 |
| Стандартные монады | 257 |
| 4.5 Разработка собственных монад | 262 |
| <i>Критерии возможности и необходимости разработки собственного монадического типа. Комбинирование монадических вычислений. Преобразователи монад. Примеры преобразования.</i> | |
| Комбинирование монадических вычислений | 263 |
| Преобразователи монад | 264 |
| Пример с преобразователем <i>StateT</i> | 267 |
| Окончательные замечания | 268 |
| Вопросы для самоконтроля | 269 |
| Задачи для самостоятельного решения | 270 |
| 5 Комбинаторная логика и λ-исчисление | 273 |
| <i>В главе рассматриваются основополагающие теоретические формализмы, которые стояли у истоков функционального программирования, а именно комбинаторная логика и λ-исчисление, разработанные в качестве расширений формальной логики и теории множеств в начале XX в. Приводятся самые основы этих направлений дискретной математики, достаточные для понимания сути того, что в свое время стояло за парадигмой функционального программирования.</i> | |
| 5.1 Основы комбинаторной логики | 274 |
| <i>Введение в комбинаторную логику. Поверхностное описание принципов комбинаторной логики. Комбинаторы и вычисления при помощи комбинаторов. Базисы в комбинаторной логике. Использование базисных комбинаторов для выражения любых вычислительных процессов. Числа и иные математические объекты в виде комбинаторов.</i> | |
| Базовые комбинаторы | 274 |
| Комбинатор неподвижной точки | 281 |
| Нумералы и арифметические операции | 283 |
| Заключительные слова | 286 |
| 5.2 Абстракция функций как вычислительных процессов | 288 |
| <i>Функция — объект математического исследования. Вычислительный процесс — функция. Описание функций как λ-выражений. Свободные и связанные идентификаторы.</i> | |

| | |
|---|------------|
| <i>фикаторы. Применение (апликация) значений к λ-выражениям. Непрерывная точка функций и теорема о непрерывной точке.</i> | |
| «Наивное» определение λ -исчисления | 289 |
| Связь с комбинаторной логикой | 292 |
| Редукция | 293 |
| Тезис Черча-Тьюринга | 294 |
| 5.3 λ-исчисление как теоретическая основа функционального программирования | 298 |
| <i>Предположение о том, что любая функция представима в виде λ-выражения. Интенционал и экстенционал функций. Формальная система. Построение формальной системы для обоснования теории функционального программирования. Правила вывода. Соответствия между вычислениями функциональных программ и редукцией λ-выражений.</i> | |
| Построение формальной системы | 299 |
| Функциональное программирование как формальная система | 303 |
| Теорема Черча-Россера | 305 |
| 5.4 Кодирование данных в λ-исчислении | 307 |
| <i>Механизм кодирования данных в λ-исчислении. λ-исчисление — достаточный формализм для представления значений истинности, упорядоченных пар, натуральных чисел, списков, а также базовых операций над этими объектами.</i> | |
| Булевские значения | 308 |
| Упорядоченные пары | 309 |
| Натуральные числа | 311 |
| Списки | 314 |
| 5.5 Редукция и вычисления в функциональных языках | 315 |
| <i>Понятие редукции. Частичные вычисления с точки зрения редукции λ-выражений. Различные редукционные стратегии и их свойства.</i> | |
| Стратегия редукции и стратегия вычислений | 315 |
| Ленивая редукция | 321 |
| Вопросы для самоконтроля | 327 |
| Задачи для самостоятельного решения | 329 |
| 6 Трансляторы программ | 331 |

В главе 6 представлено краткое описание теории построения трансляторов для интерпретации и компиляции языков программирования. Теория рассматривается на примерах, написанных на языке Haskell. Кроме того, рассматриваются способы построения парсеров, а также готовые библиотеки для синтаксического анализа. Суперкомпиляция.

| | | |
|-----|--|-----|
| 6.1 | Математическая лингвистика | 331 |
| | <i>Краткое введение в математическую лингвистику. Обзор методов и принципов математической лингвистики. Классификация языков и грамматик. Конечные автоматы и контекстно-свободные языки. Грамматики типа $LL(k)$. Контекстно-зависимые грамматики.</i> | |
| | Базовые понятия | 332 |
| | Расширенная нотация Бэкуса — Наура | 335 |
| | Классификация грамматик | 337 |
| | Конечные лингвистические автоматы | 338 |
| | Синтаксический анализ контекстно-свободных языков | 343 |
| 6.2 | Краткое введение в теорию построения трансляторов | 346 |
| | <i>Трансляторы: определения, типы и классификация, применимость для тех или иных типов языков и грамматик. Интерпретаторы и компиляторы. Компиляторы компиляторов. Методы построения трансляторов. Автоматическое построение транслятора по грамматике языка (для ограниченного множества языков). Понятие трансформационной грамматики.</i> | |
| | Классификация трансляторов и их типовые структуры | 347 |
| | Трансформационные грамматики | 354 |
| | Автоматическое построение анализатора для отдельных типов языков | 358 |
| 6.3 | Реализация трансляторов на языке Haskell | 360 |
| | <i>Функциональные языки, как естественный инструмент реализации трансляторов. Синтаксические анализаторы. Методы написания трансляторов на языке Haskell. Примеры синтаксических анализаторов для различных форматов данных. Пример вычисления арифметических выражений, записанных в различной нотации.</i> | |
| | Простейшие парсеры | 361 |
| | Комбинаторы синтаксического анализа | 363 |
| | Дополнительные комбинаторы синтаксического анализа | 366 |
| | Анализ нотации Бэкуса — Наура | 368 |

| | | |
|-----|--|------------|
| 6.4 | Библиотеки для создания трансляторов | 372 |
| | <i>Описание имеющихся библиотек для языка Haskell, предназначенных для создания трансляторов. Монодическая библиотека Parsec для самостоятельного создания трансляторов. Компилятор компиляторов Happy.</i> | |
| | <i>Монодическая библиотека парсеров Parsec</i> | <i>372</i> |
| | <i>Компилятор компиляторов Happy</i> | <i>377</i> |
| 6.5 | Частичные вычисления, трансформация программ и суперкомпиляция | 381 |
| | <i>Задача трансформации программ. Частичные вычисления, как инструмент для трансформации программ. Частичный вычислитель — инструмент для получения остаточного кода заданной функциональной программы. Интерпретатор, компилятор и компилятор компиляторов, их связь. Проекция Футамуры-Турчина. Суперкомпиляция.</i> | |
| | <i>Частичные вычисления и трансляция программ</i> | <i>382</i> |
| | <i>Проекция Футамуры — Турчина</i> | <i>385</i> |
| | <i>Трансформация программ</i> | <i>387</i> |
| | Вопросы для самоконтроля | 392 |
| | Задачи для самостоятельного решения | 394 |
| 7 | ФП и искусственный интеллект | 395 |
| | <i>Заключительная глава книги рассматривает такую область человеческого знания, как искусственный интеллект, то есть методы решения слабоформализованных задач, для которых не существует алгоритмического решения либо такое решение слишком сложно. Такой интерес связан с тем, что именно парадигма функционального программирования нашла свое непосредственное применение в рамках искусственного интеллекта.</i> | |
| 7.1 | Основные задачи искусственного интеллекта | 396 |
| | <i>Историческая справка о развитии искусственного интеллекта, как области научного исследования. Введение в базовые понятия искусственного интеллекта. Место функционального программирования в искусственном интеллекте. Функциональное и логическое программирования. Задачи искусственного интеллекта, которые могут быть решены при помощи методов и средств функционального программирования.</i> | |
| | <i>История развития искусственного интеллекта</i> | <i>398</i> |
| | <i>Различные подходы к построению систем искусственного интеллекта</i> | <i>402</i> |
| | <i>Место функционального программирования в искусственном интеллекте</i> | <i>405</i> |

| | | |
|-----|---|-----|
| 7.2 | Нечеткая математика и функциональное программирование | 407 |
| | <i>Небольшой экскурс в нечеткую математику. Функции принадлежности и лингвистические переменные. Базовые операции над функциями принадлежности. Кусочно-линейные функции принадлежности. Операции сравнения, арифметические и логические операции над кусочно-линейными функциями принадлежности. Использование языка Haskell для реализации методов обработки кусочно-линейных функций принадлежности.</i> | |
| | Базовые концепты нечеткой логики | 407 |
| | От нечеткой логики к нечеткой математике | 409 |
| | Функции принадлежности как способ описания нечетких значений | 411 |
| | Нечеткие и лингвистические переменные | 417 |
| | Операции над функциями принадлежности | 418 |
| | Пример модуля для обработки кусочно-линейных функций принадлежности | 423 |
| 7.3 | Логический вывод на знаниях | 429 |
| | <i>Знания и данные. Модели представления знаний. Понятие логического вывода на знаниях. Стратегии вывода на знаниях. Машины вывода. Эволюция машинного вывода на знаниях. Интерпретаторы функциональных языков как естественные машины вывода. Язык Haskell и его возможности в логическом выводе на знаниях. Универсальный вывод на продукционной модели знания.</i> | |
| | Знания и данные | 430 |
| | Вывод на знаниях | 434 |
| | Прямой нечеткий вывод | 437 |
| | Обратный нечеткий вывод | 439 |
| | Некоторые окончательные замечания о машинном выводе | 442 |
| 7.4 | Общение с компьютером на естественном языке | 443 |
| | <i>Принципы общения с компьютерными системами на естественном языке. Ограниченный естественный язык, деловая проза. Трансляция фраз на естественном языке во внутренний язык представления смысла. Понимание текстов на естественном языке. Использование методов функционального программирования.</i> | |
| | Обобщенная схема интеллектуальных диалоговых систем | 444 |
| | Схема анализа входного текста | 447 |
| | Некоторые окончательные замечания | 453 |
| 7.5 | Перспективы функционального программирования | 454 |

Описание видения будущего функционального программирования, функциональных языков и языка Haskell. Значение функциональной парадигмы для технологии программирования вообще.

| | |
|---|-----|
| Вопросы для самоконтроля | 460 |
| Задачи для самостоятельного решения | 461 |

Заключение 463

Ответы на задачи для самостоятельного решения 465

| | |
|--------------------------------|-----|
| Решения задач из главы 1 | 465 |
| Решения задач из главы 2 | 467 |
| Решения задач из главы 3 | 474 |
| Решения задач из главы 4 | 477 |
| Решения задач из главы 5 | 483 |
| Решения задач из главы 6 | 488 |

А Функциональные языки программирования и Интернет-ресурсы по функциональному программированию 496

Список наиболее известных и широко используемых функциональных языков программирования с краткой аннотацией. Список Интернет-ресурсов, посвященных функциональному программированию как в русском сегменте Интернета, так и во всем остальном мире.

| | |
|---|-----|
| Функциональные языки программирования | 496 |
| Русские Интернет-ресурсы | 502 |
| Иностранные Интернет-ресурсы | 503 |

В Опции различных сред разработки на языке Haskell 506

Описание типичных настроек интегрированных сред разработки и компиляторов на примере продуктов HUGS 98 и GHC для полноценной работы и выполнения различных задач на языке Haskell. Список команд, ключей командной строки и внутренних директив интерпретатора HUGS 98. Список параметров командной строки для компилятора GHC. Другие функциональные средства разработки.

| | |
|--|-----|
| Интегрированная среда разработки HUGS 98 | 506 |
| Компилятор GHC | 510 |
| Компилятор GHC | 523 |
| Компилятор компиляторов Narro | 528 |

| | | |
|----------|---|------------|
| С | Описание стандартного модуля Prelude | 531 |
| | <i>Список функций из стандартного модуля языка Haskell Prelude с более или менее подробным описанием.</i> | |
| | Функции | 531 |
| | Описание некоторых операторов языка Haskell | 586 |
| D | Краткий словарь терминов из области функционального программирования | 589 |
| | <i>Краткий словарь терминов, имеющих отношение к функциональному программированию. для каждого термина даются ссылки на страницы, где он описан, а также переводы на некоторые иностранные языки.</i> | |
| | Литература | 599 |
| | Общая литература по функциональному программированию | 599 |
| | Книги, руководства и статьи по языку Haskell | 601 |
| | Комбинаторная логика и λ -исчисление | 602 |
| | Математическая лингвистика и теория построения трансляторов | 603 |
| | Искусственный интеллект | 604 |

Введение

Данная книга является первым в России изданием, рассматривающим функциональное программирование в полном объеме, достаточном и для понимания новичку, и для использования книги в качестве справочного пособия теми, кто уже использует парадигму¹ функционального программирования в своей практике. Эту книгу можно использовать и в качестве учебника по функциональному программированию, и в качестве вспомогательного учебного пособия по смежным дисциплинам, в первую очередь по комбинаторной логике и λ -исчислению². Также книга будет интересна тем, кто всерьез занимается изучением новых компьютерных технологий, искусственного интеллекта и синергетическим слиянием научных направлений человеческой деятельности.

Необходимость написания подобной книги в первую очередь вызвана тем, что на русском языке нет полноценного справочного и учебного пособия по языку Haskell, в то время как этот функциональный язык все больше и больше завоевывает сердца программистов по всему миру. Более того, этот язык уже используют и для написания полноценных программных систем, в том числе для коммерческого использования. Однако в России изучение языка Haskell проведется лишь энтузиастами, а в строгой академической школе довольствуются традиционным рассмотрением языка Lisp в качестве иллюстрации основ функционального программирования.

¹ Под парадигмой (от греч. *παράδειγμα* — пример, модель, образец) здесь и далее понимается общая концептуальная схема постановки проблем и методов их решения. В более узком смысле под парадигмой программирования будет пониматься не просто стиль написания программ, а способ мышления, который позволяет использовать тот или иной стиль при создании таких программ.

² Основы комбинаторной логики и λ -исчисление кратко рассматриваются в главе 5 этой книги.

Однако если рассматривать англоязычные источники, руководства и учебники по языку Haskell, то налицо сложности, с которыми сталкиваются те, кто начинает самостоятельно изучать этот функциональный язык. Во-первых, это скупой и формальный подход при описании синтаксиса языка — на этом принципе основаны все руководства по языку Haskell. Во-вторых, это отсутствие какой-либо системности, или даже присутствие антисистемности в изложении как основ функционального программирования, так и способов программирования на языке Haskell. Например, в учебнике «Haskell: The Craft of Functional Programming» автор перескакивает с одной темы на другую безо всяких переходов между ними. Начинает с описания списков, а продолжает принципами создания программного обеспечения вообще. Описывает систему классов, перескакивает на описание системы ввода/вывода практически без затрагивания темы монад. А это — один из основных учебников по языку Haskell на английском языке.

Книга рассчитана на всех, кто интересуется современными тенденциями развития компьютерной науки и технологии, а также искусственным интеллектом. Она может служить основным или дополнительным учебным пособием для студентов и аспирантов, обучающихся по специальности «прикладная математика» и специализации «искусственный интеллект». Также книга будет полезна в качестве справочного материала по языку Haskell всем тем, кто использует функциональную парадигму в целом и этот функциональный язык в частности в научных исследованиях или повседневной работе.

Для чтения книги необходимо обладать базовыми познаниями в дискретной математике, а также иметь представление о методах разработки алгоритмах и алгоритмическом решении задач (теория алгоритмов). В процессе написания книги полагалось, что читатель знаком с базовыми концептами дискретной математики, поэтому лишние математические определения в книге не приводятся. Сложные разделы математики, представленные в заключительных главах, требуют более серьезных знаний, поэтому их описание сопровождается небольшими экскурсами в теорию. Это касается таких разделов дискретной математики, как комбинаторная логика, λ -исчисление, математическая лингвистика и теория построения трансляторов, а также базовые принципы искусственного интеллекта.

Рассмотрение парадигмы функционального программирования производится на примере языка Haskell, на котором написаны все исходные коды, представленные в книге. Все представленные примеры, функции, модули и исходные тексты программ проверены в интегрированной среде разработки HUGS 98 (за исклю-

чением тех, которые предназначены для компиляции в среде GHC — Glasgow Haskell Compiler). Для удобства читателя, желающего проверить и закрепить свои знания на практике, все приведенные в книге программы и функции размещены на сайте издательства www.dmkpress.com

Книгу нельзя рассматривать в качестве скрупулезного введения в язык Haskell и программирования на нем, так как цель автора — не преподать некоторую догму относительно использования языка Haskell, но направить читателя на путь, следуя которому он сам сможет понять, что и как необходимо делать для получения правильных программ. В связи с этим в книге не рассматриваются вопросы о том, какие инструкции на языке необходимо написать, чтобы получить определенный эффект (или рассматриваются в самом минимальном виде), но предлагаются пути решения разных задач, встающих перед программистом. Именно поэтому главы, непосредственно посвященные языку Haskell, написаны сухим языком. Цель этих глав — преподнести читателю как можно больше информации о синтаксисе языка без лишнего упоминания о том, как использовать предлагаемые синтаксические конструкции. Для этого необходимо обратиться к последним главам книги, где, однако, упоминание о языке Haskell сведено к минимуму. Такой формат представления информации позволит (и даже принудит) читателю самостоятельно следовать по пути использования парадигмы функционального программирования.

Также на прилагаемом CD-диске можно найти текст данной книги в формате Adobe PDF (Portable Document Format) для использования его в личных нуждах.

Краткая биография автора

Душкин Роман Викторович. С 2001 г. читает лекции по функциональному программированию для студентов четвертого курса кафедры кибернетики (№ 22) факультета «К» Московского инженерно-физического института (МИФИ). Базисом для авторского курса по функциональному программированию послужил текст лекций Г. М. Сергиевского, читавшихся до 2001 г. Данные лекции были кардинально переработаны с учетом современных веяний в науке и технике, основа была переведена с языка Lisp на Haskell (вместе с полной переработкой курса лабораторных работ), а сами лекции были дополнены строгим научным обоснованием как самой парадигмы функционального программирования, так и ее прикладных аспектов.

Р. В. Душкин является автором множества научных публикаций по темам нечеткой математики, искусственного интеллекта и функционального программирования в российских и зарубежных научных изданиях. Участвовал во множестве национальных и международных научных конференций, проводимых под эгидой Российской Ассоциации Искусственного Интеллекта. Издал ряд методических пособий, в том числе и для выполнения лабораторных работ по функциональному программированию.

Р. В. Душкин работает в области создания автоматизированных систем управления на железнодорожном транспорте, на практике используя все методы создания программных средств, применяющиеся в составе парадигм функционального и объектно-ориентированного программирования, а также искусственного интеллекта. Входил в состав команды разработчиков и управлял самим процессом разработки множества автоматизированных систем, в том числе и реального времени, для информационной и технологической поддержки процессов управления в различных областях железнодорожной отрасли России.

О пользовании книгой

Для удобства читателей при наборе текста книги использовались шрифты различных начертаний для выделения тех или иных особенностей представленной информации.

Для написания имен функций, элементов формул и математических выражений, которые приводятся непосредственно в тексте книги, используется курсивное начертание. Например: $x \in C$, f_0 , $List(A)$.

Примеры фрагментов программ на языке Haskell и изредка на абстрактном функциональном языке, который используется для объяснения теоретических аспектов применения парадигмы функционального программирования, в тексте книги приводятся при помощи моноширинной гарнитуры (машинописного шрифта):

```
length [] = 0
```

```
length (x:xs) = 1 + length xs
```

Упоминания об именах функций непосредственно в тексте также выделяются моноширинным начертанием символов: `length`, `append`. Кроме того, таким же образом выделяются наименования модулей, стандартных библиотек и дополнительных программных средств: `Prelude`, `Parsec`, `Happy`.

Иногда в тексте встречаются обозначения операций, наименования которых состоят из одного или более нелитеральных символов. Для того чтобы как-то отделять подобные обозначения от текста книги, такие операции заключаются в круглые скобки, а сами обозначения приводятся моноширинным шрифтом. Например: $(+)$, $(<@)$, $(>=)$. Сами скобки различного вида в круглые скобки не заключаются: $\{ \}$, $[]$ и т. д. в случае, если необходимо показать отдельную скобку, она заключается в кавычки: $\langle (\rangle$, $\langle \} \rangle$ и т. д.

Листинги больших и законченных модулей на языке Haskell приводятся в виде выделенных блоков:

Листинг 0.1. Пример текста программы из внешнего файла

```
-----  
-----  
--  
-- Модуль INTROEXAMPLE - пример правильно описанного модуля на языке Haskell. --  
--  
-----  
-----  
module IntroExample  
  (someFunction)  
where  
  
-----  
-- Некоторая функция, выполняющая определённые действия.  
--  
-- Входные параметры:  
--   n      - число, которое необходимо прибавить к каждому элементу списка.  
--   (x:xs) - список, к каждому элементу которого необходимо прибавить число n.  
--  
-- Возвращаемое значение:  
--   Список, составленный из сумм заданного числа n с элементами исходного  
--   списка.  
  
someFunction :: Int -> [Int] -> [Int]  
someFunction n [] = []  
someFunction n (x:xs) = (x + n) : someFunction n xs  
  
--[ КОНЕЦ МОДУЛЯ ]-----
```

Еще раз необходимо заметить, что все приведенные в книге листинги читатель сможет найти на прилагаемом к книге CD-диске или на официальном сайте книги в Интернете.

Состав и структура представления информации

Книга разбита на семь больших глав, каждая из которых посвящена отдельному аспекту парадигмы функционального программирования. Каждая глава разбита на пять разделов, которые логически делят главу на несколько отдельных областей рассмотрения.

Первая глава является своеобразным введением в проблематику — в ней приводится базовый методический материал, необходимый для понимания основ функционального программирования и дальнейшего чтения книги. По существу, первая глава — это пролог, который открывает путь в функциональный мир.

Вторая глава повествует о базовых принципах языка Haskell, в ней приводятся самые основные сведения о синтаксисе, структуре программ и модулей, взаимодействии функций, организации исходного кода и прочих аспектах использования нового языка программирования. Данную главу можно использовать в качестве учебного пособия для тех, кто впервые приступил к изучению языка Haskell.

Третья глава посвящена слиянию функциональной и объектно-ориентированной парадигм программирования, что должно представить язык Haskell в качестве современного средства разработки компьютерных приложений. Рассматривается такое базовое понятие объектно-ориентированного программирования, как «класс», и его применение в составе средств языка Haskell. Кроме того, приводится более или менее полное описание стандартных классов и их экземпляров, поставляемых в составе базового модуля `Prelude`.

Четвертая глава рассматривает такую незаурядную вещь, как «монада» — расширение парадигмы функционального программирования для включения в нее императивного подмножества структур языка Haskell для выполнения операций, связанных с использованием побочных эффектов (в первую очередь операций ввода/вывода). Так же как и в третьей главе, приводится подробное описание стандартных монад, которое поможет понять этот непростой объект и полноценно использовать язык Haskell для разработки сложных приложений.

По существу, для изучения самого языка Haskell достаточно прочитать первые четыре главы. С пятой главы начинается рассмотрение теории и услож-

ненного материала, который поможет углубить понимание функционального программирования и вывести читателя на новый уровень знаний.

Пятая глава рассматривает теоретические основы функционального программирования, из которых, собственно, данная парадигма и вышла в первой половине XX в. Приводится описание комбинаторной логики, разработанной Х. Карри, и λ -исчисления, введенного в математический аппарат А. Черчем. Приводятся основные аспекты данных математических формализмов, которые помогут понять смысл и суть работы интерпретаторов функциональных языков, в том числе и интерпретаторов языка Haskell.

Шестая глава посвящена математической лингвистике, которая является теоретическим механизмом для построения трансляторов различных языков программирования. В главе приводятся базовая теория, принципы построения трансляторов на языках программирования, а также применение теории к самому языку Haskell — на примерах показано написание интерпретатора языка Haskell на нем самом.

Седьмая глава повествует о такой области человеческого знания, как искусственный интеллект, в котором парадигма функционального программирования нашла самое непосредственное применение. Глава не ставит целью дать скрупулезное исследование темы искусственного интеллекта, но вкратце рассматривает несколько основных задач, которые традиционно решаются методами искусственного интеллекта. Естественно, что все рассмотрение происходит в канве изучения языка Haskell.

Также в книге имеются четыре приложения, где собран интересный материал, который может помочь в деле изучения стези функционального программирования, но который, однако, выходит за рамки общей канвы книги. Вдумчивый читатель найдет в этих приложениях интересные для себя вещи, которые позволят ему отправиться в свободное плавание по океану функционального программирования.

Материал представлен таким образом, что не каждую главу и не каждый раздел в любой главе необходимо читать для понимания и проникновения в суть отдельных моментов в парадигме функционального программирования. В начале каждой главы и каждого раздела внутри главы имеется краткая аннотация для соответствующего уровня рубрикации, по которой читатель сможет понять и осмыслить необходимость изучения главы или раздела.

Благодарности

Первая благодарность выражается В. А. Роганову за тот импульс, который был дан для того чтобы только сесть за написание этой книги. Без этого импульса идея книги продолжала бы лежать в ящике рабочего стола, так и не воплотившись в чем-то серьезном, пока не устарела бы морально. Без помощи А. Н. Преображенского и А. Ю. Фоменко верстка книги в системе L^AT_EX заняла бы слишком много времени, которое было бы отнято у непосредственного написания текста и изложения смысла.

Автор выражает благодарность всем своим студентам, обучавшимся на кафедре кибернетики в МИФИ, помогавшим делать курс лекций по функциональному программированию более адаптированным для понимания неподготовленными новичками. Особая благодарность всем тем, кто занимался технической работой по сбору, переводам, адаптации и верстке материалов, изданных на иностранных языках.

Автор благодарит всех людей, принявших участие в обсуждении черновиков книги и внесших свой посильный вклад в ее создание. Перечислить поименно всех таких людей ввиду их огромного количества не представляется возможным. Однако без их активного участия книга была бы неполной, пресноватой и лишенной того шарма, который привносит в любую книгу интерактивное обсуждение в процессе ее создания.

Особо необходимо отметить помощь следующих людей, бескорыстно занимавшихся чтением и правкой черновиков книги, вносящих дельные комментарии и предложения: Д. А. Антонюк, В. Г. Владимиров, М. А. Забелин, И. О. Кабанова, Ю. Д. Лобарев, В. Н. Назаров, М. П. Трескин, А. В. Тупота, Д. А. Храпов.

Огромное сердечное спасибо выражается жене Елене и сыну Кириллу за то, что они есть, за их поддержку и полное понимание во время работы над рукописью.

Контактная информация

Автор будет рад получить от своих читателей любые замечания, комментарии и просто отзывы о данной книге по следующему адресу электронной почты: `darkus.14@gmail.com`

Официальный web-ресурс книги находится по адресу `fp.haskell.ru`.

Глава 1

Основы функционального программирования

В качестве базового методического материала, на котором строится все дальнейшее изложение книги, приводится исчерпывающий список основных свойств функциональных языков, на основе которого в дальнейшем показываются плюсы и минусы конкретных реализованных языков функционального программирования, в первую очередь языка Haskell (другие языки функционального программирования рассматриваются в приложении А). Также рассматриваются типовые задачи, которые решаются методами функционального программирования проще и легче, — собственно для таких задач в свое время и разрабатывались теоретические механизмы.

Для понимания последующих глав и описанных в них методов реализации функций в разделах 1.4 и 1.5 приводится небольшой экскурс в строгую теорию по конструированию функций и доказательству их свойств. Данный материал необходим для понимания основ понятия «функция», а также для более полного представления о глубине теоретической проработки, предшествовавшей созданию первых языков функционального программирования, что называется «в коде». При первом чтении книги эти разделы можно пропустить, вернувшись к ним после прочтения главы 5.

1.1 История функционального программирования

Функциональное программирование ставит своей целью придать каждой программе простую математическую интерпретацию. Эта интерпретация должна быть независима от деталей исполнения и понятна людям, которые не имеют научной степени в предметной области.

Лоренс Паулсон

Рассмотрение череды исторических фактов в рамках истории функционального программирования и функциональных языков можно начать цитатой из книги П. Худака «Концепция, эволюция и применение функциональных языков программирования»:

«Первый функциональный язык программирования был разработан для достижения простейшей цели: получения механизма для управления поведением компьютера. не секрет, что самые первые языки программирования полностью отражали структуру ЭВМ, на которых они исполнялись».

Однако прежде чем начать описание собственно функционального программирования, необходимо обратиться к истории программирования вообще. В 40-х г. XX в. появились первые цифровые компьютеры, которые, как известно, программировались при помощи переключения различного рода тумблеров, проводков и кнопок. Число таких переключений достигало порядка нескольких сотен и неумолимо росло с ростом сложности программ. Поэтому следующим шагом развития программирования стало создание всевозможных ассемблерных языков с простой мнемоникой, отражающей используемые машинные коды.

Так, для программирования различных процессорных устройств специалисты перестали замыкать при помощи проводков необходимые клеммы и стали писать что-то типа:


```
MOV AX, BX
ADD CX
RET
```

что в дальнейшем кодировалось либо в прорези на перфокартах или перфолен-тах, либо непосредственно в машинные коды (но это был уже следующий этап развития, на котором произошел отказ от ужасных перфокарт).

Однако ассемблеры не могли стать тем инструментом, которым смогли бы пользоваться обыкновенные люди, так как мнемокоды все еще оставались слишком сложными, в том числе и требовали от разработчика знания машинных кодов в шестнадцатеричной системе счисления, тем более что всякий ассемблер был жестко связан с архитектурой технических средств, на которых он исполнялся. Таким образом, следующим шагом после ассемблера стали так называемые императивные языки высокого уровня (например, BASIC, Pascal, C, Ada и прочие, включая объектно-ориентированные). Императивными такие языки были названы по той простой причине, что главным их свойством является ориентированность в первую очередь на последовательное исполнение инструкций, оперирующих с памятью (то есть присваиваний), и итеративные циклы. Вызовы функций и процедур, даже рекурсивные, не избавляли такие языки от явной императивности (предписания)¹.

Императивное свойство новых языков программирования более высокого уровня, нежели простой ассемблер, перенималось от ассемблера по традиции, так как первоначально языки программирования высокого уровня представляли собой просто набор обобщенных команд для сокращения количества ассемблерных инструкций. Данное положение лучше всего видно на примере языка BASIC (и схожих с ним), где какой-нибудь оператор с парой параметров транслировался напрямую в ассемблер (или непосредственно в машинный код, что, собственно, не так принципиально) при помощи простейших трансформационных правил. Например:

¹ Под императивным программированием (от лат. *imperativus* — повелительный) понимается такая парадигма программирования, в которой основным методом построения программ является задание последовательности шагов для исполнения, в том числе вызов процедур, поэтому императивное программирование иногда называется процедурным.

```
10 CLS
20 INPUT V
30 PRINT "V = " V
```

Можно видеть, что операторы в подобных записях являются всего лишь сокращениями нескольких ассемблерных инструкций, то есть сведениями нескольких (может быть, даже нескольких десятков) ассемблерных строк в одну запись.

Императивное программирование было первым шагом в улучшении ситуации с программированием различных архитектур, при помощи него описывалась последовательность команд, которая изменяла состояние компьютеров. Такое положение дел недалеко ушло от ассемблера. Самое главное улучшение, которое было привнесено императивными языками программирования, — это то, что инструкции, подобные `printf`, сокращали количество записей и позволяли не писать десятки ассемблерных команд по сохранению определенных битов в графической памяти. Типичными представителями этой парадигмы программирования стали такие языки программирования, как C и Ada. Оба этих языка широко используются до сих пор: C — в разработке операционных систем, а Ada — в разработке систем реального времени.

Развитие языков высокого уровня все еще наследовало за собой свойство императивности, и уже такие довольно абстрактные языки программирования, как объектно-ориентированные (C++, Java и т. д.), все еще не могли избавиться от явного предписания. И хотя программы на таких языках все больше и больше напоминают декларативные конструкции (например, описание класса — это определенно декларация), реализация методов класса представляет собой четкое описание императивного процесса:

```
class CStub
{
    private:
        int items[];
        int arrayLength;

    public:
        // Здесь должны быть описаны конструкторы, деструктор, иные функции...
        void nofItems ();
```

```
};  
  
//...  
void CStub::nofItems () {  
    int result = 0;  
    for (int i = 0; i < arrayLength; i++)  
    {  
        if (items[i])  
        {  
            result++;  
        }  
    }  
    return result;  
}
```

Это другой подход, который был разработан в рамках объектно-ориентированной парадигмы и реализован в таких языках программирования, как C++ или Java. Такие языки предоставляют многочисленные инструменты, основанные на современных теориях создания программного обеспечения, то есть легкое расширение имеющихся программ (классов) и модульность. В рамках объектно-ориентированной парадигмы разработаны шаблоны проектирования, которые позволяют создавать проекты сложных систем «на лету». Однако и здесь было нечто потеряно, а именно выразительность.

Возвращаясь к функциональному программированию... Краеугольным камнем в парадигме функционального программирования, как будет показано далее, является понятие «функция», и именно благодаря функции рассматриваемый здесь предмет получил свое наименование. Если вспомнить историю математики, то можно оценить возраст этого концепта. Ему уже около четырехсот лет (хотя можно полагать, что зачатки понимания о функциональной связи между входными и выходными параметрами знали еще математики и философы Древней Греции), и математика придумала бесчисленное множество теоретических и практических аппаратов для оперирования функциями, начиная от обыкновенных операций дифференцирования и интегрирования и заканчивая разного рода функциональными анализами, теорией нечетких множеств и теорией функций комплексного переменного.