Содержание

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 парадигму[1] 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-исчислению[2]. 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 проповедуется 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: «(», «}» 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 LaTeX 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, вносивших 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. Тупóта, 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 перфолентах, 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 сведéниями 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.