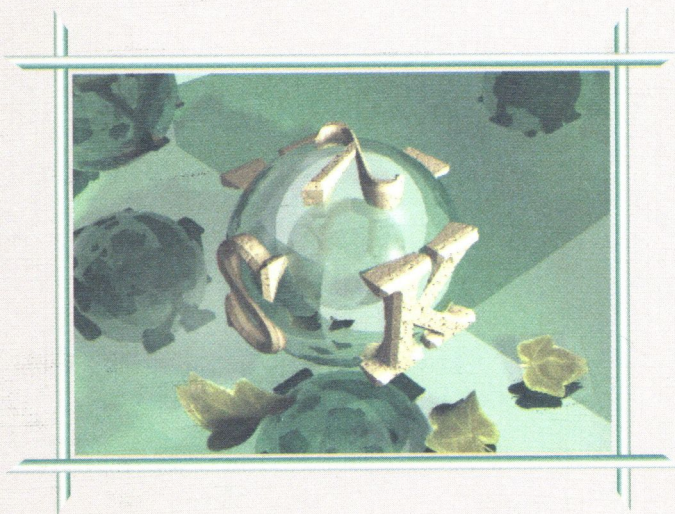


В. Э. Вольфенгаген

КОМБИНАТОРНАЯ ЛОГИКА В ПРОГРАММИРОВАНИИ



В. Э. Вольфенгаген

Комбинаторная

логика

в программировании



Серия: Компьютерные науки и информационные
технологии

Проект: *Аппликативные Вычислительные Системы*

Руководитель проекта, кандидат технических наук

Л.Ю. Исмаилова

Издано к 60-летию Московского инженерно-физического института

Институт актуального образования
“ЮрИнфоР-МГУ”

*Кафедра перспективных компьютерных исследований
и информационных технологий*

В. Э. Вольфенгаген

КОМБИНАТОРНАЯ ЛОГИКА В ПРОГРАММИРОВАНИИ

Вычисления с объектами в примерах и задачах

Издание 2-е



Москва
АО “Центр ЮрИнфоР”
2003



ББК 32.97
УДК 004
В721

Библиотека "ЮрИнфоР"

Основана в 1994 г.

Серия: Компьютерные науки и информационные
технологии

Автор: д. т. н., профессор **Вольфенгаген В. Э.**

Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах.— 2-е изд.— М.: АО "Центр ЮрИнфоР", 2003.— vi+336 с.

ISBN 5-89158-101-9

Изложен основной круг задач, сводимых к исчислению объектов — "от простого к сложному". Конкретный вариант исчисления выбирается в зависимости от решаемых вычислительных задач. В ходе последовательного решения задач читатель овладевает основными методами и средствами комбинаторной логики и λ -исчисления. Все задачи снабжены подробными и элементарными решениями. Для студентов старших курсов и аспирантов, изучающих математические основы объектно-ориентированных вычислений, начинающих и профессионально работающих над продвинутыми проектами программистов. Может быть использована в курсах дискретной математики, информатики, теории программирования. Предварительной математической подготовки не требуется. Материал частично или полностью может быть использован для самостоятельного изучения как книга "для первого чтения".

ISBN 5-89158-101-9

© В. Э. Вольфенгаген, 1987—2003

© АО "Центр ЮрИнфоР", 1997—2003

Зарегистрированный патент № 50863

Государственным реестром промышленных образцов РФ 16 августа 2002 г.

АО "Центр ЮрИнфоР"

Институт актуального образования ЮрИнфоР-МГУ

Fax: +7 (095) 956-25-12. E-mail: vew@jmsuice.msk.ru

Содержание

Предисловие редактора серии	1
Special Preface	3
Круг вопросов	5
Предисловие к первому изданию	7
Предисловие ко второму изданию	12
Введение	18
1 Предварительные сведения	27
1.1 Круг основных идей	29
1.2 Структура раздела	30
1.3 Состояние исследований	31
1.4 Типовая задача	35
1.5 Варианты задания	37
1.6 Рекомендуемый порядок выполнения задания . . .	44
2 Синтез нового объекта	45
2.1 Принцип комбинаторной полноты	46
2.1.1 Комбинаторная характеристика	46
2.1.2 Системы концептов	47

2.1.3	Комбинаторная полнота	48
2.1.4	Элементарная комбинаторная логика	49
2.2	Синтез основных комбинаторов: задачи	52
2.3	Исторические замечания	63
3	Неподвижная точка	65
3.1	Теоретические сведения	65
3.1.1	Абстракция	66
3.1.2	Мультиабстракция	67
3.1.3	Локальная рекурсия	67
3.2	Основные задачи	68
	Упражнения	73
4	Экстенциональность	75
4.1	Теоретические сведения	75
4.2	Задачи	77
	Упражнения	80
5	Нумералы	81
5.1	Числа и нумералы	81
5.2	Комбинаторная арифметика	82
5.3	Задачи	88
	Упражнения	92
6	Комбинаторы с типами	93
6.1	Представление о типе	93
6.1.1	Комбинаторные термы	96
6.1.2	λ -термы	97
6.2	Задачи	97
7	Базис I, K, S	113
7.1	Теоретические сведения	114
7.2	Задачи	114
	Упражнения	115

8	Базис I, B, C, S	117
8.1	Теоретические сведения	117
8.2	Свойство базисности	118
8.3	Элементарные примеры	120
	Упражнения	121
9	Применения неподвижной точки Y	123
9.1	Теорема о неподвижной точке	123
9.2	Элементы рекурсивных вычислений	124
9.3	Использование комбинатора Y	125
9.4	Вычисление функций	127
	Упражнения	131
10	Функция $list1$	133
10.1	Теоретические сведения	133
10.2	Задачи	134
10.3	Функтор-как-объект	137
	Упражнения	138
11	Изоморфизм д.з.к. и ABC	139
11.1	Теоретические сведения	139
11.2	Задачи	141
12	Каррирование	143
12.1	Теоретические сведения	143
12.1.1	Операторы и функции	143
12.1.2	Свертывание	144
12.1.3	Соответствие операторов и функций	144
12.2	Задачи	145
	Упражнения	147
13	Оболочка Каруби	149
13.1	Теоретические сведения	149
13.2	Задачи	150

Упражнения	155
14 Произведение и проекции	157
14.1 Теоретические сведения	157
14.2 Задача	158
14.3 Произведение и декартово замкнутая категория . .	162
15 Погружение Lisp в ABC	163
15.1 Теоретические сведения	163
15.2 Основная задача	164
15.3 Заключительные замечания	170
16 Суперкомбинаторы	171
16.1 Теоретические сведения	171
16.1.1 Понятие о суперкомбинаторе	172
16.1.2 Процесс компиляции	174
16.1.3 Приведение к суперкомбинаторам	175
16.1.4 Устранение избыточных параметров	177
16.1.5 Упорядочивание параметров	179
16.1.6 Ламбда-подъем при рекурсии	183
16.1.7 Работа алгоритма ламбда-подъема	186
16.1.8 Другие способы ламбда-подъема	189
16.1.9 Полная ленивость	191
16.1.10 Максимально свободные выражения	194
16.1.11 Ламбда-подъем с использованием МСВ . .	195
16.1.12 Полностью ленивый ламбда-подъем с <i>letrec</i>	197
16.1.13 Комплексный пример	199
16.2 Задача	201
16.3 Ответы к упражнениям	204
17 Ленивая реализация	213
17.1 Задачи	213
Упражнения	216

18 Перестановка параметров	217
18.1 Задача	217
Упражнения	222
Вопросы для самопроверки	222
19 Непосредственные вычисления	223
19.1 Задача	223
Упражнения	225
Вопросы для самопроверки	226
20 Код де Брейна	227
20.1 Задачи	227
Упражнения	232
21 Абстрактная машина: КАМ	233
21.1 Теоретические сведения	233
21.1.1 Структура КАМ	234
21.1.2 Инструкции	236
21.2 Задачи	240
Упражнения	242
22 Оптимизация КАМ-вычислений	243
22.1 Задача	243
Упражнения	252
Вопросы для самопроверки	253
23 Переменные объекты	255
23.1 Модели	255
23.1.1 Аппликативная структура	256
23.1.2 Модели с типами	260
23.1.3 Частичные элементы	267
23.1.4 Модели объектов данных	271
23.2 Основная задача	276
23.2.1 Элементарные типы	276

23.2.2 Типизация переменных объектов	278
23.2.3 Вычислительные модели	280
23.2.4 Индексированные объекты	282
23.3 Интерпретация среды вычислений	290
Библиография	291
Предметный указатель	309
Глоссарий	313
Практикум	329
Диссертации	333

Предисловие редактора серии

Компьютерные науки (computer science) и информационные технологии стали вездесущими и продолжают сулить перемены, которые еще больше затрагивают практически все сферы нашей жизни. Прежде всего новые технологии расширяют доступ к разнообразной информации и продуцируют массу информации, представленной в электронной форме, изменяя как характер труда, так и его результаты. Действительно, многие продукты, пользующиеся массовым спросом, выпускаются в виде последовательности битов, а темп их изменений необычайно высок.

Изменяются как отдельные профессии, так и целые отрасли индустрии или знания. По-сути, развитие информационных технологий дало виртуальную реальность, к которой современное общество только начинает адаптироваться и возможности которой достаточно быстро осваиваются.

Если на ранних стадиях программирование представляло собой вид искусства, когда программист писал программу для решения определенной задачи и сопровождал ее более или менее подробно составленной документацией, то теперь создана мощная индустрия программирования с сопутствующей ей инженерией программирования. В настоящее время в исследованиях по программированию или в сфере компьютерных наук, как правило, поддерживаются работы, в которых вносится некоторое небольшое улучшение в решение уже хорошо известной проблемы. Вместе с тем из виду упускаются действительно важные и фундаментальные исследования, ведущие к поиску новых концепций вычислений на компьютере и недостаточное внимание уделяется накоплению знаний в области программирования.

Публикации в этой серии задуманы как продолжающийся ряд работ в области компьютерных наук, информационных технологий и программирования, способствующих накоплению знаний в этих областях. Предполагается, что публикуемые работы могут

быть использованы для изложения отдельных курсов и вместе с тем способствовать научным исследованиям. Как правило, книги серии предназначаются для удовлетворения читателей самого различного уровня — от студентов, ставящих перед собой целью первичное ознакомление с самим предметом, до специалистов в разделах компьютерных наук.

Таким образом в публикациях серии предполагается отражать текущее состояние дел в данной области и дать основу для систематического изучения разделов компьютерных наук, информационных технологий и программирования.

Редакторы серии

Special Preface

One of the virtues of combinatory logic is that it provides a smart universe of discourse to computer scientists, programmers, applied theoreticians, practitioners. In practice, however, sophisticated applied theoretician and applied programmer must have a thorough grasp of what the system of objects does in response to evaluation request. To make the most transparent use of the evaluation system, one must have incite into such things as what kind of thing is an object or data structure, how to optimize the execution plan of a program for given set of objects, the impact of avoiding the bound variables on a program code, the importance of sub- and super-partitioning of an object set, etc. When being used or studied, all these trends are interrelated and involve some self contained local universes of mathematical ideas.

This book covers the commonly used computational ideas related to combinatory logic.

Nevertheless, the lifting from undergraduate to graduate teaching of applicative computations topics initiate unusual didactic challenges which can be explained by the visible distance between the rigorous mathematical apparatus and applied research papers.

This volume addresses this problem by providing a graduate textbook which covers the principle topics for classroom teaching. In this book the gradual progression of topics is supported by examples and exercises. Solving them, the students are introduced to the research issues at the forefront of computer science. The book is written by the author who is a specialist in the covered areas, and who have accumulated a rich experience in teaching graduate courses in those areas.

To use this book a minimal knowledge of theoretical computer science is needed to reach the following aims:

- to provide an introduction to principle areas of current research in computer science so that to give students the needed knowledge and intuition for more advanced study and research;

- to provide both the practitioners and researchers with a conceptual background of computational ideas that are expected to be used in applied computer science in the coming years.

To achieve the second aim the author have usually used the general notations of mappings between the classes of objects, so that to bring in more independency from a particular formal system. The most of attention is payed to the main computational ideas and their implementation within a formal framework.

Nobody argues that λ -calculus is a theory of functions. This theory gives a philosophy and reasons to view the idealized entities as functions. More important is that λ -calculus relates to other theories and the kind of these relations give important incites to construe different models of computation.

Spectrum of other theories usually starts with *set theory*. The question is how set theory provides a theory of functions. E.g., Zermelo's theory has a limited view of sets observing separate set A as extremely small with respect to the size of V , the *universe* of all sets. Any particular map $f : A \rightarrow B$ from one set A into other set B gives no information concerning maps from V to V , thus the classes of operations on all sets are hardly ever known within set theory. Usual assumption takes set A as an element of V , $A \in V$ in spite of *class* B which is subcollection of V , $B \subseteq V$. The connections between λ -calculus and class theory were established and studied. This does not mean that λ -calculus depends on set theory.

If we want a theory of functions be not derived from set theory we need a pure theory of functions within which functions are observed as particular entities. A *category theory* gives a universe of discourse for selected functions, or better: functional entities. The middle way between pure category theory and a set theory is given by *cartesian closed categories*.

Круг вопросов

“В течение длительного времени моя личная точка зрения состояла в том, что разделение на практическую и теоретическую работу искусственно и является вредным. Большая часть практической работы в области компьютеринга как при построении программного обеспечения, так и при проектировании аппаратуры выглядит противоречивой и неуклюжей, поскольку у тех, кто ее выполняет, нет ясного понимания фундаментальных конструкторских принципов их работы. Большая часть абстрактной математической и теоретической работы бесплодна, поскольку у нее нет точек соприкосновения с реальным компьютерингом. Одной из центральных задач Группы по Исследованиям в Программировании [Оксфордского университета] как учебной и научной структурной единицы было создание атмосферы, в которой подобного разделения просто не могло бы произойти.”

Кристофер Стрейчи

<http://vmoc.museophile.com/pioneers/strachey.html>

С момента своего возникновения комбинаторная логика и ламбда-исчисление были отнесены к “неклассическим” логиками. Дело заключается в том, что комбинаторная логика возникла в 1920-х гг., а ламбда-исчисление — в 1940-х гг. как ветвь метаматематики с достаточно очерченным предназначением — дать основания математике. Это означает, что сконструировав требуемую ‘прикладную’ математическую теорию — предметную теорию, — которая отражает процессы или явления в реальной внешней среде, можно воспользоваться ‘чистой’ метатеорией как оболочкой для выяснения возможностей и свойств предметной теории.

Комбинаторная логика и ламбда-исчисление — это такие формальные системы, в которых центральной разрабатываемой сущностью является представление об объекте. В первой из них — комбинаторной логике, — механизм связывания переменных в явном виде отсутствует, а во второй он имеется. Наличие явного механизма связывания предполагает и наличие связанных переменных, но тогда есть и свободные переменные, а также механизмы

замещения формальных параметров — связанных переменных, — на фактические параметры, то есть *подстановка*.

Изначальным назначением комбинаторной логики был именно анализ *процесса подстановки*. В качестве ее сущностей планировалось использовать объекты в виде *комбинаций констант*. Лямбда-исчислению отводилась роль средства уточнения представлений об *алгоритме* и *вычислимости*. Как следствие, комбинаторная логика дает в руки инструмент для анализа процесса подстановки. Через короткий промежуток времени оказалось, что обе эти системы можно рассматривать как *языки программирования*.

В обеих системах исчисляются объекты, они являются исчислениями или языками *высших порядков*, то есть имеются средства описания отображений или операторов, которые определяются на отображениях или операторах, а в качестве результата вырабатывают также отображения или операторы. Самое существенное, что именно *отображение* считается объектом. В этом их принципиальное отличие от всего многообразия других систем, в которых первичной сущностью обычно считают представление о *множестве* и его элементах.

К настоящему времени оба эти языка не только стали основой для всей массы исследований в области computer science, но и широко используются в теории программирования. Развитие вычислительной мощности компьютеров привело к автоматизации значительной части теоретического — логического и математического, — знания, а комбинаторная логика вместе с лямбда-исчислением признаются основой для рассуждений в терминах объектов.

Без овладения их методами нельзя полноценно развить базовую технику вычислений с объектами, поскольку все еще распространенный в объектных языках программирования и проектирования теоретико-множественный стиль заставляет вязнуть в обилии второстепенных деталей, упуская из виду действительно существенные моменты взаимодействия объектов.

Предисловие к первому изданию

Кому адресована книга

Эта книга написана в помощь тем читателям, которые изучают компьютерные науки или заняты работой в этой сфере и хотят привести свои знания в систему и переосмыслить тот круг идей, с которым приходится сталкиваться на практике. Книга призвана оказать помощь в чтении и изучении оригинальных исследовательских работ в области системного и теоретического программирования, а также в случае необходимости произвести аккуратную математическую проработку вновь создаваемых механизмов программирования или моделей данных. Подробные объяснения и большое число разобранных примеров и задач помогут войти в курс дела без чрезмерных усилий в подборе библиографии и начать собственную исследовательскую работу в этой интересной и перспективной области. Этому способствует значительная независимость в изучении отдельных разделов, что обусловлено спецификой самой математической дисциплины — комбинаторной логики. Для более математически искушенного читателя интерес могут представлять прикладные аспекты теории.

Зачем нужно исчислять объекты

Работа за компьютерами с оболочкой, способной взять на себя заботы об управлении объектами программного обеспечения, закладывает основу самой современной на сегодня методики программирования. В настоящее время с объектами работают сотни прикладных программ таких, как Windows, AutoCAD, Designer и многих других. С другой стороны инструментальные системы программирования Small Talk, C++, Actor и ряд других требуют от программиста систематических рассуждений в терминах объектов и связей между ними, которые в свою очередь могут рассматриваться как объекты. Программирование в терминах объектов тре-

бует создания и поддержания собственной математической культуры, дающей весь спектр стимулирующих идей. Программист при решении вполне конкретной задачи становится исследователем, от которого требуется создание собственного языка со своими возможностями. Эти возможности не всегда интуитивно очевидны, и могут потребоваться чисто математические оценки их выразительных возможностей. Кроме того, часто требуется не просто написать некоторый программный код, но и выполнить его оптимизацию, не теряя свойства эквивалентности исходному коду. Все это требует для аккуратного и профессионального проведения работы своей собственной “математической оболочки”, в которой поддерживаются все значимые и интересные математические приложения.

Основное — адекватный способ мышления

Хорошо известно, что в практике программирования сложились различные подходы, которые развиваются по различным направлениям. Бросающиеся в глаза различия проявляются в разном способе осмысления и написания программ. Большинство программистов занимается *процедурным* программированием. Но кроме него есть и программирование, *основанное на правилах*, *логическое* программирование, *параллельное* программирование, *визуальное* программирование, *программирование* в терминах потоков данных. При желании этот перечень можно продолжить, но он, очевидно, будет неполон, если в него не включить также и *объектно-ориентированное* программирование, которое имеет явно выраженную тенденцию роста.

Подходы и стили программирования. Подходов и стилей программирования много, и это отражает картину совершенствования и распространения все новых и новых компьютерных архитектур. Возникающие архитектуры ориентированы на новые подходы

к программированию, которые еще только зарождаются в исследовательских лабораториях.

Обилие и разнообразие подходов к программированию в computer science отражается в развитии и распространении различных подходов к построению математики. Действительно, математических теорий построено удивительно много, и каждая из них является совершенно своеобразным языком общения сравнительно ограниченного круга специалистов, которые хорошо понимают друг друга. Однако попытка “непосвященного” понять практическую пользу и значимость нового математического языка наталкивается на препятствия. Прежде всего оказывается необходимым перестроить собственный стиль мышления, чтобы на известные трудности взглянуть под новым углом зрения. Так распространение объектно-ориентированного программирования требует и привлечения других способов рассуждения, которые зачастую радикально отличаются от стереотипов рассуждения в процедурном программировании.

Рассуждения в терминах объектов. Точно также лишь немногие и сравнительно молодые математические теории ориентированы на рассуждения в терминах *объектов*, а не в терминах *операторов*, как это следует из опыта изучения математического анализа в большинстве университетов, в том числе, и технического или компьютерного профиля. К сожалению, программисту не удастся прослушать университетский курс, закладывающий основы математического мышления в терминах объектов. В лучшем случае дело ограничивается сообщением чисто математических результатов, полученных в *комбинаторной логике*, *λ -исчислении* или *теории категорий*, которые не так-то просто преломить на практическое программирование без известной теоретической искушенности.

Можно утверждать, что комбинаторная логика значительно повлияла на современную картину программирования. Начинаясь

как наука о *природе подстановок* в математических теориях, она породила *функциональное* программирование, программирование в терминах *суперкомбинаторов*, а также некоторые другие чрезвычайно плодотворные подходы к программированию. В частности, только по-настоящему проникнув в сам дух комбинаторной логики, можно понять в деталях и практически применить систему программирования с заранее нефиксированной системой инструкций.

Теория вычислений. Парадигмы программирования 90-х гг. в сильной степени выросли из математического способа рассуждений, принятого в *теории вычислений*. В частности, одной из ее начальных посылок была концепция ‘протекания информации’ вдоль некоторого ‘возможного’ русла, что привело к возникновению весьма плодотворной концепции программы, управляемой потоком данных. Другой пример связан с идеей использования некоторой части комбинаторной логики, построив в ней специальные объекты-инструкции. Эти объекты образуют систему команд *категориальной абстрактной машины*, которая может быть с успехом положена в основу вполне практических (но объектно-ориентированных) систем программирования. Более того, правила комбинаторной логики позволяют *оптимизировать компилируемый программный код*, редуцируя его к некоторой нормальной форме. Для специалистов в комбинаторной логике это почти само собой разумеется с самого начала, поскольку в этом состояла одна из целей разработки комбинаторной логики как математической дисциплины.

Современные исследования в области computer science показывают, что комбинаторная логика и ее различные категориальные диалекты становятся необходимым математическим языком программиста, пользуясь которым он обменивается идеями со своими коллегами. Дело как раз в том, что одним из предметов ее исследования являются объекты и построение различных исчис-

лений объектов, которые удовлетворяют кругу вопросов каждой конкретной прикладной задачи. Другими словами, решение всякой задачи требует построения специального точного языка. Как хорошо известно программистам, это язык интерфейса программного обеспечения. В терминах специалиста в computer science это специализированный диалект комбинаторной логики.

Объекты и системный подход

Если программистом для разработки избирается объектно-ориентированный подход, то скорее всего будет ошибкой подгонять решаемую задачу под какую-нибудь заранее известную математическую модель. Возможно, значительно лучше будет поискать нестандартное решение, которое в точности охватывает специфические особенности, связанные с самой природой прикладной области. В computer science для этого избирается *метатеория*, в рамках которой проводится исследование и которая “настраивается” на специфику прикладной области. Один из способов настройки — это *погружение* прикладной теории (“меньшей” теории) в чистую метатеорию (“большую теорию”). Кроме того, с математической точки зрения в рамках комбинаторной логики удобно строить подтеории — специальные математические модули, которые в готовом виде задают механизмы вычислений, имеющие самостоятельное значение. Такие рассуждения легко найдут отклик у программиста, вынужденного заниматься большим программным проектом, когда преимущества рассуждений в терминах объектов и их свойств становятся особенно очевидными. Комбинаторная логика позволяет на математически идеализированных объектах предварительно “проиграть” все наиболее сложные и тонкие моменты взаимодействия механизмов большого программного проекта.

г. Москва
сентябрь 1994 года

В.Э. Вольфенгаген

Предисловие ко второму изданию

Аппликативные вычислительные системы. Традиционно в состав аппликативных вычислительных систем, или АВС, включают системы исчислений объектов, основанные на комбинаторной логике и лямбда-исчислении. Единственное, что существенно разрабатывается в этих системах — это представление об *объекте*. В комбинаторной логике единственный метаоператор — *аппликация*, или, по иной терминологии, *приложение* одного объекта к другому. В лямбда-исчислении два метаоператора — *аппликация* и функциональная *абстракция*, позволяющая связывать одну переменную в одном объекте.

Возникающие в этих системах объекты ведут себя как функциональные сущности, имеющие следующие особенности:

- число аргументных мест, или *арность* объекта заранее не фиксируется, но проявляет себя постепенно, во взаимодействиях с другими объектами;

- при конструировании составного объекта один из исходных объектов — *функция*, — применяется к другому — *аргументу*, — причем в других контекстах они могут поменяться ролями, то есть функции и аргументы рассматриваются как объекты на равных правах;

- разрешается самоприменимость функций, то есть объект может применяться сам к себе.

Вычислительные системы с таким наиболее общими и наименее ограничительными свойствами оказываются в центре внимания современного сообщества computer science. Именно они в настоящее время обеспечивают необходимые метатеоретические средства, позволяя исследовать свойства целевых прикладных теорий, дают основу построения семантических средств языков программирования и обеспечивают средства построения моделей данных/метаданных в информационных системах.

Во второй половине 1970-х — начале 1980-х произошел взрыв в развитии аппликативных вычислительных систем, приведший к развитию целого спектра направлений исследований и изобилию научных публикаций. Они написаны специальным языком, их прочтение и понимание, не говоря уже об овладении существом дела, требовало основательной теоретической подготовки. Причин тому было несколько.

Во-первых, в 1980 году теория аппликативных вычислений все еще активно развивалась, а одной из целей авторов публикаций было пробудить интерес и привлечь к исследования в этой области математически одаренных студентов.

Во-вторых, с течением времени изменились роль и место, которые отводились в учебных программах для аппликативных систем. Если раньше для овладения важнейшими идеями требовалось изрядное знакомство с целым набором метаматематических дисциплин, то теперь изучение основных элементов аппликативных вычислений входит в стандартную программу, которая обязательна для студентов младших курсов. Это означает, что при написании книги предпочтение следует отдавать содержательному методу изложения, который увлекает только элементарные средства, делая предмет доступным.

В-третьих, за это время курсы по компьютерным наукам превратились из математизированных и концептуальных в довольно рецептурные, дающие нечто наподобие “быстрого взгляда” на массу готовых решений, методов, технологий и рекомендаций по использованию.

В-четвертых, отношение к компьютерным наукам стало в значительной мере практическим и даже потребительским. В большинстве случаев ожидается, что овладение тем или иным разделом компьютерных наук должно дать немедленную отдачу. У студентов имеется даже вполне заметное нежелание выполнять упражнения или решать задачи творческого и теоретического плана, носящих принципиальный характер. С целью сохранения уро-

вня обучения аппликативным вычислениям в настоящей книге получили развитие примеры и упражнения различного характера из области программирования, причем их решения даются элементарными средствами, не предполагая никакой предварительной математической подготовки.

Требования к уровню подготовки. В настоящей работе основной круг вопросов излагается элементарными средствами и, как ожидается, не вызовет трудностей у студентов, знакомых с логикой и началами методов доказательств. Предполагается, что читатель также знаком с началами программирования, имея некоторое представление о структурах данных. Вместе с тем изложение в целом и отчасти внутри отдельных разделов построено по нарастанию степени сложности, что должно удовлетворить тех продвинутых читателей, которые сторонятся книг из разряда “для легкого чтения”. Овладение рядом разделов поможет читателю встать на уровень современных исследований и стимулировать собственные научные исследования.

Примеры и упражнения. Книга содержит серии примеров и упражнений, к большинству из которых даются указания и решения. Как правило, в само решение включается достаточный запас теоретического знания, который позволит не только понять решение, но и поискать другие его варианты. Книга такого рода не может быть совершенно свободна от ошибок. В случае обнаружения ошибки или в случае, когда у вас есть предложения по ее исправлению автор будет рад узнать об этом. В случае, если у вас появятся новые задачи и упражнения, автор будет особенно рад узнать об этом, но если вы захотите их прислать, то, пожалуйста, присылайте их с решениями, воспользовавшись адресом электронной почты: vew@jmsuice.msk.ru.

Источники и практикум. Эта книга основана на курсах лекций, прочитанных автором в Московском инженерно-физическом институте. Курсы лекций, излагавшиеся в МИФИ на основе настоящей книги, оснащены практикумом. Практикум работает на IBM PC и распространяется на машинных носителях.

Благодарности. Большую заинтересованность, тонкое и профессиональное понимание проблематики, высокую эрудицию проявили к.т.н. Р.В. Храпко, Н.П. Маслий, А.С. Афанасьев, Т.В. Сырова, стремившиеся при разработке программного обеспечения применять аппликативные или чисто объектные технологии и решения, а также технику динамического формирования объектов.

Условия и возможности для применения объектного подхода при построении программного обеспечения были предоставлены В.В. Иванченко, который для организации научного исследования сделал все от него зависящее.

Содействие и доброжелательное участие И.В. Папаскири позволили создать постоянно действующий семинар и ту неповторимую атмосферу научных дискуссий, в которой производились обсуждения основного круга идей, отраженных в книге.

Проф. Л.Т. Кузин уделял значительное внимание методам и средствам работы с абстрактными объектами, содействуя исследованиям в этом направлении, их развитию, применениям и распространению на решение различных задач в области кибернетического моделирования. Целый ряд полученных результатов обсуждался на руководимых им научных семинарах секции “Прикладные проблемы кибернетики”, а их применения излагались в рамках руководимого им цикла дисциплин для студентов факультета кибернетики МИФИ при подготовке инженеров-математиков по специальности прикладная математика.

Вызывает восхищение та тщательность, трудолюбие и изобретательность, с которыми подошел К.А. Сагоян к подготовке и проверке решений всех примеров, приводимых в разделе 6, ко-

торые были использованы и в препринтных изданиях, за что ему выражается искренняя признательность. Различные варианты их решения опробованы им в ходе проведения практических занятий со студентами на факультете кибернетики МИФИ.

Огромное терпение и готовность показывать весь спектр применений исчислений объектов проявила И.В. Чепурнова, взявшая на себя труд по осуществлению издания препринтного варианта книги. Невозможно переоценить ее многолетний труд по организации и проведению занятий со студентами МИФИ и слушателями факультета повышения квалификации МИФИ. Основу ее спецкурсов составляли разделы, отраженные в настоящей работе, которые по ее предложениям дорабатывались и расширялись.

Большой труд по подбору задач для раздела 16 взяла на себя И.А. Горюнова. Проявленный ею энтузиазм и глубокое понимание механизма динамического формирования объектов позволили подобрать и подготовить упражнения, снабженные решениями.

Ряд ценных комментариев, касающихся раздела 15, был дан А.Г. Пантелеевым.

Для некоторых из примеров другие варианты решения указал Ву-Хоанг Нам.

Проф. Б.В. Бирюков и доц. А.С. Кузичев, руководившие научным семинаром по истории и методологии естественных наук в МГУ, дали ценные рекомендации в подборе библиографии. Проф. В.А. Смирнов указал на дополнительные возможные применения интенциональных методов в соединении с исчислениями объектов.

Проф. С.Х. Айтьян комментировал часть материала по мере написания книги. Отдельные аспекты применения объектов обсуждены с проф. Г.Г. Белоноговым, проф. Г.Я. Волошиным, проф. С.Н. Селетковым, проф. Е.Н. Сыромолотовым.

Проф. P.-L. Curien из Université Paris VII, LITP, один из авторов концепции категориальной абстрактной машины, любезно предоставил ряд своих публикаций, что ускорило работу над первым изданием книги.

К.т.н. В.Я. Яцук поделился методиками и опытом изложения исчислений объектов в читавшихся им курсах. И.В. Мажирин указал некоторые возможности разработки вариантов абстрактных машин. К.т.н. Г.С. Лебедев организовал преподавание дедуктивных расширений исчислений объектов.

Техническую поддержку при подготовке первоначального варианта текста оказал Э.М. Галстян.

Подготовка настоящей книги растянулась во времени и состоялась благодаря сотрудничеству со множеством заинтересованных специалистов, которые принимали участие не только в работе научных семинаров по проблематике компьютерных наук и информационных технологий в Московском инженерно-физическом институте, но и организовывали практические занятия со студентами разных курсов, осуществляли руководство курсовыми и дипломными проектами: Ю.Г. Горбанев, В.И. Васильев, О.В. Воскресенская, М.Ю. Чуприков, О.В. Баринов, И.А. Александрова, Г.К. Соколов, С.В. Косиков, И.А. Волков, С.К. Сарибекян, К.А. Сагоян, А.И. Михайлов, Т.В. Волшаник, З.И. Шаргатова, И.А. Горюнова, А.В. Гаврилов, Л.В. Гольцева, Е.В. Бурляева, В.А. Донченко, И.В. Чепурнова, А.В. Мясников, К.Е. Аксенов, С.И. Днепровский, С.В. Брызгалов, С.В. Зыков, Е.С. Пивоварова, А.Ю. Рукоданов, С.А. Кастанов, Л.А. Дмитриева, Ю.Ю. Парфенов, Р.К. Барабаш, А.И. Одрова, А.Л. Бринь, Р.В. Сницарь, А.Л. Забродин, А.М. Григорьев, Б.Б. Горелов, М.Л. Файбисович, К.В. Панькин, Н.В. Пищимова, А.И. Вайзер.

Наконец, настоящее издание книги “Комбинаторная логика в программировании” смогло увидеть свет лишь благодаря массе времени и энергии, затраченной сотрудниками Института Актуального Образования “ЮрИнфоР-МГУ”.

*г. Москва
январь 2003 года*

В.Э. Вольфенгаген

Введение

Все, что есть существенного в комбинаторной логике — это *объекты* и *способы комбинирования объектов*. Комбинирование одних объектов с другими выполняется посредством изначально выделенных объектов-констант, называемых *комбинаторами*. Этих изначальных комбинаторов всего несколько, однако пользуясь ими удастся построить такие известные формальные системы как логика высказываний, логика предикатов, арифметические системы¹ и целый ряд других. За последнее десятилетие комбинаторная логика стала одним из основных метаматематических аппаратов computer science, показав свои возможности в сфере программирования. Возникло целое семейство языков функционального программирования. Miranda, ML, KRC уже достаточно известные программистам дают представление о возможных применениях. Однако заявивший о себе в полной мере объектно-ориентированный подход к программированию и к проектированию прикладных систем в целом ставит принципиальные вопросы, касающиеся самого способа оперирования в терминах объектов. Для этого требуется заранее выбранный *универсум рассуждений*, своего рода теоретическая оболочка, которая гарантирует математическую плодотворность проводимого исследования. Это становится особенно ощутимым при реализации больших программных проектов, когда выбор систематического способа представления и оперирования объектами приобретает решающее значение.

Обсуждение структуры книги

Структура книги и расположение отдельных разделов выполнены таким образом, чтобы читателю можно было наиболее полно сосредоточить внимание на вопросах вычислений с объектами, име-

¹ Более строго: системы нумералов

ющих действительно принципиальное значение.

• Синтез нового объекта

Одна из важных задач, решаемых в комбинаторной логике, формулируется как задача синтеза объекта с заданными свойствами из имеющихся объектов применением уже известных способов комбинирования. На начальном этапе предполагается наличие всего трех объектов-комбинаторов: I , K , S , а также их свойств, задаваемых характеристическими равенствами. Для сохранения интуитивной ясности можно предполагать, что имеется система программирования с этими тремя инструкциями, пользуясь исключительно которыми предстоит построить довольно богатую по выразительным возможностям систему программирования. Результирующая система будет содержать исключительно объекты-комбинаторы.

• Характеристики комбинатора неподвижной точки

Прозрачность комбинаторной логики делает ее весьма простой в изучении. После первых продвижений может показаться, что в ней всегда имеем дело с простыми и конечными по своей природе объектами. Однако это впечатление обманчиво, и пользуясь комбинаторами, можно представлять *процессы*, в том числе циклические вычисления, которые представляют известную в программировании работу со стеком рекурсии.

• Применение принципа экстенциональности

Комбинаторная логика имеет ту особенность, что в ней строятся и применяются функции с *заранее не фиксированным числом аргументов*. Это означает, что ответ на вопрос, сколько же у применяемой функции-объекта в действительности аргументных мест требует известной осторожности. На самом деле аккуратное

использование довольно простых принципов экстенциональности (расширяемости) позволяет преодолеть эту неопределенность.

• Нумералы и их свойства

Обращаем внимание, что *с самого начала* в комбинаторной логике среди первичных объектов нет ... чисел. Дело в том, что концепцию числа можно разработать самостоятельно, пользуясь известными комбинаторами. Тогда числа предстают в несколько необычном облике — они являются объектами, проявляющими свою арность в зависимости от используемой системы постулатов. Точно так же в виде комбинаторов удастся разработать арифметические операции. Другими словами, арифметические сущности встраиваются в комбинаторную логику. Эта ситуация хорошо знакома в объектно-ориентированном программировании — приложение (арифметические объекты со своими правилами) встраивается в программную среду (комбинаторную логику).

• Исследование свойств комбинаторов с типами

Концепция класса является одной из самых основных в объектно-ориентированных рассуждениях. Класс в этом случае понимается как образец для создания экземпляров конкретных объектов. Более того, классы сами могут рассматриваться как объекты. Точно также комбинаторы классифицируются, или типизируются. Существенным для комбинаторов оказывается высокий порядок функциональных пространств. Тем не менее интуитивная ясность работы с комбинаторами как с объектами не теряется.

• Разложение термов в базисе I , K , S

Сосредоточим внимание на наипростейшей системе программирования, в которой всего только три инструкции: I , K , S . Синтезировать новый объект можно чисто механическим использованием

алгоритма разложения в базисе, который вполне аналогичен процессу компиляции.

• Разложение термов в базисе I, B, C, S

Как оказывается, базис I, K, S не единственный, и свойство базисности проявляет также набор комбинаторов I, B, C, S . Компиляция (разложение) объекта в этом базисе также решает задачу синтеза объекта с заданными свойствами. Очевидно, можно использовать свободу выбора базиса в зависимости от некоторых критериев.

• Выражение определения функции с помощью оператора неподвижной точки Y

Рассматривается случай рекурсивных определений объектов. Пользуясь фундаментальной для функционального программирования теоремой о неподвижной точке, рекурсивные определения удастся привести к обычному эквациональному виду.

• Исследование свойств функции *list1*

Показываются возможности построения функции-объекта в параметризованном виде. Придавая аргументами частные значения — а этими частными значениями могут быть и функции, — можно получить целое семейство определений частных функций.

• Установление изоморфизма декартово замкнутой категории и аппликативной вычислительной системы

Теперь начинаем продвигаться вглубь математических абстракций и будем увязывать операторный стиль мышления с комбинаторным. Отметим, что в комбинаторной логике используется единственный оператор — *оператор аппликации*, или оператор

приложения (применения) одного объекта к другому. Возникающая при этом система вычислений носит название *аппликативной вычислительной системы*. Она увязывается с традиционной операторной вычислительной системой, которая представлена специальным объектом — *декартово замкнутой категорией*.

• Построение отображения, каррирующего n -местную функцию

Используемые в операторном программировании n -местные функции-операторы в комбинаторной логике имеют образы в виде объектов, которые наследуют все их существенные свойства.

• Вывод основных свойств оболочки Каруби

Специальная категория, называемая оболочкой Каруби, позволяет лаконично выразить весь запас знаний, имеющийся относительно операторов, в терминах комбинаторной логики. При этом типы также кодируются объектами. Тем самым выполняется погружение типового приложения в бестиповую программную среду.

• Декартово произведение и проекции: погружение в ABC

Окончательное завершение начатого процесса погружения достигается введением в рассмотрение упорядоченных совокупностей объектов. Как оказывается, аппликативные вычисления также допускают их представление.

• Представление Lisp средствами λ -исчисления или комбинаторной логики

В аппликативную вычислительную систему встраивается нетривиальное приложение: значительный и, по-существу, полный фрагмент известной системы программирования Lisp.

● Реализация вычисления значений выражений с помощью суперкомбинаторов

Обсуждается, как работают объектно-ориентированные системы, встроенные в комбинаторную логику. Тем самым непосредственно удовлетворяется потребность в денотационном вычислении инструкций языков программирования, когда объектами выражается функциональный смысл программы. Существенно, что вычисление начинается с некоторого заранее известного набора инструкций. В процессе вычисления значения программы динамически возникают заранее неизвестные, но необходимые по ходу дела инструкции, которые дополнительно фиксируются в системе программирования.

● Полностью ленивая реализация суперкомбинаторов

Когда происходит динамическое формирование объектов “на лету”, то эффективность результирующего кода может теряться из-за необходимости неоднократно вычислять значение одного и того же объекта. Применение механизмов ленивого означивания позволяет этого избежать: если значение объекта уже однократно вычислено, то в дальнейшем используется именно это заранее вычисленное значение.

● Оптимизация вычислений путем перестановки параметров

Применение комбинаторов открывает возможности строить оптимизированный программный код, попутно в ходе синтеза результирующего объекта анализируя порядок возможного замещения формальных параметров на фактические.

● Реализация непосредственных вычислений выражений языков программирования

Техника вычисления значения выражений пересматривается — в свете систематического построения набора синтактико-семантических равенств, реализующих избранную парадигму объектно-ориентированных вычислений.

● Вычисление значения кода де Брейна

Вводится в рассмотрение техника переобозначения связанных переменных (формальных параметров), которая позволяет избежать коллизий связывания при замещении формальных параметров на фактические. Это прием переобозначения носит название *кодирования по де Брейну* и позволяет, фактически, аппаратом λ-исчисления пользоваться на тех же самых правах, что и аппаратом комбинаторной логики.

● Реализация машинных инструкций категориальной абстрактной машины (КАМ)

Строится специальный вариант теории вычислений, называемый *категориальной абстрактной машиной*. Для этого вводится в рассмотрение специальный фрагмент комбинаторной логики — категориальная комбинаторная логика. Она представлена набором комбинаторов, каждый из которых имеет самостоятельное значение как инструкция системы программирования. Тем самым в комбинаторную логику встраивается еще одно полезное приложение — система программирования, основанная на декартово замкнутой категории. Это позволяет еще раз на новом уровне переосмыслить связь операторного и аппликативного стиля программирования.

• Возможности оптимизации при вычислении на КАМ

Использование декартово замкнутой категории открывает дополнительные возможности оптимизации результирующего программного кода. Помимо свойств самой комбинаторной логики, используемой в качестве оболочки, допускается применение специальных категориальных равенств, заимствованных из декартово замкнутой категории как из приложения.

• Переменные объекты

Заключительная часть рассмотрения исчислений объектов касается общих вопросов математического представления объектов. Показывается, что использование концепции функтор-как-объект позволяет в компактной и лаконичной форме обозреть основные законы объектно-ориентированных вычислений. В частности, акцент делается на системах меняющихся (переменных) понятий-концептов, которые являются обычными объектами комбинаторной логики, но проявляют полезные для программирования свойства. Например, с помощью переменных концептов без особых осложнений строится не только теория вычислений, но и семантика систем программирования, а также модели объектов данных. Данные-как-объект дают новые степени свободы в компьютерных рассуждениях.

Краткие рекомендации по порядку изучения книги

Можно наметить возможный порядок чтения изложенного материала. Совсем небольшие усилия потребует автономное прочтение разделов 2–4, 6. Это материал дает представление о гибкости и выразительных возможностях языка комбинаторной логики.

К этому предварительному базису можно добавить разделы 5, 9, 7–8, где рассматриваются нумералы, рекурсия, разложения в базисах.

Затем можно прочитать разделы 10, 15–18, которые вводят в круг идей программирования посредством комбинаторов с динамической системой инструкций.

Добавлять другие разделы можно по своему вкусу. В частности, более детальное знакомство с категориальной абстрактной машиной в разделах 19–22 потребует обратиться к источникам, рекомендованным в библиографии. Разделы 11–14 ориентированы на тех читателей, которые хотят начать самостоятельное чтение оригинальных исследований в области computer science. Освоив основные идеи *теории вычислений*, можно приступить к чтению статей по данному вопросу. С другой стороны, раздел 23 может заинтересовать тех, кто пожелает более глубоко вникнуть в существо построения “встроенных приложений”, которые требуют модификации среды вычислений. В этом случае исследователь сталкивается с переменными понятиями или, по другой терминологии, с переменными концептами.

Глава 1

Предварительные сведения

К настоящему времени в теоретических исследованиях в области computer science сложился основной математический аппарат. Как оказывается, это всего несколько взаимосвязанных разделов, обеспечивающих рост как логики, так и компьютерных наук. В их число входят *λ -исчисление, комбинаторы, системы типов, теория категорий и языки программирования.*

λ -исчисление. В этом разделе изучаются способы построения чистого исчисления функциональной абстракции и аппликации (приложения) функций, которые получают применение в метаматематике, логике и компьютерных науках.

Комбинаторная логика. В ходе разработки этого раздела было показано, что связанные переменные можно исключить без потери выразительных возможностей формальной системы. Наиболее важные применения комбинаторная логика нашла в построении оснований математики, а также в развитии методов и средств реализации языков программирования.

Системы типов. К настоящему времени λ -исчисление является основным аппаратом для исследования систем типизации, а их результаты дали важные результаты как для оснований математики, так и для практики разработки и применения языков программирования.

Теория категорий. В этом разделе с математической точки зрения рассматриваются всего две сущности — *объекты* и *отображения*, — причем последние в свою очередь можно считать объектами. Применение методов теории категорий представляется особенно важным при исследовании, разработке и применении объектных систем программирования.

Языки программирования. Все эти основные направления: λ -исчисление, комбинаторную логику и системы типов, — достаточно трудно рассматривать изолированно, отделив друг от друга. Они взаимно переплетаются, и современная точка зрения заключается в их совместном изучении. Более того, именно эти три раздела дают основу разработки языков программирования.

В настоящее время появляется все больше оснований добавить к этому списку на равных правах еще один раздел — *теорию категорий*, которая оперирует единственным понятием, а именно: представлением об абстрактном *объекте*, к числу которых относятся и отображения.

На первый взгляд круг рассматриваемых в этих разделах идей не является однородным, вызывая рост числа внешне различных формализаций и математических аппаратов. На деле происходит даже большее: каждое очередное исследование, как правило, содержит построение своего собственного математического аппарата. Очень часто применяемые автором основные математические идеи оказываются завуалированными громоздкими выкладками.

Изучение названных разделов помогает устанавливать вну-

треннее единство разнородных исследований, а их систематическое применение обеспечивает исследователя или разработчика концептуально ясными и вместе с тем гибкими и мощными теоретическими средствами.

1.1 Круг основных идей

В настоящее время более внимательное рассмотрение выполненных в области компьютерных наук исследований показывает, что работы используют те или иные идеи либо из *теории категорий*, либо из *комбинаторной логики*, либо из *исчисления λ -конверсий*, либо из всех этих трех математических дисциплин сразу. О важности овладения этими разделами математики свидетельствует хотя бы тот факт, что открыв практически наугад труды любой конференции по computer science, можно найти не только чисто номинальное использование λ -обозначений, но фактическую разработку собственного математического языка, опирающегося на применение аппликаций и абстракций. Тем не менее попытки самостоятельного изучения основ λ -исчисления или комбинаторной логики наталкиваются на препятствие с самых первых шагов: требуется известное усилие, чтобы “перестроить мышление” с операторной точки зрения на математические записи *на* аппликативную, когда нет изначального разделения математических сущностей на ‘функции’ и ‘аргументы’, а есть только ‘объекты’. Интересно отметить, что один и тот же объект в зависимости от контекста может использоваться в различных *ролях*, играя то роль аргумента, то роль функции.

С синтаксической точки зрения объекты выделяются либо использованием специальной скобочной записи, либо принятием соглашения об опускании скобок, когда их при желании можно однозначно восстановить. Другая важная отличительная особенность заключается в акцентировании свойства *базисности* некоторых заранее выделенных объектов. Всякий раз оказывается, что

вновь вводимый объект может быть выражен путем *комбинирования* исходных объектов, которые в силу этого обстоятельства вполне уместно назвать *комбинаторами*. Такую разложимость произвольно введенного объекта в базисе трудно переоценить: вместо исследования свойств громоздкой прикладной теории с большим числом самых разных объектов можно заняться исследованием свойств всего нескольких объектов, нисколько не теряя при этом общности получаемого результата.

Для специалиста в области computer science это весьма желаемое свойство. Оказывается, что базисные комбинаторы могут быть приняты за *систему команд* некоторой абстрактной вычислительной системы, а все прочие объекты могут быть выражены, то есть “запрограммированы” посредством использования именно этого набора команд. Несмотря на кажущуюся очевидность, эта возможность аппарата комбинаторной логики еще не достаточно применена в практике компьютерных исследований.

1.2 Структура раздела

Обсуждение философских, чисто математических или технических аспектов аппликативных вычислений может увести далеко в сторону оснований математики. Тем не менее существует вполне приемлемый путь. Можно сперва ограничиться решением некоторых — хотя и на первый взгляд абстрактных, — задач, а затем сделать вывод, стоит ли двигаться дальше, вглубь идей аппликативных вычислений.

Данный раздел следует воспринимать как некоторое подобие меню, в котором обозначены основные вопросы, взаимодействие которых предстоит сначала увидеть непосредственно, а потом при детальном изучении выявить его более глубокую сущность.

В настоящем разделе приводятся подборки вариантов задач, которые рекомендуется использовать при самостоятельном изучении.

В случае организации аудиторной работы по изучению λ -исчисления и комбинаторной логики можно порекомендовать специальные подборки задач по вариантам, позволяющие делать не “слишком большие”, а вполне посильные шаги при освоении новых математических, или, скорее, вычислительных идей. Эти варианты задач сведены в таблицу 1.1. Варианты задания для самостоятельной работы над материалом составлены таким образом, чтобы покрыть все изложенные разделы.

1.3 Состояние исследований

Для углубленного изучения разделов настоящей книги, для тех, кто не захочет удовлетвориться вполне элементарным уровнем, можно порекомендовать некоторые работы. Часть из них вполне доступна как материал для первого чтения, тогда как другие носят характер оригинального исследования. В последнем случае открывается возможность соприкоснуться с тем, как делаются математические теории в computer science. Для этого лучше всего использовать оригинальные исследования в авторской публикации, к чтению и пониманию которых вполне можно подготовиться, решив рекомендуемые задачи.

Пионерское для computer science исследование выполнено Дж. Маккарти в (J. McCarthy, [102]). Им был разработан язык обработки списков Lisp, являющийся вполне объектно-ориентированным и к тому же функциональным языком программирования. В течение ряда лет Lisp был одной из самых популярных систем программирования в области искусственного интеллекта, заслужив название ‘ассемблер знаний’. Отметим, что Lisp является компьютерной реализацией λ -исчисления, предоставляя программисту практически все средства этой мощной математической теории. Эффективная реализация Lisp-интерпретатора, вы-

Таблица 1.1: Варианты задач

Номер варианта	Рекомендуемый набор задач					
1	1.1	2.6	3.3	4.7	5.1	6-1°
2	1.2	2.5	3.1	4.6	5.2	6-2°
3	1.3	2.4	3.2	4.5	5.3	6-3°
4	1.4	2.3	3.3	4.4	5.4	6-4°
5	1.5	2.2	3.1	4.3	5.1	6-5°
6	1.6	2.1	3.2	4.2	5.2	6-6°
7	1.7	2.5	3.3	4.1	5.3	6-7°
8	1.8	2.4	3.1	4.7	5.4	6-8°
9	1.9	2.3	3.2	4.6	5.1	6-9°
10	1.10	2.2	3.3	4.5	5.2	6-1°
11	1.11	2.1	3.1	4.4	5.3	6-2°
12	1.12	2.5	3.2	4.3	5.4	6-3°
13	1.1	2.4	3.3	4.2	5.1	6-4°
14	1.2	2.3	3.1	4.1	5.2	6-5°
15	1.3	2.2	3.2	4.7	5.3	6-6°
16	1.4	2.1	3.3	4.6	5.4	6-7°
17	1.5	2.5	3.1	4.5	5.1	6-8°
18	1.6	2.4	3.2	4.4	5.2	6-9°
19	1.7	2.3	3.3	4.3	5.3	6-1°
20	1.8	2.2	3.1	4.2	5.4	6-2°
21	1.9	2.1	3.2	4.1	5.1	6-3°
22	1.10	2.5	3.3	4.7	5.2	6-4°
23	1.11	2.4	3.1	4.6	5.3	6-5°
24	1.12	2.3	3.2	4.5	5.4	6-6°
25	1.1	2.2	3.3	4.4	5.1	6-7°
26	1.2	2.1	3.1	4.3	5.2	6-8°
27	1.3	2.5	3.2	4.2	5.3	6-9°
28	1.4	2.4	3.3	4.1	5.4	6-1°
29	1.5	2.3	3.1	4.7	5.1	6-2°
30	1.6	2.2	3.2	4.6	5.2	6-3°
31	1.7	2.1	3.3	4.5	5.3	6-4°

полненная А.Г. Пантелеевым (А.Г. Пантелеев, [41]; М.А. Булкин, Ю.Р. Габович, А.Г. Пантелеев, [5]), вскрыла множество деталей, касающихся реализации объектных и функциональных языков, а также путей и методов их эффективного использования. Методы реализации операций связывания переменных, структур данных, процедур, функций и механизмов рекурсии, установленные в ходе работ над этим проектом, стимулировали целое направление работ в области программирования.

В книгах (Л.Т. Кузин, [24], [25]) можно найти краткое и доступное инженеру введение в систему обозначений, принятую в λ -исчислении и комбинаторной логике. Исчерпывающее изложение всего круга математических идей содержится в книге Х. Барендрегта (H. Barendregt, [2]), однако для ее изучения требуется предварительная математическая подготовка. Классическое, очень подробное, ясное и обстоятельное обсуждение логического аппарата аппликативных вычислительных систем в книге Х. Карри (H. Curry, [22]) поможет сформировать мировоззрение, вероятно, на весь объектно-ориентированный подход, сложившийся в математических разделах computer science. Следует иметь ввиду, что именно благодаря усилиям Х. Карри комбинаторная логика сформировалась не только как общематематическая дисциплина, но и как необходимый фундамент компьютерных исследований.

Различную методическую помощь при решении задач, иллюстрирующих приложения АВС, можно почерпнуть из работ (А.А. Стогний, В.Э. Вольфенгаген, В.А. Кушников, В.И. Саркисян, В.В. Араксян, А.В. Шитиков, [47]), (В.Э. Вольфенгаген, [6]), (В.Э. Вольфенгаген, В.Я. Яцук, [7]), (В.Э. Вольфенгаген, К.А. Сагоян, [8]), (В.Э. Вольфенгаген, К.Е. Аксенов, Л.Ю. Исмаилова, Т.В. Волшаник, [9]), (А.А. Илюхин, Л.Ю. Исмаилова, З.И. Шаргатова, [21]), (К.Е. Аксенов, О.Т. Баловнев, В.Э. Вольфенгаген, О.В. Воскресенская, А.В. Ганночка, М.Ю. Чуприков, [1]).

Исследование (В.Э. Вольфенгаген, В.Я. Яцук, [10]) отражает раннее состояние работ и содержит подробные построения раз-

личных вариантов прикладных аппликативных систем, имеющих значение для решения прикладных задач, направленных на разработку информационных систем.

Особое место занимает работа Д. Скотта (D. Scott, [115]). Ее значимость и диапазон влияния на несколько поколений теоретиков computer science трудно переоценить. По-видимому, еще далеко не все, содержащиеся в этой работе идеи, нашли свое непосредственное применение. В частности, идея построения систем комбинаторов для специальных классов вычислений служит стимулом для глубоких и принципиальных исследований.

Работы [76], [77], [78] содержат разработку специальной комбинаторной логики, получившей название *категориальная комбинаторная логика*. На ее основе разработана абстрактная машина, которая является усовершенствованной концепцией понятия вычисления, а также самостоятельная система программирования, предназначенная для исследований в области программирования.

Несколько дополнительных работ, приводимых в библиографии, помогут начать самостоятельные исследования в различных прикладных областях, опираясь на возможности аппликативных вычислительных систем. В (R.B. Banerji, [63]), (V. Wolfengagen, [128]) можно найти применения в искусственном интеллекте.

Особо следует остановиться на исследованиях Д. Скотта (D. Scott, [111], [112], [113], [114], [115], [116], [117], [44], [118]). Эти (и многие другие) его работы заложили не только математические основы, но и, по-видимому, всю современную систему мировоззрения computer science. Теория вычислений, семантика языков программирования, вычисления, основанные на объектах — вот далеко не полный перечень направлений исследований, инициированных этим математиком.

Введение в математическую проблематику комбинаторной логики и λ -исчисления можно найти в ряде статей А.С. Кузичева (А.С. Кузичев, [27],[28], [29], [30],[31], [32], [33], [34],[35]). В этих

работах выполнено исследование выразительных возможностей систем с операторами аппликации и (функциональной) абстракции. Элементарные основы комбинаторной логики подробно изложены в (J. Hindley, H. Lercher, J. Seldin, [94]).

Работы Н. Белнапа могут принести пользу при разработке теории компьютерных информационных систем (N. Belnap, [4], [65], [66]). Эти же вопросы затронуты в (M. Coppo, M. Dezani, G. Longo, [75]).

Системы программирования в терминах объектов, основанные на технике аппликативных вычислений, в разной степени подробности изложены в [3], [20], [60], [61], [68], [85], [86], [92], [96], [103], [106], [124]. Для введения в круг вопросов, как строится семантика конструкций языков программирования можно использовать книгу (В.Э. Вольфенгаген, [16]).

В качестве основополагающей работы — для *суперкомбинаторного* программирования — следует обратить внимание на исследование (R.J.M. Hughes, [96]) и (S.L. Peyton Jones, [106]).

Развитие формальных методов содержится в (R. Amadio and P.-L. Curien, [57]), (J. Lambek and P. Scott, [100]), а в приложении к *событийно-управляемым* вычислениям, содержится в (V. Wolfengagen, [130]).

Остальные работы, приводимые в библиографии, помогут сориентироваться в смежных вопросах и начать самостоятельные исследования в области аппликативных вычислений.

1.4 Типовая задача

Общие указания к решению типовой задачи.

Формулировка задачи. Выразить через K и S объект с комбинаторной характеристикой:

$$Ia = a, \quad (I)$$

пользуясь постулатами $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии.

Решение.

I-1. Сформулируем постулаты, задающие отношение конвертируемости '=' :

$$(\alpha) \lambda x.a = \lambda z.[z/x]a; \quad (\beta) (\lambda x.a)b = [b/x]a;$$

$$(\nu) \frac{a = b}{ac = bc}; \quad (\mu) \frac{a = b}{ca = cb};$$

$$(\xi) \frac{a = b}{\lambda x.a = \lambda x.b}; \quad (\tau) \frac{a = b; b = c}{a = c}; \quad (\sigma) \frac{a = b}{b = a}.$$

I-2. Определим комбинаторные характеристики объектов K и S :

$$v(Kxy) = vx, \quad (K)$$

$$v(Sxyz) = v(xz(yz)), \quad (S)$$

которые выражаются в λ -исчислении посредством равенств $K = \lambda xy.x$ и $S = \lambda xyz.xz(yz)$.

I-3. Применяя схемы (K) и (S) , убеждаемся в том, что:

$$\begin{aligned} a &= Ka(Ka) && \text{по } (K) \\ &= SKKa. && \text{по } (S) \end{aligned}$$

Проверка. Проверим, что действительно $I = SKK$. Пусть $v = empty$ (пустой объект).

I-1. $SKKa = Ka(Ka)$, поскольку в схеме (S) можно положить $x = K, y = K, z = a$. Тогда ясно, что в силу постулата (α) :

$$Sxyz = SKKa, \quad xz(yz) = Ka(Ka), \quad SKKa = Ka(Ka).$$

I–2. Применяя аналогичным образом схему (*K*), заключаем, что $Ka(Ka) = a$.

I–3. По правилу транзитивности (τ), если выполняются равенства $SKKa = Ka(Ka)$ и $Ka(Ka) = a$, то $SKKa = a$.

Ответ. Объект *I* с заданной комбинаторной характеристикой $Ia = a$ имеет вид SKK , то есть $I = SKK$.

1.5 Варианты задания

‡ **Задание 1.** Выразить через *K* и *S* объекты с заданными комбинаторными характеристиками:

- 1) $Babc = a(bc),$
- 2) $Cabc = acb,$
- 3) $Wab = abb,$
- 4) $\Psi abcd = a(bc)(bd),$
- 5) $C^{[2]}abcd = acdb,$
- 6) $C_{[2]}abcd = adbc,$
- 7) $B^2abcd = a(bcd),$
- 8) $Ya = a(Ya)$ (Доказать, что $Y = WS(BWB).$),
- 9) $C^{[3]}abcde = acdeb,$
- 10) $C_{[3]}abcde = aebcd,$
- 11) $B^3abcde = a(bcde),$
- 12) $\Phi abcd = a(bd)(cd).$

‡ **Задание 2.** Какой комбинаторной характеристикой обладают

следующие объекты:

- 1) $(\lambda x.(P(xx)a))(\lambda x.(P(xx)a)) = Y$,
- 2) $Y = S(BWB)(BWB)$,
где $B = \lambda xyz.x(yz)$, $S = \lambda xyz.xz(yz)$, $W = \lambda xy.xyu$,
- 3) $Y = WS(BWB)$,
где $Wab = abb$, $Sabc = ac(bc)$, $Babc = a(bc)$,
- 4) $Y_0 = \lambda f.X(X)$, где $X = \lambda x.f(x(x))$,
- 5) $Y_1 = Y_0(\lambda y.\lambda f.f(y(f)))$,
где $Y_0 = \lambda f.X(X)$, $X = \lambda x.f(x(x))$?
(Указание: доказать, что $Y_ia = a(Y_ia)$.)

‡ **Задание 3.** Доказать, что:

- 1) $X = \lambda x.Xx, x \notin X$,
- 2) $Y_0 = \lambda f.f(Y_0(f))$, где $Y_0 = \lambda f.X(X)$, $X = \lambda x.f(x(x))$,
- 3) $Y_1 = \lambda f.f(Y_1(f))$, где $Y_1 = Y_0(\lambda y.\lambda f.f(y(f)))$,
 $Y_0 = \lambda f.X(X)$, $X = \lambda x.f(x(x))$.

‡ **Задание 4.** Какой комбинаторной характеристикой обладают следующие объекты¹ (доказать!):

- 1) $\Xi = C(BCF)I$,
- 2) $F = B(CB^2B)\Xi$,
- 3) $P = \Psi\Xi K$,
- 4) $\& = B^2(C\Xi I)(C(BB^2P)P)$,
- 5) $\vee = B^2(C\Xi I)(C(B^2B(B(\Phi\&))P)P)$,
- 6) $\neg = CP(\Pi I)$,
- 7) $\exists^* = B(W(B^2(\Phi P)C\Xi))K$, где $\exists[a] = \exists^*[a]$, $\exists = \exists[I]$.

¹ Обозначения: P — импликация, Ξ — формальная импликация, F — оператор функциональности, $\&$ — конъюнкция, \vee — дизъюнкция, Π — квантор общности, \neg — отрицание, \exists — квантор существования. Объекты $\Xi, F, P, \&, \vee$ — двухместные, объекты \neg, Π, \exists — одноместные.

Указания.

- 1) $Cabc = acb$, $Ia = a$, $Babc = a(bc)$.
- 2) $Babc = a(bc)$, $B^2abcd = a(bcd)$, $\Xi ab = FabI$.
- 3) $\Psi abcd = a(bc)(bd)$, $Kab = a$.
- 4) $Babc = a(bc)$, $B^2abcd = a(bcd)$, $Ia = a$, $Cabc = acb$.
- 5) $Babc = a(bc)$, $B^2abcd = a(bcd)$, $Ia = a$,
 $\Phi abcd = a(bd)(cd)$, $\&ab = \Xi(B^2(Pa)Pb)I$.
- 6) $Cabc = acb$, $Ia = a$, $\Pi = \Xi W \Xi$, $Wab = abb$.
- 7) Доказать, что $\exists[b]a = P(\Xi a(Kb))b$.

Воспользоваться равенствами:

$$\begin{aligned} Babc &= a(bc), & B^2abcd &= a(bcd), & Wab &= abb, \\ Cabc &= acb, & \Phi abcd &= a(bd)(cd). \end{aligned}$$

‡ **Задание 5.** Проверить справедливость следующих комбинаторных характеристик:

- 1) $S(KS)Kabc = a(bc)$,
- 2) $S(BBS)(KK)abc = acb$,
- 3) $B(BW(BC))(BB(BB))abcd = a(bc)(bd)$,
- 4) $B(BS)Babcd = a(bd)(cd)$.

Указание. $Kab = a$, $Sabc = ac(bc)$, $Babc = a(bc)$, $Cabc = acb$, $Wab = abb$.

‡ **Задание 6.** Выполнить следующее целевое исследование:

- 6-1° исследовать разложение термов в базисе I, K, S ;
- 6-2° исследовать разложение термов в базисе I, B, C, S ;
- 6-3° выразить определение функций, пользуясь комбинатором неподвижной точки Y ;
- 6-4° исследовать свойства функции:

$$\begin{aligned} \text{list1 } a \ g \ f \ x &= \text{if null } x \\ &\quad \text{then } a \\ &\quad \text{else } g(f(\text{car } x)) (\text{list1 } a \ g \ f(\text{cdr } x)); \end{aligned}$$

- 6-5° установить изоморфизм между декартово замкнутой категорией (д.з.к.) и аппликативной вычислительной системой (ABC);
- 6-6° получить отображение, которое соответствует отображению каррирования функций (для n -местной функции);
- 6-7° проделать вывод основных свойств оболочки Каруби;
- 6-8° закодировать термами бестипового λ -исчисления декартово произведение n объектов ($n \geq 5$) и получить соответствующие выражения для проекций;
- 6-9° представить основные функции аппликативного языка программирования Lisp средствами λ -исчисления и комбинаторной логики.

Формулировки задач для соответствующих целевых исследований в задании 6 на стр. 39.

- 6-1° Пусть определение терма $\lambda x.P$ дано индукцией по построению P :

$$\begin{aligned} 1.1) \quad \lambda x.x &= I, \\ 1.2) \quad \lambda x.P &= K P, \text{ если } x \notin FV(P), \\ 1.3) \quad \lambda x.P'P'' &= S(\lambda x.P')(\lambda x.P''). \end{aligned}$$

Исключить все переменные из приводимых ниже λ -выражений:

$$\lambda xy.xy, \lambda fx.fxx, f = \lambda x.B(f(Ax)).$$

6-2° Пусть определение терма M такого, что $x \in FV(M)$, дано индукцией по построению M :

$$2.1) \lambda x.x = I,$$

$$2.2) \lambda x.PQ = \begin{cases} (a) BP(\lambda x.Q), & \text{если } x \notin FV(P) \\ & \text{и } x \in FV(Q), \\ (b) C(\lambda x.P)Q, & \text{если } x \in FV(P) \\ & \text{и } x \notin FV(Q), \\ (c) S(\lambda x.P)(\lambda x.Q), & \text{если } x \in FV(P) \\ & \text{и } x \in FV(Q). \end{cases}$$

Исключить все переменные из приводимых ниже лямбда-выражений:

$$\lambda xy.xy, \lambda fx.fxx, f = \lambda x.B(f(Ax)).$$

6-3° Пользуясь функцией поиска неподвижной точки Y , выразить определения приводимых ниже (с помощью примеров) функций:

$$\begin{aligned} length(a_5, a_2, a_6) &= 3, \\ sum(1, 2, 3, 4) &= 10, \\ product(1, 2, 3, 4) &= 24, \\ append(1, 2)(3, 4, 5) &= (1, 2, 3, 1, 1), \\ concat((1, 2), (3, 4), ()) &= (1, 2, 3, 4), \\ map\ square(1, 2, 3, 4) &= (1, 4, 9, 16). \end{aligned}$$

Для приведенных примеров выполнить детальную проверку вычислений.

6-4° Воспользовавшись определением функции *list1* и следующими определениями: $Ix = x$, $Kxy = x$, $postfix\ x\ y = append\ y(ux)$, где (ux) — обозначение списка, состоящего из единственного элемента x , выразить функции:

(a) *length*, *sumsquares*, *reverse*, *identity*;

(б) *sum, product, append, concat, map*.

6-5° Вывести следующие равенства:

$$\begin{aligned} h &= \varepsilon \circ < (\Lambda h) \circ p, q >, \\ k &= \Lambda(\varepsilon \circ < k \circ p, q >), \end{aligned}$$

где:

$$\begin{aligned} [x, y] &= \lambda r. rxy, \\ < f, g > &= \lambda t. [f(t), g(t)] = \lambda t. \lambda z. z(ft)(gt), \\ h &: A \times B \rightarrow C, \\ k &: A \rightarrow (B \rightarrow C), \\ \varepsilon_{BC} &: (B \rightarrow C) \times B \rightarrow C, \quad x : A, \quad y : B, \\ \Lambda_{ABC} &: (A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)), \\ p &: A \times B \rightarrow A, \\ q &: A \times B \rightarrow B, \\ \varepsilon \circ < k \circ p, q > &: A \times B \rightarrow C. \end{aligned}$$

6-6° Рассматривая семейство функций h :

$$\begin{aligned} h_2 &: A \times B \rightarrow C, \\ h_3 &: A \times B \times C \rightarrow D, \\ h_4 &: A \times B \times C \times D \rightarrow E, \\ \dots &: \dots, \end{aligned}$$

найти семейство отображений

$$\Lambda_{ABC}, \Lambda_{(A \times B)CD}, \Lambda_{(A \times B \times C)DE}, \dots,$$

которые каррируют данные функции, то есть переводят их из “операторной” формы в аппликативную.

6-7° Оболочкой Каруби называют категорию, которая содержит

для $a \circ b = \lambda x.a(bx)$:

множества

объектов: $\{a \mid a \circ a = a\}$,

морфизмов: $\text{Hom}(a, b) = \{f \mid b \circ f \circ a = f\}$,

и морфизмы

тождества: $\text{id } a = a$,

композиции: $f \circ g$.

Пусть

$$[x, y] \equiv \lambda r.rxy,$$

$$< f, g > \equiv \lambda t.[f(t), g(t)] \equiv \lambda t.\lambda z.z(ft)(gt).$$

Проверить, что:

$$h = \varepsilon \circ < (\Lambda h) \circ p, q >, \quad k = \Lambda(\varepsilon \circ < k \circ p, q >),$$

где

$$\begin{aligned} h &: A \times B \rightarrow C, \\ k &: A \rightarrow (B \rightarrow C), \\ \varepsilon_{BC} &: (B \rightarrow C) \times B \rightarrow C, \quad x : A, \quad y : B, \\ \Lambda_{ABC} &: (A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)), \\ p &: A \times B \rightarrow A, \\ q &: A \times B \rightarrow B, \\ \varepsilon \circ < k \circ p, q > &: A \times B \rightarrow C. \end{aligned}$$

(**Указание.** Закодируйте функции вида $f : A \rightarrow B$ термами $B \circ f \circ A = \lambda x.B(f(Ax))$. Далее воспользуйтесь равенством: $A \circ A = A (= \lambda x.A(A(x)))$. Учтите, что $\Lambda h = \lambda xy.h[x, y]$.)

6-8° Получить терм лямбда-исчисления, соответствующий декартову произведению n объектов. Дополнительно установить n термов, которые ведут себя как проекции.

(Указание. Для случая $n=2$:

$$\begin{aligned} A_0 \times A_1 &= \lambda u. [A_0(uK), A_1(u(KI))], \\ \pi_0^2 &= \lambda u. (A_0 \times A_1)(u)K, \\ \pi_1^2 &= \lambda u. (A_0 \times A_1)(u)(KI). \end{aligned}$$

6-9° Выразить с помощью комбинаторов следующий базовый набор функций языка Lisp:

$$\{Append, Nil, Null, List, Car, Cdr\}.$$

(Указание. Для $Append \equiv \frown$ и для $Nil \equiv <>$:

- (1) $A \frown (B \frown C) = (A \frown B) \frown C$
- (2) $A \frown <> = <> \frown A = A$
- (3) $Null\ A = \begin{cases} 1, & \text{если } A = Nil, \\ 0, & \text{если } A \neq Nil, \end{cases}$
- (4) $List\ x = < x >$,
- (5) $Car\ < x_1, x_2, \dots, x_n > = x_1$,
- (6) $Cdr\ < x_1, x_2, \dots, x_n > = < x_2, \dots, x_n > .$

1.6 Рекомендуемый порядок выполнения задания

- 1) По номеру варианта выбрать соответствующую формулировку задачи.
- 2) Изучить решение типовой задачи, приводимое в тексте. По аналогии с решением типовой задачи выполнить шаги доказательства.
- 3) Проверить правильность полученного результата (ответа), проведя выкладки в обратном порядке.

Глава 2

Синтез нового объекта

Вопрос, что такое ‘данные’ или что такое ‘программа’ настолько хороший, что даже не будем пытаться ответить на него. Можно было бы допустить, что “сложность структуры данных разменивается на алгоритмическую сложность”, но для этого нужно определиться с представлением об алгоритме и о структуре данных. Во всяком случае, обычно программы и данные противопоставляются, но не всегда.

Возможно, конечно, что захотим иметь дело с *объектами*. Тогда появляется шанс рассматривать как программы, так и данные единым образом.

Одна из важных задач, решаемых в комбинаторной логике, формулируется как задача синтеза объекта с заданными свойствами из имеющихся объектов применением уже известных способов комбинирования. На начальном этапе предполагается наличие всего трех объектов-комбинаторов: I , K , S , а также их свойств, задаваемых характеристическими равенствами. Для сохранения интуитивной ясности можно предполагать, что имеется система программирования с этими тремя инструкциями, пользуясь исключительно которыми предстоит построить довольно богатую по выразительным возможностям систему программирования. Ре-

зультирующая система будет содержать исключительно объекты-комбинаторы.

2.1 Принцип комбинаторной полноты

2.1.1 Комбинаторная характеристика

Построение такой системы программирования возможно, поскольку, как известно, система комбинаторов I, K, S принципиально позволяет это сделать, или, иными словами, обладает свойством *комбинаторной полноты*.

Определение 2.1 (комбинаторная полнота). Набор комбинаторов X_1, \dots, X_m , задаваемых соответствующими преобразованиями конверсии, считается *комбинаторно*, или *функционально полным*, если для любого объекта X , составленного из различных переменных x_1, \dots, x_n , можно найти комбинатор U , выражимый в терминах X_1, \dots, X_m и такой, что

$$Ux_1 \dots x_n = X. \quad (U)$$

Свойство (U) можно рассматривать как *характеристическое правило* конверсии комбинатора U , или *комбинаторную характеристику* U .

Таким образом, система I, K, S гарантирует возможность построения такого объекта U , то есть строится, или

синтезируется “программа”, или “процедура” U , вызов которой на *фактических* параметрах $x_1 \dots x_n$ дает требуемый *результат* X — а это комбинация, составленная из переменных $x_1 \dots x_n$.

Поскольку имеем дело с аппликативной вычислительной системой, то сделаем замечание, касающееся соглашения о приме-

няемых обозначениях. Примем по соглашению, что

$$U x_1 x_2 \dots x_n \equiv \underbrace{((\dots ((U x_1) x_2) \dots) x_n)}_{n \text{ штук}},$$

то есть опущенные скобки восстанавливаются *по ассоциации влево*.

Пример 2.1.

$$U x_1 x_2 x_3 \equiv (((U x_1) x_2) x_3).$$

2.1.2 Системы концептов

Комбинаторная логика в бестиповом варианте является основным математическим аппаратом, в рамках которого исчисляются объекты, абстрактные по самой своей сути. Фактически, комбинаторная логика представляет собой чистое исчисление *концептов*, позволяя по мере необходимости создавать или модифицировать “на лету” свою собственную систему концептов. Развитие систем “переменных” концептов имеет первостепенную роль во всех значимых приложениях, включая построение объектно-ориентированных систем программирования. Несмотря на кажущуюся простоту — а, фактически, комбинаторной логикой можно начать пользоваться сразу после изучения определения всего двух комбинаторов K и S , — глубокое осмысление всех возможностей исчисления комбинаторов требует известного технического навыка. Прежде всего это касается самого аппликативного стиля используемых записей. Однако это не является чем-то неожиданным: сорокалетний опыт практического использования системы программирования Lisp давно уже подготовил почву для переосмысления “теоретического минимума программиста”, куда входят и комбинаторная логика, и λ -исчисление, и другие специальные разделы логики, по традиции называемой неклассической.

2.1.3 Комбинаторная полнота

Комбинаторная логика представляет собой ветвь математической логики, изучающую комбинаторы и их свойства. Подробнее см. книгу (Х. Карри, [22]). Комбинаторы или аналогичные им операторы можно определить в терминах λ -конверсии, и на этом основании различные исчисления λ -конверсий обычно рассматривают как часть комбинаторной логики.

Принцип комбинаторной полноты

Исчисления λ -конверсии и системы комбинаторной логики являются комбинаторно полными теориями.

Определение 2.2 (комбинаторная полнота). Комбинаторная полнота в исчислениях λ -конверсии задается схемой аксиом (β) :

$$(\lambda x.a)b = [b/x]a, \quad (\beta)$$

где λ — оператор абстракции, выражение $'[b/x]a'$ обозначает результат подстановки объекта b вместо свободных вхождений переменной x в объект a , а $'='$ — символ отношения конвертируемости.

В системах комбинаторной логики по произвольному объекту a посредством комбинаторов K, S, I строится новый объект

$$U \equiv [x_1, \dots, x_n]a,$$

где $[x_1, \dots, x_n]$ для $n > 0$ выступает в роли оператора абстракции по переменным x_1, \dots, x_m . Для оператора абстракции доказывается принцип комбинаторной полноты:

$$\begin{aligned} Ub_1 \dots b_n &\equiv ([x_1, \dots, x_n]a)b_1 \dots b_n \\ &= [b_1, \dots, b_n/x_1, \dots, x_n]a, \end{aligned}$$

где выражение $'[b_1, \dots, b_n/x_1, \dots, x_n]a'$ обозначает результат одновременной подстановки объектов b_1, \dots, b_n в объект a вместо

соответствующих вхождений графически различных переменных x_1, \dots, x_n для $n > 0$.

Содержательная интерпретация выглядит следующим образом:

“процедура” U с *формальными*, или *подстановочными* параметрами

$$x_1, \dots, x_n$$

вызывается на *фактических* параметрах $b_1 \dots b_n$ и дает требуемый *результат*

$$[b_1, \dots, b_n / x_1, \dots, x_n]a.$$

2.1.4 Элементарная комбинаторная логика

Системы комбинаторов предназначены для выполнения тех же функций, что и системы λ -конверсии, но без использования связанных переменных. Поэтому технические сложности, связанные с подстановкой и конгруэнтностью, исчезают.

Введение понятия комбинатором

Покажем, что пользуясь комбинаторами, можно сформировать *понятие*, или *концепт*, соответствующий некоторому закону. Другими словами, закон сворачивается до понятия — в этом одно из важнейших назначений комбинаторной логики как метатеории.

Рассмотрим коммутативный закон сложения в арифметике:

$$\forall x, y. x + y = y + x.$$

Этот закон можно переписать без использования связанных переменных x и y , определив

$$\forall x, y. A(x, y) = x + y$$

и введя метаоператор \mathbf{C} :

$$\forall f, x, y. (\mathbf{C}(f))(x, y) = f(y, x).$$

Этот закон примет вид:

$$\mathbf{C}A = A.$$

Метаоператор \mathbf{C} может быть назван ‘комбинатором’, выражающим закон коммутативности операции A .

В дальнейшем при записи подобных метаоператоров вместо полужирного шрифта будем использовать курсив.

Упражнение 2.1. Записать коммутативный закон умножения без использования переменных.

Простейшие комбинаторы

Начнем с того, что перечислим основные часто встречающиеся на практике комбинаторы.

Тождество. Простейшим комбинатором является комбинатор *тождества* I :

$$If = f.$$

Композитор. Элементарный *композитор* :

$$Bfgx = f(gx)$$

выражает композицию двух функций f и g .

Дубликатор. Элементарный *дубликатор* W :

$$Wfx = fxx$$

дублирует второй аргумент.

Пермутатор. Введенный выше комбинатор называется элементарным *пермутатором* и переобозначается как C :

$$Cfxy = fyx.$$

Коннектор. Элементарный *коннектор* S определяется правилом:

$$Sfgx = fx(gx).$$

Канцелятор Элементарный *канцелятор*

$$Kcx = c$$

выражает константу (константную функцию) как функцию от x .

Пример 2.2. Пусть $f \equiv \sin$ — функция ‘синус’, $g \equiv \exp^5$ — функция возведения в пятую степень. Тогда Bfg — синус от x в пятой степени:

$$\begin{aligned} Bfgx &= B \sin \exp^5 x = \sin(\exp^5 x) = \sin x^5, \\ Bgf &\text{ — пятая степень синуса,} \\ Bgfx &= B \exp^5 \sin x = \exp^5(\sin x) = \sin^5 x. \end{aligned}$$

Пример 2.3. Если Q — операция возведения в квадрат, то BQQ или WBQ — операция возведения в четвертую степень:

$$WBQx \stackrel{W}{=} BQQx \stackrel{B}{=} Q(Qx) = x^4.$$

Пример 2.4. Поскольку выполняется равенство (конверсия):

$$B(Bf) g x y = Bf(gx) y = f(gxy),$$

то, если f есть операция дифференцирования D , то $B(Bf)$ — взятие производной от функции двух аргументов:

$$B(BD) g x y = BD(gx) y = D(gxy).$$

2.2 Синтез основных комбинаторов: задачи

Теперь сосредоточим свое внимание на выработке технического навыка установления и исследования свойств нового концепта. В качестве таких концептов избираем различные комбинаторы, широко используемые в математической практике. Общая постановка задачи состоит в синтезе концепта-комбинатора по заранее заданной *комбинаторной характеристике* (см. стр. 46).

Задача 2.1. Вывод выражения для комбинатора B .

Формулировка задачи. Выразить через K и S объект с комбинаторной характеристикой:

$$Babc = a(bc), \quad (B)$$

пользуясь постулатами $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии.

Решение.

$B-1$. Сформулируем постулаты, задающие отношение конвертируемости '=' :

$$(\alpha) \quad \lambda x.a = \lambda z.[z/x]a; \quad (\beta) \quad (\lambda x.a)b = [b/x]a;$$

$$(\nu) \quad \frac{a = b}{ac = bc}; \quad (\mu) \quad \frac{a = b}{ca = cb};$$

$$(\xi) \quad \frac{a = b}{\lambda x.a = \lambda x.b}; \quad (\tau) \quad \frac{a = b; b = c}{a = c}; \quad (\sigma) \quad \frac{a = b}{b = a}.$$

$B-2$. Определим комбинаторные характеристики объектов K и S :

$$x(Kyz) = xy, \quad (K)$$

$$x(Syzw) = x(yw(zw)), \quad (S)$$

которые выражаются в λ -исчислении посредством равенств:
 $K = \lambda xy.x$ и $S = \lambda xyz.xz(yz)$.

Действительно, по постулату (β) :

$$\begin{aligned} x(Kyz) &\equiv x(\underbrace{(\lambda xy.x)yz}_{\equiv K}) \stackrel{(\beta)}{=} xy, \\ x(Syzw) &\equiv x(\underbrace{(\lambda xyz.xz(yz))}_{\equiv S}yzw) \stackrel{(\beta)}{=} x(yw(zw)), \end{aligned}$$

B-3. Применяя схемы (K) и (S) , убеждаемся, что:

$$\begin{aligned} a(bc) &= Kac(bc) && (K) \\ &= S(Ka)bc && (S) \\ &= KSa(Ka)bc && (K) \\ &= S(KS)Kabc. && (S) \end{aligned}$$

Проверка. Проверим, что $B = S(KS)K$.

B-1. $S(KS)Kabc = KSa(Ka)bc$, поскольку в схеме (S) можно положить $y \equiv (KS)$, $z \equiv K$, $w \equiv a$. Тогда:

$$Syzw = S(KS)Ka, \quad yw(zw) = (KS)a(Ka),$$

т.е. $S(KS)Ka = (KS)a(Ka)$. Удалив несущественные скобки, получим $S(KS)Ka = KSa(Ka)$. Дважды применяя к полученному выражению постулат (ν) , получим: $S(KS)Kabc = KSa(Ka)bc$.

B-2. Аналогично, применяя схему (K) , имеем $KSa = S$. Учитывая постулат (ν) , получим $KSa(Ka)bc = S(Ka)bc$.

B-3. Тем же способом, последовательно применяем схемы (S) и (K) , постулат (ν) и удаляя несущественные скобки, получаем:

$$S(Ka)bc = Kac(bc); \quad (Kac)bc = a(bc).$$

B–4. Несколько раз применяя правило транзитивности (τ), получим $S(KS)Kabc = a(bc)$. (Это выражение справедливо, поскольку если $S(KS)Kabc = KSa(Ka)bc$ и $KSa(Ka)bc = S(Ka)bc$, то $S(KS)Kabc = S(Ka)bc$ и т.д.)

Ответ. Объект *B* с комбинаторной характеристикой $Babc = a(bc)$ имеет вид $S(KS)K$, т.е. $B = S(KS)K$.

Задача 2.2. Вывод выражения для комбинатора *C*.

Формулировка задачи. Выразить через *K*, *S* и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$Cabc = acb, \quad (C)$$

пользуясь постулатами $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии.

Решение.

C–1. Сформулируем постулаты, задающие отношение конвертируемости '=' (см. предыдущую задачу).

C–2. Напомним комбинаторные характеристики, возможно, используемых объектов:

$$(K) \quad Kxy = x, \quad (S) \quad Sxyz = xz(yz),$$

$$(I) \quad Ix = x, \quad (B) \quad Bxyz = x(yz).$$

Отметим, что к до сих пор известным схемам (*K*), (*S*) и (*I*) добавлена схема (*B*), полученная в результате решения задачи 2.1 на стр. 52. Как было показано, эта последняя схема выразима в терминах схем (*K*) и (*S*).

C–3. Применив эти схемы к (acb) , получим:

$$\begin{aligned} acb &= ac(Kbc) && \text{(по схеме (K))} \\ &= Sa(Kb)c && \text{(по схеме (S))} \\ &= B(Sa)Kbc && \text{(по схеме (B))} \\ &= BBSaKbc && \text{(по схеме (B))} \\ &= BBSa(KKa)bc && \text{(по схеме (K))} \\ &= S(BBS)(KK)abc. && \text{(по схеме (S))} \end{aligned}$$

Учитывая постулат транзитивности (τ), имеем:

$$S(BBS)(KK)abc = acb,$$

т.е. $C = S(BBS)(KK)$.

Ответ. Объект с комбинаторной характеристикой $Cabc = acb$ имеет вид $C = S(BBS)(KK)$.

Задача 2.3. Вывод выражения для комбинатора W .

Формулировка задачи. Выразить комбинатор W со следующей характеристикой:

$$Wab = abb. \quad (W)$$

Решение.

$W-1$. Выпишем характеристики используемых объектов:

$$(S) Sxyz = xz(yz), \quad (I) Ix = x, \quad (C) Cxyz = xzy.$$

$W-2$. Применим эти схемы к abb :

$$\begin{aligned} abb &= ab(Ib) && (\text{по } (I)) \\ &= SaIb && (\text{по } (S)) \\ &= CSIab. && (\text{по } (C)) \end{aligned}$$

С учетом постулатов получаем: $CSIab = abb$. Таким образом, $W = CSI$.

$W-3$. Предложим еще два варианта вывода объекта W :

$$\begin{aligned} abb &= ab(Kba) & abb &= ab(Kb(Kb)) \\ &= ab(CKab) & &= ab(SK Kb) \\ &Sa(CKa)b & &= Sa(SK K)b \\ &= SS(CK)ab & &= Sa(K(SK K)a)b \\ & & &= SS(K(SK K))ab. \end{aligned}$$

Ответ. Объект W с характеристикой $Wab = abb$ имеет вид: $W = CSI(= SS(CK) = SS(K(SKK)))$.

Задача 2.4. Вывод выражения для комбинатора Ψ .

Формулировка задачи. Выразить комбинатор Ψ со следующей характеристикой:

$$\Psi abcd = a(bc)(bd). \quad (\Psi)$$

Решение.

Ψ –1. Сформулируем постулаты, задающие отношение конвертируемости.

Ψ –2. Напомним комбинаторные характеристики используемых объектов:

$$(C) Cxyz = xzy, (W) Wxy = xy, (B) Bxyz = x(yz).$$

Ψ –3. Применив эти схемы к $a(bc)(bd)$, получим:

$$\begin{aligned} a(bc)(bd) &= B(a(bc))bd && (B) \\ &= BBa(bc)bd && (B) \\ &= B(BBa)bcbd && (B) \\ &= BB(BB)abcbd && (B) \\ &= C(BB(BB)ab)bcd && (C) \\ &= BC(BB(BB)a)bbcd && (B) \\ &= W(BC(BB(BB)a))bcd && (W) \\ &= BW(BC)(BB(BB)a)bcd && (B) \\ &= B(BW(BC))(BB(BB))abcd. && (B) \end{aligned}$$

Учитывая необходимые постулаты, получаем следующий результат: $B(BW(BC))(BB(BB))abcd = a(bc)(bd)$, т.е. $\Psi = B(BW(BC))(BB(BB))$.

Ответ. Объект Ψ с комбинаторной характеристикой $\Psi abcd = a(bc)(bd)$ имеет вид $\Psi = B(BW(BC))(BB(BB))$.

Задача 2.5. Вывод выражения для комбинатора B^2 .

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$B^2 abcd = a(bcd). \quad (B^2)$$

Решение.

B^2 —1. Сформулируем необходимые постулаты, задающие отношение конвертируемости.

B^2 —2. Напомним комбинаторную характеристику используемого объекта: $(B) \ Bxyz = x(yz)$.

B^2 —3. Применяя эту схему к $a(bcd)$, получим:

$$\begin{aligned} a(bcd) &= Ba(bc)d && (\text{ по схеме } (B)) \\ &= B(Ba)bcd && (\text{ по схеме } (B)) \\ &= BBBabcd. && (\text{ по схеме } (B)) \end{aligned}$$

Учитывая постулаты, имеем: $BBBabcd = a(bcd)$, т.е. $B^2 = BBB$.

Ответ. Объект B^2 с комбинаторной характеристикой $B^2 abcd = a(bcd)$ имеет вид $B^2 = BBB$.

Задача 2.6. Вывод выражения для комбинатора B^3 .

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$B^3 abcde = a(bcde). \quad (B^3)$$

Решение.

B^3 —1. Сформулируем постулаты, которые задают отношение конвертируемости.

B^3 —2. Напомним комбинаторные характеристики используемых объектов: $(B) \ Bxyz = x(yz)$, $(B^2) \ B^2xyzw = x(yzw)$.

B^3 —3. Применяя эти схемы к $a(bcde)$, получим:

$$\begin{aligned} a(bcde) &= B^2a(bc)de && (\text{по схеме } (B^2)) \\ &= B(B^2a)bcde && (\text{по схеме } (B)) \\ &= BBB^2abcde. && (\text{по схеме } (B)) \end{aligned}$$

Учитывая постулаты, имеем: $BBB^2abcde = a(bcde)$, т.е. $B^3 = BBB^2$.

Ответ. Объект B^3 с комбинаторной характеристикой $B^3abcde = a(bcde)$ имеет вид $B^3 = BBB^2$.

Задача 2.7. Вывод выражения для комбинатора $C^{[2]}$.

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$C^{[2]}abcd = acdb. \quad (C^{[2]})$$

Решение.

$C^{[2]}$ —1. Сформулируем постулаты, которые задают отношение конвертируемости.

$C^{[2]}$ —2. Напомним комбинаторные характеристики, возможно, используемых объектов:

$$(B) \ Bxyz = x(yz), \quad (C) \ Cxyz = xzy.$$

$C^{[2]}$ –3. Применяя эти схемы к $acdb$, получим:

$$\begin{aligned} acdb &= C(ac)bd && (\text{по схеме } (C)) \\ &= BCacbd && (\text{по схеме } (B)) \\ &= C(BCa)bcd && (\text{по схеме } (C)) \\ &= BC(BC)acbd. && (\text{по схеме } (B)) \end{aligned}$$

Учитывая постулаты, имеем: $BC(BC)acbd = acbd$, то есть $C^{[2]} = BC(BC)$.

Ответ. Объект $C^{[2]}$ с заданной комбинаторной характеристикой $C^{[2]}abcd = acbd$ имеет вид $C^{[2]} = BC(BC)$.

Задача 2.8. Вывод выражения для комбинатора $C_{[2]}$.

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$C_{[2]}abcd = adbc. \quad (C_{[2]})$$

Решение.

$C_{[2]}$ –1. Сформулируем необходимые постулаты.

$C_{[2]}$ –2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$(B^2) \quad B^2xyzw = x(yzw), \quad (C) \quad Cxyz = xzy.$$

$C_{[2]}$ –3. Применяя эти схемы к $adbc$, получим:

$$\begin{aligned} adbc &= Cabdc && (\text{по схеме } (C)) \\ &= C(Cab)cd && (\text{по схеме } (C)) \\ &= B^2CCabcd. && (\text{по схеме } (B^2)) \end{aligned}$$

Учитывая постулаты, имеем: $B^2CCabcd = adbc$, т.е. $C_{[2]} = B^2CC$.

Ответ. Объект $C_{[2]}$ с заданной комбинаторной характеристикой $C_{[2]}abcd = adbc$ имеет вид $C_{[2]} = B^2CC$.

Задача 2.9. Вывод выражения для комбинатора $C^{[3]}$.

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$C^{[3]}abcde = acdeb. \quad (C^{[3]})$$

Решение.

$C^{[3]}$ —1. Сформулируем необходимые постулаты.

$C^{[3]}$ —2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$\begin{array}{ll} (C) & Cxyz = xzy, \\ (C^{[2]}) & Cxyzw = xzwy. \end{array} \quad (B) \quad Bxyz = x(yz),$$

$C^{[3]}$ —3. Применяя эти схемы к $acdeb$, получим:

$$\begin{array}{ll} acdeb &= C^{[2]}(ac)bde && (\text{по схеме } (C^{[2]})) \\ &= BC^{[2]}acbde && (\text{по схеме } (B)) \\ &= C(BC^{[2]}a)bcde && (\text{по схеме } (C)) \\ &= BC(BC^{[2]})abcde. && (\text{по схеме } (B)) \end{array}$$

Учитывая постулаты, имеем: $BC(BC^{[2]})abcde = acdeb$, то есть $C^{[3]} = BC(BC^{[2]})$.

Ответ. Объект $C^{[3]}$ с комбинаторной характеристикой $C^{[3]}abcde = acdeb$ имеет вид $C^{[3]} = BC(BC^{[2]})$.

Задача 2.10. Вывод выражения для комбинатора $C_{[3]}$.

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$C_{[3]}abcde = aeabcd. \quad (C_{[3]})$$

Решение.

$C_{[3]}$ —1. Сформулируем постулаты.

$C_{[3]}$ —2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$\begin{aligned} (B^2) \quad B^2xyzw &= x(yzw), & (C) \quad Cxyz &= xzy, \\ (C_{[2]}) \quad C_{[2]}xyzw &= xwyz. \end{aligned}$$

Применяя эти схемы к $aebcd$, получим:

$$\begin{aligned} aeabcd &= Cabecd && \text{(по схеме (C))} \\ &= C_{[2]}(Cab)cde && \text{(по схеме (C_{[2]}))} \\ &= B^2C_{[2]}Cabcede. && \text{(по схеме (B^2))} \end{aligned}$$

Учитывая постулаты, имеем: $B^2C_{[2]}Cabcede = aeabcd$, то есть $C_{[3]} = B^2C_{[2]}C$.

Ответ. Объект $C_{[3]}$ с комбинаторной характеристикой $C_{[3]}abcde = aeabcd$ имеет вид $C_{[3]} = B^2C_{[2]}C$.

Задача 2.11. Вывод выражения для комбинатора Φ .

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$\Phi abcd = a(bd)(cd). \quad (\Phi)$$

Решение.

Φ —1. Сформулируем постулаты, задающие отношение конвертируемости.

Ф–2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$\begin{aligned} (B^2) \quad B^2xyzw &= x(yzw), & (B) \quad Bxyz &= x(yz), \\ (S) \quad Sxyz &= xz(yz). \end{aligned}$$

Ф–3. Применяя эти схемы к $a(bd)(cd)$, получим:

$$\begin{aligned} a(bd)(cd) &= Babd(cd) && \text{(по схеме (B))} \\ &= S(Bab)cd && \text{(по схеме (S))} \\ &= B^2SBabcd. && \text{(по схеме (B^2))} \end{aligned}$$

Учитывая постулаты, имеем: $B^2SBabcd = a(bd)(cd)$, то есть $\Phi = B^2SB$.

Ответ. Объект Φ с комбинаторной характеристикой $\Phi abcd = a(bd)(cd)$ имеет вид $\Phi = B^2SB$.

Задача 2.12. Вывод выражения для комбинатора Y .

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:

$$Ya = a(Ya). \quad (Y)$$

Решение.

Y–1. Сформулируем постулаты, задающие отношение кон-
вертируемости.

Y–2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$(S) \quad Sxyz = xz(yz), \quad (W) \quad Wxy = xy, \quad (B) \quad Bxyz = x(yz).$$

$Y-3$. Докажем, что $Y = WS(BWB)$.

$$\begin{aligned}
 Ya &= \underline{WS(BWB)} a && \text{(по предположению)} \\
 &= \underline{S(BWB)(BWB)} a && \text{(по схеме (W))} \\
 &= \underline{BW Ba(BW Ba)} && \text{(по схеме (S))} \\
 &= W(Ba)(BW Ba) && \text{(по схеме (B))} \\
 &= Ba(BW Ba)(BW Ba) && \text{(по схеме (W))} \\
 &= a(BW Ba(BW Ba)) && \text{(по схеме (B))} \\
 &= a(\underline{S(BWB)(BWB)} a) && \text{(по схеме (S))} \\
 &= a(\underline{WS(BWB)} a) && \text{(по схеме (W))} \\
 &= a(Ya). && \text{(по предположению)}
 \end{aligned}$$

Следовательно, одно из представлений объекта Y таково: $Y = WS(BWB)$.

Ответ. Объект Y с заданной комбинаторной характеристикой $Ya = a(Ya)$ имеет вид $Y = WS(BWB)$.

2.3 Исторические замечания

Чисто хронологически комбинаторная логика как математический аппарат была введена М. Шейнфинкелем (Moses Schönfinkel) в 1920 году. Его работа была опубликована в 1924 году под названием “О строительных блоках математической логики”. В то время привлекательной казалась задача сведения логики к наипростейшему возможному базису примитивов, хотя в настоящее время это не считается столь уж важным. В этой работе было показано, что в логике можно вовсе избавиться от связанных переменных. Использование функций высших порядков позволило свести логику до языка, содержащего один *конструктор* — апплицирование функции к аргументу, — и три примитивные *константы* — U , C (которая в настоящее время имеет обозначение K) и S . Функция получает название функции ‘высшего порядка’, если ее аргументом может в свою очередь быть функция, либо в результате ее

применения получается функция. Все три константы как раз и являются функциями высшего порядка. Приведем их формальные определения.

Константа C , определяемая посредством

$$Cxy = (C(x))(y) = x,$$

является *константной функцией*, которая для всякого x возвращает x .

Константа S , определяемая посредством

$$Sfgx = ((S(f))(g))(x) = (f(x))(g(x)),$$

является *комбинирующей* для двух функций f и g . Ее результатом для x является значение, полученное применением значения функции f на аргументе x к значению функции g на аргументе x .

Константа U , определяемая посредством

$$UPQ = (U(P))(Q) = \forall x. \neg(P(x) \wedge Q(x)),$$

является обобщение *штриха Шеффера*. Она применяется к двум предикатам, а ее результатом является универсальное обобщение отрицания конъюнкции этих двух предикатов.

Замечание. Этих комбинаторов оказывается достаточно, чтобы выразить произвольные предикаты первого порядка без использования связанных переменных, которые возникают в логике первого порядка, как только речь идет о применении кванторов. Тот же самый результат достигается, когда используются алгоритмы преобразования λ -выражений в комбинаторы. В последнем случае снимаются ограничения на первопорядковость применяемых предикатов. Однако, неограниченное применение такого преобразования при рассмотрении объектов с типами может вызвать нарушение требования непротиворечивости. В комбинаторной логике стремятся избавиться от типов и связанных с ними ограничений, чтобы не столкнуться с тяжелыми требованиями сохранения непротиворечивости.

Работа М. Шейнфинкеля является введением в комбинаторную логику, которое позволяет ясно увидеть, в чем именно заключается исходная мотивация для ее разработки.

Глава 3

Неподвижная точка

Прозрачность комбинаторной логики делает ее весьма простой в изучении. После первых продвижений может показаться, что в ней всегда имеем дело с простыми и конечными по своей природе объектами. Однако это впечатление обманчиво, и пользуясь комбинаторами, можно представлять *процессы*, в том числе циклические вычисления, которые представляют известную в программировании работу со стеком рекурсии.

Эти циклические вычисления представляют собой отображения, содержащие *неподвижную точку*.

3.1 Теоретические сведения

Вычисления с неподвижной точкой являются *представлением* циклическости в программах. Одним из основных результатов λ -исчисления является теорема 3.1 о неподвижной точке.

Теорема 3.1. Для произвольного объекта F найдется объекта X такой, что $X = F X$:

$$\forall F \exists X \ (X = F X).$$

Доказательство. См. [2], с. 140. Положим $P \equiv \lambda x.F(xx)$ и $X \equiv PP$. Тогда

$$X \equiv (\lambda x.F(xx))P = F(PP) \equiv FX,$$

что устанавливает X как неподвижную точку F . □

Особенность доказательства в том, что начинаем с термина X и конвертируем его к FX , а не наоборот. Комбинаторная логика предоставляет специальный концепт-комбинатор Y , называемый *комбинатором неподвижной точки*, который математически выражает цикл в вычислениях — по заданному отображению F строит его неподвижную точку YF .

3.1.1 Абстракция

Для каждого термина M и переменной x терм $[x]M$, называемый *абстракцией M по x* , определяется¹ индукцией по построению термина M :

- (i) $[x]x = I$;
- (ii) $[x]M = KM$, если x не принадлежит M ;
- (iii) $[x]Ux = U$, если x не принадлежит U ;
- (iv) $[x](UV) = S([x]U)([x]V)$, если ни (ii), ни (iii) не подходят.

Пример 3.1. $[x]xy \stackrel{(iv)}{=} S([x]x)([x]y) \stackrel{(i)}{=} SI([x]y) \stackrel{(ii)}{=} SI(Ky)$.

Теорема 3.2 (принцип свертывания). Для произвольных объектов M, N и переменной x приложение абстракции $([x]M)$ к объекту N свертывается по принципу: ‘подставить N вместо x всюду, где x имеет свободные вхождения в M ’:

$$\forall M, N, x : ([x]M)N = [N/x]M.$$

Доказательство. См. в [94]. □

¹Терм $'[x]M'$, или $'[x].M'$ для наших целей пока можно считать аналогичным терму $'\lambda x.M'$.

3.1.2 Мультиабстракция

Для переменных x_1, \dots, x_m (не обязательно различных) определим:

$$[x_1, \dots, x_m]M \stackrel{\text{def}}{=} [x_1]([x_2](\dots([x_m]M)\dots)).$$

Пример 3.2. $[x, y]x = [x]([y]x) \stackrel{(ii)}{=} [x](Kx) \stackrel{(iii)}{=} K.$

3.1.3 Локальная рекурсия

Важное применение комбинатора неподвижной точки дает использование в программах рекурсивных определений. Элементарно рассмотрим случай локальной рекурсии. Локальная рекурсия вида

$$E1 \text{ where } x = \dots x \dots$$

преобразуется в

$$([x]E1)(Y([x](\dots x \dots))),$$

где Y — комбинатор неподвижной точки, определяемый равенством:

$$Y f = f(Y f).$$

Для функции f выражение $Y f$ — это неподвижная точка f .

При использовании в тексте программы взаимная рекурсия в *where*-предложении вида

$$\begin{aligned} E1 \text{ where } f \ x &= \dots g \dots \\ g \ y &= \dots f \dots \end{aligned}$$

транслируется сначала в выражение:

$$\begin{aligned} E1 \text{ where } f &= [x] (\dots g \dots) \\ g &= [y] (\dots f \dots), \end{aligned}$$

из которого устранены *свободные* переменные x и y . Затем пара взаимно рекурсивных определений преобразуется в одно общее рекурсивное определение:

$E1$ where $(f, g) = ([x](\dots g \dots), [y](\dots f \dots))$,

которое может быть скомпилировано в выражение:

$([f, g]E1) (Y ([f, g]([x](\dots g \dots), [y](\dots f \dots))))$

с использованием уже известного правила.

3.2 Основные задачи

Задача 3.1. Исследовать свойства заданного объекта

$$Y \equiv (\lambda x.(P(xx)a))(\lambda x.(P(xx)a)).$$

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии и схем $(K), (S)$:

$$Y = (\lambda x.(P(xx)a))(\lambda x.(P(xx)a)). \quad (Y)$$

Решение.

Y-1. Вид объекта Y известен заранее:

$$Y \equiv (\lambda x.(P(xx)a))(\lambda x.(P(xx)a)).$$

Y-2. Применяем правило подстановки (β) к его представлению:

$$\begin{aligned} Y &= (\lambda x.(P(xx)a))(\lambda x.(P(xx)a)) \\ &= (P((\lambda x.(P(xx)a))(\lambda x.(P(xx)a)))a \\ &= P(Y)a. \end{aligned} \quad (\beta)$$

Итак, имеем $Y = PYa = P(PYa)a = \dots$

Ответ. Комбинаторная характеристика исходного объекта $Y = (\lambda x.(P(xx)a))(\lambda x.(P(xx)a))$ имеет вид: $Y = PYa$.

Задача 3.2. Исследовать свойства заданного объекта:

$$Y \equiv S(BWB)(BWB).$$

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии и схем (K) , (S) :

$$Y = S(BWB)(BWB). \quad (Y)$$

Решение.

Y-1. Вид объекта Y : $Y = S(BWB)(BWB)$.

Y-2. Запишем комбинаторные характеристики следующих объектов: $Babc = abc$, $Sabc = ac(bc)$, $Wab = abb$.

Y-3. Приложим объект Y к a :

$$\begin{aligned} Ya &\equiv \underbrace{S(BWB)(BWB)}_{\equiv Y} a && \text{(по опр.)} \\ &= BWBa(BWBa) && \text{(по схеме } S) \\ &= W(Ba)(BWBa) && \text{(по схеме } B) \\ &= Ba(BWBa)(BWBa) && \text{(по схеме } W) \\ &= a(BWBa(BWBa)) && \text{(по схеме } B) \\ &= a(\underbrace{S(BWB)(BWB)}_{\equiv Y} a) && \text{(по схеме } S) \\ &\equiv a(Ya). && \text{(по опр.)} \end{aligned}$$

Таким образом, Ya является неподвижной точкой для a .

Ответ. Комбинаторная характеристика исходного объекта $Y = S(BWB)(BWB)$ имеет вид: $Ya = a(Ya)$.

Задача 3.3. Исследовать свойства заданного объекта:

$$Y \equiv WS(BWB).$$

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии и схем (K) , (S) :

$$Y = WS(BWB). \quad (Y)$$

Решение.

Y-1. Вид объекта Y : $Y = WS(BWB)$.

Y-2. Запишем комбинаторные характеристики следующих объектов:

$$Babc = abc, Sabc = ac(bc), Wab = abb.$$

Y-3. По схеме (W) получаем:

$$Y \equiv WS(BWB) = S(BWB)(BWB).$$

Таким образом, объект Y имеет ту же комбинаторную характеристику, что и объект Y из предыдущей задачи.

Y-4. Применим объект Y к a :

$$\begin{aligned} Ya &\equiv S(BWB)(BWB)a && \text{(по опр.)} \\ &= BWBa(BWBa) && \text{(по схеме } S) \\ &= W(Ba)(BWBa) && \text{(по схеме } B) \\ &= Ba(BWBa)(BWBa) && \text{(по схеме } W) \\ &= a(BWBa(BWBa)) && \text{(по схеме } B) \\ &= a(S(BWB)(BWB)a) && \text{(по схеме } S) \\ &\equiv a(Ya) && \text{(по опр.).} \end{aligned}$$

Y-5. В итоге получаем $Ya = a(Ya)$.

Ответ. Комбинаторная характеристика объекта $Y \equiv S(BWB)$ имеет вид: $Ya = a(Ya)$.

Задача 3.4. Исследовать свойства заданного объекта:

$$Y_0 \equiv \lambda f.XX, \text{ где } X \equiv \lambda x.f(xx). \quad (Y_0)$$

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии и схем (K) , (S) :

$$Y_0 \equiv \lambda f.XX, \text{ где } X \equiv \lambda x.f(xx).$$

Решение.

Y_0 —1. Вид объекта Y_0 : $Y_0 = \lambda f.XX$, где $X = \lambda x.f(xx)$.

Y_0 —2. Рассмотрим сначала объект (XX) .

$$\begin{aligned} XX &= (\lambda x.f(xx))(\lambda x.f(xx)) && \text{(по опр.)} \\ &= f((\lambda x.f(xx))(\lambda x.f(xx))) && \text{(по } \beta) \\ &= f(XX). && \text{(по опр.)} \end{aligned}$$

Значит,

$$XX = f(XX). \quad (*)$$

Y_0 —3. Теперь применим Y_0 к произвольному объекту a :

$$\begin{aligned} Y_0 a &\equiv (\lambda f.XX)a && \text{(по опр.)} \\ &= (\lambda f.f(XX))a && \text{(по } (*)) \\ &= (\lambda f.f((\lambda x.f(xx))(\lambda x.f(xx))))a && \text{(по опр. } X) \\ &= a((\lambda x.a(xx))(\lambda x.a(xx))) && \text{(по } \beta) \\ &= a((\lambda f.((\lambda f.f(xx))(\lambda x.f(xx))))a) && \text{(по } \beta, \xi) \\ &= a((\lambda f.(XX))a) && \text{(по опр. } X) \\ &\equiv a(Y_0 a). && \text{(по опр. } Y_0) \end{aligned}$$

Учитывая постулат транзитивности τ , получаем: $Y_0 a = a(Y_0 a)$.

Ответ. Комбинаторная характеристика объекта Y_0 имеет следующий вид: $Y_0 a = a(Y_0 a)$.

Задача 3.5. Исследовать свойства заданного объекта:

$$Y_1 \equiv Y_0(\lambda y. \lambda f. f(yf)), \text{ где } Y_0 \equiv \lambda f. XX, X = \lambda x. f(xx).$$

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ -конверсии и схем (K) , (S) :

$$Y_1 \equiv Y_0(\lambda y. \lambda f. f(yf)), \text{ где } Y_0 \equiv \lambda f. XX, X = \lambda x. f(xx). \quad (Y_1)$$

Решение.

Y_1 –1. Вид объекта Y_1 : $Y_1 = Y_0(\lambda y. \lambda f. f(yf))$, где выполняются равенства $Y_0 = \lambda f. XX$, и $X = \lambda x. f(xx)$.

Y_1 –2. Имеем $Y_0 a = a(Y_0 a)$,

$$\begin{aligned} Y_1 a &= Y_0(\lambda y. \lambda f. f(yf)) a && \text{(по опр.)} \\ &= (\lambda y. \lambda f. f(yf))(Y_0(\lambda y. \lambda f. f(yf))) a && \text{(по } (Y_0)) \\ &= (\lambda y f. f(yf)) Y_1 a && \text{(по } (Y_1)) \\ &= a(Y_1 a). && \text{(по } \beta) \end{aligned}$$

Итак, имеем $Y_1 a = a(Y_1 a)$.

Ответ. Комбинаторная характеристика объекта Y_1 имеет вид: $Y_1 a = a(Y_1 a)$.

Задача 3.6. Применить функцию Y для получения представления циклического списка L .

Формулировка задачи. Циклический список L , который определяется посредством

$$L = (1 : 2 : 3 : L),$$

дает бесконечный периодический список

$$L = 1, 2, 3, 1, 2, 3, 1, 2, 3, \dots$$

Указать конечное представление этой структуры данных, не используя самоссылающихся определений.

Решение. Эта циклическая конструкция ставит в соответствие L список, первый элемент которого является 1, второй — 2, третий — 3, четвертый — 1 и так далее.

$L-1$. Выполним цепочку преобразований:

$$\begin{aligned} L &= (1 : 2 : 3 : L) \\ &= (\lambda L.(1 : 2 : 3 : L))L. \text{ по } (\beta) \end{aligned}$$

$L-2$. Поскольку $L \notin (\lambda L.(1 : 2 : 3 : L))$, то по теореме о неподвижной точке

$$L = Y(\lambda L.(1 : 2 : 3 : L)).$$

Ответ. $L = Y(\lambda L.(1 : 2 : 3 : L)).$

Упражнения

Упражнение 3.1. Рассмотрим циклическое определение

```
s(n,k) =
  if k = 1
  then 1
  else if k = n
        then 1
        else s(n - 1, k - 1) + k * s(n - 1, k)
```

чисел Стирлинга второго рода. Указать конечное представление этой функции, не используя самоссылающихся определений.

Указание. Поскольку, например,

$$\begin{aligned} s(4,2) &= s(3,1) + 2 * s(3,2) \\ &= 1 + 2 * (s(2,1) + 2 * s(2,2)) \\ &= 1 + 2 * (1 + 2 * 1) = 7, \end{aligned}$$

то имеем дело с рекурсивными вычислениями. Попробуйте построить λ -абстракцию по n и k , а затем применить теорему о неподвижной точке.

Упражнение 3.2. Устранить цикличность в приводимых определениях:

```
длина x =  
  if null x  
  then 0  
  else 1 + длина (tail x)  
  
факториал n =  
  if zero n  
  then 1  
  else n × факториал (n - 1)
```

Указание. Циклическое определение приводится к стандартной форме, в которой определяемая часть является идентификатором, а определяющая часть является выражением. Это достигается введением самоссылающейся функции, которая использует функцию поиска неподвижной точки Y . Характеристическое свойство этой функции: $Yf = f(Yf)$.

Ссылающееся на себя определение функции может иметь вид $f = Ef$, в котором выражение E не содержит вхождений f . Одним из решений этого уравнения является $f = YE$.

Глава 4

Экстенциональность

Комбинаторная логика имеет ту особенность, что в ней строятся и применяются функции с *заранее не фиксированным числом аргументов*. Это означает, что ответ на вопрос, сколько же у применяемой функции-объекта в действительности аргументных мест требует известной осторожности. На самом деле аккуратное использование довольно простых принципов экстенциональности (расширяемости) позволяет преодолеть эту неопределенность.

4.1 Теоретические сведения

Для модели M обычно требуется выполнение некоторого свойства. В частности, пусть объект F построен, самое большее, из свободных переменных x_0, \dots, x_{n-1} — с помощью способов комбинирования. Помимо именно этих свободных переменных в F не могут входить никакие другие свободные переменные. Тогда *комбинаторная полнота* модели M понимается как существование в ней такого объекта (концепта) f , что для всяких переменных x_0, \dots, x_{n-1} справедливо суждение

$$f x_0 \dots x_{n-1} = F.$$

Другими словами, в модели в явном виде существует как объект самостоятельный концепт f , реализующий ту же самую смысловую нагрузку, что и F -комбинация других объектов (с ограничением на использование свободных переменных).

На практике часто используются модели, в которых из равенства функций f и g , вычисленных на произвольном аргументе d , следует равенство самих функций как объектов:

$$\frac{\forall d \in D. (fd = gd)}{f = g}, \quad (ext)$$

где по соглашению полагают $fd = (fd)$ и $gd = (gd)$. Такие модели называются экстенциональными (*ext*).

Для аппликативных структур справедлива более сильная форма комбинаторной полноты: *существует тот единственный концепт f , что для всяких свободных в F переменных x_0, \dots, x_{n-1} выполняется*

$$fx_0 \dots x_{n-1} = F(x_0, \dots, x_{n-1}),$$

или, записывая формально:

$$\mathbf{If}.\forall x_0 \dots \forall x_{n-1} (fx_0 \dots x_{n-1} = F(x_0, \dots, x_{n-1})).$$

В этой записи символ *описания* '**I**' использован как сокращение для 'тот единственный \dots , что \dots '. Это, фактически, *принцип свертывания*:

комбинация объектов F , среди которых в качестве свободных переменных использованы только лишь x_0, \dots, x_{n-1} , сворачивается до единственного объекта (концепта) f со всеми теми и только теми свойствами, которые присущи комбинации $F(x_0, \dots, x_{n-1})$.

Это выполняется только для экстенциональных аппликативных структур и позволяет от сложной формы записи переходить к анализу единственного объекта, снижая громоздкость в рассуждениях.

Особую роль играет постулат

$$\lambda x.Xx = X, x \notin X, \quad (\eta)$$

который, как покажем, в аппликативной структуре является аналогом экстенциональности.

4.2 Задачи

Задача 4.1. Доказать требуемое равенство:

$$\lambda xy.xy = \lambda x.x.$$

Формулировка задачи. Доказать, что равенство:

$$\lambda xy.xy = \lambda x.x \quad (\bar{1})$$

выводимо в $\eta\xi$ -исчислении λ -конверсии (знак '=' представляет собой отношение конвертируемости).

Доказательство.

η -1. Приведем применяемые постулаты:

$$(\eta) \quad \lambda x.Xx = X, x \notin X, \quad (\xi) \quad \frac{a = b}{\lambda x.a = \lambda x.b},$$

$$(\tau) \quad \frac{a = b, b = c}{a = c}.$$

Здесь: ' $x \notin X$ ' понимается как ' x не имеет свободных вхождений в X '.

η -2. Применяя эти постулаты, получим:

$$\begin{array}{lll} (1) & \lambda xy.xy & \equiv \lambda x.(\lambda y.xy), \quad (\text{по определению}) \\ (2) & \lambda y.xy & = x, \quad (\text{по схеме } (\eta)) \\ (3) & \lambda x.(\lambda y.xy) & = \lambda x.x, \quad ((2), (\xi)) \\ (4) & \lambda xy.xy & = \lambda x.x. \quad ((1), (3); (\tau)) \end{array}$$

Таким образом, в $\eta\xi$ -системе выводимо $\lambda xy.xy = \lambda x.x$. □

В качестве альтернативной формы записи приведем сокращенную форму изложения доказательства — в виде дерева вывода, которое надо читать в направлении ‘сверху-вниз’. Над горизонтальной чертой записываются *посылки*, а под чертой — *закключение*. Сама черта понимается как замена слова ‘следовательно’:

$$\frac{\frac{\lambda y.xy \quad x}{\lambda y.xy = x} \quad (\eta)}{\lambda xy.xy = \lambda x.x} \quad (\xi)$$

Задача 4.2. Пользуясь λ -обозначениями ([]-обозначениями), выразить конструкцию:

$$f(x) + f(f(7)) \text{ where } f(x) = \text{sqr}(x) + 3$$

Формулировка задачи. В данной задаче предлагается рассмотреть конструкции

$$M \text{ where } f(x) = N \quad \text{или} \quad \text{let } f(x) = N \text{ M},$$

частные случаи которых

$$M \text{ where } x = N \quad \text{или} \quad \text{let } x = N \text{ M},$$

представляются λ -термом:

$$(\lambda x.M)N.$$

Решение. Конструкция **where** соответствует программированию ‘сверху вниз’, когда вначале предполагается, что объекты существуют, а затем их определяют. Конструкция **let** соответствует программированию ‘снизу вверх’, когда объекты определяются до своего употребления.

η —1. Каждое из этих выражений состоит из двух частей:

- 1) выражения M , называемого *главным выражением*, или *телом*;
- 2) определения, имеющего одну из форм:

$$\text{where } f(x) = N \quad \text{или} \quad \text{let } (x) = N.$$

η -2. В общем случае выражение

$$M \text{ where } f(x) = N$$

можно привести к виду оператор/операнд в два этапа:

- 1) сначала построим выражение

$$M \text{ where } f = \lambda x. N;$$

- 2) затем получим выражение

$$(\lambda f. M) (\lambda x. N).$$

η -3. Таким образом, для объектов

$$M \equiv f(x) + f(f(7)), \quad N \equiv \text{sqr}(x) + 3$$

получаем, что

$$(\lambda f. f(x) + f(f(7))) (\lambda x. \text{sqr}(x) + 3).$$

Ответ. Исходная конструкция языка программирования представима посредством:

$$(\lambda f. f(x) + f(f(7))) (\lambda x. \text{sqr}(x) + 3),$$

или, в иных обозначениях:

$$([f]. f(x) + f(f(7))) ([x]. \text{sqr}(x) + 3).$$

Упражнения

Упражнение 4.1. Проанализируйте решение задачи 4.2 и укажите, где и в каком контексте используется постулат (η).

Упражнение 4.2. Привести циклическое определение

```
length x =
  if null x
  then 0
  else 1 + length (tail x)
```

к стандартной форме, где в левой части записан определяемый идентификатор, а в правой — определяющее выражение.

Указание. Построим цепочку преобразований:

```
length = [x].if null x
         then 0
         else 1 + length (tail x)
= ([length].[x].if null x
   then 0
   else 1 + length (tail x))length
= Y([length].[x].if null x
    then 0
    else 1 + length (tail x))
```

В ходе преобразования применяется постулат (η). Определение имеет теперь стандартную форму, а его правая часть не содержит определяемого идентификатора `length`, поскольку переменная `length` связана.

Глава 5

Нумералы

Обращаем внимание, что *с самого начала* в комбинаторной логике среди первичных объектов нет ... чисел. Дело в том, что концепцию числа можно разработать самостоятельно, пользуясь известными комбинаторами. Тогда числа предстают в несколько необычном облике — они являются объектами, проявляющими свою арность в зависимости от используемой системы постулатов. Точно так же в виде комбинаторов удастся разработать арифметические операции. Другими словами, арифметические сущности встраиваются в комбинаторную логику. Эта ситуация хорошо знакома в объектно-ориентированном программировании — приложение (арифметические объекты со своими правилами) встраивается в программную среду (комбинаторную логику).

5.1 Числа и нумералы

Как известно, одним из основных первичных математических понятий является понятие *числа*. Пользуясь числами, строят другие объекты, более близко представляющие содержательные понятия предметной области. В теоретических исследованиях “хорошим тоном” считается свести построенную теорию к какой-либо

заранее заданной арифметической системе.

Возникает вопрос, так ли уж неизбежно использовать в качестве первичного объекта понятие числа. Это один из самых сложных вопросов современной математики, попытки получения ответа на который приводят к далеко идущим последствиям.

Тем не менее при построении комбинаторной логики или λ -исчисления среди исходных объектов нет чисел. Нет ли в этих системах недостатка в выразительных возможностях? Оказывается, что в рамках комбинаторной логики или λ -исчисления можно установить такие комбинаторы или, соответственно, термы, которые ведут себя как числа. Эти представления чисел получили название *нумералов*. Нумералы как комбинаторы удовлетворяют всем законам комбинаторной логики. Более того, можно указать комбинаторы, представляющие арифметические операции, например, сложение чисел. Исследования в этой области пока все еще далеки от завершения.

5.2 Комбинаторная арифметика

Построение арифметики обычно начинается с введения натуральных чисел (см. А.С. Кузичев, [29]). В комбинаторной логике для этого вводится два объекта, при записи которых использован оператор абстракции (см. пункт 3.1.1 на стр. 66 и далее):

$$Z_0 \equiv [xy]y \text{ и } \hat{\sigma} \equiv [xyz](y(xyz)).$$

Первый из них является представлением числа 0, а второй — представлением операции прибавления единицы ‘+1’ и называется *комбинатором следования*.

Комбинаторные числа порождаются последовательно, индукцией по построению, или структурной индукцией:

- i) Z_0 считается комбинаторным числом;

- ii) комбинаторное число Z_{k+1} , являющееся представлением натурального числа $k + 1$, принимается равным $\hat{\sigma} Z_k$ для $k \geq 0$.

Утверждение 5.1. Можно показать, что комбинатор Z_n обладает свойством:

$$Z_n XY = \underbrace{X(X(\dots X(XY) \dots))}_n,$$

где X и Y — объекты, а $n \geq 0$.

Доказательство. Индукцией по n .

$$\text{i) } Z_0 = [xy]y, \text{ а } Z_0 XY = \underbrace{}_0 Y;$$

$$\text{ii) } Z_1 = \hat{\sigma} Z_0, \text{ а}$$

$$\begin{aligned} Z_1 XY &= \hat{\sigma} Z_0 XY \\ &= ([xyz](y(xyz)))([xy]y)XY \\ &= X([xy]y)XY \\ &= \underbrace{X}_1 Y; \end{aligned}$$

и т.д. □

Для введения других арифметических объектов потребуется комбинатор пары

$$\mathcal{D} \equiv [xyz](z(Ky)x)$$

со следующими свойствами:

$$1) \mathcal{D}XYZ_0 = X;$$

$$2) \mathcal{D}XY(\hat{\sigma}U) = Y \text{ для объектов } X, Y \text{ и } U.$$

Для формирования сколько-нибудь представительной арифметики добавим еще ряд арифметических объектов.

Предшественник

Операция предшествования ‘ -1 ’ обозначается через π и определяется посредством

$$\pi \equiv [x](x\bar{\sigma}(KZ_0)Z_1),$$

где $\bar{\sigma} \equiv [x](\mathcal{D}(\hat{\sigma}(xZ_0))(xZ_0))$. Комбинатор ‘предшествования’ π может рассматриваться как обращение операции ‘следования за’ $\hat{\sigma}$.

Перечислим некоторые свойства комбинатора π , которые доказываются индукцией по n , где $n \geq 0$:

- 1) $\pi Z_0 = Z_0$;
- 2) $Z_n \bar{\sigma}(KZ_0)Z_0 = Z_n$;
- 3) $\pi Z_{n+1} = Z_n$.

Модифицированное вычитание

Операция *модифицированного*, или *усеченного вычитания* в обычной арифметике обозначается через ‘ $-$ ’ и определяется посредством:

$$a-b = \begin{cases} a-b, & \text{если } a > b; \\ 0, & \text{если } a \leq b. \end{cases}$$

Для представления усеченного вычитания в комбинаторной арифметике используется комбинатор \mathbb{L} , задаваемый посредством

$$\mathbb{L} \equiv [xy](y\pi x)$$

и имеющий свойства:

- 1) $\mathbb{L}Z_nZ_0 = Z_n$;
- 2) $\mathbb{L}Z_nZ_{m+1} = \pi(\mathbb{L}Z_nZ_m)$.

Операция минимума

На основе комбинатора \mathbb{L} усеченного вычитания можно построить комбинатор $\overline{\min}$, представляющий в комбинаторной арифметике функцию $\min(a, b)$. Комбинатор $\overline{\min}$ определяется посредством

$$\overline{\min} \equiv [xy](\mathbb{L}x(\mathbb{L}xy)),$$

где

$$\overline{\min}Z_nZ_m = \begin{cases} Z_n, & \text{если } n \leq m; \\ Z_m, & \text{если } n > m. \end{cases}$$

Комбинатор взятия минимума в свою очередь может быть использован для построения других арифметических объектов.

Пример 5.1. Положим

$$\alpha \equiv \overline{\min}Z_1.$$

Очевидно, что:

- 1) $\alpha Z_0 = Z_0$;
- 2) $\alpha Z_m = Z_1$ для $m > 0$.

Сложение

Комбинатор \mathbb{A} , представляющий операцию сложения, определяется посредством

$$\mathbb{A} \equiv [x](\mathcal{E}\mathcal{E}x),$$

где

$$\begin{aligned} \mathcal{E} &\equiv [xy](Uy[z](xxz)), & U &\equiv [x](\mathcal{C}(\mathcal{M}xZ_1)x), \\ \mathcal{C} &\equiv [x](xCZ_0TV), & \mathcal{V} &\equiv [xy](xyI), \\ T &\equiv [xyz](\hat{\sigma}(y(\pi x)z)). \end{aligned}$$

Упражнение 5.1. Проверить, что комбинатор \mathbb{A} обладает всеми свойствами операции сложения '+', то есть для $n, m \geq 0$:

$$\mathbb{A}Z_0Z_m = Z_m; \quad \mathbb{A}Z_{n+1}Z_m = \hat{\sigma}(\mathbb{A}Z_nZ_m).$$

Симметрическая разность

Симметрическая, или положительная разность в обычной арифметике является операцией $'(x \dot{-} y) + (y \dot{-} x)'$. Комбинатор \mathbb{R} , представляющий симметрическую разность, определяется посредством

$$\mathbb{R} \equiv [xy](\mathbb{A}(Lxy)(Lux)).$$

Умножение

Комбинатор *умножения* \mathbb{M} определяется по аналогии с комбинатором сложения \mathbb{A} с той разницей, что вместо комбинатора \mathcal{V} используется комбинатор \mathcal{V}' :

$$\mathcal{V}' \equiv [xy](xy(KZ_0)),$$

а вместо комбинатора T используется комбинатор T' :

$$T' \equiv [xyz](\mathbb{A}(y(\pi x)z)z).$$

Упражнение 5.2. Показать, что комбинатор \mathbb{M} имеет свойства оператора умножения на арифметических объектах:

$$\mathbb{M}Z_0Z_m = Z_0; \quad \mathbb{M}Z_{n+1}Z_m = A(\mathbb{M}Z_nZ_m)Z_m.$$

Комбинаторно определимые функции

Сумма и произведение являются примерами класса *комбинаторно-определимых* функций. Построение см., например, в работах (А.С. Кузичев, [29]; Х. Барендрегт, [2]). Перечислим наиболее существенные для комбинаторно-определимых функций понятия.

Определение 5.1 (комбинаторная определимость). Функцию ϕ от n аргументов называют *комбинаторно определимой*, если найдется объект X такой, что для любого набора из n натуральных чисел r_1, \dots, r_n , для которого

$$\phi(r_1, \dots, r_n) = r,$$

где r — натуральное число, выполняется

$$X\overline{r_1} \dots \overline{r_n} = \overline{r}$$

для $n \geq 0$. Здесь: для $n = 0$ имеем $X = \overline{r}$; арифметические объекты (нумералы), представляющие натуральные числа r_i выделены надчеркиванием — $\overline{r_i}$.

Таким образом, надчеркивание рассматривается как отображение

$$\overline{} : o \mapsto \overline{o}, \quad \overline{}(o) = \overline{o},$$

которое для объекта o строит соответствующий ему арифметический объект \overline{o} .

Примитивная рекурсия

Рассмотренный способ построения комбинаторов, представляющих суммы и произведения, распространяется на произвольные функции, задаваемые схемой *примитивной рекурсии*:

$$\begin{aligned} \phi(0, r_1, \dots, r_n) &= \psi(r_1, \dots, r_n); \\ \phi(r+1, r_1, \dots, r_n) &= \psi(r, \phi(r, r_1, \dots, r_n), r_1, \dots, r_n), \end{aligned}$$

где ϕ и ψ — заранее определенные функции от соответствующего числа аргументов.

Действительно, можно положить, что ϕ и ψ комбинаторно определяются объектами g и H соответственно. В качестве объекта, комбинаторно определяющего функцию ϕ , которая задается схемой рекурсии, можно выбрать

$$\mathcal{E} \equiv [x](\mathcal{E}^* \mathcal{E}^* x),$$

где:

$$\begin{aligned} \mathcal{E}^* &\equiv [xy](Uy[z](xxz)), & U &\equiv [x](C^*(\overline{\min} \ x \ \overline{1})x), \\ C^* &\equiv [x](xC0TV), & V &\equiv [xy](xyg), \\ T &\equiv [xyz_1 \dots z_n]H(\pi x)(y(\pi x)z_1 \dots z_n)z_1 \dots z_n. \end{aligned}$$

Для такого выбора можно показать, что:

$$\begin{aligned}\frac{g \overline{0} \overline{r_1} \dots \overline{r_n}}{gr + 1 \overline{r_1} \dots \overline{r_n}} &= \frac{\overline{\psi(r_1, \dots, r_n)}}{\overline{\psi(r, \phi(r, r_1, \dots, r_n), r_1, \dots, r_n)}}.\end{aligned}$$

5.3 Задачи

Задача 5.1. Определить объекты, обладающие свойствами натуральных чисел (нумералов) и исследовать их свойства.

Формулировка задачи. Нумералы — это следующие объекты:

$$\overline{n} \stackrel{\text{def}}{=} \lambda xy. (x^n)y,$$

где n — натуральное число из множества $\{1, 2, 3, \dots\}$. Показать, что нумералы это объекты с характеристикой:

$$\overline{n} = (SB)^n(KI). \quad (\overline{n})$$

Решение.

$\overline{n}-1$. Форма записи $x^n y$ определяется по индукции:

$$\begin{aligned}(i) \quad x^0 y &= y, \\(ii) \quad x^{n+1} y &= x(x^n y), \quad n \geq 0.\end{aligned}$$

Таким образом, $x^4 y = x(x(x(xy)))$.

$\overline{n}-2$. Проверим поведение объектов $\overline{n} = (SB)^n(KI)$ для $n = 0, 1$.

$$\begin{aligned}\overline{0} &= (SB)^0(KI) = KI, \\ \overline{0}ab &= KIab = Ib = b = (\lambda xy. y)ab, \\ \overline{1} &= SB(KI), \\ \overline{1}ab &= SB(KI)ab = Ba(KIa)b = BaIb \\ &= a(Ib) = ab = (\lambda xy. xy)ab.\end{aligned}$$

$\bar{n}-3$. Проверим поведение $\bar{n} = (SB)^n(KI)$ в общем случае.

$$\begin{aligned}
 \bar{n}ab &= (SB)^n(KI)ab \\
 &= SB((SB)^{n-1}(KI))ab && (\text{по опр.}) \\
 &= Ba((SB)^{n-1}(KI)a)b && (\text{по } (S)) \\
 &= a((SB)^{n-1}(KI)ab) && (\text{по } (B)) \\
 &= a(\bar{n} - 1ab) && (\text{по опр.}) \\
 &= a(a^{n-1}b) && (\text{по опр.}) \\
 &= a^n b = (\lambda xy.x^n y)ab. && (\text{по опр.})
 \end{aligned}$$

Ответ. Нумералы $\bar{n} = (\lambda xy.x^n y)$ имеют вид $(SB)^n(KI)$.

Задача 5.2. Определить объект, представляющий операцию ‘+1’ на множестве нумералов и исследовать его свойства.

Формулировка задачи. Показать, что $\sigma = \lambda xyz.xy(yz)$ задает на множестве нумералов функцию ‘следования за’ (‘прибавление единицы’):

$$\sigma \bar{n} = \overline{n+1}. \quad (\sigma)$$

Решение.

$\sigma-1$. Комбинаторная характеристика нумералов:

$$\bar{n}ab = a^n b.$$

$\sigma-2$. Применим функцию σ к нумералу \bar{n} в общем виде:

$$\begin{aligned}
 \sigma \bar{n}ab &= (\lambda xyz.xy(yz))\bar{n}ab && (\text{по опр.}) \\
 &= \bar{n}a(ab) && (\text{по } (\beta)) \\
 &= a^n(ab) && (\text{по опр. } (\bar{n})) \\
 &= \overline{a^{n+1}b} && (\text{по опр.}) \\
 &= \overline{n+1}ab. && (\text{по опр.})
 \end{aligned}$$

Итак, $\overline{\sigma nab} = \overline{n + 1ab}$, то есть $\overline{\sigma n} = \overline{n + 1}$.

Ответ. Функция $\sigma = \lambda xyz.xy(yz)$ есть функция следования для нумералов $n = \lambda xy.x^n y$.

Задача 5.3. Определить объект, вычисляющий длину конечной последовательности (списка).

Формулировка задачи. Показать, что функция

$$Length = \lambda xy.Null\ x\ \overline{0}(\sigma(Length(Cdr\ x))y).$$

есть функция определения длины списка x :

$$Length\ < a_1, a_2, \dots, a_n > = n \quad (Length)$$

Решение.

Length-1. Введем вспомогательные функции *Null* и *Cdr* :

$$Null\ x = \begin{cases} \overline{1}, & \text{если } x = NIL = \langle \rangle \\ (x - \text{пустой список}), \\ \overline{0}, & \text{в противном случае;} \end{cases} \quad (Null)$$

$$\overline{1} = \lambda xy.xy = \lambda x.x = I, \quad (\overline{1})$$

$$\overline{0} = \lambda xy.y = KI, \quad (\overline{0})$$

$$\overline{\sigma n} = \overline{n + 1},$$

$$Cdr\ x = \begin{cases} NIL = \langle \rangle, & \text{для } x = \langle a_1 \rangle, \\ \langle a_2, \dots, a_n \rangle, & \text{для} \\ x = \langle a_1, a_2, \dots, a_n \rangle. \end{cases} \quad (Cdr)$$

Length-2. Приложим *Length* к пустому списку *Nil*:

$$\begin{aligned}
 \text{Length Nil} &= \\
 &= \lambda y. \text{Null Nil } \bar{0}(\sigma(\text{Length}(\text{Cdr Nil}))y) && (\text{ по } (\beta)) \\
 &= \lambda y. \bar{1} \bar{0}(\sigma(\text{Length}(\text{Cdr Nil}))y) && (\text{ по } (\text{Null})) \\
 &= \lambda y. I(KI)(\sigma(\text{Length}(\text{Cdr Nil}))y) && (\text{ по } (\bar{0}), (\bar{1})) \\
 &= \lambda y. KI(\sigma(\text{Length}(\text{Cdr Nil}))y) && (\text{ по } (I)) \\
 &= \lambda y. I && (\text{ по } (K)) \\
 &= \lambda y. (\lambda z. z) && (\text{ по } (I)) \\
 &= \lambda yz. z = \bar{0}. && (\text{ по } (\bar{0}))
 \end{aligned}$$

Таким образом, функция *Length* корректна по отношению к применению к пустому списку, то есть $\text{Length Nil} = \bar{0}$.

Length-3. Приложим функцию *Length* к списку *x*, состоящему из одного элемента: $x = \langle a \rangle$:

$$\begin{aligned}
 \text{Length } x &= \\
 &= \lambda y. \text{Null } x \bar{0}(\sigma(\text{Length}(\text{Cdr } x))y) && (\beta) \\
 &= \lambda y. \bar{0} \bar{0}(\sigma(\text{Length } Nil)y) && (\text{Null}), (\text{Cdr}) \\
 &= \lambda y. KI(KI)(\sigma \bar{0}y) && (\bar{0}) \\
 &= \lambda y. I(\sigma \bar{0}y) && (K) \\
 &= \lambda y. \sigma \bar{0}y && (I) \\
 &= \lambda y. \bar{1}y && (\sigma) \\
 &= \lambda y. Iy = \lambda y. y = I = \bar{1}.
 \end{aligned}$$

Функция *Length* корректна по отношению к применению к списку, состоящему из одного элемента, то есть равна $\bar{1}$.

Length-4. Теперь проверим *Length* в общем случае, для непустого списка *x* длины *n*: $x \neq Nil$, где знак ' \neq ' означает

‘не конвертируется к’:

$$\begin{aligned}
 \text{Length } x &= \lambda y. \text{Null } x \bar{0}(\sigma(\text{Length}(\text{Cdr } x))y) \\
 &= \lambda y. \bar{0} \bar{0}(\sigma(\text{Length}(\text{Cdr } x))y) \\
 &= \lambda y. \sigma(\text{Length}(\text{Cdr } x))y \\
 &= \lambda y. \sigma n - 1 y \\
 &= \lambda y. ny \\
 &= \lambda y. (\lambda xz. x^n z)y = \lambda y. \lambda z. y^n z = \lambda yz. y^n z = \bar{n}.
 \end{aligned}$$

Ответ. Функция *Length* действительно вычисляет длину списка.

Упражнения

Показать или опровергнуть следующее.

Упражнение 5.3. Сложение определяется λ -выражением

$$\lambda m n f x. m f (n f x).$$

Упражнение 5.4. Произведение определяется λ -выражением

$$\lambda m \lambda n \lambda f. m(n f).$$

Упражнение 5.5. Возведение в степень определяется λ -выражением

$$\lambda m \lambda n. n m,$$

поскольку, например, $Z_3 Z_2 f x = (Z_2)^3 f x = f^8 x$.

Глава 6

Комбинаторы с типами

Концепция класса является одной из самых основных в объектно-ориентированных рассуждениях. Класс в этом случае понимается как образец для создания экземпляров конкретных объектов. Более того, классы сами могут рассматриваться как объекты. Точно также комбинаторы классифицируются, или типизируются. Существенным для комбинаторов оказывается высокий порядок функциональных пространств. Тем не менее интуитивная ясность работы с комбинаторами как с объектами не теряется.

6.1 Представление о типе

Неформальное обсуждение понятия типа иллюстрирует довольно прозрачную идею. Всякая функция имеет область определения и область значения. Следовательно, не все аргументы представляют интерес, а только те из них, которые принадлежат выделенной области определения. Это и означает, что аргументы как объекты типизированы.

Чистые системы типов

Считается, что чистые системы типов представляют собой семейства типовых λ -исчислений, каждый член которого характеризуется тройкой $\langle S, A, R \rangle$, в которой:

S — подмножество констант системы, которые считаются *сортами*;

A — множество *аксиом* вида

$$c : s,$$

где c — константа, а s — сорт;

R — множество троек сортов, каждая из которых определяет, какие именно функциональные пространства можно построить в системе и на каких сортах строится каждое из функциональных пространств.

Каждая из чистых систем типов является формальной системой, предложения которой строятся как выводимые в ней утверждения вида

контекст \vdash приписывание типа.

Посредством этих предложений терму λ -исчисления в заданном контексте приписывается тип.

Построение класса типов

Класс типов формируется следующим образом. Начинают с того, что предъявляют конечный или бесконечный набор *базисных* типов:

$$\delta_1, \delta_2, \delta_3, \dots, \delta_n, \dots,$$

каждый из которых имеет интуитивную интерпретация в виде ассоциированного множества. Далее строится индуктивный класс типов:

- i) всякий базисный тип считается *типом*;
- ii) если a и b — типы, то $(a \rightarrow b)$ считается *типом*:

$$\frac{a \text{ — тип, } b \text{ — тип}}{(a \rightarrow b) \text{ — тип}}.$$

Интуитивной интерпретацией ' $(a \rightarrow b)$ ' является 'множество всех отображений из a в b '. По соглашению записи ' $(a \rightarrow b)$ ', ' $(a \ b)$ ', ' $(a, \ b)$ ' признаются имеющими одинаковый смысл. Опущенные в записи типа скобки восстанавливаются *по ассоциации вправо*. Это соглашение иллюстрируется приводимым далее примером.

Пример 6.1.

$$\begin{aligned} (a \ (b \ c) \ d) &\equiv (a \ ((b \ c) \ d)) \equiv (a \rightarrow ((b \rightarrow c) \rightarrow d)), \\ (a \ b \ c \ d) &\equiv (a \ b \ (c \ d)) \equiv (a(b(c \ d))), \end{aligned}$$

то есть ' $(a \ (b \ c) \ d)$ ' означает ' $(a \rightarrow ((b \rightarrow c) \rightarrow d))$ ', а ' $(a \ b \ c \ d)$ ' означает ' $(a \rightarrow (b \rightarrow (c \rightarrow d)))$ ', и это разные типы.

В аппликативных вычислительных системах в центре внимания *не* области определения функций, а сами функции как общие *законы соответствия*. Фактически, исчисляются именно законы соответствия, или, говоря иными словами, *концепты* функций, то есть в конечном счете объекты. В этом случае для учета типов применяются более тонкие рассуждения. Действительно, комбинаторы задают функции, функции от функций, функции от функций от функций, ... , то есть возникают функции высших порядков, или функционалы. Вопрос выяснения типа у объекта становится нетривиальным, и типы приходится исчислять, пользуясь точными правилами. Соответствующие области определения становятся в сильной степени взаимозависимы. В соответствующих логических конструкциях можно прийти к противоречиям.

Перейдем на более точную основу в рассуждениях. Прежде всего отметим, что в литературе для указания типа

$$'a \rightarrow b'$$

применяются различные обозначения:

$$ab, \quad b^a, \quad Fab$$

и некоторые другие.

6.1.1 Комбинаторные термы

Для комбинаторного терма X , то есть для такого терма, который получен в соответствии с постулатами комбинаторной логики, будем говорить, что

‘тип a приписан комбинатору X ’,

или

$$\vdash \#(X) = a$$

тогда и только тогда, когда данное утверждение вытекает из следующих аксиом и правила:

Схемы аксиом:

$$\begin{aligned} \vdash \#(I) &= (a, a), & (FI) \\ \vdash \#(K) &= (a, (b, a)) = (a, b, a), & (FK) \\ \vdash \#(S) &= ((a, (b, c)), ((a, b), (a, c))). & (FS) \end{aligned}$$

Правило:

$$\frac{\vdash \#(X) = (a, b), \quad \vdash \#(U) = a}{\vdash \#(XU) = b}. \quad (F)$$

Отметим, что согласно правилу (F) , выполняется приписывание схемы типа *аппликации* (XU) объекта X к объекту U , когда схемы типов этих последних объектов известны. Объект X рассматривается как функция из a в b , а U — как аргумент этой функции, берущийся из области определения a . Тогда значения результата (XU) приложения X к U берутся из области значений b .

6.1.2 λ -термы

Отметим, что при использовании λ -термов типизацию объектов удобно выполнять, пользуясь следующими двумя правилами:

Правила:

$$\frac{\vdash \#(x) = a, \quad \vdash \#(X) = b}{\vdash \#\lambda x.X = (a, b)}, \quad (\lambda)$$

$$\frac{\vdash \#(X) = (a, b), \quad \vdash \#(U) = a}{\vdash \#(XU) = b}, \quad (F)$$

где X, U — λ -термы, а x — переменная.

Отметим, что согласно правилу (F) , выполняется приписывание схемы типа *аппликации* (XU) объекта X к объекту U , когда схемы типов этих последних объектов известны. Объект X рассматривается как функция из a в b , а U — как аргумент этой функции, берущийся из области определения a . Тогда значения результата (XU) приложения X к U берутся из области значений b .

Согласно правилу (λ) по известной схеме типа a , приписанной переменной x , и известной схеме типа b , приписанной терму X , вводится схема типа (a, b) , которая приписывается терму $\lambda x.X$, а этот терм интерпретируется как функция из a в b .

6.2 Задачи

Предлагается, пользуясь аксиомами и правилом (F) , приписать типы основным комбинаторам, которые представлены в таблице 6.1. В ходе решения этих задач предполагается уяснить, что такое математические функции, как выполнять их композицию и как строить простейшие программы с помощью метода композиции. Каждый комбинатор дает идеализацию программы в виде “черного ящика”. Это значит, что внутренняя структура программы не уточняется, а важно установить ее поведение, ориентируясь только на вход и выход. Комбинации (композиции), составленные

Таблица 6.1: Список основных комбинаторов.

- (1) $\#(B)$, где $B = S(KS)K$,
- (2) $\#(SB)$,
- (3) $\#(Z_0)$, где $Z_0 = KI$,
- (4) $\#(Z_1)$, где $Z_1 = SB(KI)$,
- (5) $\#(Z_n)$, где $Z_n = (SB)^n(KI)$,
- (6) $\#(W)$, где $W = CSI$,
- (7) $\#(B^2)$, где $B^2 = BBB$,
- (8) $\#(B^3)$, где $B^3 = BBB^2$,
- (9) $\#(C^{[2]})$, где $C^{[2]} = BC(BC)$,
- (10) $\#(C^{[3]})$, где $C^{[3]} = BC(BC^{[2]})$,
- (11) $\#(C_{[2]})$, где $C_{[2]} = B^2CC$,
- (12) $\#(C_{[3]})$, где $C_{[3]} = B^2C_{[2]}C$,
- (13) $\#(\Phi)$, где $\Phi = B^2SB$,
- (14) $\#(Y)$, где $Y = WS(BWB)$,
- (15) $\#(D)$, где $D = C_{[2]}I$,
- (16) $\#(C)$, где $C = S(BBS)(KK)$.

из комбинаторов, дают возможность рассматривать произвольные программы как аппликативные формы. Аппликативная форма обладает простой структурой: ее компоненты имеют левую и правую часть, поэтому представлением формы служит бинарное дерево. Заметим, что отдельные ветви этого дерева можно вычислять независимо от других, что указывает на потенциальный параллелизм вычислений.

Задача 6.1. Определить тип объекта B .

Указание. Построение $S(KS)K$ представить в виде дерева, в котором:

$$Exp\ 1 = (a_1, (b_1, a_1))$$

$$Exp\ 2 = a_1$$

$$Exp\ 3 = ((b_1, a_1), ((a_2, b_2), (a_2, c_2)))$$

$$Exp\ 4 = (b_1, a_1)$$

$$Exp\ 5 = ((a_2, b_2), (a_2, c_2))$$

$$Exp\ 6 = (a_2, b_2)$$

$$Exp\ 7 = (a_2, c_2)$$

$$\frac{\frac{\frac{\vdash \#(K) = Exp\ 1 \quad \vdash \#(S) = Exp\ 2}{\vdash \#(S) = Exp\ 3} \quad \vdash \#(KS) = Exp\ 4}{\vdash \#(S(KS)) = Exp\ 5} \quad \vdash \#(K) = Exp\ 6}{\vdash \#(S(KS)K) = Exp\ 7} \quad (F)$$

Решение.

$\#(B)-1$. Пусть a, b, c — заданные типы. Поскольку по схеме (FK) имеем $\vdash \#(K) = (a_1, (b_1, a_1))$ и по схеме (FS) : $\vdash \#(S) = a_1$, то по правилу (F) : $\vdash \#(KS) = (b_1, a_1)$, где $a_1 = ((a, (b, c)), ((a, b), (a, c)))$.

$\#(B)-2$. Далее по схеме (FS) :

$$\vdash \#(S) = ((b_1, a_1), ((a_2, b_2), (a_2, c_2))),$$

причем $b_1 = a_2$, $a_1 = (b_2, c_2)$, то есть $b_2 = (a, (b, c))$, $c_2 = ((a, b), (a, c))$.

$\#(B)$ —3. В силу $\vdash \#(KS) = (b_1, a_1)$ и правила (F) :

$$\vdash \#(S(KS)) = ((a_2, b_2), (a_2, c_2)).$$

По схеме (FK) : $\vdash \#(K) = (a_3, (b_3, a_3))$, где $a_3 = a_2$, $(b_3, a_3) = b_2$, то есть $b_3 = a$, $a_3 = (b, c)$.

$\#(B)$ —4. Итак, $a_2 = a_3 = (b, c)$. В силу $\vdash \#(S(KS)) = ((a_2, b_2), (a_2, c_2))$ и правила (F) :

$$\vdash \#(S(KS)K) = (a_2, c_2) = ((b, c), ((a, b), (a, c))).$$

Ответ. B имеет тип: $\#(B) = ((b, c), ((a, b), (a, c)))$.

Аналогично определим типы, которые приписываются остальным комбинаторам.

Задача 6.2. Тип $\#(SB)$.

Решение.

$\#(SB)$ —1. Построение дерева:

$$Exp1 = ((a_1, (b_1, c_1)), ((a_1, b_1), (a_1, c_1)))$$

$$Exp2 = (a_1, (b_1, c_1))$$

$$Exp3 = ((a_1, b_1), (a_1, c_1))$$

$$\frac{\vdash \#(S) = Exp1 \quad \vdash \#(B) = Exp2}{\vdash \#(SB) = Exp3} (F)$$

$\#(SB)$ —2. Схема типа B : $\vdash \#(B) = ((b, c), ((a, b), (a, c)))$, но имеем $\vdash \#(B) = (a_1, (b_1, c_1))$, то есть $a_1 = (b, c)$, $b_1 = (a, b)$, $c_1 = (a, c)$.

Итак, $\vdash \#(SB) = (((b, c), (a, b)), ((b, c), (a, c)))$.

Ответ. (SB) имеет тип $(((b, c), (a, b)), ((b, c), (a, c)))$.

Задача 6.3. Тип $\#(Z_0)$.

Решение.

$\#(Z_0)$ -1. $Z_0 = KI$.

$\#(Z_0)$ -2.

$$\frac{\vdash \#(K) = (a_1, (b_1, a_1)) \quad \vdash \#(I) = a_1}{\vdash \#(KI) = (b_1, a_1)} (F)$$

$\#(Z_0)$ -3. По схеме $(FI) : \vdash \#(I) = a_1$, где $a_1 = (a, a)$; тип b_1 отличен от a_1 , то есть $b_1 = b$ (здесь: a, b — заданные типы).

Итак, $\vdash \#(KI) = (b, (a, a))$.

Ответ. $Z_0 = KI$ имеет тип $(b, (a, a))$.

Задача 6.4. Тип $\#(Z_1)$.

Решение.

$\#(Z_1)$ -1. $Z_1 = SB(KI)$.

$\#(Z_1)$ -2. $(F(KI)) : \vdash \#(KI) = (b, (a, a))$;
 $(F(SB)) : \vdash \#(SB) = (((b, c), (a, b)), ((b, c), (a, c)))$.

$\#(Z_1)$ -3. $Exp1 = (((b_1, c_1), (a_1, b_1)), ((b_1, c_1), (a_1, c_1)))$
 $Exp2 = ((b_1, c_1), (a_1, b_1))$
 $Exp3 = ((b_1, c_1), (a_1, c_1))$

$$\frac{\vdash \#(SB) = Exp1 \quad \vdash \#(KI) = Exp2}{\vdash \#(SB(KI)) = Exp3} (F)$$

$\#(Z_1)$ -4. По схеме $(F(KI)) : \vdash \#(KI) = ((b_1, c_1), (a_1, b_1))$, где $(b_1, c_1) = b, a_1 = a, b_1 = a$. Тип c_1 отличен от a и b , $c_1 = c$.

Итак, имеем $\vdash \#(Z_1) = ((a, c), (a, c))$. Отметим, что утверждение $\vdash \#(Z_1) = ((a, b), (a, b))$ также справедливо (вся разница лишь в обозначениях).

Ответ. $Z_1 = SB(KI)$ имеет тип $((a, b), (a, b))$.

Задача 6.5. Тип $\#(Z_n)$.

Решение. Определим сначала тип $\#(Z_2)$.

$$\#(Z_2)-1. \quad Z_2 = SB(SB(KI)).$$

$$\#(Z_2)-2. \quad (FZ) : \vdash \#(Z_1) = ((a, b), (a, b)).$$

$$\begin{aligned} \#(Z_2)-3. \quad Exp1 &= (((b_1, c_1), (a_1, b_1)), ((b_1, c_1), (a_1, c_1))) \\ Exp2 &= ((b_1, c_1), (a_1, b_1)) \\ Exp3 &= ((b_1, c_1), (a_1, c_1)) \end{aligned}$$

$$\frac{\vdash \#(SB) = Exp1 \quad \vdash \#(Z_1) = Exp2}{\vdash \#(Z_2) = Exp3} (F)$$

$\#(Z_2)-4.$ По схеме $(FZ) : \vdash \#(Z_1) = ((b_1, c_1), (a_1, b_1))$, где пара равенств должна выполняться одновременно: $(b_1, c_1) = (a, b)$, $(a_1, b_1) = (a, b)$, то есть:

$$\left. \begin{array}{ll} b_1 = a, & c_1 = b, \\ a_1 = a, & b_1 = b. \end{array} \right\} \quad (*)$$

Эта система равенств $(*)$ справедлива только в том случае, если $a_1 = b_1 = c_1 = a = b$. Таким образом, $\vdash \#(Z_2) = ((a, a), (a, a))$.

Теперь определим тип $\#(Z_n)$.

$$\#(Z_n)-1. \quad Z_n = (SB)^n(KI) = SB((SB)^{n-1}(KI)),$$

где $n > 2$.

$$\#(Z_n)-2. \quad (FZ_2) : \vdash \#(Z_2) = ((a, a), (a, a)).$$

$$\begin{aligned}
\#(Z_n)-3. \quad &Exp1 = (((b_1, c_1), (a_1, b_1)), ((b_1, c_1), (a_1, c_1))) \\
&Exp2 = ((b_1, c_1), (a_1, b_1)) \\
&Exp3 = ((b_1, c_1), (a_1, c_1))
\end{aligned}$$

$$\frac{\vdash \#(SB) = Exp1 \quad \vdash \#(Z_2) = Exp2}{\vdash \#(Z_3) = Exp3} (F)$$

$\#(Z_n)-4.$ По схеме $(F) : \vdash \#(Z_2) = ((b_1, c_1), (a_1, b_1))$, где $b_1 = a, c_1 = a, a_1 = a$, то есть $\vdash \#(Z_3) = ((a, a), (a, a))$.
Получаем равенство: $\#(Z_2) = \#(Z_1)$.

По индукции, придавая приращение n , можно получить, что $\vdash \#(Z_n) = ((a, a), (a, a))$, где $n > 1$.

Ответ. Объектам $Z_n = (SB)^n(KI)$, где $n > 1$ приписан один и тот же тип: $((a, a), (a, a))$.

Задача 6.6. Тип $\#(W)$.

Решение.

$$\#(W)-1. \quad W = CSI.$$

$\#(W)-2.$ $(FC) : \vdash \#(C) = ((b, (a, c)), (a, (b, c)))$. Это утверждение будет доказано далее (см. задачу 6.16).

$$\begin{aligned}
\#(W)-3. \quad &Exp1 = ((b_1, (a_1, c_1)), (a_1, (b_1, c_1))) \\
&Exp2 = (b_1, (a_1, c_1)) \\
&Exp3 = (a_1, (b_1, c_1)) \\
&Exp4 = (a_1) \\
&Exp5 = (b_1, c_1)
\end{aligned}$$

$$\frac{\vdash \#(C) = Exp1 \quad \vdash \#(S) = Exp2}{\vdash \#(CS) = Exp3 \quad \vdash \#(I) = Exp4} (F)$$

$$\frac{\vdash \#(CS) = Exp3 \quad \vdash \#(I) = Exp4}{\vdash \#(SCI) = Exp5} (F)$$

$\#(W)$ —4. По схеме $(FS) : \vdash \#(S) = (b_1, (a_1, c_1))$, однако $\vdash \#(S) = ((a, (b, c)), ((a, b), (a, c)))$. Следовательно, $b_1 = (a, (b, c))$, $a_1 = (a, b)$, $c_1 = (a, c)$. Аналогично, имеем $(FI) : \vdash \#(I) = a_1$. Но $\vdash \#(I) = (a, a)$, то есть $a_1 = (a, a)$ и $a = b$. Имеет место следующие равенства: $a = b$, $a_1 = (a, a)$, $b_1 = (a, (a, c))$, $c_1 = (a, c)$.

Итак, $\vdash \#(W) = ((a, (a, c)), (a, c))$, что эквивалентно записи: $\vdash \#(W) = ((a, (a, b)), (a, b))$.

Ответ. $W = CSI$ имеет тип $((a, (a, b)), (a, b))$.

Задача 6.7. Тип $\#(B^2)$.

Решение.

$\#(B^2)$ —1. $B^2 = BBB$.

$\#(B^2)$ —2. $(B) : ((bc)((ab)(ac)))$.

Здесь и далее предполагается, что запись вида:

$$\vdash \#(X) = (a, (b, c))'$$

аналогична записи:

$$(X) : (a(b\ c))'.$$

Договоримся в дальнейшем запятые опускать, то есть

$$(X) : (a, (b, c))'$$

эквивалентно

$$(X) : (a(b\ c))'.$$

$\#(B^2)$ —3. Найдем сначала $\#(BB)$:

$$\frac{(B) : ((b_1 c_1)((a_1 b_1)(a_1 c_1))) (B) : (b_1 c_1)}{(BB) : ((a_1 b_1)(a_1 c_1))} (F)$$

где $b_1 = (b_2 c_2)$, $c_1 = ((a_2 b_2)(a_2 c_2))$,

$(BB) : ((a_1(b_2 c_2))(a_1((a_2 b_2)(a_2 c_2))))$.

Положим: $a_1 = a$, $b_2 = b$, $c_2 = c$, $a_2 = d$. Тогда $(BB) :$
 $((a(b c))(a((d b)(d c))))$.

Теперь найдем $\#(BBB)$.

$$\frac{(BB) : ((a_1(b_1 c_1))(a_1((d_1 b_1)(d_1 c_1)))) (B) : (a_1(b_1 c_1))}{(BBB) : (a_1((d_1 b_1)(d_1 c_1)))}, (F)$$

где $(a_1(b_1 c_1)) = ((b c)((a b)(a c)))$, то есть: $a_1 = (b c)$, $b_1 = (a b)$, $c_1 = (a c)$. Пусть $d_1 = d$.

Таким образом, $(BBB) : ((b c)((d(a b))(d(a c))))$.

Ответ. $B^2 = BBV$ имеет тип $((b c)((d(a b))(d(a c))))$.

Задача 6.8. Тип $\#(B^3)$.

Решение.

$$\#(B^3)$$
—1. $B^3 = BBB^2$.

Поскольку

$$\frac{(BB) : ((a_1(b_1 c_1))(a_1((d_1 b_1)(d_1 c_1)))) (B^2) : (a_1(b_1 c_1))}{(BBB^2) : (a_1((d_1 b_1)(d_1 c_1)))}, (F)$$

где $(a_1(b_1 c_1)) = ((b c)((d(a b))(d(a c))))$, то $a_1 = (b c)$, $b_1 = (d(a b))$, $c_1 = (d(a c))$.

Пусть $d_1 = e$. Тогда $(BBB^2) : ((b c)((e(d(a b)))(e(d(a c)))))$.

Ответ. $B^3 = BBB^2$ имеет тип $((b c)((e(d(a b)))(e(d(a c)))))$.

Задача 6.9. Тип $\#(C^{[2]})$.

Решение.

$$\#(C^{[2]})-1. \quad C^{[2]} = BC(BC).$$

$$\#(C^{[2]})-2. \quad \text{Найдем } \#(BC).$$

$$\frac{(B) : ((b_1 \ c_1)((a_1 \ b_1)(a_1 \ c_1))) \quad (C) : (b_1 \ c_1)}{(BC) : ((a_1 \ b_1)(a_1 \ c_1))}, \quad (F)$$

где $(b_1 \ c_1) = ((a(b \ c))(b(a \ c)))$, то есть $b_1 = (b(c \ d))$, $c_1 = (c(b \ d))$. Пусть $a_1 = a$. Итак, $(BC) : ((a(b(c \ d)))(a(c(b \ d))))$.

$$\#(C^{[2]})-3. \quad \text{Exp 1} = ((a_1(b_1(c_1 \ d_1)))(a_1(c_1(b_1 \ d_1))))$$

$$\text{Exp 2} = (a_1(b_1(c_1 \ d_1)))$$

$$\text{Exp 3} = (a_1(c_1(b_1 \ d_1)))$$

$$\frac{(BC) : \text{Exp 1} \quad (BC) : \text{Exp 2}}{(BC(BC)) : \text{Exp 3}}, \quad (F)$$

где $(a_1(b_1(c_1 \ d_1))) = ((a(b(c \ d)))(a(c(b \ d))))$, то есть $a_1 = (a(b(c \ d)))$, $b_1 = a$, $c_1 = c$, $d_1 = (b \ d)$.

Итак, $(BC(BC)) : ((a(b(c \ d)))(c(a(b \ d))))$.

Ответ. $C^{[2]} = BC(BC)$ имеет тип $((a(b(c \ d)))(c(a(b \ d))))$.

Задача 6.10. Тип $\#(C^{[3]})$.

Решение.

$$\#(C^{[3]})-1. \quad C^{[3]} = BC(BC^{[2]}).$$

$$\#(C^{[3]})-2. \quad \text{Exp 1} = ((b_1 \ c_1)((a_1 \ b_1)(a_1 \ c_1)))$$

$$\text{Exp 2} = (b_1 \ c_1)$$

$$\text{Exp 3} = ((a_1 \ b_1)(a_1 \ c_1))$$

$$\text{Exp 4} = ((b_2 \ c_2)((a_2 \ b_2)(a_2 \ c_2)))$$

$$\text{Exp 5} = (b_2 \ c_2)$$

$$\text{Exp 6} = ((a_2 \ b_2)(a_2 \ c_2))$$

Поскольку

$$\frac{\frac{(B) : \text{Exp1} \quad (C) : \text{Exp2}}{(BC) : \text{Exp3}} (F) \quad \frac{(B) : \text{Exp4} \quad (C^{[2]}) : \text{Exp5}}{(BC^{[2]}) : \text{Exp6}} (F)}{(BC(BC^{[2]})) : (a_1 c_1)} (F),$$

где $(b_1 c_1) = ((a_3(b_3 c_3))(b_3(a_3 c_3)))$,
 $(b_2 c_2) = ((a(b(c d)))(c(a(b d))))$, $(a_1 b_1) = ((a_2 b_2)(a_2 c_2))$, то
 $a_3 = a_2 = e$, $c_2 = (b_3 c_3)$, $b_3 = c$, $c_3 = (a(b c))$.

Итак, $a_1 = (a_2 b_2) = (e(a(b(c d))))$, $c_1 = (b_3(a_3 c_3)) = (c(e(a(b d))))$, то есть $(BC(BC^{[2]})) : (a_1 c_1)$.

Ответ. $\#(C^{[3]}) = \#(BC(BC^{[2]})) = ((e(a(b(c d))))(c(e(a(b d))))).$

Задача 6.11. Тип $\#(C_{[2]})$.

Решение.

$$\#(C_{[2]})-1. \quad C_{[2]} = B^2CC.$$

$$\begin{aligned} \#(C_{[2]})-2. \quad & \text{Exp1} = ((b_1 c_1)((d_1(a_1 b_1))(d_1(a_1 c_1)))) \\ & \text{Exp2} = (b_1 c_1) \\ & \text{Exp3} = ((d_1(a_1 b_1))(d_1(a_1 c_1))) \\ & \text{Exp4} = (d_1(a_1 b_1)) \\ & \text{Exp5} = (d_1(a_1 c_1)) \end{aligned}$$

Поскольку

$$\frac{\frac{(B^2) : \text{Exp1} \quad (C) : \text{Exp2}}{(B^2C) : \text{Exp3} \quad (C) : \text{Exp4}} (F)}{(B^2CC) : \text{Exp5}} (F),$$

где

$$\begin{aligned} (d_1(a_1 b_1)) &= ((a_2(b_2 c_2))(b_2(a_2 c_2))), \\ (b_1 c_1) &= ((a(b c))(b(a c))), \end{aligned}$$

то

$$\begin{aligned}b_1 &= (a(bc)), \\c_1 &= (b(ac)), \\a_1 &= b_2, \\d_1 &= (a_2(b_2 c_2)), \\b_1 &= (a_2 c_2).\end{aligned}$$

Имеем: $d_1 = (a(b_2(bc)))$, $a_1 = b_2$, $c_1 = (b(ac))$. Пусть $b_2 = d$. Тогда $(B^2CC) : (d_1(a_1 c_1)) = ((a(d(bc)))(d(b(ac))))$.

Ответ. $C_{[2]} = B^2CC$ имеет тип $((a(d(bc)))(d(b(ac))))$.

Задача 6.12. Тип $\#(C_{[3]})$.

Решение.

$$\#(C_{[3]})-1. \quad C_{[3]} = B^2C_{[2]}C.$$

$$\begin{aligned}\#(C_{[3]})-2. \quad \text{Exp 1} &= ((b_1 c_1)((d_1(a_1 b_1))(d_1(a_1 c_1)))) \\ \text{Exp 2} &= (b_1 c_1) \\ \text{Exp 3} &= ((d_1(a_1 b_1))(d_1(a_1 c_1))) \\ \text{Exp 4} &= (d_1(a_1 b_1)) \\ \text{Exp 5} &= (d_1(a_1 c_1))\end{aligned}$$

$$\frac{(B^2) : \text{Exp 1} \quad (C_{[2]}) : \text{Exp 2}}{\frac{(B^2C_{[2]}) : \text{Exp 3} \quad (C) : \text{Exp 4}}{(B^2C_{[2]}C) : \text{Exp 5}}}, \quad (F)$$

$$\text{где } (b_1 c_1) = ((a(b(cd)))(b(c(ad)))), \quad (d_1(a_1 b_1)) = ((a_2(b_2 c_2))(b_2(a_2 c_2))).$$

Имеем $d_1 = (a_2(b_2 c_2)) = (a(b_2(b(cd))))$, $a_1 = b_2$, $c_1 = (b(c(ad)))$. Пусть $b_2 = e$. Подставим e вместо b_2 , а вместо d_1 , a_1 , c_1 подставим соответствующие выражения, получая тип:

$$(B^2C_{[2]}C) : (d_1(a_1 c_1)).$$

Ответ. $C_{[3]} = B^2C_{[2]}C$ имеет тип $((a(e(b(cd))))(e(b(c(ad))))))$.

Задача 6.13. Тип $\#(\Phi)$.

Решение.

$$\# \Phi - 1. \quad \Phi = B^2 S B.$$

$$\# \Phi - 2. \quad \text{Exp 1} = ((b_1 c_1)((d_1(a_1 b_1))(d_1(a_1 c_1))))$$

$$\text{Exp 2} = (b_1 c_1)$$

$$\text{Exp 3} = ((d_1(a_1 b_1))(d_1(a_1 c_1)))$$

$$\text{Exp 4} = (d_1(a_1 b_1))$$

$$\text{Exp 5} = (d_1(a_1 c_1))$$

Поскольку

$$\frac{(B^2) : \text{Exp 1} \quad (S) : \text{Exp 2}}{\frac{(B^2 S) : \text{Exp 3} \quad (B) : \text{Exp 4}}{(B^2 S B) : \text{Exp 5}}}, (F)$$

где

$$(b_1 c_1) = ((a(b c))((a b)(a c))),$$

$$(d_1(a_1 b_1)) = ((b_2 c_2)((a_2 b_2)(a_2 c_2))),$$

то $d_1 = (b_2 c_2) = (b_2(b c))$, $a_1 = (a_2 b_2) = (a b_2)$, $c_1 = ((a b)(a c))$.

Пусть $b_2 = d$; тогда

$$(B^2 S B) : (d_1(a_1 c_1)) = ((d(b c))((a d)((a b)(a c)))).$$

Ответ. $\Phi = B^2 S B$ имеет тип $((d(b c))((a d)((a b)(a c))))$.

Задача 6.14. Тип $\#(Y)$.

Решение.

$$\# Y - 1. \quad \text{Воспользуемся равенством } Y = W S(B W B).$$

$\# Y - 2.$ Запишем возникающие ограничения на схемы типов:

$$\frac{(W) : ((a_1(a_1 b_1))(a_1 b_1)) \quad (S) : (a_1(a_1 b_1))}{(W S) : (a_1 b_1)}, (F)$$

$$\frac{(B) : ((b_2 c_2)((a_2 b_2)(a_2 c_2))) \quad (W) : (b_2 c_2)}{(BW) : ((a_2 b_2)(a_2 c_2))}, (F)$$

$$\frac{(BW) : ((a_2 b_2)(a_2 c_2)) \quad (B) : (a_2 b_2)}{(BWB) : (a_2 c_2)}, (F)$$

$$\frac{(WS) : (a_1 b_1) \quad (BWB) : (a_2 c_2)}{(WS(BWB)) : b_1}, (F)$$

где

$$\begin{aligned} (a_1(a_1 b_1)) &= ((a(b c))((a b)(a c))), \\ (b_2 c_2) &= ((a_3(a_3 b_3))(a_3 b_3)), \\ (a_2 b_2) &= ((b_4 c_4)((a_4 b_4)(a_4 c_4))), \\ a_1 &= (a_2 c_2). \end{aligned}$$

Из равенства $(a_1(a_1 b_1)) = ((a(b c))((a b)(a c)))$ имеем: $b_1 = (a c)$, то есть тип $\#(Y)$ имеет вид $(a c)$.

$\#Y-3$. Проведем следующие рассуждения. Известно, что $Yx = x(Yx)$, то есть $\#(Yx) = \#(x(Yx))$.

Пусть $\#(x) = a$, $\#(Yx) = b$, тогда в соответствии с правилом $(F) : \#(Y) = (a, b)$, поскольку

$$\frac{(Y) : (a, b) \quad (x) : a}{(Yx) : b} (F)$$

Теперь, учитывая, что $\#(x(Yx)) = \#(Yx) = b$, получим $\#(x)$:

$$\frac{(x) : (b, b) \quad (Yx) : b}{(x(Yx)) : b} (F)$$

Следовательно, $a = (b, b)$, $(Y) : (a, b) = ((b, b), b)$.

Ответ. $Y = WS(BWB)$ имеет тип $((b, b), b)$.

Задача 6.15. Тип $\#(D)$.

Решение.

$$\#D-1. \quad D = C_{[2]}I.$$

$\#D-2.$

$$\frac{(C_{[2]}) : ((a(d(b\ c)))(d(b(a\ c)))) \quad (I) : (a(d(b\ c)))}{(C_{[2]}I) : (d(b(a\ c)))}, \quad (F)$$

где $(I) : (a_1, a_1)$, то есть $a = (d(b\ c))$.

Итак, $\#(C_{[2]}I) = (d(b((d(b\ c))c)))$.

Ответ. $D = C_{[2]}I$ имеет тип: $(a, (b, ((a, (b, c)), c)))$.

Задача 6.16. Тип $\#(C)$.

Решение.

$\#C-1.$ $C = S(BBS)(KK)$.

$\#C-2.$ $(S) : (a\ b\ c)((a\ b)(a\ c)), (B) : (b\ c)((a\ b)(a\ c)), (K) : (a(b\ a))$.

$\#C-3.$

$$\frac{(B) : (b_2\ c_2)((a_2\ b_2)(a_2\ c_2)) \quad (B) : (b_2\ c_2)}{(\mathbf{BB}) : (a_2\ b_2)(a_2\ c_2)}$$

$$\frac{(\mathbf{BB}) : (a_2\ b_2)(a_2\ c_2) \quad (S) : (a_2\ b_2)}{(\mathbf{BBS}) : (a_2\ c_2)}$$

$$\frac{(S) : (a_3(b_3\ c_3))((a_3\ b_3)(a_3\ c_3)) \quad (\mathbf{BBS}) : (a_2\ c_2)}{S(\mathbf{BBS}) : (a_3\ b_3)(a_3\ c_3)}$$

$$\frac{(K) : (a_4(b_4\ a_4)) \quad (K) : a_4}{(\mathbf{KK}) : (b_4\ a_4)}$$

$$\frac{S(\mathbf{BBS}) : (a_3\ b_3)(a_3\ c_3) \quad (\mathbf{KK}) : (b_4\ a_4)}{S(\mathbf{BBS})(\mathbf{KK}) : (a_3\ c_3)},$$

где:

$$\begin{aligned}(a_3 b_3) &= (b_4 a_4), \\ (b_2 c_2) &= ((b_6 c_6), ((a_6 b_6), (a_6 c_6))), \quad \text{берется схема для } B\end{aligned}$$

$$a_4 = (a_5(b_5 a_5)), \quad \text{берется схема для } K$$

$$\begin{aligned}(a_2 c_2) &= (a_3(b_3 c_3)), \\ (a_2 b_2) &= (a b c)(a b)(a c). \quad \text{берется схема для } S\end{aligned}$$

Имеем:

$$\left\{ \begin{array}{llll} \underline{a_3} &= a_2 &= \underline{(a b c)} &= b_4, \\ c_2 &= (b_3 c_3) &= (a_6 b_6)(a_6 c_6), \\ b_2 &= (a b)(a c) &= (b_6 c_6), \\ b_3 &= a_4 &= (a_5 b_5 a_5). \end{array} \right.$$

Далее,

$$\begin{aligned}c_6 &= (a c), \\ b_6 &= (a b), \\ b_3 &= (a_6 b_6) = (a_6 a b) = (b a b), \quad \text{поскольку} \\ & \quad b_3 = (a_5 b_5 a_5).\end{aligned}$$

Итак, $\underline{c_3} = (a_6 c_6) = (b c_6) = \underline{(b a c)}$. Теперь остается только записать тип (a_3, c_3) .

Ответ. $C = S(BBS)(KK)$ имеет тип: $((a, (b, c)), (b, (a, c)))$.

Глава 7

Базис I, K, S

При построении комбинаторной логики одна из изначальных задач состояла в предъявлении минимального числа наипростейших объектов, атомарных по своей сути, и способов комбинирования, которые дают достаточно богатые метаматематические средства. Эти наипростейшие объекты должны вести себя как *константные* функции, а с точки зрения компьютерных наук давать примитивную систему-ядро для программирования, которую можно развивать на основе “самораскрутки”. Это означает, что более сложные объекты строятся на основе более примитивных, а затем присоединяются к системе и в свою очередь могут быть использованы как элементы других конструкций.

Сосредоточим внимание на наипростейшей системе программирования, в которой всего только три инструкции: I, K, S . Синтезировать новый объект можно чисто механическим использованием алгоритма разложения в базисе, который вполне аналогичен процессу компиляции.

7.1 Теоретические сведения

Покажем, что объект, понимаемый как λ -терм (исходное представление), может быть представлен посредством комбинаторного терма (целевое представление). Процесс перевода объекта из исходного в целевое представление существенно упрощается, если использовать заранее заданный набор комбинаторов. Для этого зафиксируем набор I, K, S , который, как известно, образует базис.

7.2 Задачи

Задача 7.1. Выразить терм $\lambda x.P$ через комбинаторы I, K, S .

Формулировка задачи. Пусть определение терма $\lambda x.P$ дано индукцией по построению P :

- (1) $\lambda x.x = I$,
- (2) $\lambda x.P = KP$, если не принадлежит $FV(P)$,
- (3) $\lambda x.PQ = S(\lambda x.P)(\lambda x.Q)$.

Исключить все переменные из приводимых λ -выражений:

1. $\lambda xy.yx$; 2. $\lambda fx.xx$; 3. $f = \lambda x.B(f(Ax))$.

Решение.

$$\begin{aligned}
 P-1. \quad \lambda xy.yx &\stackrel{def}{=} \lambda x.(\lambda y.yx) \\
 &\stackrel{(3)}{=} \lambda x.(S(\lambda y.y)\lambda y.x)) \\
 &\stackrel{(1), (2)}{=} \lambda x.SI(Kx) \\
 &\stackrel{(3)}{=} S(\lambda x.SI)(\lambda x.Kx) \\
 &\stackrel{(2)}{=} S(K(SI))(S(\lambda x.K)(\lambda x.x)) \\
 &\stackrel{(1), (2)}{=} S(K(SI))(S(KK)I)
 \end{aligned}$$

$$\begin{aligned}
P-2. \quad \lambda f x. f x x & \stackrel{def}{=} \lambda f. (\lambda x. f x x) \\
& \stackrel{(3)}{=} \lambda f. (S(\lambda x. f x)(\lambda x. x)) \\
& \stackrel{(1), (3)}{=} \lambda f. S(S(\lambda x. f)(\lambda x. x))I \\
& \stackrel{(1), (2)}{=} \lambda f. S(S(K f)I)I \\
& \stackrel{(3)}{=} S(\lambda f. S(S(K f)I))(\lambda f. I) \\
& \stackrel{(2), (3)}{=} S(S(\lambda f. S)(\lambda f. S(K f)I))(KI) \\
& \stackrel{(2), (3)}{=} S(S(KS)(S(\lambda f. S(K f))(\lambda f. I)))(KI) \\
& \stackrel{(2), (3)}{=} S(S(KS)(S(S(\lambda f. S)(\lambda f. K f)))(KI)))(KI) \\
& \stackrel{(2), (3)}{=} S(S(KS)(S(S(KS)(S(\lambda f. K)(\lambda f. f)))(KI)))(KI) \\
& \stackrel{(1), (2)}{=} S(S(KS)(S(S(KS)(S(KK)I)(KI)))(KI))
\end{aligned}$$

$$\begin{aligned}
P-3. \quad f & \stackrel{def}{=} \lambda x. b(f(ax)) \\
& \stackrel{(3)}{=} S(\lambda x. b)(\lambda x. f(ax)) \\
& \stackrel{(2), (3)}{=} S(Kb)(S(\lambda x. f)(\lambda x. ax)) \\
& \stackrel{(2), (3)}{=} S(Kb)(S(K f)(S(\lambda x. a)(\lambda x. x))) \\
& \stackrel{(2), (1)}{=} S(Kb)(S(K f)(S(Ka)I))
\end{aligned}$$

Упражнения

Упражнение 7.1. Выразить комбинатор W возведения функции в квадрат в терминах комбинаторов I, K, S .

Упражнение 7.2. Для комбинатора Φ с характеристикой

$$\Phi f a b x = f(ax)(bx)$$

выполнить следующее:

1° записать предикат

$$1 < x < 5$$

без переменной.

Указание. Для этого введите в рассмотрение подходящую конструкцию Φ *and* (*greater 1*) (*less 5*);

2° выразить комбинатор

$$\Phi \text{ and } (\text{greater } 1) (\text{less } 5)$$

в терминах комбинаторов I, K, S .

Упражнение 7.3. Выразить комбинатор, представляющий функцию формирования суммы синусов двух чисел в терминах комбинаторов I, K, S .

Указание. Можно, например, воспользоваться комбинатором Φ и записать

$$\Phi \text{ plus } \sin \sin.$$

Попытайтесь преобразовать это выражение, воспользовавшись комбинаторами для устранения двухкратного вхождения '*sin*'. Например,

$$\Phi \text{ plus } \sin \sin = W(\Phi \text{ plus}) \sin = \dots$$

и т.д.

Глава 8

Базис I, B, C, S

Как оказывается, базис I, K, S не единственный, и свойство базисности проявляет также набор комбинаторов I, B, C, S . Компиляция (разложение) объекта в этом базисе также решает задачу синтеза объекта с заданными свойствами. Очевидно, можно использовать свободу выбора базиса в зависимости от некоторых критериев.

8.1 Теоретические сведения

Как можно было убедиться, представляя термы путем разложения в базисе I, K, S , существует способ систематического или даже механического перехода от одного представления объекта к другому. Можно использовать большое число различных комбинаторов, уменьшая число шагов в алгоритме разложения. Конечно, как и следовало ожидать, не все наборы комбинаторов состоят из взаимно независимых комбинаторов. В частности, комбинатор I выразим только в терминах K и S , поэтому он, строго говоря, не является необходимым. Тем не менее его использование имеет технические преимущества.

Существуют и другие наборы комбинаторов, пользуясь которыми можно получать представление всевозможных правильно построенных термов. Такие наборы комбинаторов считаются базисными, или, как принято говорить, образуют *базис*. Как и прежде, комбинатором считается объект, составленный применением аппликации из базисных комбинаторов. Следовательно, базис комбинаторов не единственный, и следует ожидать множественности представления комбинаторами одного и того же терма. В зависимости от поставленных целей можно выбрать то или иное представление¹.

8.2 Свойство базисности

Можно получить большое количество различных комбинаторов, однако оказывается, что эти комбинаторы не являются независимыми; часть их можно определить через набор комбинаторов, называемых *базисными*. Под комбинатором теперь можно понимать объект, составленный с помощью аппликаций из базисных комбинаторов.

Рассмотрим две базисные системы комбинаторов C, W, B, K и S, K :

$$\begin{array}{l|l} Cxyz = xyz, & Sxyz = xz(yz), \\ Wxy = xy, & Kxy = x. \\ Bxyz = x(yz), & \\ Kxy = x; & \end{array}$$

Для любого синтаксического объекта V , составленного из различных переменных x_1, \dots, x_n с помощью операции аппликации можно найти комбинатор X , составленный из базисных, такой, что:

$$X x_1, \dots, x_n = V.$$

¹ Например, кроме используемых в настоящей работе базисов I, K, S и I, B, C, S можно ввести в употребление и другие базисы. В частности, набор C, W, B, K также проявляет свойство базисности.

Например, если $V = xz(yz)$, то можно указать комбинатор X из базисной системы S, K , такой, что $Xxyz = xz(yz)$. Как нетрудно видеть, им оказывается комбинатор S , то есть $X = S$.

В общем случае поиск комбинатора X осуществляется посредством следующих действий:

- 1) с помощью B из V удаляются скобки;
- 2) с помощью C переменные переупорядочиваются;
- 3) с помощью W устраняются дублирования переменных;
- 4) с помощью K вводятся переменные, отсутствующие в V .

Пример 8.1. Построим в первой базисной системе комбинатор X со свойством

$$\begin{aligned}
 ac(bc) &= Xabc : \\
 \underbrace{(ac)}_x \underbrace{(b)}_y \underbrace{(c)}_z &= \underbrace{B}_x \underbrace{(a)}_y \underbrace{(c)}_z bc \\
 &\stackrel{B}{=} \underbrace{(B Ba)}_x \underbrace{(c)}_z \underbrace{(b)}_y c \\
 &\stackrel{C}{=} \underbrace{(C(B Ba)b)}_x \underbrace{(c)}_y \underbrace{(c)}_y \\
 &\stackrel{W}{=} \underbrace{W}_x \underbrace{(C(B Ba)b)}_y \underbrace{(c)}_z c \\
 &\stackrel{B}{=} \underbrace{(BW)}_x \underbrace{(C)}_y \underbrace{(B Ba)}_z bc \\
 &\stackrel{B}{=} \underbrace{(B(BW)C)}_x \underbrace{((BB))}_y \underbrace{(a)}_z bc \\
 &\stackrel{B}{=} B(B(BW)C)(BB)abc.
 \end{aligned}$$

В этом примере под объектами, к которым можно применить заранее известную *схему*, записаны переменные — в том же порядке, что и в соответствующей комбинаторной характеристике. Таким образом,

$$X = B(B(BW)C)(BB).$$

8.3 Элементарные примеры

Рассмотрим примеры разложения объекта в базисе. Исходные термы возьмем точно такими же, что и в случае разложения в базисе I, K, S . Полученные результаты можно будет сопоставить и сделать вывод о предпочтительности того или иного базиса².

Задача 8.1. Выразить терм $M = \lambda x.PQ$, пользуясь исключительно комбинаторами I, B, C, S .

Формулировка задачи. Пусть определение терма M такого, что переменная x входит в состав свободных переменных (PQ), то есть $x \in FV(PQ)$, дано индукцией по построению M (здесь ' \in ' означает 'принадлежит', а ' \notin ' — 'не принадлежит'):

$$(1) \quad \lambda x.x = I,$$

$$(2) \quad \lambda x.PQ = \begin{cases} (a) \quad BP(\lambda x.Q), & \text{если } x \notin FV(P) \text{ и } x \in FV(Q), \\ (b) \quad C(\lambda x.P)Q, & \text{если } x \in FV(P) \text{ и } x \notin FV(Q), \\ (c) \quad S(\lambda x.P)(\lambda x.Q), & \text{если } x \in FV(P) \text{ и } x \in FV(Q). \end{cases}$$

Исключить все переменные из приводимых ниже λ -выражений:

1. $\lambda xy.yx$; 2. $\lambda fx.fxx$.

Решение.

² На решение задачи разложения в базисе можно взглянуть иначе. Поскольку всякий комбинатор — это *понятие* и даже *концепт* в математическом понимании, то исходный терм считается 'исследуемым' или 'познаваемым' объектом, базис — 'системой известных понятий', а процедура разложения в базисе — 'представлением знания' об исходном объекте в терминах известных понятий. Такие рассуждения в своей основе используются в приложениях объектно-ориентированного подхода.

$$\begin{aligned}
M-1. \quad \lambda xy.yx &= \lambda x.(\lambda y.yx) \\
&\stackrel{(2)(b)}{=} \lambda x.(C(\lambda y.y)x) \\
&\stackrel{(1)}{=} \lambda x.CIx \\
&\stackrel{(2)(a)}{=} B(CI)(\lambda x.x) \\
&\stackrel{(1)}{=} B(CI)I.
\end{aligned}$$

Проверка. $B(CI)Ixy = CI(Ix)y = Iy(Ix) = Iyx = yx$.

$$\begin{aligned}
M-2. \quad \lambda fx.fxx &= \lambda f.(\lambda x.fxx) \\
&\stackrel{(2)(c)}{=} \lambda f.S(\lambda x.fx)(\lambda x.x) \\
&\stackrel{(1), (2)(a)}{=} \lambda f.S(Bf(\lambda x.x))I \\
&\stackrel{(1)}{=} \lambda f.S(BfI)) \\
&\stackrel{(2)(b)}{=} C(\lambda f.S(BfI))I \\
&\stackrel{(2)(a)}{=} C(BS(\lambda f.BfI))I \\
&\stackrel{(2)(b)}{=} C(BS(C(\lambda f.BfI))I)I \\
&\stackrel{(2)(a)}{=} C(BS(C(BB(\lambda f.f))I))I \\
&\stackrel{(1)}{=} C(BS(C(BBI)I))I.
\end{aligned}$$

Упражнения

Упражнение 8.1. Доказать, что набор комбинаторов C, W, B, K проявляет свойство *базисности*, то есть является базисом.

Указание. Воспользуйтесь определением базиса в общей форме (см. [2], с. 172):

Определение 8.1 (базис). (i) Пусть \mathcal{X} — некоторое подмножество всех λ -термов Λ , $\mathcal{X} \subset \Lambda$. Обозначим через \mathcal{X}^+ множество термов, порожденное множеством \mathcal{X} . Множество \mathcal{X}^+ — это наименьшее множество \mathcal{Y} , такое что:

$$1) \mathcal{X} \subseteq \mathcal{Y},$$

2) если термы $M, N \in \mathcal{Y}$, то их аппликация $(M N) \in \mathcal{Y}$.

(ii) Возьмем некоторое подмножество λ -термов $\mathcal{A} \subseteq \Lambda$. Множество термов $\mathcal{X} \subseteq \Lambda$ называется *базисом* для \mathcal{A} , если

$$(\forall M \in \mathcal{A})(\exists N \in \mathcal{X}^+). N = M.$$

(iii) Множество \mathcal{X} называется *базисом*, если \mathcal{X} — базис для множества всех замкнутых термов.

Упражнение 8.2. Выразить комбинатор W *mult* возведения функции в квадрат в терминах комбинаторов I, B, C, S .

Упражнение 8.3. Для комбинатора Φ с характеристикой

$$\Phi f a b x = f(a x)(b x)$$

выполнить следующее:

1° записать предикат $1 < x < 5$ без переменной.

Указание. Для этого введите в рассмотрение подходящую конструкцию Φ *and (greater 1) (less 5)*;

2° выразить комбинатор Φ *and (greater 1) (less 5)* в терминах комбинаторов I, B, C, S .

Упражнение 8.4. Выразить комбинатор, представляющий функцию формирования суммы синусов двух чисел в терминах комбинаторов I, B, C, S .

Указание. Можно, например, воспользоваться комбинатором Φ и записать

$$\Phi \text{ plus sin sin}.$$

Попытайтесь преобразовать это выражение, воспользовавшись комбинаторами для устранения двухкратного вхождения '*sin*'. Например,

$$\Phi \text{ plus sin sin} = W(\Phi \text{ plus}) \text{ sin} = \dots$$

и т.д.

Глава 9

Применения неподвижной точки Y

Рассматривается случай рекурсивных определений объектов. Пользуясь фундаментальной для функционального программирования теоремой о неподвижной точке, рекурсивные определения удастся привести к обычному эквациональному виду.

9.1 Теорема о неподвижной точке

Приводимые в настоящем разделе задачи носят характер небольшого самостоятельного исследования. Целью является установление связи конструкций языков программирования с понятиями исчисления λ -конверсии.

Характеристическое равенство функции Y имеет вид

$$Yf = f(Yf).$$

Теорема 9.1 (о неподвижной точке). Ссылающиеся на себя определение функции f :

$$f = E f$$

где f не имеет свободного вхождения в E , $f \notin E$, можно искать в виде:

$$f = Y E.$$

Доказательство. Действительно, если в определении:

$$\underbrace{f}_{=Y E} = E \underbrace{f}_{=Y E}$$

заменить f на $(Y E)$, то получаем $Y E = E(Y E)$. □

9.2 Элементы рекурсивных вычислений

Функция считается рекурсивной, если в определяющем ее выражении содержится хотя бы одно обращение к ней самой.

Рассмотрим рекурсивное определение функции вычисления факториала:

$$FAC = (\lambda n. IF(= n 0) 1 (\times n FAC(- n 1)))$$

При прямом применении этого определения происходят следующие преобразования:

$$\begin{aligned} FAC &= (\lambda n. IF(= n 0) 1 \\ &\quad (\times n FAC(- n 1))) = \\ FAC &= (\lambda n. IF(= n 0) 1 \\ &\quad (\times n (\lambda n. IF(= n 0) 1 \\ &\quad (\times n FAC(- n 1)))(- n 1))) = \\ FAC &= (\lambda n. IF(= n 0) 1 \\ &\quad (\times n (\lambda n. IF(= n 0) 1 \\ &\quad (\times n (\lambda n. IF(= n 0) 1 \\ &\quad (\times n FAC(- n 1)))(- n 1)))(- n 1))) = \\ &\dots \end{aligned}$$

Нетрудно заметить, что цепочка таких преобразований не завершается никогда. В предлагаемой записи λ -терму присвоено имя

FAC . Эта запись естественна и удобна, однако не согласуется с синтаксисом исчисления, так как в λ -исчислении именование функций не предусмотрено. В связи с этим данное выражение следует перевести на язык λ -исчисления, то есть выразить рекурсию в чистом виде (без самоссылки). Как оказывается, такой перевод возможен, причем без выхода за рамки комбинаторной логики.

9.3 Использование комбинатора Y

В качестве основного примера, выполнение которого иллюстрирует основные эффекты вычислений с неподвижной точкой воспользуемся функцией факториала FAC . Запишем функцию FAC сокращенно:

$$FAC = \lambda n.(\dots FAC \dots). \quad (9.1)$$

Применяя λ -абстракцию, получим:

$$\lambda fac.FAC \ fac = (\lambda fac.(\lambda n.(\dots fac \dots)))FAC. \quad (9.2)$$

По правилу (η) получаем:

$$\lambda fac.FAC \ fac = FAC,$$

и определение (9.2) запишется в виде:

$$FAC = (\lambda fac.(\lambda n.(\dots fac \dots)))FAC. \quad (9.3)$$

Полагая $H = \lambda fac.(\lambda n.(\dots fac \dots))$, получим:

$$FAC = H \ FAC. \quad (9.4)$$

Определение H представляет собой обычную λ -абстракцию, не использующую рекурсию. Рекурсия выражена только лишь в форме равенства (9.4). Определение (9.4) похоже на математическое

уравнение. Например, решить уравнение $(x^2 - 2) = x$ — это значит найти значения x , которому удовлетворяют $(x = -1, x = 2)$. Аналогично, для того, чтобы решить (9.4), необходимо найти λ -абстракцию для FAC , которая удовлетворяет (9.4). Уравнение $FAC = H FAC$ выражает тот факт, что когда функция H применяется к аргументу FAC , в результате снова получается FAC . Поэтому FAC называется неподвижной точкой функции H .

Пример 9.1. Числа 0 и 1 являются неподвижными точками функции $f = \lambda x. \times x x$, то есть $f 0 = 0$ и $f 1 = 1$. Действительно,

$$\begin{aligned} f 0 &= (\lambda x. \times x x) 0 = (\beta) \\ &= \times 0 0 = 0, \\ f 1 &= (\lambda x. \times x x) 1 = (\beta) \\ &= \times 1 1 = 1. \end{aligned}$$

Итак, нужно найти неподвижную точку функции H . Ясно, что эта точка зависит только от H . Введем функцию Y , которая работает по схеме:

получив на входе в качестве аргумента функцию, на выходе Y формирует неподвижную точку этой функции.

То есть, получаем $YH = H(YH)$, где Y — комбинатор неподвижной точки. Если найдем такой Y , то получим решение уравнения (9.4):

$$FAC = YH.$$

В ходе получения этого решения использован довольно общий метод. Он опирается на основную в теории рекурсивных вычислений *теорему о неподвижной точке* (см. теорему 9.1). Фактически, пришлось дать ее частное “доказательство” для функции FAC . Полученное решение дает *нерекурсивное* определение FAC . Существование теоремы о неподвижной точке как раз и состоит в том,

чтобы гарантировать переход от рекурсивных по форме записи определений к нерекурсивным по форме записи определениям. В последнем случае эффект цикличности определения (и соответствующего вычисления) оказывается завуалированным посредством комбинатора Y .

Пример 9.2. Для того, чтобы убедиться, что определенная таким образом функция FAC работает правильно, приложим ее к некоторому аргументу, например, к 1:

$$\begin{aligned}
 FAC\ 1 &= \underline{Y\ H}\ 1 = \underline{H(Y\ H)}1 && (FAC), (Y) \\
 &= (\lambda fac. \lambda n. IF(= n\ 0)1(\times n(fac(-\ n\ 1))))(Y\ H)1 && (H) \\
 &= (\lambda n. IF(= n\ 0)1(\times n((Y\ H)(-\ n\ 1))))1 && (\beta) \\
 &= IF(= 1\ 0)1(\times 1((Y\ H)(-\ 1\ 1))) && (\beta) \\
 &= \times 1(\underline{(Y\ H)}0) = \times 1(\underline{H(Y\ H)}0) && (Y) \\
 &= \times 1((\lambda fac. \lambda n. IF(= n\ 0)1(\times n(fac(-\ n\ 1))))(Y\ H)0) && (H) \\
 &= \times 1((\lambda n. IF(= n\ 0)1(\times n((Y\ H)(-\ n\ 1))))0) && (\beta) \\
 &= \times 1(IF(= 0\ 0)1(\times 0(Y\ H)(-\ 0\ 1))) && (\beta) \\
 &= \times 1\ 1 \\
 &= 1.
 \end{aligned}$$

9.4 Вычисление функций

Приведем примеры определения наиболее часто используемых в практике программирования функций. Отметим, что для простоты отобраны функции, действующие на списки как на свои аргументы. Как обычно, под *списком* понимается конечная последовательность.

Задача 9.1. Пользуясь функцией поиска неподвижной точки Y ,

выразить определения приводимых далее функций:

- 2) $sum = \lambda x. \text{if } null\ x$
 $\quad \text{then } 0$
 $\quad \text{else } (car\ x) + sum(cdr\ x),$
- 3) $product = \lambda x. \text{if } null\ x$
 $\quad \text{then } 1$
 $\quad \text{else } (car\ x) \times product(cdr\ x),$
- 4) $append = \lambda x. \lambda y. \text{if } null\ x$
 $\quad \text{then } y$
 $\quad \text{else } (list((car\ x)(append(cdr\ x)y)),$
- 5) $concat = \lambda x. \text{if } null\ x$
 $\quad \text{then } ()$
 $\quad \text{else } append(car\ x)(concat(cdr\ x)),$
- 6) $map = \lambda f \lambda x. \text{if } null\ x$
 $\quad \text{then } ()$
 $\quad \text{else } list((f(car\ x))(map\ f(cdr\ x))).$

$length(a_1, a_2, a_3) = 3;$
 $sum(1, 2, 3, 4) = 10;$
 $product(1, 2, 3, 4) = 24;$
 $append(1, 2)(3, 4, 5) = (1, 2, 3, 4, 5);$
 $concat((1, 2), (3, 4), ()) = (1, 2, 3, 4);$
 $map\ square\ (1, 2, 3, 4) = (1, 4, 9, 16).$

Для примеров “обращения” к каждой из функций выполнить проверку.

Начиная работать со списками, можно оценить возможности системы программирования Lisp¹. В этой системе нет различия

¹ По своей основе Lisp имеет, практически, все возможности бестипового лямбда-исчисления. В зависимости от применяемого диалекта внешняя форма записи объектов может варьироваться. Подчеркнем еще раз, что в этом языке нет *операторов*. Единственный вид используемых в нем объектов — это *функции*. Реализованные механизмы рекурсии с математической точки зрения

между “программами” и “данными”: и то, и другое — вполне равноправные объекты. Большой интерес, проявляемый к системе программирования Lisp, вполне оправдан. Имея четкие и краткие математические основания, эта система программирования преодолевает барьер между практическим программированием задачи и ее математическим осмыслением.

Решение. В качестве примера приведем соответствующие выкладки для функций *length* (вычисление длины списка) и *map* (функционал, “распределяющий” вдоль списка действие функции-аргумента). При вычислении этих функций проявляются основные особенности рекурсивных вычислений над списками.

length-1. Для функции *length* исходное определение

$$\begin{aligned} \text{length} &= \lambda x. \text{if } \text{null } x \\ &\quad \text{then } 0 \\ &\quad \text{else } 1 + \text{length}(\text{cdr } x), \end{aligned}$$

перепишем в виде:

$$\begin{aligned} \text{length} &= (\lambda f. \lambda x. \text{if } \text{null } x \\ &\quad \text{then } 0 \\ &\quad \text{else } 1 + f(\text{cdr } x)) \text{length}. \end{aligned}$$

length-2. Отсюда следует, что

$$\begin{aligned} \text{length} &= Y(\lambda f. \lambda x. \text{if } \text{null } x \\ &\quad \text{then } 0 \\ &\quad \text{else } 1 + f(\text{cdr } x)). \end{aligned}$$

Тем самым желаемая комбинаторная характеристика получена.

иллюстрируют эффект вычислений с неподвижной точкой. Обычно в языке дополнительно предлагаются нерекурсивные средства организации циклических вычислений.

length-3. Произведем проверку определения для списка длины 2, то есть возьмем $x = (a_1, a_2)$:

$$\begin{aligned}
 \text{length}(a_1, a_2) &= Y(\lambda f \lambda x. \text{if } \text{null } x \\
 &\quad \text{then } 0 \\
 &\quad \text{else } 1 + f(\text{cdr } x))(a_1, a_2) \\
 &= (\lambda f \lambda x. \text{if } \text{null } x \\
 &\quad \text{then } 0 \\
 &\quad \text{else } 1 + f(\text{cdr } x))(Y(\dots))(a_1, a_2) \\
 &= \text{if } \text{null } (a_1, a_2) \text{ then } 0 \text{ else } 1 + (Y(\dots))(\text{cdr } (a_1, a_2)) \\
 &= 1 + (Y(\dots))(\text{cdr } (a_1, a_2)) \\
 &= 1 + (\lambda f \lambda x. \text{if } \text{null } x \text{ then } 0 \\
 &\quad \text{else } 1 + f(\text{cdr } x))(Y(\dots))(a_2) \\
 &= 1 + \text{if } \text{null } (a_2) \text{ then } 0 \text{ else } 1 + (Y(\dots)) \text{ nil} \\
 &= 1 + (1 + (Y(\dots)) \text{ nil}) \\
 &= 1 + (1 + (\lambda f \lambda x. \text{if } \text{null } x \text{ then } 0 \\
 &\quad \text{else } 1 + f(\text{cdr } x))(Y(\dots)) \text{ nil}) \\
 &= 1 + (1 + (0)) = 2.
 \end{aligned}$$

map-1. Для функции *map* исходное определение

$$\begin{aligned}
 \text{map} &= \lambda f. \lambda x. \text{if } \text{null } x \\
 &\quad \text{then } () \\
 &\quad \text{else } (f(\text{car } x)) : (\text{map } f(\text{cdr } x))
 \end{aligned}$$

перепишем в виде:

$$\begin{aligned}
 \text{map} &= (\lambda m. \lambda f. \lambda x. \text{if } \text{null } x \\
 &\quad \text{then } () \\
 &\quad \text{else } (f(\text{car } x)) : (m f(\text{cdr } x))) \text{ map}.
 \end{aligned}$$

Отсюда следует, что

$$\begin{aligned}
 \text{map} &= Y(\lambda m. \lambda f. \lambda x. \text{if } \text{null } x \\
 &\quad \text{then } () \\
 &\quad \text{else } (f(\text{car } x)) : (m f(\text{cdr } x))).
 \end{aligned}$$

Проверка. Проверка для $f = \text{square}$, $x = (2, 3)$ (набросок):

$$\begin{aligned}
 \text{map square } (2, 3) &= \\
 &= (\lambda m \lambda f \lambda x. \text{if null } x \\
 &\quad \text{then } () \\
 &\quad \text{else } (f(\text{car } x)) : (m f(\text{cdr } x)))(Y(\dots)) \text{square } (2, 3) \\
 &= (\text{square } 2) : ((Y(\dots)) \text{square } (3)) \\
 &= (\text{square } 2) : ((\lambda m. \lambda f. \lambda x. \dots)(Y(\dots)) \text{square } (3)) \\
 &= (\text{square } 2) : ((\text{square } 3) : ((Y(\dots)) \text{square } ())) \\
 &= (\text{square } 2) : ((\text{square } 3) : ()) \\
 &= (4, 9).
 \end{aligned}$$

В данном случае символ ‘:’ принят для обозначения инфиксной формы функции *list*, поэтому принимаем в качестве соглашения об обозначениях, что $x : (y : (z : ())) \equiv \langle x, y, z \rangle \equiv (x, y, z)$.

Упражнения

Упражнение 9.1. Требуется записать функцию, реализующую алгоритм разложения в базисе I , K и S .

Указание. Как известно, все λ -термы можно исключить, заменив их конструкциями из комбинаторов I , K и S , а комбинатор I можно заменить на конструкцию SKK .

Функцию, формирующую комбинацию, обозначим, например, через **combine**. Теперь можно, например, записать функцию, которая устраняет идентификатор x из выражения E , записав ее в виде самоссылающегося определения. Затем, пользуясь теоремой о неподвижной точке, можно привести определение к стандартному виду, устранив вхождение **extract** в правую часть определения:

```

def rec extract x E
  if identifier E
  then if E = x
    then 'I'
    else combine 'K' E
  else if lambdaexp E
    then extract x (extract (bv E) (body E))
    else let F = extract x (rator E)
         let A = extract x (rand E)
         combine 'S' (combine F A)

```

Упражнение 9.2. Написать программу, которая преобразует λ -терм в комбинаторную форму, анализируя появление переменной в операторе или операнде:

- 1) если оператор не зависит от x , то вводится комбинатор B :

$$Bfgx = (f)(gx);$$

- 2) если операнд не зависит от x , то вводится комбинатор C :

$$Cf gx = (fx)g;$$

- 3) если ни оператор, ни операнд не зависят от x , то комбинатор не вводится:

$$fg = (f)(g);$$

- 4) если $g = I$, то

$$\begin{aligned}
 SfIx &= (fx)(Ix) = fxx = Wfx, \\
 BfIx &= (f)(Ix) = fx;
 \end{aligned}$$

- 5) если $f = K$, то

$$\begin{aligned}
 SKgx &= (Kx)(gx) = Ix, \\
 CKgx &= (Kx)(g) = Ix.
 \end{aligned}$$

Указание. См., например, [3], с. 45-46.

Глава 10

Функция *list1*

Покажем возможности построения функции-объекта в параметризованном виде. Придавая аргументами частные значения — а этими частными значениями могут быть и функции, — можно получить целое семейство определений частных функций.

10.1 Теоретические сведения

Определим функцию *list1* следующим образом:

$$\begin{aligned} \textit{list1 } a \ g \ f \ x \quad = \quad & \textit{if } (\textit{null } x) \\ & \textit{then } a \\ & \textit{else } g(f(\textit{car } x))(\textit{list1 } a \ g \ f(\textit{cdr } x)). \end{aligned}$$

Это пример функции достаточно общего вида, из которой, задав конкретное значение параметров, можно получить целый ряд “более простых” функций. Можно заметить, что многие функции, оперирующие со списками, имеют некоторую “общую часть”. Возникает вопрос, можно ли для класса функций над списками определить такую обобщенную функцию, из которой путем различного выбора параметров получаются конкретные предста-

вители класса функций. В качестве одной из таких обобщенных функций предлагается функция *list1*.

Основная конструкция, которой будем пользоваться, является *списком*, который может быть пустым, либо непустым. В последнем случае у него есть “голова” и “хвост”, которые в свою очередь также могут быть списками. Над списками могут выполняться следующие операции:

$null$: список \rightarrow булевский,
 car : непустой список \rightarrow (список + атом),
 cdr : непустой список \rightarrow список,
 $list$: (атом + список) \rightarrow (список \rightarrow список).

Эти операции связаны друг с другом следующим образом:

$null () = true,$
 $null (list\ x\ y) = false,$
 $car (list\ x\ y) = x,$
 $cdr (list\ x\ y) = y,$
 $list (car\ z)(cdr\ z) = z.$

Кроме того, примем сокращение:

$list\ x\ y = x : y,$

и поэтому для $n \geq 2$ воспользуемся соглашением об обозначении:

$\langle x_1, x_2, x_3, \dots, x_n \rangle = x_1 : (x_2 : (x_3 : (\dots x_n) : \dots () \dots)).$

10.2 Задачи

Задача 10.1. Исследовать свойства функции

$list1\ a\ g\ f\ x = if\ (null\ x)$
 $then\ a$
 $else\ g(f(car\ x))(list1\ a\ g\ f(cdr\ x)).$

Воспользовавшись следующими определениями:

$$Ix = x, Kxy = x, postfix\ x\ y = append\ y\ (ux),$$

где (ux) — обозначение списка, состоящего из единственного элемента x , выразить функции:

(а) *length*, *sumsquares*, *reverse*, *identity*;

(б) *sum*, *product*, *append*, *concat*, *map*.

Формулировка задачи. Воспользуемся следующими определениями функций:

(а) $ux = x : ()$,
 $length\ x = if\ null\ x\ then\ 0\ else\ 1 + length\ (cdr\ x)$,
 $sumsquares\ x = if\ null\ x$
 $then\ ()$
 $else(square(car\ x)) + sumsquares(cdr\ x)$,
 $reverse\ x = if\ null\ x\ then\ ()$
 $else\ append(reverse(cdr\ x))(ux)$,
 $identity\ x = x$,
 $square\ x = if\ null\ x\ then\ 0\ else\ x \times x$;

```

(6)  sum x = if null x
        then 0
        else(car x) + sum(cdr x),

product x = if null x
        then 1
        else(car x) × product(cdr x),

append x y = if null x
        then y
        else list(car x)(append(cdr x)y),

concat x = if null x
        then ()
        else append(car x)(concat(cdr x)),

map f x = if null x
        then ()
        else(f(car x)) : (map f(cdr x)).

```

Выполнить шаги алгоритмов для следующих примеров:

```

      sum (1, 2, 3, 4) = 10,
      product (1, 2, 3, 4) = 24,
      append (1, 2)(3, 4, 5) = (1, 2, 3, 4, 5),
      concat ((1, 2), (3, 4), ()) = (1, 2, 3, 4),
      map square (1, 2, 3, 4) = (1, 4, 9, 16).

```

Решение. Принимается, что

$$\textit{postfix } x y = \textit{append } y (ux).$$

list1–1. Рассмотрим случай (а):

```

      length = list1 0 plus (K 1),
      sumsquares = list1 0 plus square,
      reverse = list1 () postfix I,
      identity = list1 () list I.

```

list1–2. Рассмотрим случай (б):

$$\begin{aligned} \text{sum} &= \text{list1 } 0 \text{ plus } I, \\ \text{product} &= \text{list1 } 1 \text{ multiply } I, \\ \text{append } x \ y &= \text{list1 } y \ \text{list } I \ x, \\ \text{concat} &= \text{list1 } () \ \text{append } I, \\ \text{map } f &= \text{list1 } () \ \text{list } f. \end{aligned}$$

Для завершения решения требуется подставить параметры и произвести детальные вычисления. Кроме того, предполагается выполнение проверки примеров.

10.3 Функтор-как-объект

В заключение обратим внимание на некоторые особенности метода решения задачи. Прежде всего функция *list1* представляет собой функционал (и даже функтор). Определив эту функцию, на самом деле выполнили значительно больший объем работы, чем требовалось. Более конкретно, *list1* проявляет существенную зависимость от параметров: варьируя параметры, получаем целое семейство частных функций, каждая из которых имеет достаточно общий вид. Тем самым определение *list1* фиксирует *понятие*, или *концепт*. Поскольку концепт задан описанием, то задан *интенционал*. Выбирая различные значения параметров, или указывая *соотнесения*, на деле получаем целое семейство функций-индивидов. Перечислив элементы этого семейства, получаем *экстенционал* концепта *list1*.

Функции, наподобие *list1*, в программировании представляют важную идею, носящую специальное название ‘функтор-как-объект’. Как можно видеть, программа, составленная из таких объектов, проявляет высокую степень общности.

Упражнения

Упражнение 10.1. Определить типы аргументов *list1*.

Указание. Рассмотрим определение *list1* и будем действовать поэтапно.

- 1) Функция $(list1\ a\ g\ f)$ применяется к *списку*. Предположим, что он имеет следующую структуру:

список — либо пустой элемент (null), либо имеет голову (head) и хвост (tail), который в свою очередь представляет собой список.

Считаем, что такой список имеет тип A . Таким образом,

$$(list1\ a\ g\ f) \in (A \rightarrow B);$$

- 2) заключаем, что аргументы a нужно брать из B ;
- 3) функция f применяется к элементам из A , а значения принимает из C :

$$f \in (A \rightarrow C);$$

- 4) таким образом, функция g должна применяться к элементам из C , а результат применения $(list1\ a\ g\ f)$ к элементам из A принадлежит B ; отсюда заключаем, что

$$g \in (C \rightarrow (B \rightarrow B)).$$

Дополнительные детали анализа см. в [3], с. 110–111.

Глава 11

Изоморфизм д.з.к. и АВС

Теперь начинаем продвигаться вглубь математических абстракций и будем увязывать операторный стиль мышления с комбинаторным. Отметим, что в комбинаторной логике используется единственный оператор — *оператор аппликации*, или оператор приложения (применения) одного объекта к другому. Возникающая при этом система вычислений носит название *аппликативной вычислительной системы*, или АВС. Она увязывается с традиционной операторной вычислительной системой, которая представлена специальным объектом — *декартово замкнутой категорией*, или д.з.к.

11.1 Теоретические сведения

Идея построения функционального языка, в котором вовсе отсутствуют переменные, основывается на использовании комбинаторной логики. Отказываясь от переменных, проектировщик языка программирования начинает оперировать произвольными объектами, которые при построении программы разрешается применять друг к другу. Очевидно, что это наиболее рафинированная форма использования объектов.

Как известно, для корректного использования объектов следует оставаться в рамках какого-либо варианта комбинаторной логики, играющего роль теории-оболочки. Комбинаторы из этой оболочки в совокупности составляют “набор инструкций” некоторой абстрактной вычислительной системы. Может сложиться впечатление, что этот набор ничем не ограничен. В этом случае его математические свойства остаются непроявленными и потенциально содержащими ту или иную форму противоречия.

Выберем в качестве теории-оболочки теорию категорий. Зафиксируем в ней набор комбинаторов, который проявляет вполне безопасное математическое поведение: составляет декартово замкнутую категорию. Заметим, что в декартово замкнутой категории (д.з.к.) функции понимаются как операторы, действующие на свои операнды, которые записываются позиционно. С другой стороны, в аппликативной вычислительной системе (АВС) используется единственная операция — операция аппликации, трактуемая как приложение одного объекта к другому. Возникает вопрос, не происходит ли каких-либо потерь при переходе от системы понятий и определений д.з.к. к системе понятий и определений АВС и наоборот.

Воспользуемся следующими соглашениями об обозначениях:

$$[x, y] \equiv \lambda r. rxy,$$

$$< f, g > \equiv \lambda t. [f(t), g(t)] \equiv \lambda t. \lambda z. z(f(t))(g(t)).$$

Тот факт, что f является отображением из A в B (в содержательном смысле) будем обозначать посредством $f : A \rightarrow B$. Введем в

рассмотрение следующие отображения:

$$\begin{aligned}
 h &: A \times B \rightarrow C \text{ для } x : A, y : B, \\
 \Lambda_{ABC} h &: A \rightarrow (B \rightarrow C), \\
 \Lambda_{ABC} &: (A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)), \\
 k &: A \rightarrow (B \rightarrow C), \\
 \varepsilon_{BC} &: (B \rightarrow C) \times B \rightarrow C, \\
 \varepsilon \circ \langle k \circ p, q \rangle &: A \times B \rightarrow C, \\
 p &: A \times B \rightarrow A, \quad q : A \times B \rightarrow B.
 \end{aligned}$$

В дальнейшем эти отображения будут часто использоваться без явного упоминания их типов, если только не возникает двусмысленности в толковании.

11.2 Задачи

Задача 11.1. Поскольку в оболочке Каруби выводимо:

$$h = \varepsilon \circ \langle \Lambda h \rangle \circ p, q \rangle,$$

а также выводимо:

$$k = \Lambda(\varepsilon \circ \langle k \circ p, q \rangle),$$

то прежде всего из предыдущих определений следует самостоятельно получить оба равенства.

Решение.

Λ —1. Во-первых, покажем, как строить перевод выражений из операторной формы в аппликативную, то есть ищем функцию h' такую, что $h[x, y] = h'xy$. Это получается следующим

образом:

$$\begin{aligned}
 h &= \varepsilon \circ < (\Lambda h) \circ p, q >; \\
 h[x, y] &= (\varepsilon_{BC} \circ < (\Lambda h) \circ p, q >)[x, y] \\
 &= \varepsilon_{BC}(< (\Lambda h) \circ p, q >[x, y]) \\
 &= \varepsilon_{BC}(((\Lambda h) \circ p)[x, y], q[x, y]) \\
 &= \varepsilon_{BC}((\Lambda h)(p[x, y]), q[x, y]) \\
 &= \varepsilon_{BC}(((\Lambda h)x), y) \\
 &= (((\Lambda h)x)y) = \Lambda h x y \\
 &\equiv h' x y.
 \end{aligned}$$

Проделанные вычисления нуждаются в правильном приписывании объектам типов¹. Следовательно, $h' x y = (\Lambda h)x y$.

Λ-2. Во-вторых, покажем, как строить перевод выражений из аппликативной в операторную форму, то есть ищем функцию k' такую, что $k x y = k'[x, y]$. Это получается следующим образом:

$$\begin{aligned}
 k &= \Lambda(\varepsilon \circ < k \circ p, q >); \\
 kxy &= \Lambda(\varepsilon \circ < k \circ p, q >)xy \\
 &= (\lambda u. \lambda v. (\varepsilon \circ < k \circ p, q >)[u, v])xy \\
 &= \varepsilon \circ < k \circ p, q >[x, y] \\
 &\equiv k'[x, y].
 \end{aligned}$$

Таким образом, $k' = \varepsilon \circ < k \circ p, q >$ и $\Lambda k' = k$. Остается заметить, что через '=' обозначаем отношение $\eta\xi$ -конвертируемости: оно рефлексивно, симметрично и транзитивно. Следовательно, оно является отношением эквивалентности. Последнее обстоятельство позволяет прийти к заключению, что

$$(A \times B \rightarrow C) \cong (A \rightarrow (B \rightarrow C)),$$

где символ ' \cong ' читается как 'изоморфно'.

¹ У выражения $((\Lambda h)x)$ тип $B \rightarrow C$. Выражению y приписан тип B . Выражение $((\Lambda h)x)y$ получает тип C .

Глава 12

Каррирование

Используемые в операторном программировании n -местные функции-операторы в комбинаторной логике имеют образы в виде объектов, которые наследуют все их существенные свойства.

12.1 Теоретические сведения

12.1.1 Операторы и функции

В программировании часто приходится устанавливать различие между понятием ‘оператор’ и понятием ‘функция’. В первом случае руководствуются алгебраическими идеями, когда у всякого оператора заранее предполагается аридность (известное число аргументов, называемых операндами). Это число операндов известно заранее, и оно связано с самим видом конкретного оператора. Другое проявление операторности — это работа с функциями, которые хотя и имеют произвольный характер, но для каждой из них также заранее известно число аргументов. Это старо-традиционная точка зрения на вычисления и построение исчислений. В ее основе лежит относительное противопоставление символа функции или оператора с одной стороны символам аргументов с другой сто-

роны. Молчаливо предполагается, что *объекты* имеются, но они неравноправны: объекты-операторы используются по одним правилам, а объекты-операнды — по другим. Наиболее часто используемое предположение касается замещения одних объектов на другие. В рамках формализаций первого порядка операнды могут замещаться на другие объекты, а операторы обычно не могут. В этом состоит смысл ограничения, накладываемого на подстановку в этих системах. Системы первого порядка называют системами с *ограниченным принципом свертывания*, поскольку конкретно принятое определение операции подстановки реализует идею свертывания.

12.1.2 Свертывание

При работе с действительно произвольными функциями рассуждение о вычислениях приходится вести в терминах применения символа функции к соответствующему символу аргумента. Еще большая симметрия в толковании функций и аргументов достигается, если считать их объектами — без дополнительных оговорок, — и исследование процесса вычисления свести к рассуждению о приложении (апплицировании) одного объекта к другому. В этом случае на выполнение подстановки можно не накладывать обременительных ограничений, что позволяет перейти к формализациям высших порядков. Если порядок такой теории не ограничен, то это теория с *неограниченным принципом свертывания*.

12.1.3 Соответствие операторов и функций

Между обоими видами систем можно устанавливать различные связи, проявляя потенциальные возможности и того, и другого подхода. В частности, зададимся вопросом, как средствами АВС (пользуясь операторами аппликации и абстракции) выразить содержательное представление 2-местных, 3-местных, ..., n -местных функций, трактуемых как операторы. Воспользуемся следую-

щими соглашениями об обозначениях:

$$\begin{aligned} [x, y] &= \lambda r. rxy, \\ h &: A \times B \rightarrow C, \\ \text{Curry}_{ABC} &: (A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)). \end{aligned}$$

Таким образом, h считается обычным двухместным оператором, а Curry — преобразованием из операторного вида в аппликативный¹:

$$\begin{aligned} \text{Curry}_{ABC} h &= \lambda xy. h[x, y], \\ \lambda xy. h[x, y] &: A \rightarrow (B \rightarrow C). \end{aligned}$$

12.2 Задачи

Задача 12.1. Рассмотрим семейство функций h :

$$\begin{aligned} h_2 &: A \times B \rightarrow C, \\ h_3 &: A \times B \times C \rightarrow D, \\ h_4 &: A \times B \times C \times D \rightarrow E, \\ &\dots : \dots \end{aligned}$$

Найти семейство отображений:

$$\text{Curry}_{ABC}, \text{Curry}_{(A \times B)CD}, \text{Curry}_{(A \times B \times C)DE}, \dots,$$

которые каррируют данные функции, то есть переводят их в аппликативный вид.

Решение. В качестве примера рассмотрим каррирование h_3 и h_4 .

Curry—1. Действительно, пусть $h_3 : (A \times B) \times C \rightarrow D$.

Тогда $\Lambda_{(A \times B)CD} h_3 = \lambda xy. h_3[x, y] : A \times B \rightarrow (C \rightarrow D)$.

Теперь можно считать, что $\Lambda_{(A \times B)CD} h_3 = h'_2$, и поэтому

¹ В теоретических исследованиях вместо обозначения '*Curry*' часто используется обозначение ' Λ '; в дальнейшем будем использовать именно это последнее обозначение.

следующая идея состоит в подстановке вместо первой переменной — упорядоченной пары переменных, то есть

$$\begin{aligned}
 \Lambda_{AB(C \rightarrow D)}(\Lambda_{(A \times B)CD}h_3) &= \\
 &= \lambda uv.(\Lambda_{(A \times B)CD}h_3)[u, v] \\
 &= \lambda uv.(\lambda xy.h_3[x, y])[u, v] \\
 &= \lambda uv.(\lambda y.h_3[[u, v], y]) \\
 &= \lambda uv.y.(h_3[[u, v], y]) : A \rightarrow (B \rightarrow (C \rightarrow D)) \\
 &= (\Lambda_{AB(C \rightarrow D)} \circ \Lambda_{(A \times B)CD}) h_3.
 \end{aligned}$$

Curry-2. Пусть теперь $h_4 : A \times B \times C \times D \rightarrow E$, где предполагается, что

$$A \times B \times C \times D = (A \times B \times C) \times D = ((A \times B) \times C) \times D.$$

Тогда рассмотрим преобразование каррирования по шагам.

Шаг 1:

$$\Lambda_{((A \times B) \times C)DE}h_4 = \lambda xy.h_4[x, y] : ((A \times B) \times C) \rightarrow (D \rightarrow E).$$

Шаг 2:

$$\begin{aligned}
 \Lambda_{(A \times B)C(D \rightarrow E)}(\Lambda_{((A \times B) \times C)DE}h_4) &= \\
 &= \lambda uv.(\Lambda_{((A \times B) \times C)DE}h_4)[u, v] \\
 &= \lambda uv.y.h_4[[u, v], y] : A \times B \rightarrow (C \rightarrow (D \rightarrow E)).
 \end{aligned}$$

Шаг 3:

$$\begin{aligned}
 \Lambda_{AB(C \rightarrow (D \rightarrow E))}(\lambda uv.y.h_4[[u, v], y]) &= \\
 &= \lambda \bar{x} \bar{y}.(\lambda uv.y.h_4[[u, v], y])[\bar{x}, \bar{y}] \\
 &= \lambda \bar{x} \bar{y}v.y.h[[\bar{x}, \bar{y}], v], y].
 \end{aligned}$$

Обсуждая полученное в пунктах (*Curry-1*) и (*Curry-2*) решение, можно заметить, что функции каррирования имеют вид:

$$\Lambda_{(A \times B \times C)DE} = \Lambda_{AB(C \rightarrow (D \rightarrow E))} \circ \Lambda_{(A \times B)C(D \rightarrow E)} \circ \Lambda_{(A \times B) \times C)DE}.$$

Для целей отыскания решения в общем случае перепишем это равенство в виде:

$$\begin{aligned}\Lambda_{(A_1 \times A_2 \times A_3)A_4B} &= \\ &= \Lambda_{A_1A_2(A_3 \rightarrow (A_4 \rightarrow B))} \circ \Lambda_{(A_1 \times A_2)A_3(A_4 \rightarrow B)} \circ \Lambda_{((A_1 \times A_2) \times A_3)A_4B}.\end{aligned}$$

Упражнения

Упражнение 12.1. Остается самостоятельно получить соответствующее равенство для n -местных функций

Указание. Это выполняется индукцией по числу аргументных мест.

Упражнение 12.2. Установить связь каррированных и некаррированных функций.

Решение. Для всякой функции

$$f : [D_1 \times D_2 \times \dots \times D_n] \rightarrow D$$

найдется эквивалентная ей каррированная функция

$$Curry f : D_1 \rightarrow [D_2 \rightarrow [\dots [D_n \rightarrow D] \dots]]$$

Пользуясь λ -обозначениями, нетрудно заключить, что

$$Curry f = \lambda x_1 x_2 \dots x_n. f(x_1, x_2, \dots, x_n)$$

Далее, $Curry$ в свою очередь является функцией высших порядков:

$$\begin{aligned}Curry &: [[D_1 \times D_2 \times \dots \times D_n] \rightarrow D] \\ &\quad \rightarrow [D_1 \rightarrow D_2 \rightarrow \dots \rightarrow D_n \rightarrow D] \\ Curry &= \lambda f. \lambda x_1 x_2 \dots x_n. f(x_1, x_2, \dots, x_n) \\ &= \lambda f x_1 x_2 \dots x_n. f(x_1, x_2, \dots, x_n)\end{aligned}$$

Пример 12.1.

$$\begin{aligned}
 plus &= \lambda x_1 x_2. + x_1 x_2 \\
 &= \lambda x_1 x_2. plus(x_1, x_2) \\
 &= (\lambda f. \lambda x_1 x_2. f(x_1, x_2)) plus \\
 &= (\lambda f x_1 x_2. f(x_1, x_2)) plus \\
 &= Curry plus
 \end{aligned}$$

Для функции *Curry* можно указать функцию *unCurry* с противоположным эффектом:

$$\begin{aligned}
 unCurry &: [D_1 \rightarrow D_2 \rightarrow \dots \rightarrow D_n \rightarrow D] \\
 &\quad \rightarrow [[D_1 \times D_2 \times \dots \times D_n] \rightarrow D] \\
 unCurry &= \lambda f. \lambda(x_1, x_2, \dots, x_n). f x_1 x_2 \dots x_n
 \end{aligned}$$

Пример 12.2.

$$\begin{aligned}
 plus &= \lambda(x_1, x_2). + [x_1, x_2] \\
 &= \lambda(x_1, x_2). plusc x_1 x_2 \\
 &= (\lambda f. \lambda(x_1, x_2). f x_1 x_2) plusc \\
 &= unCurry plusc.
 \end{aligned}$$

Пример без труда обобщается и дает следующий результат.

Утверждение 12.1. Если $Curry h = k$, то:

$$\begin{aligned}
 unCurry(Curry h) &= h \\
 Curry(unCurry k) &= k
 \end{aligned}$$

Глава 13

Оболочка Каруби

Специальная категория, называемая оболочкой Каруби, позволяет лаконично выразить весь запас знаний, имеющийся относительно операторов, в терминах комбинаторной логики. При этом типы также кодируются объектами. Тем самым выполняется погружение типового приложения в бестиповую программную среду.

13.1 Теоретические сведения

Исследовательская работа в программировании часто начинается с выбора объемлющей теории, то есть теории-оболочки, в которой удобно представлять и исследовать вновь разрабатываемые механизмы. При необходимости установления и поддержания *системы типов* возникает необходимость в использовании такой теории-оболочки, которая позволяет устанавливать и рассматривать различные идеи, касающиеся типов. По всей вероятности, в оболочке лучше совсем отказаться от какой-либо априорной идеи типизации. Другими словами, приходится иметь дело с бестиповой теорией, и хорошим примером такого рода служит бестиповое λ -исчисление.

В настоящем разделе исследуется установление связи теоретико-категорных понятий с понятиями бестипового λ -исчисления. Пусть \mathcal{L} — совокупность термов некоторого исчисления λ -конверсий. Оболочкой Каруби для \mathcal{L} , обозначаемой через $\mathcal{C}(\mathcal{L})$, будем считать категорию, определяемую следующим образом. Положим

$$a \circ b \equiv \lambda x. a(bx)$$

для a, b принадлежащих \mathcal{L} , где ‘ \circ ’ — знак композиции функций.

Помимо того, воспользуемся следующими обозначениями:

$$\begin{aligned} \{a \in \mathcal{L} \mid a \circ a = a\} &— \text{множество объектов категории,} \\ \{f \in \mathcal{L} \mid b \circ f \circ a = f\} &— \text{множество морфизмов, } \text{Hom}(a, b), \\ id &— \text{тождественный морфизм, } id \circ a = a \\ f \circ g &— \text{композиция морфизмов.} \end{aligned}$$

13.2 Задачи

Задача 13.1. Показать, что $\mathcal{C}(\mathcal{L})$ — категория. Предлагается это проверить самостоятельно, выполнив согласование с какой-либо формой определения категории.

Формулировка задачи. Примем следующие определения необходимых для проведения исследования объектов. Напомним, что в данном случае необходимость выполнения следующих равенств предполагается заранее:

$$A = \lambda x. A(A(x)) \quad = A \circ A, \quad (A)$$

$$F = \lambda x. B(f(A(x))) = B \circ f \circ A. \quad (f)$$

1. Декартово произведение:

$$A \times B = \lambda u \lambda z. z(A(u(\lambda x \lambda y. x)))(B(u(\lambda x y. y))).$$

2. Проекция на первый и на второй элемент соответственно:

$$\begin{aligned} p_{AB} &= \lambda u.(A \times B)(u)(\lambda x \lambda y.x), & p_{AB} : A \times B &\rightarrow A; \\ q_{AB} &= \lambda u.(A \times B)(u)(\lambda x \lambda y.y), & q_{AB} : A \times B &\rightarrow B. \end{aligned}$$

3. Спаривание функций:

$$\begin{aligned} \langle f, g \rangle &= \lambda t \lambda z. z(f(t))(g(t)) = \lambda t. [f(t), g(t)], \\ f : C &\rightarrow A, \quad g : C \rightarrow B, \quad \langle f, g \rangle : C \rightarrow (A \times B). \end{aligned}$$

4. Множество отображений (функциональное пространство):

$$(A \rightarrow B) = \lambda f. B \circ f \circ A.$$

5. Аппликация (приложение функций к аргументу):

$$\begin{aligned} \varepsilon_{BC} &= \lambda u. C(u(\lambda xy.x)(B(u(\lambda xy.y)))), \\ \varepsilon_{BC} &: (B \rightarrow C) \times B \rightarrow C. \end{aligned}$$

6. Функция каррирования, то есть перевода “обычных” функций в аппликативный вид, названа в честь Х. Карри (Еще раз напоминаем: часто эту функцию обозначают через ‘*Curry*’; в данном случае полагаем, что $Curry \equiv \Lambda$, то есть функцию каррирования обозначаем через ‘ Λ ’):

$$\begin{aligned} \Lambda_{ABC} h &= \lambda x \lambda y. h(\lambda z. z(x)(y)), \\ h : (A \times B) &\rightarrow C, \quad \Lambda_{ABC} h : A \rightarrow (B \rightarrow C). \end{aligned}$$

Требуется доказать следующее:

- Свойства проекций:

$$\begin{aligned} p_{AB} \circ \langle f, g \rangle &= f, \quad q_{AB} \circ \langle f, g \rangle = g, \\ \langle p_{AB} \circ h, q_{AB} \circ h \rangle &= h. \end{aligned}$$

- Пусть $h : (A \times B) \rightarrow C$, $k : A \rightarrow (B \rightarrow C)$. Тогда

$$\begin{aligned}\varepsilon \circ < (\Lambda h) \circ p, q > &= h, \\ \Lambda(\varepsilon \circ < k \circ p, q >) &= k,\end{aligned}$$

где $\Lambda = \Lambda_{ABC}$, $p = p_{AB}$, $\varepsilon = \varepsilon_{BC}$, $q = q_{AB}$.

Решение. Доказательство сводится к проверке свойств введенных объектов.

$C(\mathcal{L})$ —1. Заметим, что отображение $h : (A \times B) \rightarrow C$ имеет перевод в терм λ -исчисления, причем выполняется равенство:

$$h = \lambda x. C(h((A \times B)(x))),$$

где $x = [x_1, x_2]$. Это непосредственное следствие (f).

$C(\mathcal{L})$ —2. Установим комбинаторную характеристику h :

$$\begin{aligned}h[x_1, x_2] &= C(h((A \times B)[x_1, x_2])) \\ &= C(h(\lambda z. z(A([x_1, x_2]K))(B([x_1, x_2](KI))))) \\ &= C(h(\lambda z. z(A x_1)(B x_2))) = C(h[A x_1, B x_2]).\end{aligned}$$

Таким образом, $h[x_1, x_2] = C(h[A x_1, B x_2])$.

$C(\mathcal{L})$ —3. Еще раз обратим внимание на необходимость учета следующих равенств:

$$\begin{aligned}x &= \lambda z. a(x(1 z)) = A \circ x, \\ f &= \lambda z. B(f(A z)) = B \circ f \circ A,\end{aligned}$$

где $1 = \lambda y. y = I$.

$C(\mathcal{L})$ —4. Нетрудно видеть (доказать самостоятельно!), что

$$(A \times B) = \lambda u. [A(u K), B(u(K I))],$$

где $K = \lambda xy.x$, $I = \lambda x.x$. Тогда непосредственной проверкой устанавливается следующая комбинаторная характеристика декартова произведения:

$$\begin{aligned}(A \times B)[u, v] &= \lambda z.z(A([u, v]K))(B([u, v](KI))) \\ &= \lambda z.z(Au)(Bv) \\ &= [Au, Bv].\end{aligned}$$

$\mathcal{C}(\mathcal{L})$ —5. Проверка свойств проекций приводит к следующим характеристикам:

$$\begin{aligned}p_{AB}([u, v]) &= (A \times B)[u, v]K = [Au, Bv]K = Au, \\ q_{AB}([u, v]) &= (A \times B)[u, v](KI) = [Au, Bv](KI) = Bv.\end{aligned}$$

$\mathcal{C}(\mathcal{L})$ —6. Рассматривая упорядоченную пару $[f, x]$, покажем, как с помощью отображения ε_{BC} получить аппликацию f к x :

$$\begin{aligned}\varepsilon_{BC}([C \circ f \circ B, B \circ x]) &= \varepsilon_{BC}([f, x]) \\ &= C([C \circ f \circ B, B \circ x]K(B([C \circ f \circ B, B \circ x](KI)))) \\ &= C((C \circ f \circ B)(B(B \circ x))) \\ &= C(f(Bx)) \\ &= (C \circ f \circ B)(x) \\ &= f(x).\end{aligned}$$

Отметим необходимость учета свойств композиции.¹ По существу, далее будем обосновывать простое равенство:

$$\Lambda_{ABC} h x y = h([x, y])$$

(см. раздел 12, где определена функция каррирования).

¹Напомним, что $C \circ f = f$, $B \circ x = x$.

$C(\mathcal{L})$ –7. Пусть t представимо упорядоченной парой, то есть $t = [t_1, t_2]$. Тогда

$$\begin{aligned}
 (\varepsilon \circ < (\Lambda h) \circ p, q >) t &= \varepsilon[(\Lambda h)(p t), q t] \\
 &= \varepsilon[(\lambda x y. h[x, y])(p t), q t] \\
 &= \varepsilon[(\lambda x y. h[x, y]) t_1, t_2] \\
 &= \varepsilon[\lambda y. h[t_1, y], t_2] \\
 &= (\lambda y. h[t_1, y]) t_2 \\
 &= h[t_1, t_2].
 \end{aligned}$$

Получено характеристическое равенство:

$$\varepsilon \circ < (\Lambda h) \circ p, q > = h.$$

$C(\mathcal{L})$ –8. Установим теперь второе характеристическое равенство, где $\#(t_1) = A$, $\#(t_2) = B$, $\#(t) = A \times B$, а $t = [t_1, t_2]$, то есть t представлено в виде упорядоченной пары; через ‘#’ обозначаем функцию ‘вычисления типа’:

$$\begin{aligned}
 \Lambda(\varepsilon \circ < k \circ p, q >) t_1 t_2 &= (\varepsilon \circ < k \circ p, q >)([t_1, t_2]) \\
 &= \varepsilon[k(p t), q t] = \varepsilon[k t_1, t_2] \\
 &= (\lambda u. C(u K (B(u(K I)))))[k t_1, t_2] \\
 &= C(k t_1 (B(t_2))) \\
 &= C(k t_1 t_2) \\
 &= k t_1 t_2.
 \end{aligned}$$

Тем самым² обосновано характеристическое равенство

$$\Lambda(\varepsilon \circ < k \circ p, q >) = k.$$

Все введенные равенства действительно имеют место, что завершает доказательство.

² Следует учесть, что $B(t_2) = t_2$.

Упражнения

Упражнение 13.1. Выполнить представление объекта метаданных, пользуясь комбинаторами.

Решение. Предполагается, что читатель знаком с техникой переменных объектов (см., например, раздел 23 настоящей книги). Построение объекта метаданных будем проводить по шагам, используя принцип свертывания.

(1) Использование свертывания. Для заданных объектов данных свертывание порождает объект метаданных. Далее верхний индекс используется для указания относительного уровня общности. Таким образом, метаобъект уровня $j + 1$ порождается объектами уровня j . Отметим, что свертывание использует дескрипцию, так что метаобъект z^{j+1} идентифицируется посредством x^{j+1} .

$$x^{j+1} \equiv \mathbf{I}z^{j+1} : \underbrace{[\dots [D] \dots]}_{j+1 \text{ раз}} \forall x^j : \underbrace{[\dots [D] \dots]}_{j \text{ раз}} (z^{j+1}(x^j) \leftrightarrow \Phi^j)$$

(2) Кодирование свертывания. Предположим следующее:

- истинностные значения $[\]$ представляют собой *переменный домен*;
- функция g является *предикатной* функцией $g : \mathcal{U} \rightarrow [\] \equiv [\mathcal{U}]$, следовательно, *предикатный концепт* является объектом

$$c \equiv \lambda A. \lambda g. x^2 \circ g \circ A \in [\mathcal{U}];$$

- j -й концепт определяется посредством

$$\begin{aligned} x^j &\equiv \mathbf{I}z^j : \underbrace{[\dots [D] \dots]}_{j \text{ раз}} \forall x^{j-1} : \underbrace{[\dots [D] \dots]}_{j-1 \text{ раз}} (z^j(x^{j-1}) \leftrightarrow \Phi^{j-1}) \\ &\equiv \{x^{j-1} \mid \Phi^{j-1}\} \\ &\subseteq \{x^{j-1} \mid x^{j-1} : A \rightarrow x^j\} \\ &= H_{x^j}(A) \equiv [\mathcal{U}]_A; \end{aligned}$$

- переменный домен $[\mathcal{U}]_A$ кодируется выражением

$$\lambda g. x^{j+1} \circ g \circ A;$$

- уточнение ‘моментами времени’ дает:

$$c \equiv \lambda A. (\underbrace{(\lambda h. [\] \circ h \circ A)}_{[\]_A}) \circ c \circ (\underbrace{(\lambda x^{j-1}. x^j \circ x^{j-1} \circ A)}_{\mathcal{U}_A});$$

(3) Заключительные шаги кодирования. Для завершения кодирования свертывания предположим следующее:

- фиксирование A и варьирование параметров f , B дает *предикатный концепт*

$$c \in [\mathcal{U}]_A, \quad c_f \subseteq \mathcal{U}_B;$$

- объект данных c_f определяется как *переменный предикат*

$$c_f \equiv \{t \in \mathcal{U}_B \mid [t/x^{j-1}](\|\bar{\Phi}^j\|_f(B))\}$$

для эвольвенты $f : B \rightarrow A$;

- метаобъект c_f кодируется посредством

$$c_f \in \lambda c. [\mathcal{U}]_B \circ (c \circ f) \circ B.$$

Замечание. Использование оболочки Каруби привлекает те же самые идеи, что и рассмотренные в разделе 10 при анализе функции *list1*. Действительно, λ -исчисление дает широкий спектр различных термов. Среди них отбираются только такие, на синтаксическую форму которых наложены специальные ограничения. Совокупность этих ограничений, перечисленная в начале настоящего раздела, задает *соотнесение*. В качестве теории-концепта (общая оболочка) выступает бестиповое λ -исчисление, рассматриваемое как теория вычислений. В результате ее применения к соотнесению получается теория-индивид, которая в свою очередь обладает свойствами оболочки. В частности, в ее рамках представим специальный класс вычислений — категориальные вычисления. В данном случае вопрос о сравнении выразительных возможностей теории-концепта и теории-индивида имеет математический характер.

Глава 14

Произведение и проекции

Окончательное завершение начатого процесса погружения достигается введением в рассмотрение упорядоченных совокупностей объектов. Как оказывается, аппликативные вычисления также допускают их представление.

14.1 Теоретические сведения

Задача погружения объектов одной теории в другую возникает при рассмотрении содержательно заданных отображений, когда требуется осуществить их трансляцию в аппликативный (функциональный) язык.

При выполнении такой работы предварительно потребуется небольшой запас определений:

$$K = \lambda xy.x, I = \lambda x.x, [x, y] = \lambda r.rxy$$

(упорядоченная пара).

14.2 Задача

Формулировка задачи. Получить терм λ -исчисления, соответствующий декартову произведению n -объектов. Дополнительно установить n термов, которые ведут себя как проекции.

Указание. Для случая $n = 2$ получаем

$$\begin{aligned} A_0 \times A_1 &= \lambda u. [A_0(u K), A_1(u(K I))], \\ \pi_0^2 &= \lambda u. (A_0 \times A_1)(u)K, \\ \pi_1^2 &= \lambda u. (A_0 \times A_1)(u)(K I). \end{aligned}$$

Решение. Прежде всего попытаемся отыскать закономерности в построении терма.

$\times \pi - 1$. Рассмотрим случай $n = 3$.

$$\begin{aligned} A_0 \times A_1 \times A_2 &= (A_0 \times A_1) \times A_2 \\ &= \lambda u. [(A_0 \times A_1)(u K), A_2(u(K I))] \\ &= \lambda u. [(\lambda v. [A_0(v K), A_1(v(K I))])(u K), A_2(u(K I))] \\ &= \lambda u. [A_0((u K)K), A_1((u K)(K I)), A_2(u(K I))]. \end{aligned}$$

Действительно, непосредственной проверкой можно убедиться, что

$$((A_0 \times A_1) \times A_2)[[x_0, x_1], x_2] = [[A_0(x_0), A_1(x_1)], A_2(x_2)].$$

Для установления соответствующих проекций сформулируем следующее утверждение (весьма простое) и выполним его проверку.

Утверждение 14.1.

$$\begin{aligned} \pi_0^3 &= \lambda u. (A_0 \times A_1 \times A_2)(u)(K)(K), \\ \pi_1^3 &= \lambda u. (A_0 \times A_1 \times A_2)(u)(K)(K I), \\ \pi_2^3 &= \lambda u. (A_0 \times A_1 \times A_2)(u)(K I). \end{aligned}$$

Доказательство.

$$\begin{aligned}
 \pi_0^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2] K K \\
 &= K[x_0, x_1] x_2 K \\
 &= [x_0, x_1] K \\
 &= x_0; \\
 \pi_1^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2] K (K I) \\
 &= K[x_0, x_1] x_2 (K I) = x_1; \\
 \pi_2^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2] (K I) \\
 &= (K I)[x_0, x_1] x_2 = I x_2 = x_2.
 \end{aligned}$$

Действительно, для определенного ранее вида произведения объекты π_j^3 , $j = 0, 1, 2$ ведут себя как проекции. \square

$\times \pi - 2$. Рассмотрим случай $n = 4$.

$$\begin{aligned}
 A_0 \times A_1 \times A_2 \times A_3 &= (A_0 \times A_1 \times A_2) \times A_3 \\
 &= \lambda u. [(A_0 \times A_1 \times A_2)(u K), A_3(u(K I))] \\
 &= \lambda u. [[[A_0(((u K) K) K), A_1(((u K) K)(K I))], \\
 &\quad A_2((u K)(K I))], A_3(u(K I))].
 \end{aligned}$$

Утверждение 14.2.

$$\begin{aligned}
 \pi_0^4 &= \lambda u. (A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K)(K), \\
 \pi_1^4 &= \lambda u. (A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K)(K I), \\
 \pi_2^4 &= \lambda u. (A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K I), \\
 \pi_3^4 &= \lambda u. (A_0 \times A_1 \times A_2 \times A_3)(u)(K I).
 \end{aligned}$$

Доказательство.

$$\begin{aligned}
 \pi_0^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3] K K K = x_0, \\
 \pi_1^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3] K K (K I) = x_1, \\
 \pi_2^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3] K (K I) = x_2, \\
 \pi_3^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3] (K I) = x_3.
 \end{aligned}$$

\square

Перейдем теперь к обобщению произведения.

Утверждение 14.3 (обобщение). По индукции можно доказать, что:

$$\begin{aligned}\pi_0^1 &= I, \pi_0 = \lambda y. y K, \pi_1 = \lambda y. y(K I); \\ \pi_n^{n+1} &= \lambda y. y(K I), i \leq 1; \pi_j^{n+1} = \pi_j^n \circ \pi_0, 0 \leq j \leq n-1.\end{aligned}$$

Доказательство. Предоставляется читателю. □

Указание. Перечислим некоторые соображения, относящиеся к доказательству обобщения. Частные случаи:

$n = 1$ (упорядоченные 2-ки).

$$\pi_1^2 = \pi_1, \pi_0^2 = \pi_0^1 \circ \pi_0 = I \circ \pi_0 = \pi_0.$$

$n = 2$ (упорядоченные 3-ки).

$$\pi_2^3 = \lambda y. y(K I) = \pi_1,$$

$$\pi_1^3 = \pi_1^2 \circ \pi_0 = \pi_1 \circ \pi_0,$$

$$\pi_0^3 = \pi_0^2 \circ \pi_0 = \pi_0^1 \circ \pi_0 \circ \pi_0.$$

$n = 3$ (упорядоченные 4-ки).

$$\pi_3^4 = \pi_1,$$

$$\pi_2^4 = \pi_2^3 \circ \pi_0 = \pi_1 \circ \pi_0,$$

$$\pi_1^4 = \pi_1^3 \circ \pi_0 = \pi_1^2 \circ \pi_0 \circ \pi_0 = \pi_1 \circ \pi_0 \circ \pi_0,$$

$$\begin{aligned}\pi_0^4 &= \pi_0^3 \circ \pi_0 = \pi_0^2 \circ \pi_0 \circ \pi_0 = \pi_0^1 \circ \pi_0 \circ \pi_0 \circ \pi_0 \\ &= \pi_0 \circ \pi_0 \circ \pi_0.\end{aligned}$$

Частные случаи проекций декартова произведения в преобразованном виде:

$n = 2.$

$$\pi_1^2 = \lambda y. y(K I),$$

$$\pi_0^2 = \lambda y. y K.$$

$n = 3.$

$$\begin{aligned}
 \pi_2^3 &= \lambda y. y(KI), \\
 \pi_1^3 &= \lambda y. \pi_1(\pi_0 y) \\
 &= \lambda y. \pi_1(y K) \\
 &= \lambda y. (y K)(K I), \\
 \pi_0^3 &= \lambda y. \pi_0(\pi_0 y) \\
 &= \lambda y. \pi_0(y K) \\
 &= \lambda y. (y K)K.
 \end{aligned}$$

$n = 4.$

$$\begin{aligned}
 \pi_3^4 &= \lambda y. y(KI), \\
 \pi_2^4 &= \lambda y. \pi_1(\pi_0 y) = \lambda y. \pi_1(y K) = \lambda y. (y K)(K I), \\
 \pi_1^4 &= \lambda y. \pi_1(\pi_0(\pi_0 y)) = \lambda y. \pi_1(\pi_0(y K)) \\
 &= \lambda y. \pi_1((y K)K) = \lambda y. y((y K)K)(K I), \\
 \pi_0^4 &= \lambda y. \pi_0(\pi_0(\pi_0 y)) = \lambda y. \pi_0(\pi_0(y K)) \\
 &= \lambda y. \pi_0((y K)K) = \lambda y. y((y K)K)K.
 \end{aligned}$$

Неформальное обсуждение решенной задачи сводится к следующим соображениям. Упорядоченные n -ки широко распространены, в частности, из них конструируются отношения реляционной базы данных. Напомним, что язык манипулирования данными реляционных систем управления базами данных рассматривает совокупности n -ок в виде самостоятельных *объектов*, называемых отношениями. Часто используемый запрос к базе данных состоит в усечении имеющихся отношений и может быть сведен к операциям проекции. Для выполнения этих операций нужен уже реализованный язык запросов.

В данном случае в качестве системы программирования используется АВС. Более точное рассуждение предполагает оговорки, что в качестве системы-оболочки используется оболочка Каруби (см. раздел 13). В рамках этой оболочки построены объекты-произведения и объекты-проекции. В свою очередь они состоят из объектов-комбинаторов. Программирование выполнено полностью в терминах объектов.

14.3 Произведение и декартово замкнутая категория

В настоящем разделе ограничимся тем, что приведем определение декартово замкнутой категории, или д.з.к. Это определение важно для лучшего понимания того, что изложено в разделе 11, к прочтению которого рекомендуется вернуться еще раз.

Пусть \mathcal{C} — категория. Тожественный морфизм объекта A будем обозначать через id_A .

\mathcal{C} называется *декартово замкнутой категорией*, если выполняются следующие условия:

- 1) в \mathcal{C} имеется *терминальный объект* T такой, что для произвольного объекта $A \in \mathcal{C}$ существует единственный морфизм $!_A : A \rightarrow T$;
- 2) для любых объектов $A_1, A_2 \in \mathcal{C}$ имеется объект $A_1 \times A_2$ — их *декартово произведение*, — с морфизмами $p_i : A_1 \times A_2 \rightarrow A_i$ — *проекциями*, — такими, что для любых $f_i : C \rightarrow A_i$ для $i = 1, 2$ имеется единственный морфизм $\langle f_1, f_2 \rangle : C \rightarrow A_1 \times A_2$ такой, что $p_i \circ \langle f_1, f_2 \rangle = f_i$;
- 3) для $A, B \in \mathcal{C}$ имеется объект $B^A \in \mathcal{C}$ — *степень*, $B^A \equiv A \rightarrow B$ — с морфизмом

$$\epsilon = \epsilon_{AB} : B^A \times A \rightarrow B,$$

такой, что для любого $f : C \times A \rightarrow B$ имеется единственный

$$\Lambda f : C \rightarrow B^A,$$

удовлетворяющий равенству $f = \epsilon \circ (\Lambda f \times id_A)$.

Обсуждение плодотворности этого определения требует знакомства с разделами 11-12, а также с разделами 19-22, в которых изложено одно из наиболее ярких применений д.з.к. к построению абстрактной машины.

Глава 15

Погружение Lisp в ABC

В аппликативную вычислительную систему встраивается нетривиальное приложение: значительный и, по-существу, полный фрагмент известной системы программирования Lisp (от англ. List processing).

15.1 Теоретические сведения

Одна из наиболее трудных проблем при разработке пользовательского (языкового) интерфейса состоит в правильном и подходящем встраивании приложения в программную среду. Успеху в ее решении, как известно, способствует применение объектного подхода и, естественно, объектно-ориентированное программирование. На практике определяется специальный набор классов, экспортирующий *методы* для прикладных программ.

В математике такой прием известен под названием *погружения*, когда в объемлющей теории, называемой *метатеорией*, строятся объекты, совокупность которых составляет *встроенную теорию*. Метатеория выступает в роли оболочки, а объекты встроенной теории могут адаптироваться по мере необходимости,

причем не происходит выхода за рамки метатеории. Рассмотрим использование этого приема на примере. В качестве метатеории используем бестиповую комбинаторную логику. Будем считать ее оболочкой объектов. В качестве встроенной теории построим систему объектов вместе с их характеристическими равенствами, которые реализуют интерфейс Lisp, то есть образуют функционально полный универсум рассуждений о *списках* и *атомах*. Под списком объектов, как обычно, понимаем конечную последовательность объектов, среди которых могут быть другие списки.

Язык Lisp, или ЛИСП, является, по существу бестиповым языком. Его основные конструкции совпадают с конструкциями бестипового λ -исчисления. Хорошо известно, что эти конструкции выразимы средствами комбинаторной логики. Целью настоящего исследования является установление комбинаторных характеристик некоторых функций языка программирования.

Воспользуемся $\eta\xi$ -исчислением λ -конверсий. Приведем постулаты, задающие отношение конвертируемости '*conv*' (обозначаемое знаком '='):

$$\begin{array}{ll}
 (\alpha) & \lambda x.a = \lambda z.[z/x]a, & (\beta) & (\lambda x.a)b = [b/x]a, \\
 (\nu) & \frac{a = b}{ac = bc}, & (\mu) & \frac{a = b}{ca = cb}, \\
 (\tau) & \frac{a = b; \quad b = c}{a = c} & (\sigma) & \frac{a = b}{b = a} & (\rho) & a = a \\
 (\eta) & \lambda x.bx = b, \quad x \notin b. & (\xi) & \frac{a = b}{\lambda x.a = \lambda x.b}
 \end{array}$$

15.2 Основная задача

Задача 15.1. Выразить с помощью комбинаторов следующий набор функций Lisp-системы:

$$\{\text{Append}, \text{Nil}, \text{Null}, \text{List}, \text{Car}, \text{Cdr}\}. \quad (\text{Lisp})$$

Формулировка задачи. Рассмотрим свойства этих функций языка Lisp.

- Посредством функции *Append* строится конкатенация двух списков. Эта функция обладает свойством *ассоциативности*:

$$A \frown B \frown C = (A \frown B) \frown C, \quad (\text{Append})$$

где A, B, C — произвольные списки, знак ' \frown ' — инфиксная форма записи функции *Append*.

- Пустой список обозначается как $< >$ и эквивалентен объекту *Nil*. Очевидно, что

$$A \frown < > = < > \frown A = A. \quad (\text{Nil}), (< >)$$

- Функция *Null* распознает пустой список:

$$\text{Null } A = \begin{cases} \bar{1}, & \text{если } A = \text{Nil}, \\ \bar{0}, & \text{в противном случае.} \end{cases} \quad (\text{Null})$$

- Функция *List* строит из атома список длины 1:

$$\text{List } x = < x >, \quad (\text{List})$$

где x — атом, а $< x_1, x_2, \dots, x_n >$ — список длины n .

- Функция *Car* выбирает первый элемент списка:

$$\text{Car } < x_1, x_2, \dots, x_n > = x_1. \quad (\text{Car})$$

- Функция *Cdr* убирает первый элемент из списка:

$$\text{Cdr } < x_1, x_2, \dots, x_n > = < x_2, \dots, x_n >. \quad (\text{Cdr})$$

На основании этих свойств сформулируем следующие схемы аксиом:

$$\text{Append } a (\text{Append } b c) = \text{Append}(\text{Append } a b)c, \quad (15.1)$$

$$\text{Append Nil } a = \text{Append } a \text{ Nil}, \quad (15.2)$$

$$\text{Null Nil} = \bar{1}, \quad (15.3)$$

$$\text{Null}(\text{Append}(\text{List } a)b) = \bar{0}, \quad (15.4)$$

$$\text{Car}(\text{Append}(\text{List } a)b) = a, \quad (15.5)$$

$$\text{Cdr}(\text{Append}(\text{List } a)b) = b, \quad (15.6)$$

где a, b, c — произвольные объекты. Докажем, что аксиомы (15.1)–(15.6) выводимы в $\eta\xi$ -исчислении λ -конверсии.

Решение. Осуществим последовательный перевод содержательных равенств (15.1)–(15.6) в термы и формулы комбинаторной логики.

Lisp-1. Покажем, что функции *Append* соответствует объект B с комбинаторной характеристикой $(B) : Babc = a(bc)$ (схема аксиом (15.1)):

$$\begin{aligned} Ba(Bbc) x &= a(Bbcx) && (\text{по } (B)) \\ &= a(b(cx)) && (\text{по } (B)) \\ &= Bab(cx) && (\text{по } (B)) \\ &= B(Bab)cx. && (\text{по } (B)) \end{aligned}$$

Учитывая правило транзитивности (τ) , имеем:

$$Ba(Bbc) x = B(Bab)cx.$$

В $\eta\xi$ -исчислении доказуемо, что для переменной x :

$$\frac{z_1 x = z_2 x}{z_1 = z_2},$$

или, в линейной записи, $z_1x = z_2x \Rightarrow z_1 = z_2$, то есть, полагая $z_1 = Ba(Bbc)$ и $z_2 = B(Bab)c$, получаем следующее:

- | | | |
|-----|---|----------------------------|
| (1) | $z_1x = z_2x \Rightarrow \lambda x.z_1x = \lambda x.z_2x$, | (по (ξ)) |
| (2) | $\lambda x.z_1x = z_1$, | (по (η)) |
| (3) | $z_1 = \lambda x.z_1x$, | (по (σ) , (2)) |
| (4) | $\lambda x.z_2x = z_1$, | (по (τ) , (1), (3)) |
| (5) | $z_2 = \lambda x.z_2x$, | (по (η)) |
| (6) | $z_1 = z_2$. | (по (τ) , (4), (5)) |

Итак, схема аксиом (15.1) доказана.

Lisp-2. Докажем схему аксиом (15.2), принимая во внимание, что $Nil \leftrightarrow I$ выполняется¹, причем $(I) : Ia = a$.

$$\begin{aligned}
 BIax &= I(ax) && \text{(по (B))} \\
 &= ax && \text{(по (I))} \\
 &= a(Ix) && \text{(по (I))} \\
 &= BaIx. && \text{(по (B))}
 \end{aligned}$$

Поскольку $BIax = BaIx$, то $BIa = BaI$. Это заключение устанавливается приемом, аналогичным примененному в ходе обоснования предыдущей аксиомы. Поскольку он будет применяться достаточно часто, то специально сформулируем соответствующее правило:

$$(\nu^{-1}) : \quad \frac{ux = vx}{u = v}.$$

Это правило, как оказалось, работает в случае, когда x является переменной. Если сопоставить его с одним из правил монотонности (ν) , то можно заметить, что посылка и заключение в нем поменялись местами. На этом основании (в случае, когда x — переменная) это правило (ν^{-1}) может быть названо правилом *обращения монотонности*.

¹ знак ' \leftrightarrow ' обозначает взаимно однозначное соответствие.

Lisp–3. Перепишем схему аксиом (15.3) в виде равенства $\bar{I} = \text{Null Nil}$ (по правилу (σ)), где $\text{Nil} = I$, \bar{I} – нумерал, комбинаторная характеристика которого имеет вид $\bar{I}ab = ab$, или в терминах λ -исчисления, $\bar{I} = \lambda xy.xy$. Заметим, что

$$\begin{aligned}(D) : \quad Dabc &= cab, \\ (\bar{0}) : \quad \bar{0}ab &= b, \text{ или } \bar{0} \leftrightarrow \lambda xy.y.\end{aligned}$$

Следует учесть, что $KI \leftrightarrow \bar{0}$, то есть $KI = \bar{0}$. Теперь найдем объект, соответствующий функции Null :

$$\begin{aligned}\bar{I} &= \lambda xy.xy && (\text{ по определению } \bar{I}) \\ &= \lambda x.x && (\text{ доказуемо } \lambda xy.xy = \lambda x.x) \\ &= I && (\text{ по определению } I) \\ &= KI(K(K\bar{0})) && (\text{ по схеме } (K)) \\ &= I(KI)(K(K\bar{0})) && (\text{ по схеме } (I)) \\ &= D(KI)(K(K\bar{0}))I && (\text{ по схеме } (D)) \\ &= D\bar{0}(K(K\bar{0}))I. && (\text{ по схеме } (\bar{0}))\end{aligned}$$

Сравнивая полученное выражение $D\bar{0}(K(K\bar{0}))I = \bar{I}$ и схему $\text{Null Nil} = I$ (или, более строго, схему $\text{Null Nil} = \bar{I}$), получаем что $D\bar{0}(K(K\bar{0}))I \leftrightarrow \text{Null Nil}$.

Lisp–4. Проведем следующие преобразования:

$$\begin{aligned}\bar{0} &= K\bar{0}(b\bar{0}) && (\text{ по схеме } (K)) \\ &= K(K\bar{0})a(b\bar{0}) && (\text{ по схеме } (K)) \\ &= Da(b\bar{0})(K(K\bar{0})) && (\text{ по схеме } (D)) \\ &= B(Da)b\bar{0}(K(K\bar{0})) && (\text{ по схеме } (B)) \\ &= D\bar{0}(K(K\bar{0}))(B(Da)b). && (\text{ по схеме } (D))\end{aligned}$$

Сравним полученное выражение $D\bar{0}(K(K\bar{0}))(B(Da)b) = \bar{0}$ со схемой аксиом (15.4): $\text{Null}(\text{Append}(\text{List } a)b) = \bar{0}$. Если учесть, что $D\bar{0}(K(K\bar{0})) \leftrightarrow \text{Null}$, $B \leftrightarrow \text{Append}$, то $D \leftrightarrow \text{List}$.

Lisp–5. Аналогично, как и в предыдущем пункте, найдем объект, соответствующий функции *Car*:

$$\begin{aligned}
 a &= Ka(bc) && (\text{ по схеме } (K)) \\
 &= Da(bc)K && (\text{ по схеме } (D)) \\
 &= B(Da)bcK && (\text{ по схеме } (B)) \\
 &= DcK(B(Da)b). && (\text{ по схеме } (D)).
 \end{aligned}$$

Очевидно, что $DcK \leftrightarrow Car$.

Lisp–6. Тем же способом построим объект, соответствующий функции *Cdr*:

$$\begin{aligned}
 bz &= I(bz) && (\text{ по схеме } (I)) \\
 &= KIa(bz) && (\text{ по схеме } (K)) \\
 &= Da(bz)(KI) && (\text{ по схеме } (D)) \\
 &= B(Da)bz(KI) && (\text{ по схеме } (B)) \\
 &= (\lambda xy.xy(KI))(B(Da)b)z. && (\text{ по } (\beta), (\sigma))
 \end{aligned}$$

Поскольку

$$bz = (\lambda xy.xy(KI))(B(Da)b)z,$$

то

$$(\lambda xy.xy(KI))(B(Da)b) = b$$

для переменной z (применяется правило обращения монотонности), то есть

$$\lambda xy.xy\bar{0} \leftrightarrow Cdr.$$

Ответ. Итоги представления основных функций системы программирования Lisp приведем в виде таблицы соответствия:

№ п/п	Функция Lisp	Объект комбинаторной логики или λ -исчисления
1	<i>Append</i>	B
2	<i>Nil</i>	I
3	<i>Null</i>	$D\bar{0}(K(K\bar{0}))$
4	<i>List</i>	D
5	<i>Car</i>	DcK
6	<i>Cdr</i>	$\lambda xy.xy\bar{0}$

15.3 Заключительные замечания

Решение поставленной задачи, безусловно, выполнено методом погружения. Переформулируем его в терминах объектов.

Система равенств (15.1)–(15.6) в совокупности рассматривается как *соотнесение*, для которого индивидуализируется теория-оболочка. Теория-оболочка представляется собой концепт, а результат индивидуализации будет зависеть от выбора концепта. В качестве оболочки выберем, например, $\eta\xi$ -исчисление λ -конверсий. Тогда каждый из объектов-концептов

Append, *Nil*, *Null*, *List*, *Car*, *Cdr*

в результате выполнения соотнесения даст соответствующий объект-индивид

B , I , $D\bar{0}(K(K\bar{0}))$, D , DcK , $\lambda xy.xy\bar{0}$.

Объекты-индивиды образуют теорию-индивид, которая представляет собой язык Lisp (и является оболочкой). У $\eta\xi$ -исчисления λ -конверсий проявляется одна замечательная особенность: как объекты-концепты, так и объекты-индивиды относительно соотнесения (15.1)–(15.6) остаются в рамках одного и того же универсума.

Глава 16

Суперкомбинаторы

Суперкомбинаторы устанавливают чисто объектную систему программирования, встроенную в комбинаторную логику. Тем самым непосредственно удовлетворяется потребность в денотационном вычислении инструкций языков программирования, когда объектами выражается функциональный смысл программы. Существенно, что вычисление начинается с некоторого заранее известного набора инструкций. В процессе вычисления значения программы динамически возникают заранее неизвестные, но необходимые по ходу дела инструкции, которые дополнительно фиксируются в системе программирования.

16.1 Теоретические сведения

Имеется два подхода к применению суперкомбинаторов для реализации аппликативных языков программирования. При первом из них программа компилируется посредством фиксированного набора суперкомбинаторов (в неоптимизированном варианте — S , K , I) с заранее известными определениями. Сначала остановимся на втором подходе, при котором определения суперкомбинаторов генерируются самой программой в процессе компиляции.

16.1.1 Понятие о суперкомбинаторе

Определение 16.1. Суперкомбинатор $\$S$ арности n представляет собой лямбда-выражение

$$\$S \stackrel{\text{def}}{=} \lambda x_1. \lambda x_2. \dots \lambda x_n. E,$$

или, что эквивалентно, абстракцию вида:

$$\$S \stackrel{\text{def}}{=} [x_1]. [x_2]. \dots [x_n]. E,$$

где E не является абстракцией. Таким образом, все “ведущие” символы абстракции $[\cdot]$ относятся только к x_1, x_2, \dots, x_n , при этом выполняются условия:

- (1) $\$S$ не содержит свободных переменных;
- (2) каждая абстракция в E является суперкомбинатором;
- (3) $n \geq 0$, то есть наличие символов $[\cdot]$ не обязательно.

Суперкомбинаторный редекс — это аппликация суперкомбинатора к n аргументам, где n — его арность. Подстановка аргументов в тело суперкомбинатора вместо свободных вхождений соответствующих формальных параметров называется *редукцией* суперкомбинатора.

Можно вспомнить обычное определение комбинатора и произвести сравнение. Другими словами, можно сказать, что комбинатор — это такая лямбда-абстракция, которая не содержит вхождений свободных переменных. Некоторые лямбда-выражения являются комбинаторами, а некоторые комбинаторы являются суперкомбинаторами.

Пример 16.1. Выражения

$$3, \quad + \ 2 \ 5, \quad [x].x, \quad [x].+ \ x \ x, \quad [x].[y].xy$$

представляют собой суперкомбинаторы.

Пример 16.2. Приводимые термы не являются суперкомбинаторами:

$$\begin{aligned} [x].y & - \text{(переменная } y \text{ входит свободно),} \\ [y].-y\ x & - \text{(переменная } x \text{ входит свободно).} \end{aligned}$$

Пример 16.3. Терм $[f].f([x].f\ x\ 2)$ является комбинатором, так как все переменные (f и x) связаны, но не являются суперкомбинатором, поскольку во внутренней абстракции переменная f свободна и нарушается пункт (2) определения 16.1. В соответствии с этим определением комбинаторы S , K , I , B , C являются суперкомбинаторами. Следовательно, представленная в теории категориальных вычислений SK -машина реализует один из методов использования суперкомбинаторов.

Суперкомбинаторы арности 0 считаются *константными аппликативными формами* (КАФ).

Пример 16.4. Выражения:

$$a) 3, \quad б) +\ 4\ 6, \quad в) +\ 2$$

являются КАФ.

Пункт в) показывает, что КАФ может быть функцией, хотя и не содержит абстракций. Поскольку в КАФ нет символа абстракции, для них код не компилируется.

Упражнение 16.1. Показать, что следующие выражения являются суперкомбинаторами:

$$1\ 0, \quad [x].+\ x\ 1, \quad [f].([x].+\ x\ x).$$

Упражнение 16.2. Объясните, почему следующие выражения не являются суперкомбинаторами:

$$[x].x\ y\ z,\quad [x].[y].+ (+\ x\ y)\ z.$$

Упражнение 16.3. Привести пример комбинатора, не являющегося суперкомбинатором.

16.1.2 Процесс компиляции

Реальные программы содержат значительное число абстракций. Программу следует преобразовать таким образом, чтобы она содержала только суперкомбинаторы. Согласимся с тем, что имена суперкомбинаторов будут начинаться с символа ‘\$’, например:

$$\$XY = [x].[y].- \ y\ x.$$

Для того, чтобы подчеркнуть особенности суперкомбинаторов, перепишем это определение в виде:

$$\$XY\ x\ y = -\ y\ x.$$

Избираемая стратегия заключается в преобразовании абстракции, которую следует откомпилировать, в:

- (i) совокупность суперкомбинаторных определений,
- (ii) вычисляемое выражение.

Будем изображать это посредством:

Определения суперкомбинаторов
.....
.....

Вычисляемое выражение

Пример 16.5. Выражение $([x].[y].- \ y\ x)^3\ 4$ представимо в виде:

$$\begin{array}{c} \$XY\ x\ y = -\ y\ x \\ \hline \$XY\ 3\ 4 \end{array}$$

Пример 16.6. Выражение $(\$XY\ 3)$ не является редексом и не может быть вычислено. Таким образом, определения суперкомбинаторов задаются в виде набора правил перезаписи. Редукция заключается в перезаписи выражения, которое совпадает с левой частью правила, заменяя его на выражение, стоящее в правой части. Такие системы считаются системами перезаписи термов.

Упражнение 16.4. Можно ли вычислить выражения:

$$\$XY\ 5, \quad \$XY\ 5\ 7, \quad \$XY\ 3\ 4\ 7?$$

16.1.3 Приведение к суперкомбинаторам

Суперкомбинаторы легко компилируются. Дадим описание *алгоритма* преобразования абстракций в комбинаторы. Рассмотрим программу, не содержащую ни одного суперкомбинатора:

$$([x] . ([y] . +\ y\ x)\ x)\ 4.$$

Выберем самую внутреннюю абстракцию, то есть такую абстракцию, которая не содержит других абстракций:

$$([y] . +\ y\ x).$$

В нее входит свободная переменная x , поэтому эта абстракция не является суперкомбинатором.

- (1) С помощью простого преобразования (обычной β -редукции) получим суперкомбинатор

$$([x] . [y] . +\ y\ x)x.$$

- (2) Подставим его в исходную программу:

$$([x] . ([w] . [y] . +\ y\ w)\ x\ x)4.$$

- (3) Присвоим суперкомбинатору имя $\$Y$.
- (4) Теперь видно, что $[x]$ -абстракция тоже является суперкомбинатором. Дадим ему имя $\$X$ (скомпилируем абстракцию) и сопоставим его скомпилированному коду:

$$\begin{array}{r}
 \$Y \ w \ y = + \ y \ w \\
 \$X \ x = \$Y \ x \ x \\
 \hline
 \$X \ 4
 \end{array}$$

Можно выполнить полученную программу, осуществляя редукцию суперкомбинаторов:

$$\$X \ 4 \ \rightarrow \$Y \ 4 \ 4 = + \ 4 \ 4 = 8.$$

Таким образом, алгоритм преобразования абстракций в суперкомбинаторы приобретает следующий вид:

ЦИКЛ-*while*: есть абстракции?

- (1) выбрать любую абстракцию, без других абстракций,
- (2) вынести все свободные в этой абстракции переменные в качестве экстрапараметров,
- (3) абстракции приписать некоторое имя (например, $\$X34$),
- (4) заменить вхождение абстракции в программу на имя суперкомбинатора, которое приложено к свободным переменным,
- (5) произвести компиляцию абстракции и скомпилированному коду сопоставить имя.

КОНЕЦ-*while*

В ходе преобразования объем программы возрос. Это является

своеобразной платой за простоту правил редукции. Заметим, что процедура преобразования приводит исходную программу к виду:

... определения суперкомбинаторов ...

E

Поскольку выражение **E** является вычисляемым выражением самого высокого уровня, то оно не содержит свободных переменных. Его можно считать суперкомбинатором арности 0, то есть КАФ:

... определения суперкомбинаторов ...

\$Prog = E

\$Prog

Процесс преобразования абстракций в суперкомбинаторы называется *ламбда-подъемом*, поскольку все абстракции поднимаются на верхний уровень.

Упражнение 16.5. Преобразовать и выполнить программы:

- 1) $([x] . ([y] . - \ y \ x) \ x) \ 5$,
- 2) $([z] . + \ z(([x] . ([y] . \times \ y \ x) \ x) \ 4)) \ 2$.

16.1.4 Устранение избыточных параметров

Рассмотрим простую оптимизацию алгоритма ламбда-подъема. Пусть имеется выражение:

$[x] . [y] . - \ y \ x$.

Хотя оно и является суперкомбинатором, применим к нему алгоритм ламбда-подъема.

- (1) Выберем самую внутреннюю абстракцию $[y] . - \ y \ x$. Переменная x входит в нее свободно. Вынесем ее в качестве экстрапараметра: $([x] . [y] . - \ y \ x) \ x$. Положим:

$$\$Y = [x] . [y] . - y x.$$

Таким образом:

$$\begin{array}{r} \$Y \ x \ y = - y \ x \\ \hline [x] . \$Y \ x \end{array}$$

(2) Пусть $\$X = [x] . \$Y \ x$. Тогда имеем:

$$\begin{array}{r} \$Y \ x \ y = - y \ x \\ \$X \ x = \$Y \ x \\ \hline \$X \end{array}$$

(3) В данном случае определение $\$X$ упрощается до $\$X = \Y (в силу η -редукции).

Таким образом, суперкомбинатор $\$X$ является избыточным и может быть заменен на $\$Y$:

$$\begin{array}{r} \$Y \ x \ y = - y \ x \\ \hline \$Y \end{array}$$

В результате оказывается, что имеется две оптимизации:

- 1) устранение избыточных параметров из определений с помощью η -редукции,
- 2) удаление избыточных определений.

16.1.5 Упорядочивание параметров

Во всех рассмотренных выше программах порядок вынесения переменных как экстрапараметров был произвольным. Например, рассмотрим программу:

$$\begin{aligned} & \dots\dots\dots \\ & \quad (\dots \\ & ([x] . [z] . + y (\times x z)) \\ & \quad \dots), \end{aligned}$$

где '...' указывает на объемлющий $[x]$.-абстракцию контекст. Начнем выполнять алгоритм лямбда-подъема.

(1) Выберем самую внутреннюю абстракцию

$$[z] . + y (\times x z) .$$

Эта абстракция не является суперкомбинатором, так как содержит две свободные переменные x и y . На следующем шаге алгоритма следует вынести свободные переменные в качестве экстрапараметров.

Возникает вопрос, в каком порядке располагать переменные: можно сначала поместить x , а затем y , а можно — сначала y , затем — x . Выполним оба варианта.

Вариант 1.

(2) Вынесем переменные, расположив их в порядке x , y :

$$([x] . [y] . [z] . + y (\times x z)) x y .$$

(3) Присвоим полученному суперкомбинатору имя:

$$\$S = ([x] . [y] . [z] . + y (\times x z)) .$$

(4) Подставим $\$S$ в исходную программу:

$$\begin{array}{c} \$S \ x \ y \ z = + \ y(\times \ x \ z) \\ \hline \dots \\ ([x].\$S \ x \ y) \\ \dots \end{array}$$

Выражение $[x].\$S \ x \ y$ не является суперкомбинатором, поэтому к нему, в свою очередь, следует применить алгоритм лямбда-подъема.

(2) Вынесем свободную переменную y :

$$([y]. [x].\$S \ x \ y)y.$$

(3) Присвоим суперкомбинатору имя $\$T \ y \ x = \$S \ x \ y$.

(4) Подставим комбинатор в программу:

$$\begin{array}{c} \$T \ y \ x = \$S \ x \ y \\ \hline \$T \ y. \end{array}$$

Вернемся к основному алгоритму, в котором теперь получаем:

$$\begin{array}{l} \$S \ x \ y \ z = + \ y(\times \ x \ z) \\ \$T \ y \ x = \$S \ x \ y \\ (\dots \ \$T \ y \ \dots). \end{array}$$

Вариант 2.

(2) Вынесем переменные, расположив их в порядке y , x :

$$([y] . [x] . [z] . + y(\times x z)) y x.$$

(3) Присвоим полученному суперкомбинатору имя:

$$\$S = ([y] . [x] . [z] . + y(\times x z)).$$

(4) Подставим $\$S$ в исходную программу:

$$\begin{array}{c} \$S y x z = + y(\times x z) \\ \hline (\dots \\ ([x] . \$S y x) \\ \dots) \end{array}$$

Выражение $[x] . \$S y x$ не является суперкомбинатором, поскольку содержит свободную переменную y . Применим алгоритм подъема к $[x] . \$S y x$.

(1) Самая внутренняя абстракция совпадает с программой.

(2) Вынесем переменную y : $([y] . [x] . \$S y x)y$.

(3) Присвоим суперкомбинатору имя $\$T = [y] . [x] . \$S y x$.

(4) Подставим комбинатор в программу:

$$\begin{array}{c} \$T y x = \$S y x \\ \hline \$T y \end{array}$$

Вернемся к основному алгоритму. Получим:

$$\begin{array}{c} \$S x y z = + y (\times x z) \\ \$T y x = \$S y x \\ \hline (\dots \$T y \dots) \end{array}$$

В соответствии с правилом устранения избыточных параметров (см. подраздел 16.1.4) получаем: $\$T = \S , поэтому $\$T$ можно устранить. Тогда скомпилированный код примет вид:

$$\begin{array}{c} \$S \ y \ x \ z = + \ y(\times x \ z) \\ \hline \$S \ y \end{array}$$

В первом варианте такая оптимизация невозможна. В исходной программе

$$(\dots ([x] \cdot [z] \cdot + \ y(\times x \ z)) \dots)$$

имеются две связанные переменные: x и z . Во втором варианте произведено упорядочивание свободных переменных на шаге (2) так, что в полученном суперкомбинаторе связанные в программе переменные x и z стоят последними: $\$S \ y \ x \ z$. Только в этом случае код можно оптимизировать. Таким образом, свободные переменные необходимо упорядочивать так, чтобы связанные переменные оказались последними в списке параметров суперкомбинатора.

С каждой абстракцией связывается лексический номер уровня, который определяется числом объемлющих ее символов абстракции.

Пример 16.7. В выражении

$$([x] \cdot [y] \cdot + \ x(\times y \ y))$$

оператор $[x] \cdot$ -абстракции находится на уровне 1, а $[y] \cdot$ -абстракция — на уровне 2.

Сформулируем правила, позволяющие устанавливать лексический номер уровня:

- 1) лексический номер абстракции на единицу больше числа объемлющих ее абстракций; если таких абстракций нет, то номер равен 1;

- 2) лексический номер переменной — это номер абстракции, связывающей данную переменную; если номер x меньше номера y , то говорят, что x свободнее y ;
- 3) лексический номер константы равен 0.

Для того, чтобы повысить возможность оптимизаций, экстрапараметры следует отсортировать по возрастанию их лексических номеров.

16.1.6 Ламбда-подъем при рекурсии

Заметим, что ламбда-абстракции, как правило, не имеют имен. В отличие от них суперкомбинаторы имеют имена. Помимо этого суперкомбинаторы могут ссылаться сами на себя. Это означает, что рекурсивные суперкомбинаторы реализуются непосредственно, без привлечения комбинатора неподвижной точки Y . Конечно, рекурсивные определения можно преобразовать в нерекурсивные, воспользовавшись Y , но это потребует введения в употребление дополнительных правил.

Пример 16.8. Для того, чтобы $\$F$ стал нерекурсивным, следует ввести определения:

$$\left. \begin{aligned} \$F &= Y \ \$F1 \\ \$F1 \ E \ x &= \$G \ (F(- \ x \ 1)) \ 0. \end{aligned} \right\} \quad (*)$$

Дополнительное определение помечено символом '*'. Поскольку Y необходимо редуцировать, то такое определение $\$F$ потребует больше редукций, чем рекурсивная версия.

Обозначение:

$$\begin{aligned} \$S1 \ x \ y &= B1 \\ \$S2 \ f &= B2 \\ &\dots \\ E \end{aligned}$$

эквивалентно выражению:

```
letrec
  $S1 = [x] . [y] . B1
  $S2 = [f] . B2
  ...
in
  E.
```

Оно означает, что в E входят $\$S1$, $\$S2$, ..., рекурсивные определения которых приведены в `letrec`. Алгоритм лямбда-подъема работает точно так же, как и ранее: выражения, встречающиеся в `letrec`, понимаются так же, как и любые другие выражения. Тем не менее возникает вопрос, какой лексический номер уровня следует приписать переменным, связанным в `letrec`. Поскольку такие переменные означиваются, когда непосредственно объемлющая абстракция прикладывается к аргументу, то их лексический номер совпадает с номером данной абстракции. Если же объемлющей абстракции нет, то лексический номер равен 0. Такой номер приписывается константам и суперкомбинаторам. Внутри `letrec`, у которого нет абстракций, не может быть никаких свободных переменных, кроме тех переменных, которые уже определены в `letrec`. Такой `letrec` является комбинатором. Для того, чтобы превратить его в суперкомбинатор, необходимо выполнить лямбда-подъем, устраняющий все внутренние абстракции. Переменные, связанные в `letrec` уровня 0, не будут выноситься как экстрапараметры, поскольку константы (напомним, что они имеют уровень 0) не выносятся.

Пример 16.9. Приведем программу, дающую бесконечный список единиц:

```
-----
letrec x = cons 1 x
in x
```

В этой программе `letrec` находится на уровне 0, и абстракций нет, поэтому `x` уже является суперкомбинатором:

```

      $x = cons 1 x
      -----
      x

```

Пример 16.10. Рассмотрим рекурсивную функцию вычисления факториала:

```

-----
letrec fac = [n].IF (= n 0) 1
                  (× n (fac(- n 1)))
in fac 4

```

В данном случае `letrec` имеет номер 0 и внутри `[n]` -абстракций нет. Следовательно, `fac` является суперкомбинатором:

```

$fac n = IF (= n 0) 1 (× n(fac(- n 1)))
$Prog = $fac 4
-----
$Prog

```

Упражнение 16.6. Скомпилировать программу:

```

-----
let
  inf = [v].(letrec vs = cons v vs in vs)
in
  inf 4

```

Указание: `let` означает, что в выражении `inf 4` функция `inf` имеет определение, указанное в `let`. Функция `inf v` возвращает бесконечный список символов `v`.

16.1.7 Работа алгоритма лямбда-подъема

Рассмотрим программу, суммирующую первые 100 целых чисел:

```
SumInts m = sum (count 1)
           where
             count n = [], n > m
                   = n:count(n + 1)

sum [] = 0
sum (n:ns) = n + sum ns

-----
SumInts 100
```

`SumInts` представляет собой композицию функций `sum` и `count`: сначала функция `count` применяется к 1, а затем ее результат поступает на вход функции `sum`. Функция `count` работает следующим образом:

```
count 1 = 1:count 2 (так как n = 1, m = 100,
                   и условие n > m не выполняется)
        = 1:2:(count 3) (вычисляется count 2)
...
        = 1:2: ... :100:(count 101)
        = 1:2: ... :100:[] (поскольку
                           n = 101, m = 100,
                           то n > m, (count 101 = []))
```

Теперь список `1:2:3: ... :100:[]` передается функции `sum`. Во втором определяющем равенстве для `sum` аргумент `(n:ns)` обозначает список, в котором первым элементом является число `n`, а “хвостом” — список чисел `ns`:

```
sum 1:2:3: ... :100:[] = 1 + sum 2:3: ... :100:[]
                        ...
                        = 1 + 2 + 3 + ... + 100 + sum[]
                        = 1 + 2 + 3 + ... + 100 + 0 (так как sum [] = 0)
```

Запишем эту функцию в терминах абстракций с использованием `letrec`:

```
letrec
  SumInts
    [m].letrec
      count = [n].IF (> n m) NIL
                      (cons n (count (+ n 1) ))
      in sum (count 1)
    sum = [ns].IF (= ns NIL) 0
              (+ (head ns)(sum (tail ns)) )
in SumInts 100
```

В данном случае: `NIL` — пустой список `[]`, `head` — функция, возвращающая первый элемент списка, `tail` — функция, возвращающая хвост списка (список без первого элемента). Переменные `SumInts` и `sum` имеют номер 0, однако `SumInts` содержит внутреннюю `[n]`.-абстракцию со свободными переменными `m` и `count`. Необходимо выполнить алгоритм лямбда-подъема и “поднять” эти переменные.

(1) Внутренняя абстракция имеет вид:

```
[n].IF (> n m) NIL (cons n (count(+ n 1) ))
```

(2) Вынесем переменные `count` и `m` в указанном порядке (так как связанная в исходной программе переменная `m` должна находиться на последнем месте):

```
([count]. [m]. [n].IF (> n m) NIL
                      (cons n (count (+ n 1)))) count m
```

(3) Полученному суперкомбинатору присвоим имя `$count`:

```
$count count m n = IF (> n m) NIL
                      (cons n (count (+ n 1)))
```

(4) Заменим вхождение `[n]`.-абстракции в программу на конструкцию `$count count m n`:

```

$count count m n = IF (> n m) NIL
                      (cons n (count (+ n 1)))
-----
letrec
  SumInts
    = [m].letrec
      count = $count count m
      in sum (count 1)
  sum = [ns].IF (= ns NIL) 0
              (+ (head ns)(sum (tail ns)))
in SumInts 100

```

(5) В выражениях `SumInts` и `sum` нет внутренних абстракций, их уровень равен 0, поэтому они являются суперкомбинаторами. Непосредственно применяя к ним подъем и добавляя суперкомбинатор `$Prog`, получим окончательный результат:

```

$count count m n = IF (> n m) NIL
                      (cons n(count (+ n 1)))
$sum ns = IF (= ns NIL) 0 (+ (head ns)
                             ($sum (tail ns)))
$SumInts m = letrec count = $count count m
              in $sum (count 1)
$Prog = $SumInts 100
-----
$Prog

```

Упражнение 16.7. Скомпилировать программу, которая прикладывает функцию `f = КВАДРАТ` к каждому элементу списка целых чисел от 1 до 5:

```

-----
apply m = fold КВАДРАТ (constr 1)
          where constr n = [], n > m
                      = n:constr(n + 1)

fold f [] = []
fold f (n:ns) = fn:fold f ns

```

16.1.8 Другие способы лямбда-подъема

Представленная в предыдущих подразделах методика не является единственным способом лямбда-подъема рекурсивных функций. Существует алгоритм, который строит суперкомбинаторы для структур данных, а не для функций. Этот алгоритм работает следующим образом. Пусть имеется программа, содержащая рекурсивную функцию f со свободной переменной v :

```

-----
( ...
  letrec f = [x].( ... f ... v ... )
  in ( ... f ... )
... )

```

По f строится рекурсивный суперкомбинатор $\$f$, однако при этом абстрагируется не сама функция f , а переменная v : все вхождения v замещаются на $\$f v$. В результате замещения получаем:

```

-----
$f v x = ... ($f v) ... v ...
( ...
  ( ... ($f v) ... )
  ... )

```

Посмотрим, как работает данный алгоритм на примере из подраздела 16.1.6. Исходная программа имеет вид:


```

-----
letrec
  SumInts
    [m].letrec
      count = [n].IF (> n m) NIL
                    (cons n (count (+ n 1)) )
      in sum (count 1)
    sum
      = [ns].IF (= ns NIL) 0
               (+ (head ns) (sum (tail ns)) )
  in SumInts 100

```

Поднимем `[n]`.-абстракцию, абстрагируя свободную переменную `m`, но не `count`, и заменим все вхождения `count` на `($count m)`:

```

-----
$count m n = IF (> n m) NIL
              (cons n ($count m (+ n 1)))

letrec
  SumInts = [m].sum($count m 1)
  sum = [ns].IF (= ns NIL) 0
               (+ (head ns) (sum (tail ns)))
  in
    SumInts 100

```

В исходной программе имеется два обращения к `count`: в `[n]`.-абстракции и в определении `SumInts`; оба эти обращения заменены на `($count m)`. Теперь видно, что и `SumInts`, и `sum` являются суперкомбинаторами, поэтому можно выполнить их лямбда-подъем:

```

$count m n = IF (> n m) NIL
                (cons n ($count m (+ n 1)))
$sum ns = IF (= ns NIL) 0
            (+ (head ns) ($sum (tail ns)))
$SumInts m = sum ($count m 1)
$Prog = $SumInts 100
-----
$Prog

```

Основное преимущество этого метода по отношению к предыдущему заключается в следующем. В примере из подраздела 16.1.6 рекурсивное обращение к `$count` в суперкомбинаторе `$count` осуществлялось через его параметр, названный *count*. В новом методе выполняется вызов самого суперкомбинатора `$count` непосредственно. Построенный на этом методе компилятор работает более эффективно.

Упражнение 16.8. Выполнить предложенным методом компиляцию программы из упражнения 16.7.

16.1.9 Полная ленивость

Рассмотрим функцию `f = [y].+ y (sqrt 4)`, где `sqrt` — функция извлечения квадратного корня. Каждый раз, когда эта функция применяется к аргументу, подвыражение `(sqrt 4)` приходится вычислять заново. Однако независимо от значения аргумента `y` выражение `(sqrt 4)` редуцируется к 2. Следовательно, хотелось бы не выполнять повторных вычислений подобных константных выражений, а, вычислив его единственный раз, использовать сохраненный результат.

Пример 16.11. Рассмотрим программу:

```

-----
f = g 4
g x y = y + (sqrt x)
(f 1) + (f 2)

```

Запись в виде лямбда-выражения дает:

```
-----
letrec f = g 4
      g = [x].[y].+ y (sqrt x)
      in + (f 1) (f 2)
```

При вычислении этого выражения получаем следующий результат:

```
+ (f 1) (f 2) -->
--> + (. 1) (. 2)
      .-----> (([x].[y].+ y (sqrt x)) 4)
--> + (. 1)(. 2)
      .-----> ([y].+ y (sqrt 4))
--> + (. 1)(+ 2 (sqrt 4))
      .-----> ([y].+ y (sqrt 4))
--> + (. 1)4
      .-----> ([y].+ y (sqrt 4))
--> + (+ 1 (sqrt 4))4
--> + (+ 1 2)4
--> + 3 4
--> 7
```

В этом примере подвыражение `(sqrt 4)` вычисляется дважды, при каждом применении выражение `[y].(sqrt 4)` считается динамически создаваемым константным подвыражением `[y].-абстракции`. Точно такой же эффект наблюдается и при переходе к суперкомбинаторам. Рассмотренное выражение компилируется следующим образом:

```
$g x y = + y (sqrt x)
$f = $g 4
$Prog = + ($f 1)($f 2)
-----
$Prog
```

Редукция выглядит следующим образом:

```

$Prog --> + (. 1)(. 2)
          .----.----> ($g 4)
--> + (. 1)(+ 2 (sqrt 4))
          .----> ($g 4)
--> + (. 1) 4
          .----> ($g 4)
--> + (+ 1 (sqrt 4))4
--> + (+ 1 2)4
--> + 3 4
      7

```

И в этом случае подвыражение `(sqrt 4)` вычисляется дважды.

После выписывания этих примеров, носящих наводящий характер, сформулируем следующую основную проблему, для преодоления которой потребуются определенные усилия:

после связывания всех его переменных каждое выражение должно вычисляться максимум один раз.

Такое свойство вычисления выражений считается *полной ленивостью* вычислений.

Упражнение 16.9. Пусть имеется программа:

```

f = g 2
g x y = y * (КВАДРАТ x)
(f 3) * (f 1)

```

а) Записать программу в виде абстракции и произвести вычисления.

б) Скомпилировать программу и произвести вычисления.

16.1.10 Максимально свободные выражения

Для достижения полной ленивости нет необходимости производить вычисления тех выражений, которые не содержат (свободных) вхождений формального параметра.

Определение 16.2. Выражение E называется *собственным* подвыражением F , если и только если E является подвыражением F и E не совпадает с F .

Определение 16.3. Подвыражение E из абстракции считается *свободным* в лямбда-абстракции L , если все переменные E свободны в L .

Определение 16.4. . *Максимально свободное выражение*, или МСВ, в L — это такое свободное выражение, которое не является собственным подвыражением другого свободного выражения в L .

Пример 16.12. В приводимых ниже абстракциях подчеркнуты МСВ:

- (1) $([x].\underline{\text{sqrt } x}),$
- (2) $([x].x(\underline{\text{sqrt } 4}))$
- (3) $([y].[x].\underline{+(* y y) x}).$

Для обеспечения полной ленивости при выполнении β -редукции не следует означивать максимально свободные абстракции.

Пример 16.13. Вернемся к функции из примера 16.11:

```
-----
letrec f = g 4
      g = [x].[y].+ y (sqrt x)
      in + (f 1) (f 2)
```

Последовательность редукций начинается, как в этом примере:

```

+ (f 1)(f 2) -->
--> + (. 1)(. 2)
      .-----> (([x].[y].+ y (sqrt x)) 4)
--> + (. 1)(. 2)
      .-----> ([y].+ y (sqrt 4))

```

(здесь: выражение `(sqrt 4)` является МСВ в `[y].-`абстракции, поэтому при приложении `[y].-`абстракции к аргументу `(sqrt 4)` не следует вычислять:

```

--> + (. 1)(+ 2 .)
      .-----> ([y].+ y (sqrt 4)) )

```

Означивание имеет указатель на `(sqrt 4)` в теле абстракции:

```

+ (. 1)(+ 2 .)
      .----->([y].+ y 2)
--> + (. 1)4
      .----> ([y].+ y 2)
--> + (+ 1 2)4
--> + 3 4
7

```

В этом случае `(sqrt 4)` вычисляется только один раз.

Упражнение 16.10. Вычислить выражение из упражнения 16.9, воспользовавшись МСВ.

16.1.11 Ламбда-подъем с использованием МСВ

Алгоритм, использующий МСВ, отличается от приведенного ранее алгоритма ламбда-подъема тем, что абстрагирует не переменные, а МСВ. Вернемся к разбираемому примеру. Функция `g` содержит абстракцию:

$$[x].[y].+ y (\text{sqrt } x)$$

Проиллюстрируем в этой ситуации работу алгоритма.

- (1) Самой внутренней абстракцией является

$$[y].+ y (\text{sqrt } x).$$

- (2) Выражение $(\text{sqrt } x)$ является МСВ. Вынесем МСВ в качестве экстрапараметра:

$$([\text{sqrtx}].[y].+ y (\text{sqrtx}))(\text{sqrt } x),$$

где sqrtx является именем экстрапараметра. Подставим полученное выражение в исходную абстракцию:

$$[x].([\text{sqrtx}].[y].+ y \text{sqrtx})(\text{sqrt } x).$$

- (3) Присвоим полученному суперкомбинатору имя:

$$\begin{array}{l} \$g1 = [\text{sqrtx}].[y].+ y \text{sqrtx} \\ \hline [x].\$g1 (\text{sqrt } x) \end{array}$$

- (4) Имеющаяся $[x].$ -абстракция также является суперкомбинатором, которому дадим имя и который сопоставим скопированному коду:

$$\begin{array}{l} \$g1 \text{sqrtx } y = + y \text{sqrtx} \\ \$g \text{ } x = \$g1 (\text{sqrt } x) \\ \$f = \$g \text{ } 4 \\ \$\text{Prog} = + (\$f \text{ } 1)(\$f \text{ } 2) \\ \hline \$\text{Prog} \end{array}$$

Дополнительный суперкомбинатор получен потому, что потеряна возможность использования η -редукции из-за абстрагирования по (`sqrtx`) вместо абстрагирования по `x`. Однако этот эффект компенсируется наличием полной ленивости. Следовательно,

для обеспечения полной ленивости необходимо абстрагировать МСВ, а не свободные переменные, используя возникающие абстракции в качестве экстра-параметров.

Приведенный алгоритм считается полностью ленивым лямбда-подъемом.

Упражнение 16.11. Выполнить полностью ленивый лямбда-подъем программы, рассмотренной в упражнении 16.9.

16.1.12 Полностью ленивый лямбда-подъем с *letrec*

Рассмотрим программу:

```
-----
let f = [x].letrec fac = [n].( ... )
      in + x (fac 100)
      in
      + (f 3)(f 4)
```

Алгоритм из подраздела 16.1.6 скомпилирует ее в:

```
$fac fac n = ( ... )
$f x = letrec fac = $fac fac
      in + x (fac 100)
      + ($f 3)($f 4)
```

Функция `fac` определена локально в теле функции `f` и, следовательно, выражение `(fac 100)` не может быть поднято как МСВ

из тела `f`. Это означает, что `(fac 100)` будет вычисляться заново всякий раз, когда выполняется `$f`. Таким образом, происходит потеря свойства полной ленивости.

Выход из создавшегося положения достаточно прост: достаточно заметить, что определение `fac` не зависит от `x`, поэтому можно вынести `letrec` для `fac`:

```
-----
letrec
  fac = [n].( ... )
  in let
    f = [x].+ x (fac 100)
  in
    + (f 3)(f 4)
```

Теперь ничто не препятствует применению полностью ленивого подъема, который построит полностью ленивую программу:

```
$fac n = ( ... )
$fac100 = $fac 100
$f x = + x $fac100
$Prog = + ($f 3)($f 4)
-----
$Prog
```

В данном случае произведена некоторая оптимизация: выражение, не имеющее свободных переменных, не абстрагируется, а получает имя и становится суперкомбинатором — `$fac100`.

Таким образом, стратегия воплощается в два этапа:

- 1) вынесение `letrec`- (и `let`-) определений как можно “дальше”,
- 2) выполнение полностью ленивого лямбда-подъема.

Упражнение 16.12. Выполнить полностью ленивый лямбда-подъем программы:

```

let
  g = [x].letrec el = [n].[s].(IF (= n 1)(head s)
                                   (el (- n 1)(tail s)))
    in (cons x (el 3 (A,B,C)))
in
  (cons (g R)(g L))

```

(здесь: R и L — константы).

16.1.13 Комплексный пример

Покажем действие полностью ленивого ламбда-подъема на более детализированном примере¹:

```

-----
SumInts n = foldl + 0 (count 1 n)
count n m = [],    n > m
count n m = n:count (n + 1) m
fold op base [] = base
foldl op base (x:xs) = foldl op (op base x) xs

```

Приведенная программа в терминах абстракций записывается следующим образом:

```

-----
letrec
  SumInts = [n].foldl + 0 (count 1 n)
  count = [n].[m].IF (> n m) NIL
                (cons n (count (+ n 1) m))
  foldl = [op].[base].[xs].IF (= xs NIL) base
                (foldl op (op base (head xs))(tail xs))
in SumInts 100

```

¹Дополнительную разработку техники ленивых вычислений с суперкомбинаторами см. в [106]. В этой же работе можно познакомиться с реализацией абстрактной машины, обеспечивающей поддержку всего спектра возможностей.

В этой программе:

- (1) внутренней абстракцией является (`[xs]. ...`),
- (2) максимально свободными выражениями являются выражения (`fold op`), (`op base`) и `base`.

Вынесем их в качестве экстрапараметров `p`, `q` и `base` соответственно:

```
$R1 p q base xs = IF (= xs NIL) base
                  (p (q (head xs)) (tail xs))
-----
letrec
  SumInts = [n].foldl + 0 (count 1 0)
  count = [n].[m].IF (> n m) NIL
                  (cons n (count (+ n 1) m))
  foldl = [op].[base].$R1 (foldl op) (op base) base
in
  SumInts 100
```

В этой программе внутренней абстракцией является '`[base].`'. Максимально свободными выражениями служат выражения (`$R1(foldl op)`) и `op`, которые вынесем как `r` и `op` соответственно:

```
$R1 p q base xs = IF (= xs NIL) base
                  (p (q (head xs))(tail xs))
$R2 r op base = r (op base) base
-----
letrec
  SumInts = [n].foldl + 0 (count 1 n)
  count = [n].[m].IF (> n m) NIL
                  (cons n (count (+ n 1) m))
  foldl = [op].$R2($R1 op)op
in
  SumInts 100
```

(3) все определения `letrec` являются суперкомбинаторами, поскольку выполнен подъем всех внутренних абстракций. С учетом всех замечаний получаем окончательный результат:

```
$SumInts n = $foldlPlus0 ($count1 n)
$foldlPlus0 = $foldl + 0
$count1 = $count 1
$count n m = IF (> n m)NIL(cons n ($count (+ n 1) m))
$foldl op = $R2 ($R1 ($foldl op))op
$Prog = $SumInts 100
$R1 p q base xs = IF (= xs NIL) base
                  (p (q (head xs))(tail xs))
$R2 r op base = r (op base) base
-----
$Prog
```

Завершая рассмотрение, отметим, что в данном случае нельзя устранить параметр `op` из `$foldl`, поскольку он используется дважды в правой части.

16.2 Задача

Задача 16.1. Записать следующее определение исходного языка с помощью суперкомбинаторов:

$$el\ n\ s = if\ n = 1\ then\ (hd\ s) \\ \qquad\qquad\qquad else\ el\ (n - 1)\ (tl\ s).$$

Функция *el* выбирает *n*-й элемент из последовательности *s*.

Решение. В терминах λ -исчисления определение функции принимает вид:

$$el = Y(\lambda el.\lambda n.\lambda s.if\ (= n\ 1)\ (hd\ s)(el\ (- n\ 1)\ (tl\ s))). \quad (el)$$

Напомним алгоритм перевода λ -выражения в аппликативную форму. Возьмем произвольное λ -выражение $\lambda V.E$.

SC—1. Тело исходного выражения преобразуется в аппликативную форму рекурсивным вызовом компилятора. Результатом является: $\lambda V.E'$.

SC—2. Свободные переменные в λ -выражении идентифицируются литерами P, Q, \dots, R , затем λ -выражение снабжается префиксом ' λ ', связывающим все эти переменные. Результатом служит:

$$\lambda P.\lambda Q.\dots\lambda R.\lambda V.E'$$

Результирующее выражение заведомо является комбинатором, поскольку E' — аппликативная форма, все свободные переменные P, Q, \dots, R которой связаны.

SC—3. Назовем возникающий комбинатор *alpha* и сопоставим ему определение (определяющее равенство):

$$\textit{alpha } P Q \dots R V \rightarrow E'.$$

Исходное λ -выражение заменяется формой

$$(\textit{alpha } P R \dots R).$$

Применительно к выражению аппликацию a можно сократить. Выражение связано с V , а каждая свободная переменная принимает свое собственное значение в $'$. Следовательно, аппликативная форма действительно полностью эквивалентна исходному λ -выражению. Теперь определим пошаговую трансляцию.

1) Рассмотрим самое внутреннее λ -выражение:

$$\lambda s.\textit{if } (= n 1) (\textit{hd } s) (\textit{el } (- n 1) (\textit{tl } s)).$$

Его свободными переменными являются n и el , поэтому комбинатор $alpha$ вводится определением:

$$alpha\ n\ el\ s \rightarrow if\ \begin{matrix} (= n\ 1)\ (hd\ s) \\ (el\ (-\ n\ 1)\ (tl\ s)). \end{matrix} \quad (alpha)$$

- 2) Все λ -выражения заменим на $(alpha\ n\ el)$, следовательно,

$$el = Y(\lambda el. \lambda n. alpha\ n\ el).$$

- 3) Повторим шаги 1) и 2) для $\lambda n. alpha\ n\ el$. Введем комбинатор $beta$ определением:

$$beta\ el\ n \rightarrow alpha\ n\ el, \quad (beta)$$

откуда получаем, что

$$el = Y(\lambda el. beta\ el).$$

- 4) Повторим шаги 1) и 2) для $(\lambda el. beta\ el)$. Введем комбинатор $gamma$ определением:

$$gamma\ el \rightarrow beta\ el, \quad (gamma)$$

откуда получаем, что

$$el = Y\ gamma.$$

Ответ. $el = Y\ gamma$.

Контрольные вопросы.

1. Чем вызвано использование суперкомбинаторов для реализации редукции?
2. Что представляет собой язык константных аппликативных форм (КАФ)?

3. Какие специфические свойства комбинаторов S , K , I допускают их непосредственное использование в РГ-машине (машине редукции графа)?

Упражнение 16.13. Записать с помощью суперкомбинаторов выражение:

$$fac\ n = if\ n = 0\ then\ 1\ else\ n \times (fac(n - 1)).$$

16.3 Ответы к упражнениям

16.1

1 0 не имеет свободных переменных (см. Опр. 16.1, п.(1)) и символов абстракции (см. Опр. 16.1, п.(3)); $[x]. +\ x\ 1$ имеет переменную x , которая является связанной (см. Опр. 16.1, п.(1)); $+ \ x\ 1$ не содержит абстракций (см. Опр. 16.1, п.(2)); $[f]. f\ ([x]. +\ x\ x)$ не имеет свободных переменных (см. Опр. 16.1, п.(1)), $[x]. +\ x\ x$ является суперкомбинатором (см. Опр. 16.1, п.(2)).

16.2

$[x]. x\ y\ z$ содержит свободные переменные y и z , нарушается пункт (1) определения 16.1; $[x]. [y]. +\ (+\ x\ y)\ z$ имеет свободную переменную z .

16.3

Например, $[x]. [y]. x\ y\ ([z]. z\ x)$.

16.4

Выражение $\$XY\ 5$ вычислить нельзя, так как оно не является редексом; $\$XY\ 5\ 7$ вычислимо, $\$XY\ 3\ 4\ 7$ вычислимо.

16.5

1) $([x]. ([y]. -\ y\ x)x)5$:

- (1) самой внутренней абстракцией является $[y]. - y x$;
 (2) вынесем x как экстрапараметр $([x]. [y]. - y x)x$ и подставим его в исходную программу $([x]. ([v]. [y]. - y v) x x)5$;
 (3) присвоим суперкомбинатору имя $\$Y$:

$$\begin{array}{l} \$Y v y = - y v \\ \hline ([x]. \$Y x x)5 \end{array}$$

- (4) $[x].$ -абстракция также является суперкомбинатором, которому можно дать имя и который можно сопоставить скомпилированному коду:

$$\begin{array}{l} \$Y v y = - y v \\ \$X x = \$Y x x \\ \hline \$X 5 \end{array}$$

Полученная программа выполняется следующим образом:

$$\$X 5 = \$Y 5 5 = - 5 5 = 0.$$

2) $([z]. + z (([x]. ([y]. * y x) x) 4)2)$:

- (1) внутренняя абстракция: $[y]. * y x$;

- (2) вынесем x как экстрапараметр:

$$\begin{array}{l} ([x]. [y]. * y x) x \\ ([z]. + z (([x]. ([w]. [y]. * y w) x x)4))2; \end{array}$$

- (3)

$$\begin{array}{l} \$Y w y = * y w \\ \hline ([z]. + z ([x]. \$Y x x) 4) 2 \end{array}$$

- (4) $[x].$ -абстракция является суперкомбинатором:


```

$Y w y = * y w
$X x = $Y x x
-----
([z].+ z ($X 4)) 2

```

Теперь видно, что `[z].`-абстракция является суперкомбинатором:

```

$Y w y = * y w
$X x = $Y x x
$Z z = + z ($X 4)
-----
$Z 2

```

Выполнение программы:

```

$Z 2 = + 2 ($X 4) = + 2 ($Y 4 4) =
      = + 2 (* 4 4) = + 2 16 = 18.

```

16.6

inf находится на уровне 0 и не содержит внутренних абстракций. Поэтому *inf* уже является суперкомбинатором:

```

$inf v = letrec vs = cons 0 vs in vs
$Prog = $inf 4
-----
$Prog

```

16.7

Запишем эту программу в терминах абстракций:

```

-----
letrec apply = [m].letrec constr = [n].(IF > n m) NIL
                                     (cons n (constr (+ n 1)))
in fold КВАДРАТ (constr 1)
   fold = [f].[ns].IF (= ns NIL) NIL
              (cons (f (head ns)) (fold (f (tail ns)))) )
in apply 5

```

(1) внутренняя абстракция имеет вид:

```
[n].IF (> n m) NIL (cons n (constr (+ n 1)));
```

(2) вынесем переменные `constr` и `m` в указанном порядке:

```
[constr].[m].[n].IF (> n m) NIL  
      (cons n (constr (+ n 1))) constr m;
```

(3) присвоим полученному суперкомбинатору имя `$constr`:

```
$constr constr m n = IF (> n m) NIL  
      (cons n (constr (+ n 1)));
```

(4) заменим вхождение `[n].`-абстракции в программу на выражение `($constr constr m n)`;

```
$constr constr m n = IF (> n m) NIL  
      (cons n (constr (+ n 1)))
```

```
-----  
letrec  
  apply = [m].letrec  
    constr = $constr constr m  
    in fold КВАДРАТ (constr 1)  
  fold = [f].[ns].IF (= ns NIL) NIL  
    (cons (f (head ns)) (fold f (tail ns))) )  
in apply 5
```

(5) выражения `apply` и `fold` являются суперкомбинаторами:

```
$constr constr m n = IF (> n m) NIL  
      (cons m (constr (+ n 1)))  
$fold f ns = IF (= ns NIL) NIL  
      (cons (f (head ns))  
        (fold (f (tail ns))))  
$apply m = letrec constr = $constr constr m
```

```

      in $fold КВАДРАТ (constr 1)
$Prog = $apply 5

```

```

-----
$Prog

```

16.8

Поднимем `[n]`-абстракцию, абстрагируя переменную `m`, но не `constr`, и заменим все вхождения `constr` на `($constr m)`:

```

$constr m n = IF (> n m) NIL
              (cons n ($constr m (+ n 1)))

```

```

-----
letrec
  apply = [m].fold (КВАДРАТ) ($constr m 1)
  fold = [f].[ns].IF (= ns NIL) NIL
            (cons (f (head ns)) (fold (f (tail ns)))) )
in apply 5

```

В данном случае и `apply`, и `fold` являются суперкомбинаторами:

```

$constr m n = IF (> n m) NIL
              (cons n ($constr m (+ n 1)))
$fold f ns = IF (= ns NIL) NIL (cons
              (f (head ns))
              ($fold f (tail ns)) )
$apply m = $fold КВАДРАТ ($constr m 1)
$Prog = $apply 5

```

```

-----
$Prog

```

16.9

а) Запись в виде абстракции:

```
-----
letrec f = g 2
      g = [x].[y]. * y (КВАДРАТ x) in * (f 3)(f 1)
```

Вычисление выражения:

```
* (f 3)(f 1) -->
--> (. 3)(. 1)
      .----.----> ([x].[y]. * y (КВАДРАТ x))2
--> * (. 3)(. 1)
      .----.----> ([y]. * y (КВАДРАТ 2))
--> * (. 3)(* 1 (КВАДРАТ 2))
      .----> ([y]. * y (КВАДРАТ 2))
--> * (. 3) 4
      .----> ([y]. * y (КВАДРАТ 2))
--> * ( * 3 (КВАДРАТ 2)) 4
--> * 12 4
--> 48.
```

б) Данное выражение компилируется в:

```
$g x y = * y (КВАДРАТ x)
$f = $g 2
$Prog = * ($f 3) ($f 1)
-----
$Prog
```

Последовательность редукций:

```

$Prog --> (* (. 3)(. 1))
          .----.----> ($g 2)
--> * (. 3)(* 1 (КВАДРАТ 2))
          .----> ($g 2)
--> * (. 3) 4
          .----> ($g 2)
--> * (* 3 (КВАДРАТ 2)) 4
--> * (* 3 4) 4
--> * 12 4
--> 48.

```

16.10

```

* (f 3)(f 1) -->
--> * (. 3)(. 1)
          .----.----> (([x].[y].* y (КВАДРАТ x)) 2)
--> * (. 3)(* 1 .)
          .-----.----> ([y].* y (КВАДРАТ 2))
--> * (. 3) 4 (* 1 .)
          .-----.----> ([y].* y 4)
--> * (. 3) 4
          .----> ([y].* y 4)
--> * (* 3 .) 4
          .----> 4
--> * 12 4
--> 48

```

Выражение `--> (КВАДРАТ 2)` вычисляется единственный раз.

16.11

Функция `g` содержит абстракцию:

$$[x].[y]. * y \text{ (КВАДРАТ } x \text{)}$$

Далее:

(1) самой внутренней абстракцией является

$$[y]. * y \text{ (КВАДРАТ } x);$$

(2) (КВАДРАТ x) представляет собой МСВ, которую вынесем в качестве экстрапараметра:

$$([\text{КВАДРАТ} x]. [y]. * y \text{ (КВАДРАТ} x)) \text{ (КВАДРАТ } x)$$

Подставим полученное выражение в абстракцию:

$$[x]. ([\text{КВАДРАТ} x]. [y]. * y \text{ (КВАДРАТ} x)) \text{ (КВАДРАТ } x)$$

(3) присвоим полученному суперкомбинатору имя:

$$\begin{aligned} \$g1 &= [\text{КВАДРАТ} x]. [y]. * y \text{ КВАДРАТ} x \\ &\text{-----} \\ [x]. \$g1 &\text{ (КВАДРАТ } x) \end{aligned}$$

(4) $[x].$ -абстракция также является суперкомбинатором, которому дадим имя $\$g$ и который сопоставим скомпилированному коду:

$$\begin{aligned} \$g1 \text{ КВАДРАТ} x \ y &= * y \text{ КВАДРАТ} x \\ \$g \ x &= \$g1 \text{ (КВАДРАТ } x) \\ \$f &= \$g \ 2 \\ \$Prog &= * (\$f \ 3) (\$f \ 1) \\ &\text{-----} \\ \$Prog & \end{aligned}$$

16.12

Данная программа компилируется в:

$$\begin{aligned} \$el \ e1 \ n \ s &= (IF (= n 1) (head s) (e1 (- n 1) (tail s))) \\ \$g \ x &= letrec \ e1 = \$el \ e1 \\ &\quad \text{in (cons } x \ (e1 \ 3 \ (A,B,C)) \) \\ &\text{-----} \\ &(\text{cons } (\$g \ R) (\$g \ L)) \end{aligned}$$

Поскольку определение $e1$ не зависит от x , можно вынести $letrec$ для $e1$:

```

-----
letrec el = [n].[s].(IF (= n 1)(head s)
                        (el(- n 1)(tail s)))
in let g = [x].cons x (el 3 (A,B,C))
in (cons (g R) (g L))

```

В результате применения лямбда-подъема получим:

```

$el n s = (IF (= n 1)(head s)(el (- n 1)(tail s)))
$el3 (A,B,C) = $el 3 (A, B, C)
$g x = cons x $el3 (A,B,C)
$Prog = cons ($g R)($g L)
-----
$Prog

```

В ходе компилирования программы порождаются новые комбинаторы, параметрами которых являются конкретные МСВ. Другими словами, компилятор выполняет *соотнесение* с заложенным в нем способом вычленения из исходного кода программы объектов и порождает объекты-индивиды. Поскольку порожденные объекты являются комбинаторами, то вполне безопасно добавить их к системе-оболочке в качестве новых инструкций. Все порожденные комбинаторы дают вполне индивидуализированное представление исходной программы: это система объектов.

Возможно, лучше объяснять в терминах соотнесения системы-оболочки (λ -исчисления) с совокупностью МСВ. Тогда λ -исчисление как *концепт* дает систему *индивидов* (скомпилированных программ): они образуют класс эквивалентности (конвертируются друг к другу) относительно критериев оптимальности.

Глава 17

Ленивая реализация

Когда происходит динамическое формирование объектов “на лету”, то эффективность результирующего кода может теряться из-за необходимости неоднократно вычислять значение одного и того же объекта. Применение механизмов ленивого означивания позволяет этого избежать: если значение объекта уже однократно вычислено, то в дальнейшем используется именно это заранее вычисленное значение.

17.1 Задачи

Задача 17.1. Для примера определения

$$el\ n\ s = \underline{if}\ n = 1\ then\ (hd\ s)\ else\ el\ (n - 1)\ (tl\ s)\ \underline{fi},$$

где функция *el* возвращает *n*-ый элемент из списка (конечной последовательности) *s*, выбрать суперкомбинаторы, дающие полностью ленивую реализацию, и рассмотреть частный случай применения *el* 2.

Формулировка задачи. Введем некоторые определения. Оказывается, что те выражения, которые подвергаются повторным означиваниям, легко идентифицируются. Это относится и ко всякому

подвыражению λ -выражения, которое не зависит от связанной переменной. Такие выражения называют свободными выражениями λ -выражения (по аналогии с введением понятия свободной переменной). Свободные выражения, которые не являются частью никакого другого большего свободного выражения, называют максимальными свободными выражениями λ -выражения (МСВ).

Рассмотрим схему трансляции. Максимальные свободные выражения каждого λ -выражения преобразуются в параметры соответствующего комбинатора. Остановимся на схеме преобразования максимальных свободных выражений в параметры соответствующего комбинатора.

Замечание. Сначала установим, что такая схема трансляции значима, то есть получены настоящие комбинаторы и каждое λ -выражение заменено на аппликативную форму. Рассмотрим применимость новой схемы к отдельному λ -выражению, тело которого является аппликативной формой. Тот комбинатор, который при этом возникает, должен удовлетворять определению, так как его тело должно быть аппликативной формой и не должно содержать свободных переменных. Тело комбинатора заведомо будет аппликативной формой, поскольку оно в свою очередь возникло из аппликативной формы (тела исходного λ -выражения) путем подстановки вместо некоторых выражений новых имен параметров. В нем не может быть свободных переменных, поскольку любая свободная переменная должна быть частью некоторого максимального свободного выражения и поэтому подлежит удалению как часть параметра. Все это подтверждает, что построен настоящий комбинатор. Окончательный результат, который замещает исходное λ -выражение, представляет собой новый комбинатор, приложенный к максимальным свободным выражениям, каждое из которых — уже аппликативная форма.

Решение. Вернемся к исходной задаче.

lazy-1. Пусть максимальные свободные выражения для исходного выражения

$$\lambda s. el\ n\ s = \lambda s. if\ (= \ n\ 1)(hd\ s)(el\ (-\ n\ 1)(tl\ s))$$

представляют собой

$$p \equiv (if(= n 1)) \text{ и } q \equiv (el(- n 1)).$$

Следовательно, новый комбинатор *alpha* определяется посредством

$$alpha\ p\ q\ s \rightarrow p\ (hd\ s)\ (q\ (tl\ s)).$$

Отметим, что в этом случае

$$\lambda s.alpha\ p\ q\ s = alpha\ p\ q \rightarrow \lambda s.p\ (hd\ s)\ (q\ (tl\ s)).$$

Поскольку теперь

$$\begin{aligned} \lambda s.el\ n\ s &= \lambda s.if\ (= n 1)(hd\ s)(el(- n 1)(tl\ s)) \\ &= \lambda s.p\ (hd\ s)\ (q\ (tl\ s)) = alpha\ p\ q = el\ n, \end{aligned}$$

то

$$el\ n = alpha\ \underbrace{(if\ (= n 1))}_p\ \underbrace{(el\ (- n 1))}_q.$$

Отсюда вытекает следующий результат: определением функции *el* служит выражение

$$el = Y\ (\lambda el.\lambda n.alpha\ (if\ (= n 1))\ (el\ (- n 1))).$$

Продолжая этот процесс, получаем дополнительные комбинаторы *beta* и *gamma*, задаваемые определениями:

$$\begin{aligned} beta\ el\ n &\rightarrow \underbrace{alpha\ (if\ (= n 1))\ (el\ (- n 1))}_{el\ n}, \\ gamma\ el &\rightarrow beta\ el, \end{aligned}$$

откуда заключаем, что выражение *el* эквивалентно выражению $(Y\ gamma)$, то есть $el = Y\ gamma$, что совпадает с прежним результатом¹.

¹ Действительно, $gamma\ el = el$, откуда $el = (\lambda f.(gamma\ f))\ el$. По теореме о неподвижной точке (см. теорему 9.1 на стр. 123) $el = Y(\lambda f.(gamma\ f)) = Y\ gamma$.

lazy—2. Рассмотрим теперь частный случай, когда применяется (*el* 2):

$$\begin{aligned}
 el\ 2 &\rightarrow \underbrace{(Y\ gamma)}_{el}\ 2 \\
 &\rightarrow gamma\ el\ 2 \\
 &\rightarrow beta\ el\ 2 \\
 &\rightarrow alpha\ (if\ (= 2\ 1))\ (el\ (-\ 2\ 1)).
 \end{aligned}$$

Всегда при использовании (*el* 2) употребляются одни и те же копии свободных выражений, следовательно, они означаются только один раз. Фактически получается редукция:

$$el\ 2 \rightarrow alpha\ if-FALSE\ (alpha\ if-TRUE\ (el\ (-\ 1\ 1))).$$

Более подробно:

$$\begin{aligned}
 el\ 2 &\rightarrow alpha\ (if\ (= 2\ 1))\ (el\ (-\ 2\ 1)) \\
 &\rightarrow alpha\ (if-FALSE)\ (el\ 1) \\
 &\rightarrow alpha\ (if-FALSE)\ (alpha\ (if\ (= 1\ 1))\ (el\ (-\ 1\ 1))) \\
 &\rightarrow alpha\ (if-FALSE)\ (alpha\ (if-TRUE)\ (el\ (-\ 1\ 1)))
 \end{aligned}$$

Рассмотренная схема приводит к субоптимальным комбинаторам.

Упражнения

Упражнение 17.1. Для выражения

$$fac\ n = if\ n = 0\ then\ 1\ else\ n \times (fac\ (n - 1))$$

осуществить полностью ленивую реализацию с помощью суперкомбинаторов и вычислить *fac* 3.

Глава 18

Перестановка параметров

Применение комбинаторов открывает возможности строить оптимизированный программный код, попутно в ходе синтеза результирующего объекта анализируя порядок возможного замещения формальных параметров на фактические.

18.1 Задача

Задача 18.1. Для исходного определения:

$$el = Y (\lambda el. \lambda n \lambda s. IF (= n 1) (hd s) (el (- n 1) (tl s)))$$

получить выражение с оптимальным (по двум критериям) упорядочением параметров комбинатора.

Формулировка задачи. Напомним, что двумя основными критериями оптимальности порядка параметров комбинатора являются минимальное число МСВ и максимальное устранение избыточных параметров. Оптимизируем наше определение последовательно по обоим критериям.

Замечание. Для максимизации размера и минимизации числа МСВ следующего вложенного λ -выражения все МСВ подвергнутого компиляции λ -выражения, которые в то же самое время являются свободными выражениями следующего вложенного λ -выражения, должны появиться прежде МСВ, не обладающих указанным свойством. Оптимальное упорядочение параметров по этому критерию можно сформулировать следующим образом. Все E_i являются свободными выражениями λ -выражения, которое подлежит компилированию, однако оно может быть свободным выражением одного или большего числа вложенных λ -выражений. Назовем самое внутреннее λ -выражение, в котором выражение E_i не является свободным, *порождающим* λ -выражением. Если порождающее λ -выражение E_i включает порождающее λ -выражение E_j , то в оптимальном упорядочении E_i предшествует E_j . Отсюда не следует, что с необходимостью однозначно определяется оптимальное упорядочение, поскольку выражения с одним и тем же порождающим λ -выражением могут входить в любом порядке. Тем не менее всякое упорядочение, удовлетворяющее этому условию, является оптимальным, как и всякое другое.

Решение.

opt—1. Прежде всего примем во внимание результаты решения задачи 17.1 на стр. 213.

opt—2. Рассмотрим аппликативную форму

$$(\alpha (hd\ s)\ n\ (tl\ s)),$$

в которой параметры α можно расположить в любом порядке. Если непосредственно вложенное λ -выражение связывает n , то максимальные свободные выражения (МСВ) из формы представляют собой

$$(\alpha (hd\ s))\ и\ (tl\ s).$$

Если же выполнить переупорядочение формы вида

$$(\alpha (hd\ s) (tl\ s)\ n),$$

то МСВ единственно:

$$(\alpha (hd\ s) (tl\ s)).$$

opt—3. Если, с другой стороны, непосредственно вложенное λ -выражение связывает s , то оптимальным упорядочением параметров служит

$$(\alpha\ n\ (hd\ s) (tl\ s)),$$

откуда единственное МСВ представляет собой $(\alpha\ n)$.

opt—4. Получив оптимальное упорядочение по одному критерию, обратимся к другому. В качестве “естественных” параметров можно принять

$$\alpha, \beta, \epsilon, hd, tl.$$

На компилятор возлагается задание так расположить параметры, чтобы происходило максимальное устранение избыточных параметров. У компилятора только тогда есть выбор, когда один комбинатор прямо определяется как вызов другого. В качестве примера рассмотрим редукцию:

$$\beta\ p\ q\ r\ s \rightarrow \alpha\ \dots\ s\ \dots$$

Кроме последнего параметра, других избыточных параметров нет, но последний параметр комбинатора всегда должен быть связанной переменной того λ -выражения, из которого был осуществлен вывод. В данном случае параметр s является связанной переменной λ -выражения, непосредственно

включающего *alpha*. Если параметры уже упорядочены оптимально по рассмотренным правилам, то все параметры, в которых участвует *s*, перемещаются в конец его списка параметров. Если имеется только один такой параметр, и это *s*, то *s* оказывается избыточным параметром *beta* и может быть устранен. На этом основании вызов *alpha* следует задавать в виде

$$(\alpha E_1 \dots E_n s),$$

где *s* не имеет вхождений ни в одно из выражений E_i . Это означает, что все E_i свободны в *beta*, что распространяется и на $(\alpha E_1 \dots E_n s)$. Следовательно, фактически, если имеются какие-либо E_i , то *beta* надо определить посредством редукции

$$\beta p s \rightarrow p s,$$

где *p* соответствует $(\alpha E_1 \dots E_n s)$.

Если у *alpha* единственный параметр *s*, то *beta* следует определять посредством

$$\beta s \rightarrow s.$$

В первом случае *beta* эквивалентен комбинатору *I*. Порождающее *beta* λ -выражение замещается на аппликацию

$$(\beta (\alpha E_1 \dots E_n s)),$$

то есть на $(I (\alpha E_1 \dots E_n s))$. Кроме того, *beta* можно целиком опустить, и λ -выражение замещается непосредственно на $(\alpha E_1 \dots E_n s)$.

Во втором случае *beta* эквивалентно *alpha*. Можно видеть, что оптимальное упорядочение, получаемое по второму критерию, удовлетворяет также первому и, кроме того, существенно упрощает работу по нахождению избыточных параметров.

opt–5. Рассматриваемое выражение *el* определяется равенством:

$$el = Y(\lambda el. \lambda n. \lambda s. IF (= n 1) (hd s) (el(- n 1) (tl s))).$$

У самого внутреннего λ -выражения имеются два МСВ:

$$(IF (= n 1)) \text{ и } (el(- n 1)).$$

Примем их за параметры *p* и *q*. Оба этих МСВ имеют одно и то же порождающее λ -выражение, поэтому их порядок несущественен, и *alpha* можно определить редукцией:

$$alpha\ p\ q\ s \rightarrow p\ (hd\ s)\ (q\ (tl\ s)),$$

поэтому

$$el = Y(\lambda el. \lambda n. alpha\ (IF (= n 1))\ (el(- n 1))).$$

Теперь оказывается, что у следующего λ -выражения только одно МСВ и это *el*. Следовательно, *beta* определяется редукцией

$$beta\ el\ n \rightarrow alpha\ (IF (= n 1))\ (el(- n 1)),$$

по которой

$$el = Y(\lambda el. beta\ el).$$

Далее следует применять редукцию

$$gamma\ el \rightarrow beta\ el,$$

и поскольку комбинатор *gamma* эквивалентен комбинатору *beta*, то *gamma* порождать не следует.

Окончательно получаем, что *gamma* эквивалентен $(Y\ beta)$.

Упражнения

Упражнение 18.1. Получить выражение с оптимальным порядком суперкомбинаторов для определения:

$$fac\ n = IF\ n = 0\ then\ 1\ else\ n \times (fac(n - 1)).$$

Вопросы для самопроверки

Попробуйте дать ответы на приводимые вопросы.

1. Определите понятия ‘ленивое означивание в λ -исчислении’ и ‘полная ленивость’ в языке КАФ и укажите связь между ними.
2. Что означает термин ‘МСВ λ -выражения’?
3. Что представляет собой окончательный результат замещения исходного λ -выражения при полностью ленивой реализации суперкомбинаторов?
4. Какие преимущества дает использование правил оптимизации?
5. Назовите два основных критерия оптимизации.
6. На что влияет порядок параметров суперкомбинаторов?

Глава 19

Непосредственные вычисления

В настоящем разделе техника вычисления значения выражений пересматривается в свете систематического построения набора синтактико-семантических равенств, которые реализуют избранную парадигму объектно-ориентированных вычислений.

19.1 Задача

Задача 19.1. Для выражения:

$$\textit{let } x = \textit{plus in } x \ (4, (x \textit{ where } x = 3));;$$

построить λ -выражение и выражение комбинаторной логики, а также вычислить их значение.

Решение.

dc–1. Фиксируем выражение:

$$\textit{let } x = \textit{plus in } x \ (4, (x \textit{ where } x = 3));;$$

dc-2. Выразим его в виде λ -выражения:

$$M = (\lambda x.x(4, (\lambda x.x)3)) + .$$

dc-3. Подготовим выражение к переводу на язык комбинаторной логики (КЛ), то есть произведем каррирование:

$$N = (\lambda x.x\ 4\ ((\lambda x.x)3))\oplus,$$

где знак сложения ' \oplus ' отличается от знака '+'. Отложим пока обсуждение их различия до рассмотрения вопросов, связанных с вычислением выражений на категориальной абстрактной машине.

dc-4. Напомним, что процедура перевода в КЛ задается по индукции:

$$\begin{aligned} (i) \quad \lambda x.x &= I, \\ (ii) \quad \lambda x.c &= Kc, \ c \neq x, \\ (iii) \quad \lambda x.PQ &= S(\lambda x.P)(\lambda x.Q). \end{aligned}$$

Таким образом,

$$\begin{aligned} N &= (\lambda x.x\ 4((\lambda x.x)3))\oplus \\ &= S(\lambda x.x\ 4)(\lambda x.((\lambda x.x)3))\oplus \\ &= S(S(\lambda x.x)(\lambda x.4))(S(\lambda x.(\lambda x.x))(\lambda x.3))\oplus \\ &= S(SI(K\ 4))(S(\lambda x.I)(K3))\oplus \\ &= S(SI(K\ 4))(S(K\ I)(K\ 3))\oplus . \end{aligned}$$

dc-5. Пользуясь этой процедурой, получили

$$N = S(SI(K\ 4))(S(K\ I)(K\ 3))\oplus,$$

где $Sxyz = xz(yz)$, $Kxy = x$, $Ix = x$.

dc-6. Используя какой-либо способ вычисления, получаем в результате число '7':

$$\begin{aligned}
 & S(SI(K4))(S(KI)(K3))\oplus \rightarrow \\
 & \rightarrow (SI(K4)\oplus)(S(KI)(K3)\oplus) \\
 & \rightarrow (I\oplus)(K4\oplus)(S(KI)(K3)\oplus) \\
 & \rightarrow \oplus(K4\oplus)(S(KI)(K3)\oplus) \\
 & \rightarrow \oplus4(S(KI)(K3)\oplus) \rightarrow \oplus4(KI\oplus)(K3\oplus) \\
 & \rightarrow \oplus4(I(K3\oplus)) \rightarrow \oplus43 \rightarrow 7.
 \end{aligned}$$

Проверить результат легко прямым выполнением β -редукции для λ -выражения:

$$M = (\lambda x.x(4, (\lambda x.x)3)) + \rightarrow + (4, (\lambda x.x)3) \rightarrow + (4, 3) \rightarrow 7,$$

(еще раз обращаем внимание на использование другого символа операции сложения: '+' вместо ' \oplus ').

Упражнения

Упражнение 19.1. Выразите через комбинаторы K и S объект I с комбинаторной характеристикой $Ia = a$.

Указание. Воспользуйтесь тем, что $I = \lambda z.z$. Получите $\lambda z.z = (\lambda x y z.xz(yz))(\lambda x y.x)(\lambda x y.x)$ и вспомните, что $K = \lambda x y.x$, $S = \lambda x y z.xz(yz)$.

Ответ. $I = SKK$.

Упражнение 19.2. Для выражения:

$$let\ x = \pi/2\ in\ let\ z = sin\ in\ sqr(z\ x);;$$

постройте выражение комбинаторной логики и вычислите его значение.

Указание.

1. λ -выражение имеет вид:

$$\begin{aligned}
 (\lambda x. (\lambda z. \text{sqr}(z\ x)) \sin) \pi / 2 &= \\
 &= (\lambda x. \text{sqr}(\sin\ x)) \pi / 2, & (\beta) \\
 &= \text{sqr}(\sin\ \pi / 2). & (\beta)
 \end{aligned}$$

2. Воспользуйтесь тем, что

$$f(gx) = (f \circ g)x = S(KS)K\ f\ g\ x.$$

Ответ. $S(KS)K\ \text{sqr}\ \sin\ \pi / 2 = 1.$

Вопросы для самопроверки

1. Выпишите комбинаторные характеристики K, I, S, B, W, C стандартных комбинаторов.
2. Почему для реализации β -редукции оказался удобным переход к комбинаторной логике?

Глава 20

Код де Брейна

Вводится в рассмотрение техника переобозначения связанных переменных (формальных параметров), которая позволяет избежать коллизий связывания при замещении формальных параметров на фактические. Это прием переобозначения носит название *кодирования по де Брейну* и позволяет, фактически, аппаратом λ -исчисления пользоваться на тех же самых правах, что и аппаратом комбинаторной логики.

20.1 Задачи

Задача 20.1. Для выражения:

$$\text{let } x = \text{plus in } x (4, (x \text{ where } x = 3));;$$

построить λ -выражение и выражение кода де Брейна и вычислить последнее с помощью *SECD*-машины.

Формулировка задачи. Известно, что в ходе выполнения λ -конверсии возникают коллизии переменных. Например, “прямое” выполнение β -редукции для $(\lambda xy.x)y$ могло бы дать $\lambda y.y$:

$$(\lambda xy.x)y = \lambda y.y,$$

что совершенно недопустимо, поскольку:

$$\begin{aligned} (\lambda xy.x)y &\stackrel{(\alpha)}{=} (\lambda uv.u)y \stackrel{(\beta)}{=} (\lambda v.y) \\ &\neq (\lambda y.y) = I. \end{aligned}$$

Заметим, далее, что в замкнутом терме существенной важной информацией о переменной служит глубина ее связывания, то есть количество символов λ , стоящих между переменной и связыванием λ (не считая последний оператор). Тогда переменная оказывается замененной на число, которое, однако, нельзя смешивать с обычным натуральным числом. Для отличия числа, заменяющие переменные, от обычных натуральных чисел первые будем называть числами де Брейна. Теперь, например, для

$$P = \lambda y.(\lambda xy.x)y$$

кодирование по де Брейну приобретает вид:

$$\lambda.(\lambda\lambda.\underline{1})\underline{0}.$$

Скажем, правило (β) , примененное к этому выражению, дает $\lambda\lambda.\underline{1}$, и нет необходимости в преобразовании $\lambda xy.x$ в $\lambda xv.x$, которое ликвидирует коллизию. Основной вопрос — это описание смысла выражений. Это зависит от ассоциаций значений и идентификаторов, то есть от среды. Таким образом, означивание представляет собой функцию $\| M \|$, ассоциирующую значение со средой. Представим обычные семантические равенства, где приложение функции к аргументу представляется просто записью непосредственно вслед за символом функции символа аргумента:

$$\begin{aligned} \|x\|env &= env(x), \\ \|c\|env &= c, \\ \|(M\ N)\|env &= \|M\|env (\|N\|env), \\ \|\lambda x.M\|env\ d &= \|M\|env [x \leftarrow d], \end{aligned}$$

где:

- $env(x)$ - значение x в среде env ;
 c - константа, обозначающая значение, также называемое c , что соответствует обычной практике;
 $env[x \leftarrow d]$ - среда env , где x замещен на значение d , то есть выполнена *подстановка* d вместо x в env .

Вообще, формализм де Брейна может быть рассмотрен по аналогии с комбинаторной логикой с соответствующей адаптацией правил. Для осуществления перехода от обычных λ -выражений к кодировке переменных числами де Брейна рассмотрим ряд правил и соглашений.

Пусть среда env имеет вид

$$(\dots (((), w_n) \dots, w_0),$$

где значение w_i ассоциировано с числом i де Брейна. Такое предположения учитывает довольно сильные ограничения. Среды, в которых происходит вычисление выражений, считаются связанными структурами, а не массивами. Такой выбор тесно связан с выполнением требования эффективности. Прежде всего данный выбор ведет к простому машинному описанию:

$$\begin{aligned}
 \parallel \underline{0} \parallel (env, d) &= d, \\
 \parallel \underline{(n+1)} \parallel (env, d) &= \parallel \underline{n} \parallel env, \\
 \parallel c \parallel env &= c, \\
 \parallel M N \parallel env &= \parallel M \parallel env (\parallel N \parallel env) \\
 \parallel \lambda.M \parallel env d &= \parallel M \parallel (env, d).
 \end{aligned}$$

Интерес представляют не только значения сами по себе, а значения с точки зрения обеспечиваемых ими вычислений. При комбинаторном подходе подчеркивается, что значением, например, MN служит комбинация значений M и N .

Вводится три комбинатора:

$$\begin{array}{ll} S & \text{с арностью } 2, \\ \Lambda & \text{с арностью } 1, \\ ' & \text{с арностью } 1 \end{array}$$

и бесконечно много комбинаторов $n!$ в том смысле, что:

$$\begin{aligned} \|\underline{n}\| &= n!, \\ \|c\| env &= c, \\ \|MN\| &= S(\|M\|, \|N\|), \\ \|\lambda.M\| &= \Lambda(\|M\|). \end{aligned}$$

Отсюда легко устанавливается процедура перехода от семантических равенств к чисто синтаксическим:

$$\begin{aligned} 0!(x, y) &= y, \\ (n+1)!(x, y) &= n!x, \\ ('x)y &= x, \\ S(x, y)z &= xz(yz), \\ \Lambda(x)yz &= x(y, z) \end{aligned}$$

Такие правила близки к SK -правилам: первые три указывают на “забывающее” аргумент свойство (наподобие комбинатора K); четвертое — некаррированная форма правила S ; пятое — в точности каррирование, то есть преобразование функции от двух аргументов в функцию от первого аргумента, задающую функцию от второго аргумента.

Введем также комбинатор пары $\langle \cdot, \cdot \rangle$, где

$$\|(M, N)\| = \langle \|M\|, \|N\| \rangle,$$

и снабдим его выделителями (проекциями) Fst и Snd . Введем также оператор композиции ‘ \circ ’ и новую команду ϵ . Рассмотрим $S(\cdot, \cdot)$ и $n!$ как сокращения для ‘ $\epsilon \circ \langle \cdot, \cdot \rangle$ ’ и ‘ $Snd \circ Fst^n$ ’ соответственно, где $Fst^{n+1} = Fst \circ Fst^n$. Сведем теперь все комбинаторные

равенства воедино:

(ass)	$(x \circ y)z = x(yz),$
(fst)	$Fst(x, y) = x,$
(snd)	$Snd(x, y) = y,$
$(dpair)$	$< x, y > z = (xz, yz),$
(ac)	$\varepsilon(\Lambda(x)y, z) = x(y, z),$
$(quote)$	$('x)y = x,$

где $(dpair)$ устанавливает связь спаривания и образования совокупности, а (acc) - композиции и аппликации. Можно заметить, что $S(x, y)z = \varepsilon(xz, yz)$. Тем самым придается однородность рассмотрению операторов Fst , Snd и ε . Кроме того, справедливо равенство:

$$'M = \Lambda(M \circ Snd),$$

откуда следует, что

$$('x)yz = xz.$$

Решение.

DB-1. Пользуясь синтаксическими и семантическими равенствами, получаем для $M = (\lambda x.x(4, (\lambda x.x)3)) + :$

$$\begin{aligned} M' &= \| M \| = \| (\lambda \underline{0}.4, (\lambda \underline{0}.3)) + \| \\ &= S(\| \lambda \underline{0}.4, (\lambda \underline{0}.3) \|, \| + \|) \\ &= S(\Lambda(\| \underline{0}.4, (\lambda \underline{0}.3) \|), \| + \|) \\ &= S(\Lambda(S(0!, \| (4, (\lambda \underline{0}.3) \|)), \| + \|) \\ &= S(\Lambda(S(0!, <' 4, S(\Lambda(0!), ' 3) >)), \Lambda(+ \circ Snd)). \end{aligned}$$

DB-2. Теперь произведем вычисления по методу Лендина для *SECD*-машины, то есть M следует означивать путем приложения M' к среде. В данном случае среда пуста, поскольку терм замкнут. При означивании M' применим стратегию самого левого и притом самого внутреннего выражения. Для краткости обозначим:

$$A = S(0!, <' 4, B >), B = S(\Lambda(0!), 3).$$

Приведем полную последовательность редукций:

$$(\Lambda(A), \Lambda(+ \circ Snd))() \rightarrow \varepsilon(\Lambda(A)(), \Lambda(+ \circ Snd)()) \rightarrow A \text{ env}$$

здесь принято сокращение $\text{env} \equiv ((), \Lambda(+ \circ Snd)())$:

$$\begin{aligned} &\rightarrow \varepsilon(0! \text{ env}, <' 4, B > \text{ env}) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), ('4 \text{ env}, B \text{ env})) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), (4, B \text{ env})) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), (4, \varepsilon(\Lambda(0!) \text{ env}, '3 \text{ env}))) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), (4, \varepsilon(\Lambda(0!) \text{ env}, 3))) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), (4, 0! (\text{env}, 3))) \\ &\rightarrow \varepsilon(\Lambda(+ \circ Snd)(), (4, 3)) \rightarrow (+ \circ Snd)((), (4, 3)) \\ &\rightarrow +(Snd((), (4, 3))) \rightarrow +(4, 3) \rightarrow 7. \end{aligned}$$

Видно, что результат совпадает с результатом, полученным в ходе непосредственных вычислений выражения.

Упражнения

Упражнение 20.1. Постройте выражения де Брейна для приводимых далее λ -выражений.

1) $\lambda y. yx$. *Ответ.* $\parallel \lambda y. yx \parallel = \Lambda(S(0!, 1!))$.

2) $(\lambda x. (\lambda z. zx)y)((\lambda t. t)z)$.

Указание. Обозначим исходное выражение через Q и перейдем к $R = \lambda zxy. Q$. Тогда, рассмотрев дерево представления R , можем записать его в виде: $R' = (\lambda. (\lambda. \underline{01}) \underline{1})((\lambda. \underline{0}) \underline{2})$, и простой заменой ' λ ' на ' Λ ', ' o ' на ' S ', ' \underline{n} ' на ' $n!$ ' получить код де Брейна для Q .

Ответ. $Q_{DB(z,x,y)} = S(\Lambda(S(\Lambda(S(0!, 1!))1!)), S(\Lambda(0!), 2!))$ (порядок имен переменных в индексе Q соответствует порядку их связывания в промежуточном выражении R и позволяет восстановить исходное определение с соответствующими свободными переменными).

Глава 21

Абстрактная машина: КАМ

Строится специальный вариант теории вычислений, называемый *категориальной абстрактной машиной*. Для этого вводится в рассмотрение специальный фрагмент комбинаторной логики — категориальная комбинаторная логика. Она представлена набором комбинаторов, каждый из которых имеет самостоятельное значение как инструкция системы программирования. Тем самым в комбинаторную логику встраивается еще одно полезное приложение — система программирования, основанная на декартово замкнутой категории. Это позволяет еще раз на новом уровне переосмыслить связь операторного и аппликативного стиля программирования.

21.1 Теоретические сведения

Аббревиатура ‘КАМ’ принята в качестве сокращения для ‘Категориальная Абстрактная Машина’. Как можно будет увидеть, такое название само по себе несет значительную смысловую нагрузку.

21.1.1 Структура КАМ

Сначала обоснуем введение самого термина ‘категориальная’. В *категориях* пользуются композицией и тождеством (тождественное преобразование служит целям оптимизации), в *декартовых категориях* дополнительно вводят произведение с использованием спаривания $\langle \cdot, \cdot \rangle$, а также проекций Fst и Snd . *Декартово замкнутые категории* (д.з.к.) дополнительно содержат каррирование Λ , апплицирование ϵ и средства экспоненцирования, то есть средства построения функциональных пространств.

Перечисленные в предыдущем разделе правила позволяют выполнять означивание категориальных термов вида M' , построенных из указанных комбинаторов путем аппликации (приложения) терма к среде, которая построена из совокупности различных компонентов:

- 1) комбинаторы аппликации и построения совокупности в M' не использованы;
- 2) аппликация возникает, когда записываем $M'()$, то есть когда категориальный терм прикладывается к (пустой) среде;
- 3) построение совокупности происходит всякий раз, когда используется правило (*dpair*).

Машинные инструкции

Машинные инструкции такой КАМ строятся следующим образом:

статические операторы можно принять за базисные машинные инструкции.

Сперва отметим, что в применяемых при редукции $M'()$ правилах все редексы имеют вид Mw , где w — значение, то есть терм в нормальной форме по отношению к применяемым правилам. Терм преобразован по де Брейну, то есть является кодом де Брейна. Далее:

1. рассматриваем терм как код, действующий на w , причем образован из элементарных составляющих кода;
2. проекции Fst и Snd с легкостью причисляются к набору инструкций: Fst действует на значение (w_1, w_2) , образуя доступ к первому порожденному элементу пары, если считать, что значение представлено в виде бинарного дерева;
3. для совокупностей рассматривается действие $\langle M, N \rangle$ на w в соответствии с равенством:

$$\langle M, N \rangle w = (Mw, Nw).$$

Действия M и N на w должны выполняться независимо, а затем результаты объединяются в виде дерева, вершина которого является совокупностью, а листья — полученными значениями w_1 и w_2 .

В последовательной машине сперва выполняется означивание, например M . Получается w_1 . Перед началом обработки w его следует сохранить в памяти, чтобы иметь возможность восстановить w при означивании N , когда получается w_2 . Наконец w_1 и w_2 собираются в совокупность, но в предположении, что запомнено значение w_1 , а это выполняется именно в тот момент, когда восстанавливаем w .

Структура машины

Исходя из рассмотренных соображений возникает *структура машины*:

- T — терм как структурированное значение, например граф;
- C — код;
- S — стек, или дамп (вспомогательная память).

Состоянием машины считается тройка $\langle T, C, S \rangle$.

На основании предыдущего *кодом* для $\langle M, N \rangle$ считается следующая последовательность инструкций:

‘<’, за которой следует последовательность инструкций, соответствующая коду M , за которой следует ‘;’, за которой следует последовательность инструкций, соответствующая коду N , за которой следует ‘>’.

21.1.2 Инструкции

Отметим, что категориальная абстрактная машина (КАМ) совершает только символьные преобразования, компиляция нигде не проводится. Достигается это с помощью следующих инструкций.

Инструкции < , >

Представим действие инструкций ‘<’, ‘;’, ‘>’:

- < : выталкивает терм на вершину стека,
- , : меняет местами терм и вершину стека,
- > : строит совокупность из вершины стека и терма, заменяет терм на эту совокупность и проталкивает стек.

Тот конкретный синтаксис, который используется для спаривания, в точности соответствует образованию управляющих инструкций. Эти инструкции должны замещать конструкцию спаривания. Тем самым достигается комбинирование означиваний M и N в означивание $\langle M, N \rangle$.

Инструкции Fst Snd $\Lambda(C)$

Применительно к структуре машины проекции Fst и Snd можно описать более точно:

Fst : получает терм (s, t) и замещает его на s ,

Snd : получает терм (s, t) и замещает его на t ,

Код для $n!$ формируется из n и инструкции ' Fst ', за которой следует инструкция ' Snd '. Для построения кода можно не предпринимать дополнительных усилий, если воспользоваться соглашениями:

примем $x\ y$ или $x|y$ для обозначения композиции, которая в обычной математической практике обозначается как $y \circ x$. При каррировании кодом $\Lambda(M)$ служит $\Lambda(C)$, где C — код. Действие ' Λ ' описывается следующим образом:

$\Lambda(C)$: замещает терм s на $C : s$, где C — код, инкапсулированный в Λ ;

обозначение $C : s$ служит сокращением для ' $\Lambda(M)s$ '. С точки зрения правил перезаписи действие ' Λ ' пусто, поскольку $\Lambda(M)w$ является значением в силу того, что w — значение, следовательно его нельзя перезаписать. Дадим перефразировку в терминах действий:

действием $\Lambda(M)$ на w является $\Lambda(M)w$.

Описание команды ' Λ ' приводит к построению замыкания, как и в $SECD$ -машине. Действительно, как подчеркивается самим обозначением $C : s$, в качестве значения обрабатывается совокупность, построенная из кода, соответствующего телу λ -выражения, и значения, которое представляет собой среду декларации функции, описанной посредством абстракции.

Инструкция '

Дополнительно нужны константы. Они требуются для базисных констант, когда кодом для ' C ' является ' C ' со следующим действием:

' C : замещает терм на инкапсулированную константу C .

Для конструкций типа встроенной функций, например, для символа сложения, используется кодирование, ранее уже рассмотренное:

кодом для ' (op) ' служит $\Lambda((op))$, где (op) — это инструкция Snd , за которой следует ' op '.

Инструкция ϵ

Вернемся к операции аппликации из λ -исчисления. Ее запишем, как $\epsilon o < M, N >$. В виде правила запишем равенство:

$$(\epsilon o < M, N >)w = \epsilon(Mw, Nw) \quad (= \epsilon[Mw, Nw]).$$

Последняя запись, содержащая символы квадратных скобок, используется в обозначениях комбинаторной логики.

Пусть (Mw, Nw) означено как (w_1, w_2) , что является действием кода, ассоциированного с $< M, N >$. Осталась инструкция ' ϵ ', которая получает $w_1 = \Lambda(P)w'_1$ и производит $\epsilon(\Lambda(P)w'_1, w_2) = P(w'_1, w_2)$.

В терминах КАМ кодом, соответствующим $\epsilon o < M, N >$, служит код для $< M, N >$ за которым идет ' ϵ ' со следующим эффектом:

ϵ получает терм $(C : s, t)$, замещает его на (s, t) , а оставшейся части кода устанавливает префикс C .

Цикл работы КАМ

Прежде чем приступить к построению полной таблицы инструкций, вспомним соглашения об обозначениях для списков инструкций и элементов стека:

пустой список L обозначается через $[]$; обозначение $[e_1; \dots; e_n]$ принято для списка с n элементами e_1, \dots, e_n ;

Таблица 21.1: Цикл работы КАМ

Начальная конфигурация			Результирующая конфигурация		
Терм	Код	Стек	Терм	Код	Стек
(s, t)	$car.C$	S	s	C	S
(s, t)	$cdr.C$	S	t	C	S
s	$(quoteC).C$	S	C	C	S
s	$(curC).C1$	S	$(C : s)$	$C1$	S
s	$push.C$	S	s	C	$s.S$
t	$swap.C$	$s.S$	s	C	$t.S$
t	$cons.S$	$s.S$	(s, t)	C	S
$(C : s, t)$	$app.C1$	S	(s, t)	$C@C1$	S

$a.L$ добавляет в начало L ;
 $L1@L2$ присоединяет $L2$ к $L1$.

Для удобства дадим инструкциям мнемонические наименования в зависимости от их действия:

Fst	Snd	$<$	$,$	$>$	ϵ	Λ	$'$
car	cdr	$push$	$swap$	$cons$	app	cur	$quote$

Представим таблицу 21.1, описывающую работу КАМ. В ее левой части — начальные состояния (“старые” состояния), а в правой — результирующие (“новые” состояния). Фактически, считаем КАМ λ -исчислением с явными произведениями.

Фактически, в этой таблице описана система программирования с небольшим числом исходных команд (инструкций). Заметим, что среди них нет инструкции условного вычисления (условного перехода). Эту инструкцию в дальнейшем приходится добавлять, пользуясь некоторыми соглашениями, касающимися представления о работе категориальной абстрактной машины. Кроме

того, при вычислении рекурсивных определений приходится дополнительно обрабатывать (неявный) оператор неподвижной точки. Для этого снова требуются дополнительные соглашения.

21.2 Задачи

Задача 21.1. Для выражения:

$$\text{let } x = \text{plus in } x(4, (x \text{ where } x = 3));;$$

построить его запись с помощью “категориального кода” и написать программу вычисления в терминах КАМ-инструкций.

Решение.

САМ–1. Воспользуемся математической символикой, которая подчеркивает связь с правилами перезаписи. Пусть A , B обозначают коды A и B соответственно, $+$ служит сокращением для *plus*, $S(x, y) \equiv S[x, y] = \varepsilon \circ \langle x, y \rangle$, $\oplus \equiv (Snd+) : ()$.

Результирующие вычисления представим в табл. 21.2. Отметим, что вычисления начинаются с пустой средой, то есть в позиции терма записан $()$. Исходный терм представлен в виде кода де Брейна. В самом начале вычислений стек, или дамп (“вспомогательная память”) также предполагается пустым $[]$.

Таким образом, КАМ позволила получить ожидаемый результат еще одним способом, легко реализуемым на компьютере. Для этого следует иметь ввиду, что операция $+$ не является собственно инструкцией КАМ, но является встроенной в хост-систему программирования функцией.

Таблица 21.2: Вычисление на КАМ

Терм	Код	Стек
$()$	$< \Lambda(A), \Lambda(Snd+) > \varepsilon$	$[]$
$()$	$\Lambda(A), \Lambda(Snd+) > \varepsilon$	$[()]$
$A : ()$	$, \Lambda(Snd+) > \varepsilon$	$[()]$
$()$	$\Lambda(Snd+) > \varepsilon$	$[A : ()]$
...
\oplus	$> \varepsilon$	$[A : ()]$
$(A : (), \oplus)$	ε	$[]$
$(((), \oplus)$	$< Snd, <' 4, B >> \varepsilon$	$[]$
$(((), \oplus)$	$Snd, <' 4, B >> \varepsilon$	$[(((), \oplus)]$
\oplus	$, <' 4, B >> \varepsilon$	$[(((), \oplus)]$
$(((), \oplus)$	$<' 4, B >> \varepsilon$	$[\oplus]$
$(((), \oplus)$	$'4, B >> \varepsilon$	$[(((), \oplus); \oplus]$
4	$, B >> \varepsilon$	$[(((), \oplus); \oplus]$
$(((), \oplus)$	$>> \varepsilon$	$[4; \oplus]$
$(((), \oplus)$	$(Snd), '3 > \varepsilon >> \varepsilon$	$[(((), \oplus); 4; \oplus]$
$Snd : (((), \oplus)$	$, 3 > \varepsilon >> \varepsilon$	$[(((), \oplus); 4; \oplus]$
$(((), \oplus)$	$3 > \varepsilon >> \varepsilon$	$[Snd : (((), \oplus); 4; \oplus]$
3	$> \varepsilon >> \varepsilon$	$[Snd : (((), \oplus); 4; \oplus]$
$(Snd : (((), \oplus), 3)$	$\varepsilon >> \varepsilon$	$[4; \oplus]$
$(((), \oplus), 3)$	$Snd >> \varepsilon$	$[4; \oplus]$
3	$>> \varepsilon$	$[4; \oplus]$
$(4, 3)$	$> \varepsilon$	$[\oplus]$
$(\oplus, (4, 3))$	ε	$[]$
$(((), (4, 3))$	$Snd+$	$[]$
$(4, 3)$	$+$	$[]$
7	$[]$	$[]$

Упражнения

Упражнение 21.1. Представить таблицу машинных инструкций КАМ для вычисления выражения:

$$\text{let } x = 3 \text{ in } (op(7, x) \text{ where } op = sub).$$

Указание. Вначале получите соответствующее λ -выражение

$$(\lambda x. (\lambda op. op \ 7 \ x) \ sub) 3,$$

на основе постулатов λ -исчисления преобразуйте его к виду

$$(\lambda op. op(7, (\lambda x. x) 3)) \ sub$$

и получите код де Брейна:

$$S(\Lambda(S(0!, <' 7, S(\Lambda(0!), ' 3) >)), \Lambda(sub \circ Snd)).$$

Теперь, переписав его в виде инструкций КАМ и выполнив необходимые преобразования, можно получить ответ.

Ответ. 4.

Глава 22

Оптимизация КАМ-вычислений

Использование декартово замкнутой категории открывает дополнительные возможности оптимизации результирующего программного кода. Помимо свойств самой комбинаторной логики, используемой в качестве оболочки, допускается применение специальных категориальных равенств, заимствованных из декартово замкнутой категории как из приложения.

22.1 Задача

Задача 22.1. Для выражения:

$$\textit{let } x = 5 \textit{ in let } z \textit{ y} = y + x \textit{ in let } x = 1 \textit{ in } (zx) \times 2;;$$

составить КАМ-программу вычисления значения и по возможности оптимизировать ее.

Решение.

Таблица 22.1: Вычисление значения подстановки

Терм	Код	Стек
s	$push.(cur\ C).swap.C1@[cons;\varepsilon]$	S
s	$(cur\ C).swap.C1@[cons;\varepsilon]$	$s.S$
$C : s$	$swap.C1@[cons;\varepsilon]$	$s.S$
s	$C1@[cons;\varepsilon]$	$(C : s).S$
w	$[cons;\varepsilon]$	$(C : s).S$
$(C : s, w)$	$[\varepsilon]$	S
(s, w)	C	S

opt–1. Запишем исходное выражение:

$$P = let\ x = 5\ in\ let\ z\ y = y + x\ in\ let\ x = 1\ in\ (zx) \times 2.$$

Его сведение к λ -выражению дает:

$$P = (\lambda x.(\lambda z.(\lambda y.(zx) \times 2)1)(\lambda y.y + x))5.$$

Заметим, что эта форма записи приводит к простой оптимизации во время компиляции. Дело в том, что код, соответствующий выражению $(\lambda.M)N$, представляет собой инструкцию ‘*push*’, за которой следует ‘*cur C*’ (где C – код M), за которой следует ‘*swap*’, за которой следует код $C1$ для N , за которым следует ‘*cons*’ и ‘ ε ’. Предполагая, что означивание $C1$ на терме s дает значение w , приведем основные шаги вычисления:

$$\begin{aligned} \text{код } ((\lambda.M)N) &= push.cur\ C.swap.C1@[cons;\varepsilon], \\ C &= \text{код}(M),\ C1 = \text{код}(N). \end{aligned}$$

В таблице 22.1 представлены вычисления значения.

Заметим, что

$$\| (\lambda.M)N \| = < \Lambda(\| M \|), \| N \| > \varepsilon.$$

Рассмотрим средства получения оптимизированного кода. Кроме возможности означивания выражений относительно среды, как это уже рассматривалось, в рамках категориальной комбинаторной логики возможно весьма естественно смоделировать β -редукцию. Для этого используется множество правил, отличающихся от рассмотренных ранее, причем используются только чистые “категориальные” комбинаторы, то есть исключаются составление совокупностей и апплицирование. Отправной точкой служит правило:

$$(Beta) \quad \varepsilon \circ < \Lambda(x), y > = x \circ < Id, y > .$$

Это правило обосновывается следующим образом:

$$\begin{aligned} (\varepsilon \circ < \Lambda(x), y >)t &= \varepsilon(< \Lambda(x), y > t) \\ &= \varepsilon(\Lambda(x)t, yt) \\ &= x(t, yt) = x(Id\ t, yt) \\ &= (x \circ < Id, y >)t, \end{aligned}$$

откуда следует принцип свертывания (*Beta*). Отметим, что $Id\ x = x$. По правилу (*Beta*) выражению ‘ $let\ x = N\ in\ M$ ’ сопоставляется код $push.skip.swap@C1@cons.C$, где $skip$ замещает Id (с пустым действием). Действие конструкции $[push.skip.swap]$ точно такое же, как и у $[push]$, то есть у рассмотренной оптимизации кода имеется теоретическое обоснование.

opt–2. Покажем, что оптимизация $[push.skip.swap]$ путем замены на $[push]$ правомерна. Действительно, в терминах λ -исчисления получаем:

$$< f, g > = \lambda t. \lambda r. r(ft)(gt) = \lambda t. (ft, gt),$$

а также получаем:

$$\langle g \rangle = \lambda t. \lambda r. r(t)(gt) = \lambda t. (t, gt) = \lambda t. (Id\ t, gt).$$

Замещая в первом равенстве f на Id , получаем:

$$\langle Id, g \rangle = \langle g \rangle,$$

что обосновывает употребление символа ' $\langle \cdot \rangle$ ' для единственного выражения (вырожденный случай). Далее,

$$\begin{aligned} \langle Id, g \rangle &= [push; skip; swap; g; cons], \\ \langle g \rangle &= [push; g; cons]. \end{aligned}$$

Отсюда требуемая оптимизация получается как графическое равенство.

opt–3. Введем еще одно правило оптимизации кода $[\oplus]^1$. Обоснование такой оптимизации трудности не представляет. Действительно, справедливы следующие равенства:

$$\begin{aligned} [\oplus] \quad \varepsilon \circ \langle \Lambda(+ \circ Snd), \langle M, N \rangle \rangle &= \\ &= (+ \circ Snd) \circ \langle Id, \langle M, N \rangle \rangle \\ &= + \circ \langle M, N \rangle = \langle M, N \rangle +. \end{aligned}$$

Более подробно:

$$\begin{aligned} (\varepsilon \circ \langle \Lambda(+ \circ Snd), \langle M, N \rangle \rangle) t &= \\ &= \varepsilon(\Lambda(+ \circ Snd)t, \langle M, N \rangle t) \\ &= (\Lambda(+ \circ Snd)t)(\langle M, N \rangle t) \\ &= (+ \circ Snd)(t, \langle M, N \rangle t) \\ &= +(Snd(t, \langle M, N \rangle t)) \\ &= (+ \circ \langle M, N \rangle)t, \end{aligned}$$

¹ Оказывается, что, например, можно провести оптимизацию, скомпилировав $M + N$ в код $\langle M, N \rangle$, за которым следует 'plus'.

откуда и следует требуемое равенство. Кроме того, последовательность $[cons; plus]$ можно заменить на $[plus]$, если считать что ' $plus$ ' представляет собой двухместную функцию, а в качестве аргументов обрабатывает терм и вершину стека. Следует отметить, что при этом частично теряется апплицируемость операции ' $plus$ ' к ее аргументам (так как осуществлен переход от λ -терма $(\lambda x. \lambda y. + xy)$ к двухместному оператору $x + y$, требующему сразу оба операнда), однако эта потеря общности для операций компенсируется возможностью аналогичного проведения оптимизации для частных случаев трех- и вообще, n -местных функций. Это обосновывается использованием следующих правил (случай трехместных функций):

$$\begin{aligned} \lambda t. (ft, gt, kt) &= \lambda t \lambda r'. r'(\lambda r. r(ft)(gt))(kt) = \\ &= \lambda t. (< f, g > t, kt) = << f, g >, k >, \end{aligned}$$

следовательно:

$$+(M, N, O) = + << M, N >, O > .$$

Действительно,

$$\begin{aligned} \varepsilon \circ < \Lambda(+ \circ Snd), \varepsilon \circ < \Lambda(Snd), << M, N >, O >>> &= \\ = + \circ Snd \circ < Id, \varepsilon \circ < \Lambda(Snd), << M, N >, O >>> &= \\ = + \circ (\varepsilon \circ < \Lambda(Snd), << M, N >, O >>>) &= \\ = + \circ Snd \circ < Id, << M, N >, O >> &= \\ = + \circ << M, N >, O > . & \end{aligned}$$

Таким образом, для исходной задачи можно выделить следующие основные шаги вычисления:

s	$push.C1@cons.C$	S
s	$C1@cons.C$	$s.S$
w	$cons.C$	$s.S$
(s, w)	C	S

Такое оптимизированное вычисление использует комбинатор тождества, без которого используемую комбинаторную логику трудно назвать категориальной.

opt—4. Вернемся к вычислению на КАМ терма P . Воспользуемся обозначением $x \mid y \equiv y \circ x$ для упрощения компилируемой записи. Введем сокращения:

$$Fst = F, Snd = S,$$

Изложение сделаем замкнутым, приведя все подробности выкладок. Формулировку принципа оптимизации (*Beta*) возьмем в виде:

$$(Beta) \quad < \Lambda(X), Y > \mid \varepsilon = x \circ < Id, y > = < y > \mid x.$$

Формулировку принципа оптимизации $[\oplus]$ возьмем в виде:

$$[\oplus] \quad < \Lambda(+ \circ Snd), < M, N > > \mid \varepsilon = \\ = (+ \circ Snd) \circ < Id, < M, N > >.$$

Кодирование по де Брейну дает:

$$P = (\lambda x. (\lambda z. (\lambda x. (z \ x) \times 2) 1) (\lambda y. y + x)) 5,$$

затем

$$P' = (\lambda. (\lambda. (\lambda. (\underline{1} \ \underline{0}) \times 2) 1) (\lambda. \underline{0} + \underline{1})) 5 \\ = (\lambda. (\lambda. (\lambda. \times ((\underline{1} \ \underline{0}), 2)) 1) (\lambda. + (\underline{0}, \underline{1}))) 5.$$

Далее вычисление значения P' дает:

$$\parallel P' \parallel = < ' 5 > \parallel (\lambda. (\lambda (\underline{1} \ \underline{0}) \times 2) 1) (\lambda. \underline{0} + \underline{1}) \parallel;$$

$$\begin{aligned} & \parallel (\lambda. (\lambda (\underline{1} \ \underline{0}) \times 2) 1) (\lambda. \underline{0} + \underline{1}) \parallel = \\ & = < \parallel \lambda. (\lambda. (\underline{1} \ \underline{0}) \times 2) 1 \parallel, \parallel \lambda. \underline{0} + \underline{1} \parallel > \varepsilon \\ & = < \Lambda \parallel \lambda. (\underline{1} \ \underline{0}) \times 2) 1 \parallel, \parallel \lambda. \underline{0} + \underline{1} \parallel > \varepsilon \\ & \stackrel{(Beta)}{=} < \parallel \lambda. \underline{0} + \underline{1} \parallel > \mid \parallel (\lambda. (\underline{1} \ \underline{0}) \times 2) 1 \parallel; \end{aligned}$$

$$\begin{aligned}
\| (\lambda.(\underline{1} \ \underline{0}) \times 2) 1 \| &= < \| \lambda.(\underline{1} \ \underline{0}) \times 2 \|, '1 > \varepsilon \\
&= < \Lambda \| (\underline{1} \ \underline{0}) \times 2 \|, '1 > \varepsilon \\
&= < '1 > | \| (\underline{1} \ \underline{0}) \times 2 \|; \\
< \| \lambda.\underline{0} + \underline{1} \| > &= < \Lambda \| +(\underline{0}, \underline{1}) \| > \\
&= < \Lambda < ' +, < 0!, 1! > > \varepsilon > \\
&= < \Lambda < \Lambda(+ \circ S), < 0!, 1! > > \varepsilon >; \\
(\oplus) & < \Lambda(+ \circ < 0!, 1! >) > \\
&= < \Lambda(< 0!, 1! > | +) > \\
&= < \Lambda(< S, F | S > | +) >; \\
\| (\underline{1} \ \underline{0}) \times 2 \| &= < \| \times \|, < \| (\underline{1} \ \underline{0}) \|, '2 > > \varepsilon \\
&= < \Lambda(\times \circ S), < \| (\underline{1} \ \underline{0}) \|, '2 > > \varepsilon \\
(\oplus) & \times \circ < \| (\underline{1} \ \underline{0}) \|, '2 > \\
&= \times \circ < < 1!, 0! > \varepsilon, '2 > \\
&= \times \circ < < F | S, S > \varepsilon, '2 > \\
&= < < F | S, S > | \varepsilon, '2 > | \times.
\end{aligned}$$

Для экономичной записи вычислений введены сокращения. С учетом сокращений для D , где $D = (< S, F | S > | +)$ и для C , где $C = < F | S, S > | \varepsilon$, получим:

$$\| P' \| = < '5 > | < \Lambda(D) > | < '1 > | < C', 2 > | \times.$$

Для сокращения громоздкости КАМ-вычислений в порядке соглашения обозначим:

$$B = < C', 2 > | \times, C = < F | S, S > | \varepsilon, D = < S, F | S > | +.$$

Обращаем внимание на то обстоятельство, что приводимые переобозначения, очевидно, имеют связь с суперкомбинаторами. Действительно, каждый объект, введенный в употребление в виде математического соглашения об обозначениях, представляет собой вполне определенный набор КАМ-инструкций. Этот набор может быть вычислен (заранее скомпилирован) на КАМ, и при

необходимости достаточно в нужном месте использовать именно результат предварительного вычисления. Нетрудно заметить, что эти предварительные вычисления лучше производить не бессистемно, но придерживаясь вполне определенной “дисциплины” вычислений². Сама по себе категориальная абстрактная машина является относительно удачно сбалансированной системой “математических” вычислений. Эти вычисления с большой наглядностью можно расположить в виде таблицы с тремя столбцами, отражающими текущее значение терма, кода и стека. Напомним, что именно текущая тройка $\langle \text{терм, код, стек} \rangle$ является в точности текущим состоянием (вычисления). Поэтому последовательность строк в таблице КАМ-вычислений задает последовательность состояний процесса вычисления³.

²Методы оптимизации вычислений, когда удастся выделить относительно независимые параметры, получили развитие не только в различных вариантах “суперкомбинаторного” программирования, но и широко используются в функциональном программировании. Сами подходы разнятся по используемой семантике вычислений: как правило, используется денотационная семантика “с продолжениями”. Это означает, что для всякого правильного в некотором смысле вычисления известен “остаток программы”.

³Вычисление может с математической точки зрения рассматриваться как последовательность состояний, то есть как *процесс* в точном понимании этого термина. Эта точка зрения вполне в духе теории вычислений Д. Скотта.

Приведем полную последовательность КАМ-вычислений:

Терм	Код	Стек
$()$	$<' 5 > < \Lambda(D) > <' 1 > B$	$[]$
$()$	$'5 > < \Lambda(D) > <' 1 > B$	$[()]$
(5)	$> < \Lambda(D) > <' 1 > B$	$[()]$
$((), 5)$	$< \Lambda(D) > <' 1 > B$	$[]$
$((), 5)$	$\Lambda(D) > <' 1 > B$	$[(), 5]$
$(D : ((), 5))$	$> <' 1 > B$	$[(), 5]$
$(((), 5), D : ((), 5))$	$<' 1 > B$	$[]$
$(((), 5), D : ((), 5))$	$'1 > B$	$[...]$
(1)	$> B$	$[...]$
$(((), 5), D : ((), 5)); 1$	B	$[]$
$(((), 5), D : ((), 5)); 1$	$< C, '2 > \times$	$[]$
$(((), 5), D : ((), 5)); 1$	$C, '2 > \times$	$[...; 1]$
$(((), 5), D : ((), 5)); 1$	$< F S, S > \varepsilon, '2 > \times$	$[...; 1]$
$(((), 5), D : ((), 5)); 1$	$F S, S > \varepsilon, '2 > \times$	$[...; ...; 1]$
$((D : ((), 5)))$	$, S > \varepsilon, '2 > \times$	$[...; ...; 1]$
$(((), 5), D : ((), 5)); 1$	$S > \varepsilon, '2 > \times$	$[D : ((), 5); ...]$
(1)	$> \varepsilon, '2 > \times$	$[D : ((), 5); ...]$
$((D : ((), 5)); 1)$	$\varepsilon, '2 > \times$	$[...]$
$(((), 5); 1)$	$D, '2 > \times$	$[...]$
$(((), 5); 1)$	$< S, F S > +, '2 > \times$	$[...]$
$(((), 5); 1)$	$S, F S > +, '2 > \times$	$[(((), 5); 1); ...]$
(1)	$, F S > +, '2 > \times$	$[(((), 5); 1); ...]$
$(((), 5); 1)$	$F S > +, '2 > \times$	$[1; ...]$
(5)	$> +, '2 > \times$	$[1; ...]$
$(1, 5)$	$+, '2 > \times$	$[...]$
(6)	$, '2 > \times$	$[...]$
$(...)$	$'2 > \times$	$[6]$
(2)	$> \times$	$[6]$
$(6, 2)$	\times	$[]$
(12)	$[]$	$[]$

Упражнения

Упражнение 22.1. Напишите оптимизированную программу для вычисления на КАМ выражения:

$$\text{let } x = 3 \text{ in let } z = x + y \text{ in } fz \text{ where } y = 1 \text{ where } f = \text{sqr}.$$

Указание. Переходим к λ -выражению:

$$(\lambda x.(\lambda z.(\lambda f.fz)\text{sqr})((\lambda y.+xy)1))3,$$

получаем код де Брейна:

$$(\lambda.(\lambda.(\lambda.\underline{0}\ \underline{1})\text{sqr})((\lambda.+ \ \underline{1}\ \underline{0})\ 1))3.$$

Далее,

$$\begin{aligned} \mathcal{S} \parallel \lambda.(\dots)(\dots), 3 \parallel &= \mathcal{S}(\Lambda(\parallel (\dots)(\dots) \parallel), '3); \\ \parallel (\dots)(\dots) \parallel &= \mathcal{S}(\parallel (\dots) \parallel, \parallel (\dots) \parallel); \\ \parallel (\dots) \parallel &= \Lambda(\mathcal{S}(\Lambda(\mathcal{S}(0!, 1!)), \parallel \text{sqr} \parallel)); \\ \parallel (\dots) \parallel &= \mathcal{S}(\Lambda(\mathcal{S}(\mathcal{S}(\parallel + \parallel, 1!), 0!)), '1); \end{aligned}$$

В выражениях индексы будут указывать нумерацию парных скобок, например,

$$0(\dots)_0, \dots, 6(\dots)_6.$$

Запишем теперь

$$\begin{aligned} \mathcal{S}_0(\Lambda_1(\mathcal{S}_2(\Lambda_3(\mathcal{S}_4(\Lambda_5(\mathcal{S}_6(0!, 1!)_6)_5, \parallel \text{sqr} \parallel)_4)_3, \\ \mathcal{S}_3(\Lambda_4(\mathcal{S}_5(\mathcal{S}_6(\parallel + \parallel, 1!)_6, 0!)_5)_4, '1)_3)_2)_1, '3)_0 \equiv R. \end{aligned}$$

Прделаем проверку, непосредственно выполнив вычисления:

$$\begin{aligned} R\rho &\equiv \mathcal{S}_0(\dots, '3)_0\rho \\ &= (\Lambda_1(\dots)_1\rho)('3\rho) \\ &= {}_1(\dots)_1(\rho, 3) \\ &\equiv {}_1(\dots)_1\rho'; \end{aligned}$$

$$\begin{aligned} {}_1(\dots)_1\rho' &\equiv S_2(\dots, \dots)_2\rho' \\ &= S_4(\dots, \dots)_4(\rho', {}_4(\dots)_4(\rho', 1)); \end{aligned}$$

$$\begin{aligned} {}_4(\dots)_4(\rho', 1) &= S_5(\dots, \dots)_5(\rho', 1) \\ &= S_6(\dots, \dots)_6(\rho', 1)1 \\ &= \Lambda(+ \circ Snd)(\rho', 1)\rho' 1 \\ &= (+ \circ Snd)((\rho', 1), 3)1 \\ &= + 3 1 = 4; \end{aligned}$$

$$\begin{aligned} S_4(\dots, \dots)_4(\rho', 4) &= \Lambda_5(\dots)_5(\rho', 4)(\Lambda_5(\dots)_5(\rho', 4)) \\ &= {}_5(\dots)_5((\rho', 4), (\Lambda_5(\dots)_5(\rho', 4))) \\ &\equiv S(0!, 1!)(\dots, \dots) \\ &= \Lambda_5(\dots)_5(\rho', 4)4 \\ &= {}_5(\dots)_5((\rho', 4), 4) \\ &\equiv (sqr \circ Snd)((\rho', 4), 4) \\ &= sqr(4) = 16. \end{aligned}$$

Полученное выражение R следует предварительно оптимизировать по правилам ($Beta$) и $[\oplus]$. Для этого полезно предварительно использовать равенство:

$$S(x, y) = \varepsilon \circ \langle x, y \rangle.$$

Ответ. $sqr(4) = 16$.

Вопросы для самопроверки

1. Назовите альтернативные способы устранения коллизии переменных в λ -выражениях.
2. Обоснуйте необходимость введения оператора композиции.
3. Покажите связь и различие между понятиями ‘пара’ и ‘совокупность’.

Указание. Можно воспользоваться теоретико-множественными определениями для этих понятий, положив

$$f : D \rightarrow E, \quad g : D \rightarrow F.$$

Отсюда получаем, что

совокупность : $h = \langle f, g \rangle : D \rightarrow E \times F$;

пара : $[f, g] = (f, g) : (D \rightarrow E) \times (D \rightarrow F)$.

4. Назовите три основных подхода к реализации языков функционального программирования, лежащих в основе концепции категориальной абстрактной машины.
5. Что представляют собой базисные машинные инструкции КАМ?
6. Опишите проекции Fst и Snd применительно к структуре машины.
7. Сформулируйте основные правила оптимизации.
8. Какое преимущество дает использование принципов (*Beta*) и \oplus при написании КАМ-программ ?

Глава 23

Переменные объекты

Заключительная часть рассмотрения исчислений объектов касается общих вопросов математического представления объектов. Показывается, что использование концепции функтор-как-объект позволяет в компактной и лаконичной форме обозреть основные законы объектно-ориентированных вычислений. В частности, акцент делается на системах меняющихся (переменных) понятий-концептов, которые являются обычными объектами комбинаторной логики, но проявляют полезные для программирования свойства. Например, с помощью переменных концептов без особых осложнений строится не только теория вычислений, но и семантика систем программирования, а также модели объектов данных. Данные-как-объект дают новые степени свободы в компьютерных рассуждениях.

23.1 Модели

В такой аппликативной вычислительной системе, как *бестиповое* λ -исчисление одни и те же объекты в зависимости от контекста вычислений могут выступать и в роли аргументов, и в роли функций, которые применяются к этим аргументам. Если аргументы бе-

ругся из области D , то функции нужно брать из области $D \rightarrow D$, а поскольку они могут меняться ролями, то нужно иметь изоморфизм области аргументов D области отображений $D \rightarrow D$, то есть:

$$D \simeq (D \rightarrow D),$$

а это в общем случае невозможно. Однако в частном случае, когда область $D \rightarrow D$ ограничивается до множества функций, непрерывных в некоторой топологии на D это возможно. Как показал Д. Скотт (D. Scott, [113]), для целей построения семантики λ -исчисления, покрывающей имеющиеся в computer science и, в частности, в программировании механизмы вычислений достаточно выбрать категорию *полных решеток* с непрерывными отображениями. В этом случае удастся построить объект D_∞ , который изоморфен объекту $D_\infty \rightarrow D_\infty$, то есть:

$$D_\infty \simeq (D_\infty \rightarrow D_\infty),$$

что приводит к построению экстенциональной модели λ -исчисления.

23.1.1 Аппликативная структура

Аппликативная структура, в силу своей простоты и концептуальной ясности, дает основу для формулирования, развития и исследования многообразия моделей данных.

Предположим, что все элементы можно собрать в одну совокупность — домен D , — на которой установлена единственная операция *аппликации* $(\cdot \cdot)$:

$$(\cdot \cdot) : D \times D \rightarrow D : \forall x, y \in D \exists z \in D. [x, y] \mapsto z \ \& \ z = (x y),$$

или, в иных обозначениях,

$$\frac{x \in D \quad y \in D}{(x y) \in D}, \quad (\cdot \cdot)$$

то есть для произвольной пары элементов x, y из D можно построить элемент (xy) , называемый ‘аппликацией, или приложением x к y ’. Эту пару D и (\cdot) :

$$\mathcal{M} = (D, (\cdot))$$

называют *аппликативной структурой*.

Отметим, что в литературе в качестве обозначения явной операции аппликации вместо ‘ (\cdot) ’ часто используют ‘ ε ’:

$$\begin{aligned} \varepsilon : D \times D &\rightarrow D : \\ \forall x, y \in D \exists z \in D. [x, y] &\mapsto z \ \& \ z = \varepsilon[xy] \equiv (xy), \end{aligned}$$

Эту запись можно эквивалентным образом упростить до:

$$\varepsilon : D \times D \rightarrow D, \varepsilon : [x, y] \mapsto (xy);$$

тогда аппликативная структура принимает вид:

$$\mathcal{M} = (D, \varepsilon).$$

Аппликативная структура \mathcal{M} считается *экстенсиональной*, если выполняется следующее условие:

$$\frac{\forall a, b \in D \forall x \in D. ax = bx}{a = b}.$$

Отметим, что для $\mathcal{M} = (D, \varepsilon)$ часто вместо ‘ $d \in D$ ’ пишут ‘ $d \in \mathcal{M}$ ’.

Термы

Предъявим класс всех *термов* в \mathcal{M} , который обозначим через $\mathcal{T}(\mathcal{M})$. Воспользуемся индукцией по сложности терма:

i) включим в класс термов все *переменные* v_1, \dots, v_i, \dots :

$$v_1, \dots, v_i, \dots \in \mathcal{T}(\mathcal{M});$$

ii) включим в класс термов все *константы* c_a :

$$\frac{a \in \mathcal{M}}{c_a \in \mathcal{T}(\mathcal{M})};$$

iii) аппликативная структура *замыкается* операцией аппликации, то есть

$$\frac{M, N \in \mathcal{T}(\mathcal{M})}{(M N) \in \mathcal{T}(\mathcal{M})}.$$

Здесь: M, N обозначают произвольные термы из $\mathcal{T}(\mathcal{M})$.

Приписывание

Под *приписыванием* в аппликативной структуре \mathcal{M} понимается отображение ρ :

$$\rho : \text{переменные} \rightarrow \mathcal{M},$$

то есть $\rho(v_i) \in \mathcal{M}$ для произвольной переменной v_i .

Оценивающее отображение

Начнем с того, что зафиксируем аппликативную структуру \mathcal{M} и приписывание ρ . *Оценка*, или интерпретация терма M — это отображение:

$$\| \cdot \| : \text{термы} \times \text{приписывания} \rightarrow \text{элементы из } \mathcal{M}.$$

Оценка термов из $\mathcal{T}(\mathcal{M})$ в \mathcal{M} при приписывании ρ строится индукцией по сложности терма:

i) все *переменные* $x \in \mathcal{T}(\mathcal{M})$ интерпретируются как:

$$\|x\|\rho = \rho(x),$$

где $\rho(x) \in \mathcal{M}$;

ii) все константы $c_a \in \mathcal{T}(\mathcal{M})$ интерпретируются как:

$$\|c_a\|\rho = a,$$

где $a \in \mathcal{M}$;

iii) аппликация терма M к терму N интерпретируется как:

$$\|(M\ N)\|\rho = (\|M\|\rho)(\|N\|\rho),$$

то есть по принципу: ‘оценка аппликации есть аппликация оценок’.

Тот факт, что равенство $M = N$ термов $M, N \in \mathcal{T}(\mathcal{M})$ истинно в \mathcal{M} при приписывании ρ , обозначается посредством:

$$\mathcal{M}, \rho \models M = N$$

и имеет смысл равенства $\|M\|\rho = \|N\|\rho$.

Как видно из определения оценки $\|M\|\rho$, она зависит от приписывания значений ρ на множестве свободных переменных M . Таким образом, в случае *замкнутых* термов M , когда $FV(M) = \emptyset$, оценка $\|M\|\rho$ не зависит от приписывания ρ , что обозначают посредством $\|M\|$.

Подстановка

Одной из наиболее часто используемых конструкций в языках программирования является подстановка. Ее смысл проявляется при рассмотрении приписывания. Зафиксируем приписывание ρ и возьмем элемент $a \in \mathcal{M}$. Тогда конструкция ‘приписывание подстановки a вместо переменной x ’ обозначается посредством ‘ $\rho([a/x])$ ’ и задается как:

$$\rho([a/x]) \stackrel{\text{def}}{=} \rho'(y)|_a^x \equiv \begin{cases} a, & \text{если } y \equiv x, \\ \rho(y), & \text{если } y \not\equiv x, \end{cases}$$

где y — некоторая переменная.

23.1.2 Модели с типами

Отметим, что изначально λ -исчисление как в бестиповом, так и в типовом варианте рассматривалось как инструмент или метатеория для формализации представления о *правиле* или *процессе*. Для оперирования типами потребуется ввести синтаксис типового языка.

Типы

Символы типов зададим индукцией по построению:

- i) базовый тип σ считается символом типа;
- ii) если σ и τ — типы, то (σ, τ) — тип:

$$\frac{\sigma - \text{тип}, \quad \tau - \text{тип}}{(\sigma, \tau) - \text{тип}}.$$

Переменные

Каждая *переменная* x_i снабжается некоторым типом σ , а этот факт записывается посредством одного из способов:

$$x_i^\sigma, \quad \text{либо } x_i : \sigma, \quad \text{либо } \#(x_i) = \sigma.$$

Объектам s , соответствующим λ -термам, содержащим множество *свободных переменных* $FV(s)$ и множество *связанных переменных* $BV(s)$ ставятся в соответствие типы индукцией по построению объекта:

- i) переменная x_i^σ является термом типа σ , где $FV(x_i^\sigma) = \{x_i^\sigma\}$, а $BV(x_i^\sigma) = \emptyset$;
- ii) если s — терм типа (σ, τ) , а t — терм типа σ , то (st) — терм типа τ :

$$\frac{s : (\sigma, \tau) \quad t : \sigma}{(st) : \tau}, \quad (\cdot \cdot)$$

множество свободных переменных в составном терме является объединением множеств свободных переменных в компонентных термах:

$$FV((s, t)) = FV(s) \cup FV(t),$$

и множество связанных переменных в составном терме является объединением множеств связанных переменных в компонентных термах:

$$BV((s, t)) = BV(s) \cup BV(t);$$

iii) если s — терм типа τ , а y — переменная типа σ , то $\lambda y.s$ — терм типа (σ, τ) :

$$\frac{y : \sigma \quad s : \tau}{(\lambda y.s) : (\sigma, \tau)}, \quad (\lambda \cdot \cdot)$$

из множества свободных в s переменных вычитается y :

$$FV((\lambda y.s)) = FV(s) - \{y\},$$

а к множеству связанных в s переменных добавляется y :

$$BV((\lambda y.s)) = BV(s) \cup \{y\}.$$

Таким образом, метаоператор абстракции $(\lambda \cdot \cdot)$ связывает одну переменную в одном терме. Совокупность всех λ -термов будем обозначать через Tm .

Предструктура

Под *предструктурой* будем понимать семейство

$$(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\})$$

с параметрами σ, τ , где D^σ — непустое множество, поставленное в соответствие произвольному символу типа σ , а $\varepsilon_{\sigma\tau}$ — отображение

$$\varepsilon_{\sigma\tau} : D^{(\sigma, \tau)} \times D^\sigma \rightarrow D^\tau, \quad \varepsilon_{\sigma\tau} : [x, z] \mapsto x(z)$$

для функции $x \in D^{(\sigma\tau)}$ и аргумента $z \in D^\sigma$, действующее для всяких типовых символов σ, τ .

Предструктуру считаем *экстенсиональной*, если для элементов $x, y \in D^{(\sigma, \tau)}$ при произвольном элементе $z \in D^\sigma$ из равенства $\varepsilon_{\sigma\tau}[x, z] = \varepsilon_{\sigma\tau}[y, z]$ следует, что $x = y$:

$$\frac{x, y \in D^{(\sigma, \tau)} \quad \forall z \in D^\sigma. \varepsilon_{\sigma\tau}[x, z] = \varepsilon_{\sigma\tau}[y, z]}{x = y}.$$

Приписывание

Приписыванием в системе $(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\})$ считается функция ρ , чью область определения составляет множество всех переменных и такая, что

$$\rho(x_i^\sigma) \in D^\sigma.$$

Множество всех приписываний обозначим через Asg .

Если x, y — переменные, то $\rho(\cdot)|_a^x$ задается посредством:

$$\rho(y)|_a^x \equiv \begin{cases} \rho(x)|_a^x = a, & \text{если } y \equiv x, \\ \rho(y), & \text{если } y \not\equiv x. \end{cases}$$

Сравните с определением приписывания подстановки в 23.1.1 на стр. 259.

Структура

Структурой считается семейство

$$(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\}, \|\cdot\|),$$

с параметрами σ, τ , где $(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\})$ — предструктура, а $\|\cdot\|$ — оценивающее отображение

$$\|\cdot\| : Tm \times Asg \rightarrow \bigcup_{\sigma} D^\sigma,$$

которое поэлементно определяется индукцией по построению термина:

$$\text{i) } \|x_i^\sigma\| \rho = \rho(x_i^\sigma);$$

$$\text{ii) } \|(st)\| \rho = \varepsilon_{\sigma\tau}[\|s\| \rho, \|t\| \rho], \text{ где терму } s \text{ приписан тип } (\sigma, \tau), \text{ а } t - \text{ тип } \sigma;$$

iii) для произвольного элемента $a \in D^\sigma$ выполняется

$$\varepsilon_{\sigma\tau}[\|(\lambda x.s)\| \rho, a] = \|s\| \rho|_a^x,$$

где терму s приписан тип τ , а переменная x имеет тип σ .

Для произвольной структуры $(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\}, \|\cdot\|)$ записываем:

$$(\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\}) \models s = t[\rho]$$

в том случае, если

$$\|s\| \rho = \|t\| \rho.$$

Пример 23.1. Рассмотрим один специальный вид структуры, обозначаемой через T_B :

$$T_B = (\{B^\sigma\}, \{\varepsilon_{\sigma\tau}\}),$$

которая определена на B . Для этой структуры:

$$B^\circ = B, \quad B^{\sigma \rightarrow \tau} = B^\sigma \rightarrow B^\tau, \quad \varepsilon_{\sigma\tau}[x, y] = x(y).$$

Это пример структуры *полного типа* на B . Действительно, для типа $(\sigma, \tau) \equiv (\sigma \rightarrow \tau)$ соответствующий домен (множество) всевозможных элементов этого типа представляет собой в точности $B^{\sigma \rightarrow \tau}$. Каждый его элемент является отображением типа $\sigma \rightarrow \tau$. С другой стороны $B^\sigma \rightarrow B^\tau$ в качестве домена имеет множество всевозможных отображений из B^σ в B^τ . Для структуры полного типа считаем, что оба эти множества должны совпадать.

Подстановка

Если s — терм, x — переменная, а t — терм того же самого типа, что и x , то запись

$$[t/x]s$$

читается как

‘подстановка терма t вместо каждого свободного вхождения x в s ’.

Подстановка определяется индукцией по сложности s следующим образом:

для *атомарного* значения s :

$$\text{i) } [t/x]x = t;$$

$$\text{ii) } [t/x]y = y \text{ для переменных } y \neq x;$$

для *неатомарного* значения s :

$$\text{iii) } [t/x](r s) = ([t/x]r)([t/x](r s));$$

$$\text{iv) } [t/x](\lambda x.s) = (\lambda x.s);$$

$$\text{v) } [t/x](\lambda y.s) = (\lambda y.[t/x]s) \text{ для переменных } y \neq x.$$

Подстановка является функцией $Subst$ из всех переменных в термы такая, что $Subst(x)$ имеет тот же самый тип, что и x . Точно также примем $s(Subst)$ для обозначения одновременной подстановки каждого свободного вхождения каждой переменной y в s , то есть $Subst(y)$.

Непротиворечивость

Для фиксированной структуры $M = (\{D^\sigma\}, \{\varepsilon_{\sigma\tau}\}, \|\cdot\| \cdot \|\cdot\|)$ проверим выполнимость теоремы о непротиворечивости.

Лемма 23.1. В случае, если связанные переменные терма s и свободные переменные терма t различны, то есть

$$BV(s) \cap FV(t) = \emptyset,$$

для подстановки терма t вместо переменной x в терм s выполняется равенство:

$$\|[t/x]s\|\rho = \|s\|\rho|_t^x.$$

Доказательство. Для фиксированного t применим индукцию по построению s . □

Лемма 23.2. Если переменная y отличается как от свободных, так и от связанных в s переменных:

$$y \notin FV(s) \cup BV(s),$$

то выполняется равенство:

$$\|[y/x]s\|\rho|_d^y = \|s\|\rho|_d^x.$$

Доказательство. Следует из леммы 23.1. □

Лемма 23.3. Если переменная y отличается как от свободных, так и от связанных в s переменных:

$$y \notin FV(s) \cup BV(s),$$

то выполняется равенство:

$$M \models (\lambda x.s) = (\lambda y.[y/x]s).$$

Доказательство. Из леммы 23.2 следует, что

$$\begin{aligned} \varepsilon[\|(\lambda x.s)\|\rho, d] &= \|s\|\rho|_d^x \\ &= \|[y/x]s\|\rho|_d^y \\ &= \varepsilon[\|(\lambda y.[y/x]s)\|\rho, d]. \end{aligned}$$

□

Лемма 23.4. В случае, если связанные переменные терма s и свободные переменные терма t различны, то есть

$$BV(s) \cap FV(t) = \emptyset,$$

для подстановки терма t вместо переменной x в терм s выполняется равенство:

$$\|(\lambda x.s)t\|\rho = \|[t/x]s\|\rho.$$

Доказательство. Из леммы 23.1 следует, что

$$\begin{aligned} \|(\lambda x.s)t\|\rho &= \varepsilon[\|(\lambda x.s)\|\rho, \|t\|\rho] \\ &= \|s\|\rho|_{\|t\|\rho}^x \\ &= \|[t/x]s\|\rho. \end{aligned}$$

□

Лемма 23.5. В случае, если переменная x не совпадает ни с одной из свободных в терме s переменных, то есть

$$x \notin FV(s),$$

выполняется равенство:

$$\|(\lambda x.sx)\|\rho = \|s\|\rho.$$

Доказательство.

$$\begin{aligned} \varepsilon[\|(\lambda x.sx)\|\rho, d] &= \|(sx)\|\rho|_d^x \\ &= \varepsilon[\|s\|\rho|_d^x, \|x\|\rho|_d^x] \\ &= \varepsilon[\|s\|\rho, d]. \end{aligned}$$

□

Лемма 23.6. Если $M \models s = t$, то $M \models (\lambda x.s) = (\lambda x.t)$.

Доказательство. Зафиксируем ρ . Тогда:

$$\begin{aligned}\varepsilon[\|(\lambda x.s)\| \rho, d] &= \|s\| \rho|_d^x \\ &= \|t\| \rho|_d^x \\ &= \varepsilon[\|(\lambda x.t)\| \rho, d].\end{aligned}$$

□

Теорема 23.1 (непротиворечивость). Если

$$\vdash s = t,$$

то в каждой структуре

$$M \models s = t.$$

Доказательство. Воспользуемся индукцией по доказательству равенства $s = t$, используя леммы 23.1–23.6. □

23.1.3 Частичные элементы

Для построения модели объектов данных приходится применять формальные средства логики. Но в случае использования стандартных средств они оказываются корректными только для непустых областей. Практика построения *открытых информационных систем* приводит к рассмотрению подобластей, определяемых неразрешимыми предикатами.

Частичные элементы

В случае подобластей, определяемых неразрешимыми предикатами, что характерно для Web, вопрос о том,

существует ли данный объект x как элемент этой подобласти, может оказаться неразрешимым. Как оказывается, если рассматривать более, чем один сорт переменных, то логические средства

приходится приспособливать к оперированию областями, относительно которых нельзя ни выяснить, являются ли они “обитаемыми”, ни выяснить, существуют ли их элементы полностью.

Налицо ситуация, когда в рассуждениях об объектах далеко не всегда можно предполагать их существование.

В этом случае правильно построенные конструкции языка могут ничего не обозначать.

Предикат существования

Для рассмотрения термов, которые могут не иметь значения — ничего не обозначающих, — вводится специальный *предикат существования* E . Для термина t конструкция

$$Et$$

читается как ‘ t существует’. Предполагается, что свободные переменные сортов из t пробегают область *потенциальных*, или *возможных* элементов, выбираемых из *неявной* внешней области.

Как оказывается, любую область D частичных элементов можно представить в виде подобласти некоторой области \tilde{A} *потенциальных* элементов, а эти последние рассматриваются как объективизации возможных элементов множества D . Принятие такой точки зрения означает следующее:

предикат физического существования E осуществляет селектирование *действительных* в D элементов.

Обычное допущение, применяемое в логике, состоит в том, что связанные переменные пробегают только действительные элементы. Это значит, что связанные переменные рассматриваются как *ограниченные* предикатом E .

Равенство частичных элементов

Установление всякой области предполагает, что в рамках этой области можно строить суждения об *одинаковости* объектов. Поскольку используются частичные элементы, то равенство среди них рассматривается в следующем смысле:

$$(t = s) \supset (Et \ \& \ Es),$$

то есть установленное равенство частичных элементов влечет их существование.

Эквивалентность частичных элементов

Разработка представления об *эквивалентности* элементов начинается с того, что все элементы вне предиката *E* считаются несуществующими и рассматриваются как эквивалентные в своем несуществовании. Более формально, принимается биусловие

$$t \equiv s \Leftrightarrow (Et \supset t = s) \ \& \ (Es \supset t = s).$$

Таким образом, эквивалентные элементы становятся попросту неразличимыми.

Экстенциональность

Требование корректности определения предикатов предполагает, что они экстенциональны не только относительно равенства, но и относительно эквивалентности. Иное название для этого принципа — *подстановочность эквивалентных*:

$$[t/x]\phi \ \& \ t = s \supset [s/x]\phi. \quad (ext)$$

По сути, отношение эквивалентности \equiv и предикат существования *E* выбираются в качестве исходных, а равенство становится производным и определяется посредством принятия биусловия

$$t = s \Leftrightarrow t \equiv s \ \& \ Et \ \& \ Es.$$

Сорта

Логические средства построения модели объектов данных представляют собой *многосортную* теорию с высшими типами. Это означает, что разрешается задавать структуры высшего порядка. Для этих целей каждой конечной последовательности сортов

$$(T_1, T_2, \dots, T_n)$$

ставится в соответствие сорт

$$[T_1, T_2, \dots, T_n],$$

который считается *сортом-степенью* всех n -арных отношений на данной последовательности областей. Как оказывается, остальные конструкции типов могут быть выражены посредством сортов-степеней.

Дескрипции

При построении логических средств в числе термов используются *определенные описания*, или *дескрипции* — конструкции ‘тот . . . , который . . .’. Такие конструкции удобны, а при их использовании не возникает осложнений, поскольку имеется предикат существования. В этом случае возникает запас термов с богатыми выразительными возможностями, которые можно использовать в зависимости от поставленных целей, как было показано Д. Скоттом (D. Scott, [114]) и М. Фурманом (M. Fourman, [49]). Таким образом вводимый терм

$$\mathbf{I}x\Phi$$

канонически читается как

‘тот единственный x , для которого выполняется (верно) Φ ’.

Возможные миры

Возможные миры будем понимать как различные собрания индивидов с дополнительной структурой или без нее. Термины ‘элемент’ и ‘индивид’ будем использовать взаимозаменяемо. Различные совокупности индивидов идентифицируются *индексными выражениями*. Как известно, произвольная система структур может быть индексирована. Это осуществляется выбором в качестве индексов элементов подходящего множества, причем такой выбор можно произвести различными способами.

Возьмем некоторое фиксированное множество I этих индексов, а также будем рассматривать совокупность *возможных* индивидов D и *виртуальных* индивидов V . Затем отметим индексам систему *действительных* индивидов A_i , положив для всякого индекса $i \in I$, возможно различные $A_i \subseteq D$. Ясно, что выполняется

$$A_i \subseteq D \subseteq V,$$

поскольку виртуальные индивиды вводятся лишь для увеличения закономерности. Отметим, что между A_i и i не предполагается взаимно однозначного соответствия, поскольку элементы множества I могут обладать структурой, которая не полностью отражается в отличиях различных совокупностей A_i друг от друга.

23.1.4 Модели объектов данных

При построении модели объектов данных с формальной точки зрения вместо термина ‘приписывание’, использованного на стр. 258 и на стр. 262, воспользуемся термином ‘соотнесение’, которое родственно представлению о возможном мире или системе индексов.

Концепты, индивиды и состояния

Среди *объектов* будем различать концепты, индивиды и состояния. Этими сущностями воспользуемся как основными единицами

построения *модели объекта данных*, или МОД. Как и следует ожидать, потребуются дополнительные средства формализации этой модели, которые систематизируем на рис.23.1. Интерпретация соответствий, представленных на этом рисунке приводит к формулированию специального подхода к установлению взаимосвязи объектов данных, который можно назвать *принципом концептуализации*.

Принцип концептуализации

Взаимодействие концептов, индивидов, состояний позволяет сформулировать основной *принцип концептуализации* следующем образом:

$$\|(\text{индивидуальный концепт})\| = \text{функция : соотнесения} \rightarrow \text{индивиды},$$

то есть *значение* индивидуального концепта может рассматриваться как функция из совокупности соотнесений в совокупность индивидов.

Это означает, что для описания концептов пользуемся формулами, которые идентифицируют функции из соотнесений в индивиды. Другими словами, концепт рассматривается как процесс в математическом смысле. Исходя из сформулированного принципа концептуализации, установим основу для схемы исследования предметной области, нацеленной на выделение и фиксацию объектов данных:

$$\begin{array}{l} \|(\text{индивидуальный концепт})\| \in \text{индивид}^{\text{соотнесение}} \\ \text{индивид}^{\text{соотнесение}} \rhd \text{индивид} \\ \text{индивид} \rhd \text{состояние}^{\text{соотнесение}} \\ \text{состояние}^{\text{соотнесение}} \rhd \text{состояние} \end{array}$$

В сформулированной схеме использованы следующие обозначения:

- знак ‘ \succ ’ отмечает переход от инварианта к семейству;
- запись *индивид*^{соотнесение} означает множество всех отображений из соотнесений в индивиды, то есть экспоненту;
- запись *состояние*^{соотнесение} означает множество всех отображений из соотнесений в состояния.

Основная особенность схемы заключается в переходе от индивидуальных концептов (в языке) к индивидуальным концептам в предметной области. Такой подход обеспечивается функцией вычисления значения $\| \cdot \|$. Далее концепт ПО¹ трактуется как процесс, на основании которого выделяются и фиксируются индивиды. Индивид также рассматривается как процесс, позволяющий указать состояния. Более нейтральной является терминология вида ‘состояние–метасостояние’, которая может применяться взамен ‘индивид–концепт’ либо взамен ‘состояние–индивид’. Как принцип концептуализации, так и схема исследования предметной области могут быть сформулированы не только на качественном уровне, но и в виде формальной диаграммы, показанной на рис.23.1. В соответствии с этим рисунком индивидуальный концепт описывается формулой Φ , которая дополнительно снабжается оператором дескрипции **I**. Напомним, что запись

$$\mathbf{I}x\Phi(x)$$

понимается как описательное предложение вида:

‘тот (единственный) x , который (обладает свойством) $\Phi(x)$ ’.

Описание объектов

Как оказалось, с интуитивной точки зрения оператор дескрипции, конечно, служит целям описания того или иного объекта. Сперва для такого описания вычисляется значение, которое позволяет

¹ПО – предметная область.

языковой конструкции поставить в соответствие объект предметной области. Для полученного таким способом образа языковой конструкции учтем соотносительное, что формально выражается индексом. При фиксации индивидов в МОД наиболее важно соблюдать справедливость условия для $i \in I$:

$$\|Ix\Phi(x)\|i = d \Leftrightarrow \{d\} = \{d \in D \mid \|\Phi(\bar{d})\|i = 1\},$$

которое считаем основной характеристикой формализации, где символ ‘1’ использован как истинностное значение *true*.

Варианты этого принципа в математике хорошо известны, но они значительно в меньшей степени используются в сфере исследования такого предмета, как объекты данных. Действительно, все, что здесь оказалось сказанным — это единственность индивидуализации объекта d посредством формулы Φ .

Другим аспектом характеристики формализации служит следующее соображение. Применяемые дескрипции обладают достаточной “избирательностью” для вычленения индивида из предметной области. В силу этого приведенная характеристика формализации МОД, кроме фиксации индивида в предметной области, предназначена для обеспечения связи с системами символьной обработки данных. Разработка и использование систем символьной обработки данных приводит к применению понятия функции в смысле определения. В основу исследования кладется процесс перехода от аргумента к значению, когда этот процесс кодируется определением. Определения обычно задаются предложениями языка исследователя. Затем они применяются к аргументам, заданным также предложениями этого языка исследователя. В случае компьютерных систем символьной обработки определения понимаются как программы, которые в свою очередь применяются к программам. Поскольку объектами изучения являются как функции, так и аргументы, то возникает бестиповая система, допускающая самоприменимость функции, что считается невозможным для “обычных” математических функций. В качестве основ-

ной бестиповой системы обычно используется λ -исчисление, опирающееся на понятие связанной переменной, или комбинаторная логика, вовсе не использующая понятие переменной.

23.2 Основная задача

Сформулируем достаточно общую постановку задачи, сводящуюся к формированию представления об *объекте* — *объекте данных* или *объекте метаданных*.

Задача 23.1. Построить объект заданного вида, исходя из его определяющих свойств Φ .

Формулировка задачи. Установить определение объекта, применив *принцип свертывания*:

$$C = \text{Iy} : [D] \forall x : D(y(x) \leftrightarrow \Phi) = \{x : D \mid \Phi\}, \quad (C)$$

где $[D]$ обозначает *степень-множество* (множество всех подмножеств) D .

Решение. Решение этой основной задачи опирается на диаграмму, показанную на рис.23.1. Конкретное применение зависит от целого ряда факторов и прежде всего — от целей использования объекта. К таким объектам относятся *элементарные типы и индексированные концепты*. Кроме этого вид результирующего объекта может изменяться в зависимости от применяемой вычислительной модели. В последующих подразделах приводятся различные варианты решения основной задачи — задачи построения модели объекта данных.

23.2.1 Элементарные типы

Рассмотрим построение определений для элементарных типов.

$H_T(I)$ —1. *Тип*. Обычно тип рассматривается как подмножество множества, идентифицированного символом сорта. Таким образом, для сорта D тип T зададим дескрипцией

$$T = \mathbf{I}y : [D] \forall x : D(y(x) \leftrightarrow \Phi) = \{x : D \mid \Phi\},$$

для которой справедливо включение $T \subseteq D \in [D]$.

$H_T(I)$ —2. *Отношение*. Объект данных, называемый отношением, рассматривается как подмножество декартова произведения областей, идентифицированных символами сортов. Следовательно, для сортов A, B отношение R определяется дескрипцией

$$\begin{aligned} R &= \mathbf{I}z : [A, B] \forall x : A \forall y : B(z[x, y] \leftrightarrow \Psi) \\ &= \{[x : A, y : B] \mid \Psi\}, \end{aligned}$$

для которой справедливо включение $R \subseteq A \times B$. В данном случае, чтобы избежать громоздкости, рассмотрено определение двухместного отношения R .

$H_T(I)$ —3. *Значение функции*. При построении баз данных значительное внимание уделяется классу отношений, называемому функциональными отношениями. С этой целью введем определение

$$R'(t) = \mathbf{I}y : B.R([t, y]),$$

где t — терм сорта A . В данном случае справедлива запись $R'(t) \in B$.

$H_T(I)$ —4. *Функциональная абстракция*. Такой объект данных отличается особенно частым употреблением в аппликативных системах программирования для указания на определение функции. Для переменной u сорта A и терма s сорта B функциональная абстракция определяется дескрипцией

$$\begin{aligned} \lambda u : A.s &= \mathbf{I}w : [A, B] \forall u : A \forall v : B(w[u, v] \leftrightarrow v = s) \\ &= \{[u, v] \mid v = s\}, \end{aligned}$$

для которой справедливо включение $\lambda u.s \subseteq A \times B$.

Как оказалось, аппарат дескрипций проявляет достаточную мощность — позволяет вторичным образом выразить оператор абстракции. Отметим, что в комбинаторной логике оператор абстракции также выражается через комбинаторы. В этом смысле можно усмотреть известную аналогию между средствами комбинаторной логики и средствами, предоставляемыми дескрипциями.

23.2.2 Типизация переменных объектов

Построим обобщение аппарата типизации объектов данных на случай определения совокупностей, которые изменяются в зависимости от внешних условий. В этом случае в рассмотрение вводятся *переменные концепты*, каждый из которых считается инвариантом соответствующей совокупности объектов для заданных условий.

$H_T(I)$ —1. Условия учитываются в виде индекса, который характеризует выбранное соотнесение. Наиболее типичными случаями определения переменных концептов следует считать унарный концепт-тип и бинарный концепт-отношение.

$H_T(I)$ —2. *Переменный концепт-тип*. Возникает при рассмотрении пар соотнесение-индивид и соответствует дескрипции

$$\begin{aligned} C &= C(I) = \mathbf{I}z : [I, T] \forall i : I \forall hi : T(z[i, hi] \leftrightarrow \Phi) \\ &= \{[i, hi] \mid \Phi\} \subseteq \{h \mid h : I \rightarrow T\} \\ &= H_T(I). \end{aligned}$$

В рассмотренном случае $H_T(I)$ считается совокупностью всех индивидов для соотнесений из I и для типа T . Данное определение допускает вывод одного частного случая, который является центральным при построении вычислительных

моделей с использованием λ -исчисления. Если в качестве I взять T , а индивиды считать константными, то C переводит такой индивид h в одноэлементное множество $\{h\}$:

$$C : h \mapsto h$$

и, конечно, $C(h) \in \{h\}$. Отсюда немедленно следует равенство

$$C = \mathbf{1}_C : C \rightarrow C,$$

то есть C ведет себя как единичное отображение $\mathbf{1}_C$ со свойством

$$C = C \circ C.$$

$H_T(I)$ —3. *Бинарный концепт-отношение.* Поскольку это наиболее общий случай зависимости объектов, то недостаточно ограничиться дескрипцией

$$\begin{aligned} \phi &= \phi(I) \\ &= \mathbf{I}z : [I, (T, T)]. \forall i \forall ui : T \forall vi : T (z[i, [ui, vi]] \leftrightarrow \Phi) \\ &= \{[i, [ui, vi]] \mid \Phi\} \\ &\subseteq \{< u, v > \mid < u, v > : I \rightarrow T \times T\} \\ &= H_{T \times T}(I). \end{aligned}$$

Существенный момент заключается в том, что u является элементом $H_T(I)$, а v — элементом $H_T(I)$, то есть $u \in H_T(I)$, $v \in H_T(I)$. Более того, при определении дескрипции для ϕ справедливы следующие равенства:

$$\begin{aligned} \phi &= \phi(I) \\ &= \mathbf{I}z : [(I, T), (I, T)]. \forall i \forall ui \forall vi (z[[i, ui], [i, vi]] \leftrightarrow \Phi) \\ &= \{[[i, ui], [i, vi]] \mid \Phi\} \\ &\subseteq \{[u, v] \mid \Phi\} \\ &= H_T(I) \times H_T(I). \end{aligned}$$

Оба выражения для ϕ оказываются изоморфными и могут быть положены в основу системы типизации.

23.2.3 Вычислительные модели

Рассмотрим типизированные вычислительные модели объектов данных. Основной целью системы типизации ОД² является ее использование для моделирования выполнения либо конструкций ЯМОД³, либо программы, возможно, использующей конструкции ЯМОД. Рассмотрим способ интерпретации конструкций ЯМОД или ЯМОД, которые, возможно, входят в объемлющую программу, либо, наоборот, содержат программу (функции) системы программирования. Способ основан на определении и использовании семейств аппликативных предструктур вида

$$(\{H_B\}, \{\varepsilon_{BC}\})$$

для произвольных типов B , C из системы типизации ОД, где H — домен объектов типа B , ε_{BC} — аппликатор вычисления значения функции на аргументе из H_B .

$H_T(I)-1$. *Строение модели.* При построении вычислительной модели объектов данных (ВМОД) предполагаются следующие действия:

- введение в рассмотрение пространства функций как явных объектов в представляющей категории;
- построение по заданным объектам A и B самостоятельного объекта $(A \rightarrow B)$;
- установление для его использования отображения-аппликатора $\varepsilon_{BC} : (B \rightarrow C) \times B \rightarrow C$, которое обеспечивает по $f : B \rightarrow C$ и $x : B$ вычисление $f(x) : C$, то есть $f, x \mapsto f(x)$;
- при вычислении $h(x, y)$ двухаргументного отображения-оператора $h : A \times B \rightarrow C$ фиксируется x , а $h(x, y)$ считается функцией от y ;
- для связи h со значениями вводится специальная функция

²ОД — объекты данных.

³ЯМОД — язык манипулирования объектами данных.

абстрактор Λ_{ABC} , типизируемая посредством

$$(\Lambda_{ABC}h) : A \rightarrow (B \rightarrow C);$$

• для заданного оператора h и аргумента x абстрактор имеет вид $(\Lambda_{ABC}h)(x) : B \rightarrow C$, что позволяет при вычислении значения от второго аргумента использовать аппликатор.

Принципиальной особенностью этой ВМОД является возможность ее использования для концептов, индивидов и состояний по отдельности. Вместе с тем допустимо применять ВМОД и для объекта данных в целом.

$H_T(I)$ –2. *Абстракция по двум переменным.* При абстрагировании по двум переменным проявляются все особенности разработанного аппарата, которые обычным образом переносятся на случай большего числа переменных. Для сокращения записи дескрипции не указаны сорта переменных:

$$\lambda[x, y].t = \lambda z \mathbf{I} w \forall x \forall y (z = [x, y] \& t = w).$$

В свою очередь w трактуется как отношение

$$w = \mathbf{I} d \forall z \forall \tau [d[z, \tau] \leftrightarrow \Psi]$$

и $\tau \in w'(z)$. Поскольку терм t в зависимости от z принимает значения из $w'(z)$, то полагаем $t \in w'(z)$. Соответствующие упорядоченные пары $[z, t]$ кладутся в основу определения объекта

$$\lambda z.t = \lambda[x, y].t,$$

что дает требуемую дескрипцию.

$H_T(I)$ –3. *Функциональное пространство.* Этот объект служит целям представления множества функций из типа **A** в тип **B**, что записывается в виде дескрипции

$$\mathbf{A} \rightarrow \mathbf{B} = \{z : [\mathbf{A}, \mathbf{B}] \mid \forall x \in \mathbf{A} \exists y \in \mathbf{B}. z[x, y]\}$$

или в более слабой форме для сортов A , B и $A \rightarrow B$. В последнем случае определяющая формула изменяется для учета связи сортов A , B с типами, \mathbf{B} .

$H_T(I)$ –4. *Аппликатор*. На основе двух предыдущих дескрипций укажем дескрипцию для оценочного морфизма:

$$\varepsilon_{AB} = \lambda[z, x] \mathbf{I}y.z[x, y],$$

которая по функции $z \in \mathbf{A} \rightarrow \mathbf{B}$ и аргументу $x \in \mathbf{A}$ указывает значение $y \in \mathbf{B}$, то есть

$$\varepsilon_{AB} : [z, x] \mapsto z(x),$$

или в более общем случае, $\varepsilon_{AB} : [z, x] \mapsto y$.

23.2.4 Индексированные объекты

Построение ВМОМД предполагает выделение и фиксацию основных строительных блоков расширяемой среды программирования. Каждый из этих блоков является динамичным объектом метаданных в том смысле, что учитывает соотношение либо разворачивание событий. Перечисленный порядок действий, нацеленный на учет динамики, предполагает введение средствами ЯООД⁴ целого ряда ОМД⁵, соединяющих в себе возможности АВС и средств типизации. Наиболее важными из таких ОМД являются индексированный концепт и индексированное отношение.

$H_T(I)$ –1. *Индексированный концепт*. Вычисление значения выражений, построенных в виде λ -абстракций, опирается на свойство расширяемости. Это свойство проявляется в добавлении к среде и увязке индивидов, удовлетворяющих телу λ -абстракции. Рассматривая аппликацию $(\lambda.\Phi)\bar{h}$, где

⁴ЯООД – язык определения объектов данных.

⁵ОМД – объекты метаданных.

Φ — тело λ -абстракции, h — индивид (индивидуальная константа) из предметной области, получаем :

$$\| (\lambda.\Phi)\bar{h} \| i = \| \Phi \| [i, hi]$$

для соотнесения i . Тем самым увязка индивида h с телом λ -абстракции Φ сводится к построению концепта, соответствующего $\| \Phi \|$ и проверке принадлежности $h \in \| \Phi \|$. Проверка принадлежности производится операцией селекции реляционного языка манипулирования данными. Проверка осуществляется согласно следующей процедуре:

- 1) устанавливается соотнесение i в соответствии с базой данных;
- 2) устанавливается тип \mathbf{T} индивида h , выбранного из базы данных;
- 3) проверяется, удовлетворяет ли индивид h ограничению Φ ;
- 4) отбираются все такие индивиды $h' = hi$, которые кладутся в основу экстенционала концепта $C'(i)$; для него справедливо включение

$$C'(i) \text{ ISA } \mathbf{T};$$

- 5) все отобранные пары $[i, hi]$ идентифицируют индивиды h в мире i из совокупности I и принимаются за экстенционал переменного концепта $C(I)$;
- 6) сохраняется естественное включение $C(I) \subseteq H_T(I)$ для переменного домена $H_T(I)$, задаваемого определением

$$H_T(I) = \{h \mid h : I \rightarrow \mathbf{T}\}.$$

$H_T(I)$ —2. *Динамика концепта.* Наиболее действенным применением семейств индексированных концептов является возможность рассмотрения их изменения в зависимости от изменений, происшедших в предметной области. Пусть в предметной области события разворачиваются по закону f , где $f : B \rightarrow I$, то есть из мира I осуществляется переход в

мир B . В этом случае применение свойства расширяемости при вычислении значений λ -абстракций имеет определенную специфику. Для аппликации $(\lambda.\Phi)\bar{h}$ получаем:

$$\| (\lambda.\Phi)\bar{h} \| i = \| \Phi \|_f [b, (h \circ f)b]$$

для соотнесения $i = fb$. Таким образом, увязка индивида h с телом λ -абстракции Φ сводится не к проверке принадлежности индивида h значению $\| \Phi \|$, а к проверке принадлежности образа h при разворачивании событий по закону f в мире b для смещенной оценки $\| \Phi \|_f$. Следовательно, проверку принадлежности осуществляет модифицированная процедура:

- 1) устанавливается новое соотнесение B как альтернатива старому соотнесению при разворачивании событий по закону f ;
- 2) устанавливается тип \mathbf{T} образа индивида h , выбранного из образа базы данных при преобразовании f ;
- 3) проверяется, удовлетворяет ли образ индивида h ограничению Φ в мире b ;
- 4) отбираются все такие индивиды $h' = (h \circ f)$, которые кладутся в основу экстенционала концепта $C'_f(b)$; для него справедливо включение $C'_f(b) \text{ ISA } \mathbf{T}$;
- 5) все отобранные пары $[b, hb]$ идентифицируют индивиды h в мире b из совокупности B и принимаются за экстенционал переменного концепта $C_f(B)$;
- 6) сохраняется естественное включение $C_f(B) \subseteq H_T(B)$ для домена $H_T(B)$, задаваемого определением

$$H_T(B) = \{h \mid h : B \rightarrow \mathbf{T}\}.$$

$H_T(I)$ —3. *Статика концепта.* Предельным случаем закона разворачивания событий является тождественное (единичное) преобразование. Тогда запись

$$f = \mathbf{1}_I : I \rightarrow I$$

означает, что закон разворачивания событий f не приводит к изменениям в предметной области. При вычислении значения λ -абстракции получаем:

$$\| (\lambda.\Phi)\bar{h} \| i = \| \Phi \|_{1_I} [i, hi].$$

Таким образом, увязка индивида h с телом λ -абстракции Φ не требует выполнения модифицированной процедуры, а оказывается полностью аналогичной процедуре для индексированного концепта. Исследование статики концепта имеет важное следствие. Поскольку в предметной области (и в базе данных D) изменений не происходит, то отождествим соотнесение I и D , положив $I = D$. Тогда

$$C_{1_I}(I) : h \rightarrow h,$$

то есть индивид h переводится сам в себя и

$$C_{1_I}(I) = C_{1_D}(D) = 1_D.$$

С другой стороны, получаем $C_{1_I}(I) = C(I)$ и $C_I : C_I \rightarrow C_I$, откуда следует правило:

индексированный концепт ведет себя как единичное преобразование.

Более подробно, для произвольных концептов A , B и отображения $f : A \rightarrow B$ справедливы равенства

$$A = A \circ A; \quad f = B \circ f \circ A,$$

которые вытекают непосредственно из предыдущего рассмотрения и полученного правила.

$H_T(I)$ -4. *Функторная характеристика концепта.* Анализ динамики и статики концептов позволяет разделить управляющее воздействие на концепты, с одной стороны, и саму

систему концептов — с другой. Это означает, что управление осуществляет переключение системы концептов в зависимости от закона, по которому разворачиваются события. Представление системы меняющихся концептов приводит к формулированию *функторной характеристики концептов*. Существенным оказывается способ учета закона f разворачивания событий — C ведет себя как контравариантный функтор:

1) закон $f : B \rightarrow I$ понимается как переход от мира I к миру B , то есть от уровня знания I к уровню знания B и т.п.;

2) получаем: $f = 1_I \circ f \circ 1_B$ и $C_{1_B} = C(B)$, $C_{1_I} = C(I)$;

3) для концепта $C_f = C_f(B)$ справедливо следующее включение:

$$\begin{aligned} C(f) : C(I) &\rightarrow C(B); \\ C_f &\subseteq C(B), \end{aligned}$$

поскольку проверка типа \mathbf{T} для индивида h и его образа при преобразовании f сохраняет естественные включения для переменных доменов:

$$\begin{aligned} C(I) &\subseteq H_T(I) = \{h \mid h : I \rightarrow \mathbf{T}\}; \\ C(B) &\subseteq H_T(B) = \{h \mid h : B \rightarrow \mathbf{T}\}; \\ C_f &= \{h \circ f \mid h \circ f : B \rightarrow \mathbf{T}\} \subseteq C(B). \end{aligned}$$

$H_T(I)$ —5. *Индексированное отношение*. До сих пор рассматривалось индексирование одноместных концептов. В этом случае уже проявляются все основные моменты, связанные с учетом закона разворачивания событий. Однако в случае взаимодействия с базой данных требуется определение и поддержание многоместных концептов. Для упрощения изложения рассмотрим индексирование двухместного концепта, соответствующего бинарному отношению. Полная его характеристика следует из рассмотрения аппликации

$(\lambda\lambda.\Phi)\overline{u}\overline{v}$ для формулы Φ и индивидов \overline{u} и \overline{v} . Получаем:

$$\begin{aligned}
 \|(\lambda\lambda.\Phi)\overline{u}\overline{v}\| i &= (\|(\lambda\lambda.\Phi)\| i)(\|\overline{u}\| i)(\|\overline{v}\| i) \\
 &= \Lambda \| \lambda.\Phi \| i(ui)(vi) \\
 &= (\| \lambda.\Phi \| [i, ui])(vi) \\
 &= \Lambda \| \Phi \| [i, ui](vi) \\
 &= \| \Phi \| [[i, ui], vi]
 \end{aligned}$$

для соотнесения i . Возможен и иной способ рассуждения:

$$\begin{aligned}
 \|(\lambda.\Phi)[\overline{u}, \overline{v}]\| i &= \Lambda \| \Phi \| i(\|[\overline{u}, \overline{v}]\| i) \\
 &= \Lambda \| \Phi \| i[ui, vi] \\
 &= \| \Phi \| [i, [ui, vi]],
 \end{aligned}$$

при применении которого предполагается, что Φ - двухместный оператор (некаррированный). Увязка индивида $[ui, vi]$ с телом λ -абстракции Φ сводится к построению концепта, соответствующего $\| \Phi \|$, и к проверке принадлежности $[ui, vi] \in \| \Phi \|$. В данном случае индивид представляет собой упорядоченную пару, каждый элемент которой определен на соответствующем одноместном концепте:

$$\begin{aligned}
 \|(\lambda.\Psi_1)\overline{u}\| i &= \| \Psi_1 \| [i, ui] = \mathbf{U}(\{i\}); \\
 \|(\lambda.\Psi_2)\overline{v}\| i &= \| \Psi_2 \| [i, vi] = \mathbf{V}(\{i\}).
 \end{aligned}$$

Проверка принадлежности индивида $wi = [ui, vi]$ концепту $\phi = \| \Phi \|$ осуществляется согласно следующей процедуре:

- 1) устанавливается соотнесение i в соответствии с базой данных;
- 2) устанавливаются типы \mathbf{T} и \mathcal{T} индивидов ui и vi , выбранных из базы данных (и спаренных);
- 3) проверяется, удовлетворяет ли индивид wi ограничению Φ ;
- 4) отбираются все такие индивиды wi , которые кладутся в основу экстенционала концепта $\phi'(i)$; для него справедливо включение

$$\phi'(i) \text{ ISA } (\mathbf{T} \times \mathcal{T});$$

5) все отобранные пары $[i, wi]$ идентифицируют индивиды w в мире i из совокупности I и принимаются за экстенционал переменного концепта $\phi = \phi(I)$;

6) сохраняется естественное включение $\phi(I) \subseteq H_{\mathbf{T} \times \mathcal{T}}(I)$ для переменного домена $H_{\mathbf{T} \times \mathcal{T}}(I)$, задаваемого определением

$$H_{\mathbf{T} \times \mathcal{T}}(I) = \{w \mid w : I \rightarrow (\mathbf{T} \times \mathcal{T})\}.$$

$H_{\mathbf{T}}(I)$ —6. *Динамика отношения.* Учет изменений, происходящих в предметной области, проявляется в ассоциированных изменениях базы данных. Приняв в качестве закона разворачивания событий $f : B \rightarrow I$, применим свойство расширяемости при вычислении значений λ -абстракции. Для аппликации $(\lambda.\Phi)[\bar{u}, \bar{v}]$ получаем:

$$\| (\lambda.\Phi)[\bar{u}, \bar{v}] \| i = \| \Phi \|_f [b, (< u, v > \circ f)b].$$

Это выражение может быть записано в несколько модифицированном виде. Суть модификации состоит в следующем:

$$\begin{aligned} \| (\lambda.\Psi_1)\bar{u} \| (fb) &= \| \Psi_1 \|_f [b, (u \circ f)b], \\ \| (\lambda.\Psi_2)\bar{v} \| (fb) &= \| \Psi_2 \|_f [b, (v \circ f)b], \\ u \circ f &\in \mathbf{U}_f, v \circ f \in \mathbf{V}_f. \end{aligned}$$

Тогда

$$\phi_f \subseteq [\mathbf{U}_f, \mathbf{V}_f]$$

и

$$< u, v > \circ f \in \phi_f$$

для $\phi_f \subseteq (B \times \mathbf{T}) \times (B \times \mathcal{T})$. Следовательно, увязка индивида w с телом λ -абстракции Φ сводится к проверке принадлежности образа w при разворачивании событий по закону f в мире b для смещенной оценки $\| \Phi \|_f$. Детали проверки принадлежности производятся согласно следующей процедуре:

- 1) в качестве альтернативы соотнесению i устанавливается новое соотнесение b для закона разворачивания событий f ;
- 2) устанавливается тип $\mathbf{T} \times \mathcal{T}$ образа индивида w , выбранного из образа базы данных при преобразовании f ;
- 3) проверяется, удовлетворяет ли образ индивида w ограничению Φ в мире b ;
- 4) отбираются все такие индивиды $\langle u, v \rangle \circ f$, которые кладутся в основу экстенционала концепта $\phi'_f(b)$; для него справедливо включение

$$\phi'_f(b) \text{ ISA } \mathbf{T} \times \mathcal{T};$$

- 5) все отобранные пары

$$[b, (\langle u, v \rangle \circ f)b]$$

идентифицируют индивиды

$$\langle u, v \rangle \circ f$$

в мире b из совокупности B и принимаются за экстенционал переменного концепта $\phi_f(B) = \phi_f$;

- 6) сохраняется естественное включение

$$\phi_f(B) \subseteq \mathbf{U}_f(B) \times \mathbf{V}_f(B) \subseteq H_T(B) \times H_T(B)$$

для переменного домена $H_T(B) \times H_T(B)$, задаваемого определением

$$H_T(B) \times H_T(B) = \{\bar{h} \mid \bar{h} : (B \rightarrow \mathbf{T}) \times (B \rightarrow \mathcal{T})\}.$$

23.3 Интерпретация среды вычислений

Среда вычислений, представленная в виде объектов, на практике обычно оснащается управляющей структурой — системой *сценариев*, — которая определяет ее поведение. Пример реализаций таких структур хорошо известны. С интуитивной точки зрения сценарий является системой предписаний, которая задает переходы от одних состояний — *старых состояний* вычисления, — к другим — *новым состояниям*.

Это достаточно специальная точка зрения на вычисления, хотя и предоставляющая ряд удобств. Для ее осуществления требуется рассматривать *параметризованные объекты*. В качестве основной модели объекта избирается представление о *переменном множестве* — переменном концепте, — сравнительно новым в computer science представлении. Объем интуитивного знания о том, что такое объект, заменяется на идеализированную сущность, называемую *функтор-как-объект*. Как известно, функторы позволяют оперировать языками высших порядков, когда объект рассматривается как отображение *из* объектов *в* объекты.

Владение таким понятийным запасом и соответствующей техникой рассуждений, не говоря о технике их реализации, относят к возможности работать на “переднем крае” работ в области программирования.

Библиография

- [1] Аксенов К.Е., Баловнев О.Т., Вольфенгаген В.Э., Воскресенская О.В., Ганночка А.В., Чуприков М.Ю. *Лабораторный практикум "Системы искусственного интеллекта"*. — М.: МИФИ, 1985. — 92 с.
- [2] Барендрегт Х. *Лямбда-исчисление. Его синтаксис и семантика*. — М.: Мир, 1985.
- [3] Бердж В. *Методы рекурсивного программирования*. — М.: Машиностроение, 1983.
- [4] Белнап Н., Стил Т. *Логика вопросов и ответов*. — М.: Прогресс, 1981. — 288 с.
- [5] Булкин М.А., Габович Ю.Р., Пантелеев А.Г. *Методические рекомендации по программированию и эксплуатации интерпретатора для алгоритмического языка ЛИСП в ОС ЕС*. — Киев : НИИАСС, 1981. — 91 с.
- [6] Вольфенгаген В.Э. *Вычислительная модель реляционного исчисления, ориентированного на представление знаний*. — М.: препринт МИФИ, 004-84, 1984.
- [7] Вольфенгаген В.Э., Яцук В. Я. *Вычислительная модель реляционной алгебры*. — Программирование, № 5, М.: АН СССР, 1985. с. 64-76.

- [8] Вольфенгаген В.Э., Сагоян К.А. *Методические указания к проведению практических занятий по курсу "Дискретная математика". Специальные главы дискретной математики.* — М.: МИФИ, 1987. — 56 с.
- [9] Вольфенгаген В.Э., Аксенов К.Е., Исмаилова Л. Ю., Волшаник Т. В. *Лабораторный практикум по курсу "Дискретная математика. Аппликативное программирование и технология поддержки реляционных систем".* — М.: МИФИ, 1988. — 56 с.
- [10] Вольфенгаген В.Э., Яцук В.Я. *Аппликативные вычислительные системы и концептуальный метод проектирования систем знания.* — МО СССР, 1987.
- [11] Вольфенгаген В.Э., Чепурнова И.В., Гаврилов А.В. *Методические указания к проведению практических занятий по курсу "Дискретная математика". Специальные главы дискретной математики.* — М.: МИФИ, 1990. — 104 с.
- [12] Вольфенгаген В.Э., Гольцева Л.В. *Аппликативные вычисления на основе комбинаторов и λ -исчисления.* — (Руководитель проекта "Аппликативные вычислительные системы" к.т.н. Л.Ю. Исмаилова.) — М.: МИФИ, 1992. — 41 с.
Основы аппликативных вычислительных систем изложены элементарными средствами, что обеспечивает студентов и аспирантов кратким и исчерпывающим руководством, которое может использоваться 'для первого чтения'. Настоящее руководство в различных вариантах течение ряда лет использовалось для проведения практических занятий и лабораторных работ по соответствующим разделам курса компьютерных наук. Охватываются вопросы использования комбинаторов и λ -исчисления при реализации аппликативных вычислений, Приводится необходимый теоретический минимум, основное внимание уделено выполнению упраж-

нений, иллюстрирующих применения основных вычислительных идей, понятий и определений. Для облегчения овладения предметом руководство снабжено простой обучающей программой, которая может использоваться в качестве вводного лабораторного практикума. При практической работе с обучающей программой следует иметь ввиду, что решение задач предполагает проведение дополнительных преобразований с целью оптимизации выражений, устранения в них переменных, упрощения целевого исполняемого выражения. Аппликативные вычисления излагаются как набор таких методов и средств.

Для студентов и аспирантов всех специальностей. Может быть использована для первоначального самостоятельного изучения предмета.

- [13] Вольфенгаген В.Э. *Теория вычислений*. — М.:МИФИ, 1993. — 96 с.
- [14] Вольфенгаген В.Э. *Категориальная абстрактная машина*.— М.:МИФИ, 1993. — 96 с.; — 2-е изд. — М.: АО “Центр ЮрИнфоР”, 2002. — 96 с.
- [15] Вольфенгаген В.Э. *Проектирование языков программирования и теория вычислений*.— М.:МИФИ, 1993. — 189 с.
- [16] Вольфенгаген В.Э. *Конструкции языков программирования. Приемы описания*.— М.: АО “Центр ЮрИнфоР”, 2001. — 276 с.

В работе изложены основы, касающихся разработки, реализации и применения конструкций как императивных, так и функциональных языков программирования. Значительное внимание уделяется применению денотационной семантики, позволяющей в полной мере извлечь преимущества объектно-ориентированного подхода, что, в конечном счете, позволяет построить результирующую вычислительную модель чисто функционального типа.

Изложение материала сопровождается детально разобранными которые снабжены комментариями, помогающими уяснить реализацию конструкций различных языков.

Книгу можно использовать в качестве учебника или справочного пособия. Она будет полезна как студентам и аспирантам, так и профессионалам в области компьютерных наук, информационных технологий и программирования.

- [17] Вольфенгаген В.Э. *Логика. Конспект лекций: техника рассуждений*.— М.: АО “Центр ЮрИнфоР”, 2001. — 137 с. В работе рассмотрены способы переформулирования текста фактического типа на символьный язык, допускающий применение классических логических средств. Показаны приемы и способы записи аргументации и проверки ее значимости. На большом числе примеров проиллюстрирована техника логических рассуждений, выводов и доказательств. Отмечены способы включения в вывод аннотаций (комментариев), пользуясь которыми можно проверить истинность или установить ложность приводимых выводов.
Для студентов и аспирантов гуманитарных специальностей. Может быть использована для первоначального изучения предмета, а также для самостоятельного изучения.
- [18] Голдблатт Р. *Топосы: категорный анализ логики*.— М.: Мир, 1985.
- [19] Джонстон П. Т. *Теория топосов*.— М.: Наука, 1986.
- [20] Захарьящев М.В., Янов Ю.И. (ред.) *Математическая логика в программировании*.— М.:Мир, 1991. — 408 с.
- [21] Илюхин А.А., Исмаилова Л.Ю., Шаргатова З.И. *Экспертные системы на реляционной основе*.— М.: МИФИ, 1990.
- [22] Карри Х.Б. *Основания математической логики*.— М.: Мир, 1969.

- [23] Клини С.К. *Введение в метаматематику*.— М.: ИЛ, 1957.
- [24] Кузин Л.Т. *Основы кибернетики, т.2.* — М.: Энергия, 1979, 15-9.
- [25] Кузин Л.Т. *Основы кибернетики. Т.1. Математические основы кибернетики: Учеб. пособие для вузов.* — 2-е изд., перераб. и доп. — М.: Энергоатомиздат, 1994. — 576 с.
- [26] Кузичев А.С. *Некоторые свойства комбинаторов Шейнфинкеля-Карри*.— Комбинаторный анализ, вып. 1. Изд-во МГУ, 1971.
- [27] Кузичев А.С. *Дедуктивно-комбинаторное построение теории функциональностей*.— ДАН СССР, 1973, т. 209, № 3.
- [28] Кузичев А.С. *Непротиворечивые расширения чистой комбинаторной логики*.— Вестн. Моск. ун-та, матем., механ., No 3, 76-81, 1973.
- [29] Кузичев А.С. *О предмете и методах комбинаторной логики*.— История и методология естественных наук, М.:МГУ, вып.14, 1973, с. 131-141.
- [30] Кузичев А.С. *Система лямбда-конверсии с дедуктивным оператором формальной импликации*.— ДАН СССР, 212, № 6, 1973, 1290-1292.
- [31] Кузичев А.С. *Дедуктивные операторы комбинаторной логики*.— Вестн. Моск. ун-та, матем., механ., № 3, 13-21, 1974.
- [32] Кузичев А.С. *О выразительных возможностях дедуктивных систем лямбда-конверсии и комбинаторной*

- логики.— Вестн. Моск. ун-та, матем., механ., № 6, 19-26, 1974.
- [33] Кузичев А.С. *Принцип комбинаторной полноты в математической логике*.— История и методология естественных наук, сб. МГУ, вып. 16, 1974. — с. 106-127.
- [34] Кузичев А.С. *Комбинаторно полные системы с операторами $\Xi, F, Q, \Pi, \exists, P, \neg, \&, \vee, =$* .— Вестн. Моск. ун-та, сер. матем., мех., № 6, 1976.
- [35] Кузичев А.С. *Операция подстановки в системах с неограниченным принципом комбинаторной полноты*.— Вестн. Моск. ун-та, сер. матем., мех., № 5, 1976.
- [36] Мальцев А.И. *Алгоритмы и рекурсивные функции*.— М.: Наука, 1965.
- [37] Марков А.А. *Невозможность некоторых алгоритмов в теории ассоциативных систем*.— ДАН СССР, 1947, т. 55, № 7; т. 58, №3.
- [38] Марков А.А. *О логике конструктивной математики*.— Вестн. Моск. ун-та, матем., механ., № 2, 7-29, 1970.
- [39] Марков А.А. *О логике конструктивной математики*.— М.: Знание, 1972.
- [40] Мендельсон Э. *Введение в математическую логику*.— М.: Наука, 1971.
- [41] Пантелеев А.Г. *Об интерпретаторе с языка ЛИСП для ЕС ЭВМ*. — Программирование, 1980, № 3, с. 86-87
- [42] Пратт Т., Зелковиц М. *Языки программирования. Разработка и реализация*. — СПб.: Питер, 2002. — 688 с.

- [43] Себеста Р. *Основные концепции языков программирования*. 5-е изд.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2001. — 672 с.
- [44] Скотт Д.С. *Советы по модальной логике*.— Семантика модальных и интенциональных логик (ред.: Смирнов В.А.). — М.: Прогресс, 1981, с. 280-317 (Scott D. Advice on modal logic.— Philosophical problems in logic. Some recent developments.— Lambert K. (ed.), Reidel, Dordrecht, Holland, 1970).
- [45] *Семантика модальных и интенциональных логик*.— Под ред. Смирнова В.А. — М.: Прогресс, 1981. — 424 с.
- [46] Смирнов В.А., Карпенко А.С., Сидоренко Е.А. (ред.) *Модальные и интенциональные логики и их применение к проблемам методологии науки*.— М.:Наука, 1984.— 368 с.
- [47] Стогний А.А., Вольфенгаген В.Э., Кушников В.А., Саркисян В.И., Араксян В.В., Шитиков А.В. *Проектирование интегрированных баз данных*.— Киев: Техніка, 1987.
- [48] Такеути Г. *Теория доказательств*.— М.: Мир, 1978.
- [49] Фурман М. *Логика топосов*.— Справочная книга по математической логике: В 4-х частях (под. ред. Дж. Барвайса).— Ч. IV. Теория доказательств и конструктивная математика: Пер. с англ. — М.: Наука. Главная редакция физико-математической литературы, 1983. — с. 241-277
- [50] Хендерсон П. *Функциональное программирование. Применение и реализация*.— М.: Мир, 1983.
- [51] Хопкрофт Дж., Мотвани Р., Ульман Дж. *Введение в теорию автоматов, языков и вычислений*. 2-е изд.: Пер. с англ.— М.: Издательский дом “Вильямс”, 2002.— 528 с.

- [52] Шабунин Л.В. *О непротиворечивости некоторых исчислений комбинаторной логики.*— Вестн. Моск. ун-та, сер. матем. мех., 1971, №6.
- [53] Шенфилд Дж. *Математическая логика.*— М.: Наука, 1975.
- [54] Энгелер Э. *Метаматематика элементарной математики.*— М.: Мир, 1986.
- [55] Яновская С.А. *Основания математики и математическая логика.*— Математика в СССР за тридцать лет. 1917-1947.— М.-Л.: Гостехиздат, 1948.
- [56] *Nested relations and complex objects in databases.*— Lecture Notes in Computer Science, 361, 1989.
- [57] Amadio R.M., Curien P.-L. *Domains and lambda-calculi.*— Cambridge University Press, 1998. — 484 p.
В книге изложены математические аспекты семантики языков программирования. Основными целями являются построение формальных средств для анализа значения конструкторов программирования как независимым от языка, так и независимым от машины способами, а также для обоснования свойств программ, например, их завершаемости либо возможности получения решения той задачи, для которой они предназначены.
Преследуется двойная цель: дать специалистам в компьютерных науках необходимую мотивацию для получения математических результатов, а специалистам в математике указать на возможные приложения в области компьютерных наук.
- [58] Appel A. *Compiling with continuations.*— Cambridge University Press, 1992.
- [59] Avron A., Honsel F., Mason I., and Pollak R. *Using typed lambda-calculus to implement formal systems on a machine.*— Journal of Automated Reasoning, 1995.

- [60] Backus J.W. *Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.*— Comm. ACM, 1978, v.21, № 8, p. 614-641.
- [61] Backus J.W. *The algebra of functional programs: functional level reasoning, linear equations and extended definitions.*— Int. Col. on formalization of programming concepts, LNCS, v. 107, 1981, pp. 1-43.
- [62] Backus J.W., Williams J.H., Wimmers E.L. *FL language manual (preliminary version).*— IBM research report No RJ 5339(54809), 1987.
- [63] Banerji R.B. (ed.) *Formal techniques in artificial intelligence: a sourcebook.*— Studies in computer science and artificial intelligence, 6, North-Holland, 1990.
- [64] Beery G., Levy J-J. *Minimal and optimal computations of recursive programs.*— J. Assoc. Comp. Machinery, Vol.26, No 1, 1979.
- [65] Belnap N.D. (Jr.) *A useful four-valued logic.* Modern uses of multiple-valued logic.— Epstein G., Dunn J.M. (eds.) Proceedings of the 1975 International Symposium of multiple-valued logic, Reidel, 1976.
- [66] Belnap N.D. (Jr.) *How a computer should think.*— Contemporary aspects of philosophy, Proceedings of the Oxford International Symposium, 1976.
- [67] Bird R.S. *An introduction to the theory of lists.*— Logic programming and calculi of discrete design (ed. Broy M.), Springer-Verlag, 1986, pp. 5-42.
- [68] Böhm C. (ed.) *Lambda calculus and computer science theory.*— Proceedings of the Symposium held in Rome. March 25-29, LNCS, vol.37, Berlin: Springer 1975.

- [69] Bonsanque M. *Topological dualities in semantics*.— PhD thesis, Vrije Universiteit Amsterdam, 1996.
- [70] Bunder M.V.W. *Set Theory based on Combinatory Logic*.— Doctoral thesis, University of Amsterdam, 1969.
- [71] Bunder M.V.W. *Propositional and predicate calculuses based on combinatory logic*.— Notre Dame Journal of Formal Logic, Vol. XV, 1974, pp. 25-34.
- [72] Bunder M.V.W. *The naturalness of illative combinatory logic as a basis for mathematics*.— To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.— Seldin J.P., Hindley J.R. (eds.), Academic Press, 1980, pp. 55-64.
- [73] Bucciarelli A. *Logical reconstruction of bi-domains*.— In: Proc. Typed Lambda Calculi and Applications, Springer Lecture Notes in Comp. Sci., 1210, 1997.
- [74] Church A. *The calculi of lambda-conversion*.— Princeton. 1941, ed. 2, 1951.
- [75] Coppo M., Dezani M., Longo G. *Applicative information systems*.— LNCS, 159, 1983, pp. 35-64.
- [76] Cousineau G., Curien P.-L., Mauny M. *The categorical abstract machine*.— LNCS, 201, Functional programming languages computer architecture.— 1985, pp. 50-64.
- [77] Curien P.-L. *Categorical combinatory logic*.— LNCS, 194, 1985, pp. 139-151.
- [78] Curien P.-L. *Typed categorical combinatory logic*.— LNCS, 194, 1985, pp. 130-139.
- [79] Curry H.B., Feys R. *Combinatory logic*.— Vol. 1. Amsterdam: North-Holland, 1958.

- [80] Curry H.B., Hindley J.R., Seldin J.P. *Combinatory logic*.— Vol. II. Amsterdam, 1972.
- [81] Curry H.B. *Some philosophical aspects of combinatory logic*.— Barwise J., Keisler H.J., Kunen K. (eds.) *The Kleene Symposium*.— North-Holland Publ. Co, 1980, p. 85-101.
- [82] Danforth S., Tomlison C. *Type theories and object oriented programming*.— *ACM Computing Surveys*, 1988, v.20, No 1, pp. 29-72.
- [83] Darlington J., Henderson P., Turner D.A. (eds.) *Functional programming and its applications*.— Cambridge Univ. Press, Cambridge, 1982.
- [84] de Bruijn N.G. *Lambda-calculus notations with nameless dummies: a tool for automatic formula manipulation*.— *Indag. Math.* 1972, N 34, pp. 381-392.
- [85] Eisenbach S. (ed.) *Functional programming: languages, tools and architectures*.— Chichester: Horwood, 1987.
- [86] Fasel J.H., Keller R.M. (eds.) *Graph reduction*.— LNCS, 279, 1986.
- [87] Fenstad J.E. et al. *Situations, language and logic*. — Dordrecht: D. Reidel Publ. Comp., 1987.
- [88] Fitting M. *First-order logic and automated theorem proving*.— Springer-Verlag, 1990.
- [89] Frandsen G.S., Sturtivant C. *What is an efficient implementation of the lambda-calculus?*— LNCS, 523, 1991, pp. 289-312.
- [90] Friedman H. *Equality between functionals*.— *Proceedings of the Symposium on Logic*. Boston, 1972-1973.— *Lecture Notes in Mathematics*, 453, 1975, pp. 22-37.

- [91] Gardenfors P. *Induction, Conceptual spaces and AI.*— Proceedings of the workshop on inductive reasoning, Riso National Lab, Roskilde, 1987.
- [92] Henson M.S. *Elements of functional languages.*— Oxford: Blackwell, 1987.
- [93] Hindley J.R. *The principal type-scheme of an object in combinatory logic.*— Trans. Amer. Math. Soc., 1969, vol. 146.
- [94] Hindley J., Lercher H., Seldin J. *Introduction to combinatory logic.*— Cambridge University Press, 1972.
- [95] Howard W. *The formulas-as-type notion of construction.*— Seldin J.P., Hindley J.R. (eds.), To H.B. Curry: Essays on combinatory logic, lambda-calculus and formalism.— Amsterdam: Academic Press, 1980.
- [96] Hughes R.J.M. *Super combinators: a new implementation method for applicative languages.*— Proceedings of the 1982 ACM symposium on LISP and functional programming, pp. 1–10.
- [97] Hughes R.J.M. *The design and implementation of programming languages.*— PhD Thesis, University of Oxford, 1984.
- [98] Hunt L.S. *A Hope to FLIC translator with strictness analysis.*— MSc dissertation, Department of Computing, Imperial College, University of London, 1986.
- [99] Kelly P.H.J. *Functional languages for loosely-coupled multiprocessors.*— PhD Thesis, Imperial College, University of London, 1987.
- [100] Lambek J., Scott P.J. *Introduction to higher order categorical logic.* — Cambridge Studies in Advanced Mathematics 7, Cambridge University Press, 1986, 1988, 1989, 1994. — 293 p.

- [101] Landin P. *The next 700 programming languages*.— Communications of the ACM, 3, 1966.
- [102] McCarthy J. *A basis for a mathematical theory of computation*.— Computer programming and formal systems (eds.: Braffort and Hirshberg), Amsterdam: North-Holland, 1963, pp. 33-69.
- [103] Michaelson G. *An introduction to functional programming through lambda-calculus*. — Addison Wesley Publ.Co, 1989, 320 p.
- [104] Milner R., Parrow J., Walker D. *A calculus of mobile process*. — Parts 1-2. — Information and Computation, 100(1), 1992, pp. 1-77
- [105] Mycroft A. *Abstract interpretation and optimizing transformations for applicative programmes*.— PhD Thesis, Department of Computer Science, University of Edinburgh, 1981.
- [106] Peyton Jones S.L. *The implementation of functional programming languages*.— Prentice Hall Int., 1987.
- [107] Pitts A. *Relational properties of domains*. — Information and Computation, 127, 1996, pp. 66-90.
- [108] Rosser J.B. *A mathematical logic without variables*.— Ann. of Math., vol. 36, No 2, 1935. pp. 127-150.
- [109] Schönfinkel M. *Über die Bausteine der mathematischen Logik*.— Math. Annalen, vol. 92, 1924, pp. 305-316.
- [110] Schroeder-Heister P. *Extensions of logic programming*.— LNAI, 475, 1991.

- [111] Scott D.S. *Advice on modal logic*.— Philosophical problems in logic. Some recent developments.— Lambert K. (ed.), Dordrecht; Holland: Reidel, 1970.
- [112] Scott D.S. *Outline of a mathematical theory of computation*.— Proceedings of the 4-th Annual Princeton conference on information sciences and systems, 1970.
- [113] Scott D.S. *The lattice of flow diagrams*.— Lecture Notes in Mathematics, 188, Symposium on Semantics of Algorithmic Languages.— Berlin, Heidelberg, New York: Springer-Verlag, 1971, pp. 311-372.
- [114] Scott D.S. *Identity and existence in intuitionistic logic*.— In: Applications of Sheaves. Berlin: Springer, 1979, pp. 660-696.
- [115] Scott D.S. *Lambda calculus: some models, some philosophy*.— The Kleene Symposium. Barwise, J., et al.(eds.), Studies in Logic 101, North-Holland, 1980, pp. 381-421.
- [116] Scott D.S. *Relating theories of the lambda calculus*.— Hindley J., Seldin J. (eds.) To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.— N.Y.& L.: Academic Press, 1980, pp. 403-450.
- [117] Scott D.S. *Lectures on a mathematical theory of computation*.— Oxford University Computing Laboratory Technical Monograph PRG-19, 1981. — 148 p.
- [118] Scott D.S. *Domains for denotational semantics*.— LNCS, 140, 1982, pp. 577-613.
- [119] Stoy J.E. *Denotational semantics: The Scott-Strachey approach to programming language theory*.— M.I.T. Press, Cambridge, Mass., 1977.— xxx+414 p.

- [120] Stoye W.R. *The implementation of functional languages using custom hardware*.— PhD Thesis, University of Cambridge, 1985.
- [121] Szabo M.E. *Algebra of proofs*.— Studies in Logic foundations of mathematics, v. 88. North-Holland Publ. Co, 1978.— 297 p.
- [122] Talcott C. *Rum: An intensional theory of function and control abstractions*.— Foundations of logic and functional programming, LNCS, 306, 1986, pp. 3-44.
- [123] Tello E.R. *Object-Oriented Programming for Windows / Covers Windows 3.x*.— Wiley and Sons, Inc., 1991.
- [124] Turner D.A. *A New Implementation Technique for Applicative Languages*.— Software Practice and Experience.— No 9, 1979, pp. 31-49.
- [125] Turner D.A. *Aspects of the implementation of programming languages*.— PhD Thesis, University of Oxford, 1981.
- [126] Turner R. *A theory of properties*.— J. Symbolic logic, v. 52, 1987, pp. 455-472.
- [127] Wodsworth C.P. *Semantics and pragmatics of the lambda calculus*.— PhD Thesis, University of Oxford, 1981.
- [128] Wolfengagen V.E. *Frame theory and computations*.— Computers and artificial intelligence. V.3, No 1, 1984, pp. 1-31.
- [129] Wolfengagen V.E. *Building the access pointers to a computational environment*. — In: electronic Workshops in Computing, Berlin Heidelberg New York: Springer-Verlag, 1998. pp. 1-13
<http://ewic.springer.co.uk/adbis97/>

- [130] Wolfengagen V.E. *Event driven objects*. — In: Proceedings of the 1st International Workshop on Computer Science and Information Technologies, Moscow, Russia, 1999, Vol. 1. pp. 88-96
- [131] Wolfengagen V.E. *Functional notation for indexed concepts*. — In: Proceedings of The 9th International Workshop on Functional and Logic Programming WFLP'2000, Benicassim, Spain, September 28-30, 2000
<http://www.dsic.upv.es/~wflp2000/>
- [132] Zhang *The largest cartesian closed category of stable domains*. — Theoretical Computer Science, 166, 1995, pp. 203-219.
- [133] <http://www.rbjones.com>
Этот электронный ресурс FACTASIA имеет своей целью “развить предвидение будущего и дать ресурсы для построения такого видения и самого будущего”, а также “сделать вклад в те ценности, которые определяют будущее, и в те технологии, которые помогают его строить”. Воспользовавшись навигационными средствами в разделе Logic можно найти анализ основных разделов *комбинаторной логики* и *λ -исчисления*, библиографию по данному вопросу, а также важнейшие связи с другими разделами.
См. <http://www.rbjones.com/rbjpub/logic/cl/>
- [134] <http://ling.ucsd.edu/~barker/Lambda/ski.html>
Представлен простой учебник по комбинаторной логике в режиме on-line.
- [135] <http://www.cwi.nl/~tromp/cl/cl.html>
Представлено руководство, необходимые библиографические ссылки и Java-апплет, позволяющий интерпретировать объекты — выражения аппликативного языка, — которые строятся по правилам комбинаторной логики.

[136] <http://www.brics.dk>

BRICS — Basic Research in Computer Science (Фундаментальные исследования в компьютерных науках). Исследовательский центр и Международная школа аспирантов. В разделе “BRICS Publications” представлена “Lecture Series”, в которой содержатся материалы курсов в 9-ти основных областях: *дискретной математики, семантики вычислений, логики в компьютерных науках, сложности вычислений, построения и анализа алгоритмов, языков программирования, тестирования, распределенных вычислений, криптологии и безопасности данных.*

[137] <http://www.afm.sbu.ac.uk>

Formal Methods (Формальные методы). Документ содержит ряд указателей, имеющих в Web на формальные методы, которые полезны для математического описания и обсуждения свойств компьютерных систем. Особое внимание отведено тем формальным методам, которые полезны для сокращения числа ошибок, возникающих в системах, в особенности на ранних этапах их проектирования. Эти методы являются дополнительными по отношению к такому методу выявления ошибок, как тестирование.

[138] <http://liinwww.ira.uka.de/bibliography/>

The Collection of Computer Science Bibliographies (Коллекция библиографий по компьютерным наукам). Представлена коллекция библиографий научной литературы в области компьютерных наук, которая собрана из разнообразных источников и покрывает многие их аспекты. Библиографии ежемесячно обновляются в связи с их исходным размещением, поэтому поддерживаются наиболее актуальные версии. Коллекция включает более 1.2 млн. ссылок на журнальные статьи, доклады на конференциях и технические отчеты, сгруппированные в виде приблизительно 1400 библиографий. Более 150000 ссылок содержат URLы на электронные версии статей, имеющих в доступе on-line.

- [139] <http://www.ncstr1.org>
NCSTR1 – Networked Computer Science Technical Reference Library (Сетевая библиотека технических ссылок по компьютерным наукам.) Эта библиотека создана в рамках инициативы Open Arhives Initiative (<http://www.openarchives.org>).
- [140] <http://xxx.lanl.gov/archive/cs/intro.html>
CoRR – The Computing Research Repository (Исследовательский репозиторий в области компьютеринга). Содержит статьи, начиная с 1993 г. в области компьютерных наук, которые классифицируются двумя способами: по предметному признаку и с применением Классификационной Системы ACM по компьютерингу 1998 года. Классификационная схема ACM относительно стабильна и покрывает все разделы компьютерных наук. Предметные же признаки не являются взаимно исключающими и не претендуют на исчерпание всех разделов. Тем не менее они отражают области активного исследования в компьютерных науках.
- [141] <http://web.cl.cam.ac.uk/DeptInfo/CST/node13.html>
Foundations of Computer Science (Основания компьютерных наук).
Курс лекций Кембриджского университета, задачей которого является закладывание базовых основ программирования.
- [142] <http://web.comlab.ox.ac.uk/oucl/courses/topics01-02/lc>
Lambda Calculus (Ламбда-исчисление).
Курс лекций Оксфордского университета, который охватывает формальную теорию, системы перезаписи, комбинаторную логику, полноту по Тьюрингу и системы типов. Является кратким введением во многие отрасли компьютерных наук, демонстрируя их связи с ламбда-исчислением.

Предметный указатель

Curry, 147

unCurry, 148

‘+1’

$\sigma = \lambda xyz.xy(yz)$, 89

$\hat{\sigma} \equiv [xyz](y(xyz))$, 82

Аксиома

(FI), 96

(FK), 96

(FS), 96

Базис

I, K, S , 114

I, B, C, S , 120

Декартово произведение, 150

Дескрипция

I, 76

Домен

$H_T(I)$, 276

Значение

$\| (M N) \|$, 228

$\| \lambda x.M \|$, 228

$\| c \|$, 228

$\| x \|$, 228

Индивид, 272

Инструкция

car, 239

cdr, 239

cons, 239

cur, 239

push, 239

quote, 239

skip, 246

swap, 239

Каррирование

Curry, 147

Категория

$C(\mathcal{L})$, 150

декартово замкнутая, 162

Код

категориальный, 245

оптимизированный, 245

Код де Брейна

$\underline{0}$, 228

$\underline{1}$, 228

Комбинатор

B , 37, 52

B^2 , 37, 57

B^3 , 37, 57

C , 37, 54

$C^{[2]}$, 37, 58

$C^{[3]}$, 37, 60

$C_{[2]}$, 37, 59

- $C_{[3]}$, 37, 60
- F , 38
- $I = \lambda x.x$, 157
- I , 35
- $K = \lambda xy.x$, 157
- P , 38
- W , 37, 55
- Y , 37, 62, 68, 69, 71, 72, 201
- Y_0 , 38, 71, 72
- Y_1 , 38, 71, 72
- $\&$, 38
- Φ , 37, 61
- Ψ , 37, 56
- Ξ , 38
- \exists , 38
- \neg , 38
- \vee , 38
- дубликатор, W , 50
- канцелятор, K , 51
- композитор, B , 50
- коннектор, S , 51
- минимума \min , 85
- пары \mathcal{D} , 83
- пермутатор, C , 51
- предшествования π , 84
- разности \mathbb{R} , 86
- следования за $\hat{\sigma}$, 82
- сложения A , 85
- тождества, I , 50
- умножения M , 86
- усеченного вычитания \mathbb{L} , 84
- Композиция
 - $f \circ g$, 150
- Конструктор
 - Append*, 164
 - Car*, 164
 - Cdr*, 164
 - List*, 164
 - Nil*, 164
 - Null*, 164
 - else*, 128
 - fi*, 213
 - hd*, 202
 - if*, 128, 202
 - if-FALSE*, 216
 - if-TRUE*, 216
 - in*, 223
 - let*, 223
 - then*, 128
 - tl*, 202
 - where*, 223
- Концепт, 272
 - индивидный, 272
 - предикатный, 155
- Нумерал
 - $\bar{n} = (SB)^n(KI)$, 88
 - $\bar{n} = \lambda xy.(x^n)y$, 88
 - $\bar{0}$, 88
 - $\bar{1}$, 88
- Оболочка Каруби
 - \mathcal{L} , 150
- Объект
 - Append*, 44
 - Car*, 44
 - Cdr*, 44, 90
 - Fst*, 240, 248
 - List*, 44
 - Nil*, 44, 90
 - Null*, 44, 90
 - Snd*, 240, 248
 - Y , 129
 - Λ , 42, 232, 240
 - S , 231, 240
 - ε , 42, 232, 240

- $\varepsilon_{BC} : (B \rightarrow C) \times B \rightarrow C$,
 151
append, 41, 128, 135
car, 127, 134, 135
cdr, 128, 134, 135
concat, 41, 128, 135
false, 134
 $h : (A \times B) \rightarrow C, \Lambda_{ABCh} :$
 $A \rightarrow (B \rightarrow C)$, 151
identity, 135
length, 41, 90, 128, 135
list, 134
list1, 40, 133
map, 41, 128, 135
null, 129, 134
postfix, 135
product, 41, 128, 135
reverse, 135
sum, 41, 128, 135
sumsquares, 135
times, 128, 204
true, 134
 арифметический
 \mathbb{Z}_0 , 82
 \mathbb{A} , 85
 \mathbb{L} , 84
 \mathbb{M} , 86
 \mathbb{R} , 86
 $\overline{\min}$, 85
 π , 84
 $\hat{\sigma}$, 82
 способ комбинирования, 18
- Пара
 $[f, g]$, 254
 $[x, y] = \lambda r. rxy$, 145, 157
 $[x, y]$, 140
- Параметр
 подстановочный, 49
 фактический, 49
 формальный, 49
- Погружение, 11
- Полнота
 комбинаторная, 75
- Постулат
alpha, α , 36
eta, η , 77
mu, μ , 36
nu, ν , 36
sigma, σ , 36
tau, τ , 36
xi, ξ , 36
- Правило
 (F) , 96, 97
 (λ) , 97
 характеристическое, 46
- Предикат
 переменный, 156
- Принцип
 $(Beta)$, 245
 $[\oplus]$, 248
 свертывания, 276
 свертывающая, 155
- Проекция
 $p : A \times B \rightarrow A$, 141
 $q : A \times B \rightarrow B$, 141
- Равенство
 (ac) , 231
 (ass) , 231
 $(dpair)$, 231
 (fst) , 231
 $(quote)$, 231
 (snd) , 231
 $< Id, g > = < g >$, 246

- $\langle M, N \rangle w = (Mw, Nw)$, 235
 $\langle f, g \rangle = \lambda t. [f(t), g(t)]$, 140
 $[x, y] = \lambda r. rxy$, 140
 $\lambda xy. xy = \lambda x. x$, 77
 $a \circ b = \lambda x. a(bx)$, 150
 $id\ a = a$, 150
- Рекурсия
 стек, 65
- Соотнесение, 272
- Состояние, 272
- Спаривание
 $\langle f, g \rangle$, 254
 $\langle x, y \rangle$, 140
- Суперкомбинатор
 $alpha$, 203, 215
 $beta$, 203, 215
 $gamma$, 203, 215
- Теория
 метатеория, 11
 подтеория, 11
- Терм
 $\lambda V. E$, 201
 $\lambda x. P$, 40
 $\lambda x. PQ$, 41
- Тип
 $\#(B)$, 99
 $\#(B^2)$, 104
 $\#(B^3)$, 105
 $\#(C)$, 111
 $\#(C^{[2]})$, 105
 $\#(C^{[3]})$, 106
 $\#(C_{[2]})$, 107
 $\#(C_{[3]})$, 108
 $\#(D)$, 110
 $\#(SB)$, 100
 $\#(W)$, 103
 $\#(X)$, 95
 $\#(Y)$, 109
 $\#(Z^n)$, 102
 $\#(Z_0)$, 100
 $\#(Z_1)$, 101
 $\#(\Phi)$, 109
- Функция
 $Curry_{ABC}$, 145
 Λ_{ABC} , 141
 $\Lambda_{ABC}\ h$, 141
 \oplus , 224
 $\varepsilon \circ \langle k \circ p, q \rangle: A \times B \rightarrow C$, 141
 ε_{BC} , 141
 el , 202, 213
 fac , 204, 216, 222
 $h: A \times B \rightarrow C$, 145
 $plus$, 223
 предикатная, 155
- Характеристика
 комбинаторная, 46
- Число
 комбинаторное Z_0 , 82
- Язык
 Lisp, 164

Глоссарий

Алгебра

Алгеброй часто считают систему, вовсе не использующую связанных переменных, то есть все используемые переменные являются свободными.

Алгоритм (неформально)

Алгоритмом считается детерминированная процедура, которую можно применять к любому элементу некоторого класса символических *входов* и которая для каждого такого входа дает в конце концов соответствующий символический *выход*.

Существенные черты алгоритма:

- *1) алгоритм задается как набор инструкций конечных размеров;
- *2) имеется вычислитель, который умеет обращаться с инструкциями и производить вычисления;
- *3) имеются возможности для выделения, запоминания и повторения шагов вычисления;
- *4) пусть P — набор инструкций в соответствии с *1), а L — вычислитель из *2). Тогда L взаимодействует с P так, что для любого данного входа вычисление происходит дискретным образом по шагам, без использования аналоговых устройств и соответствующих методов;

- *5) L взаимодействует с P так, что вычисление продвигается вперед детерминированно, без обращения к случайным методам или устройствам, например к игральным костям.

Алфавит

Алфавитом считается определенный набор объектов, называемых *символами* или буквами, которые обладают свойством неограниченной воспроизводимости (на письме).

Аксиоматическая теория множеств

Характеристики этой теории (см., например, [22]): (1) пропозициональные функции рассматриваются экстенционально (функции, имеющие одни и те же значения истинности для одних и тех же аргументов, отождествляются), (2) пропозициональные функции более чем от одного аргумента сводятся к пропозициональным функциям одного аргумента, т. е. к классам; (3) имеется класс, элементы которого называются множествами; класс может быть элементом другого класса тогда и только тогда, когда он является множеством; (4) множества характеризуются генетически, согласно их построению, чтобы слишком обширные классы, например, класс всех множеств, не допускались в качестве множеств.

Аппликативные вычислительные системы

Традиционно в состав *аппликативных вычислительных систем*, или АВС, включают системы исчислений объектов, основанные на комбинаторной логике и лямбда-исчислении. Единственное, что существенно разрабатывается в этих системах — это представление об *объекте*. В комбинаторной логике единственный метаоператор — *аппликация*, или, по иной терминологии, *приложение* одного объекта к другому. В лямбда-исчислении два метаоператора — *аппликация* и функциональная *абстракция*, позволяющая связывать одну переменную в одном объекте.

Возникающие в этих системах объекты ведут себя как функциональные сущности, имеющие следующие особенности:

число аргументных мест, или арность объекта заранее не фиксируется, но проявляет себя постепенно, во взаимодействиях с другими объектами;

при конструировании составного объекта один из исходных объектов — функция, — применяется к другому — аргументу, — причем в других контекстах они могут поменяться ролями, то есть функции и аргументы рассматриваются как объекты на равных правах;

разрешается самоприменимость функций, то есть объект может применяться сам к себе.

Высказывание

Высказывание определено тем способом, относительно которого можно рассматривать его доказательство.

Возможные миры

Возможные миры будем понимать как различные собрания индивидов с дополнительной структурой или без нее.

Дескрипция

При построении логических средств в числе термов часто используются *определенные описания*, или *дескрипции* — конструкции ‘тот ..., который ...’. Дескрипция соответствует терму $Ix\Phi$, который канонически читается как ‘тот единственный x , для которого выполняется (верно) Φ ’.

Дефинициальное (определяющее) равенство

Бинарное отношение дефинициального равенства обозначается посредством “ $\stackrel{def}{=}$ ”. Его левая часть считается *определяемым*, а его правая часть — *определяющим*. Это отношение эквивалентности (рефлексивное, симметричное и транзитивное).

Доктрина типов

Доктрина типов восходит к Б. Расселу, согласно которому всякий тип рассматривается как диапазон значимости пропозициональной (высказывательной) функции. Более того, считается, что у всякой функции имеется тип (ее домен). В доктрине типов выполняется *принцип замены типа (высказывания) на дефинициально эквивалентный тип (высказывание)*.

Значение

Значения образуют концептуальный класс, состоящий из содержательных объектов, которые приписываются посредством *оценки* формальным объектам.

Имя

Имя называет некоторый действительный или воображаемый объект.

Интерпретация (— теории)

Под интерпретацией теории относительно содержательной области (предметной области) понимается много-однозначное соответствие между элементарными высказываниями теории и определенными содержательными высказываниями, относящимися к этой содержательной области.

Интерпретация (— терма)

Оценка, или интерпретация терма M в структуре \mathcal{M} — это отображение:

$$\| \cdot \| : \text{термы} \times \text{приписывания} \rightarrow \text{элементы из } \mathcal{M}$$

(см. также *Оценка*).

Интерпретация (— адекватная)

Адекватная, или относительно полная интерпретация каждому содержательному высказыванию (интерпретанту из со-

держательной области) ставит в соответствие теорему из теории.

Инфикс

Инфиксами считаются бинарные функторы (“связки”, операторы), которые пишутся между аргументами.

Исчисление

Исчислением называют систему, использующую связанные переменные. В частности, λ -исчисление использует связанные переменные, и единственным оператором, связывающим переменную, то есть превращающим переменную в формальный параметр, является оператор функциональной абстракции λ .

Категория

*Категория E содержит объекты X, Y, \dots и стрелки f, g, \dots . Каждой стрелке f соответствует объект X , называемый *доменом* и объект Y , называемый *кодоменом*. Этот факт записывается в виде $f : X \rightarrow Y$. Дополнительно накладываются ограничения на использование композиции — с учетом единичного (тождественного) отображения. Стрелки рассматриваются как представления отображений. Более строго, если g — произвольная стрелка $g : Y \rightarrow Z$ с доменом Y , который совпадает с кодоменом f , то имеется стрелка $g \circ f : X \rightarrow Z$, называемая *композицией g с f* . Для каждого объекта Y существует стрелка $1_Y = 1_Y : Y \rightarrow Y$, называемая тождественной стрелкой для Y . Предполагается выполнение аксиом тождества и ассоциативности для всех стрелок $h : Z \rightarrow W$:*

$$1_Y \circ f = f, g \circ 1_Y = g, h \circ (g \circ f) = (h \circ g) \circ f : X \rightarrow W.$$

Класс

Понятие класса считается интуитивно ясным. Классы объ-

ектов обычно рассматриваются как некоторые объекты.

Собственные классы (например, класс чисел, домов, людей и т.п.) — это такие классы, которые не являются членами самих себя. *Несобственные* классы — это такие классы, которые являются членами самих себя (например, класс всех понятий).

Класс (— концептуальный)

В широком смысле слова — это совокупность допустимых элементов этого класса.

Класс (— индуктивный)

Индуктивный класс — это концептуальный класс, порожденный из определенных исходных элементов посредством выделенных способов комбинации. Более строго, класс K индуктивен, если:

- (1) класс K включает в себя базис;
- (2) класс K замкнут относительно способов комбинации;
- (3) класс K является подклассом любого класса, удовлетворяющего условиям (1) и (2).

Понятие индуктивного класса обычно используют в двух случаях:

- (1) элементы являются *объектами*, а способы комбинации — *операциями*;
- (2) элементы являются *высказываниями*, а способы комбинации — *связками*.

Комбинатор

Комбинатором считается объект, который относительно означивания проявляет свойство константности. С точки зрения λ -исчисления комбинатор является замкнутым термом.

Комбинаторная логика

В узкой формулировке это ветвь математической логики, изучающая комбинаторы и их свойства. В комбинаторной логике функциональная абстракция выражима в терминах обычных операций, то есть *без использования формальных переменных* (параметров).

Конструкция

Процесс получения объекта X , принадлежащего индуктивному классу K (см. **Класс индуктивный**), посредством итерации способов комбинации считается *конструкцией* X относительно K .

λ -терм (лямбда-терм)

λ -термом, или λ -выражением считается объект, полученный индукцией по построению с возможным применением операторов аппликации и абстракции.

Логика

“Логика есть анализ и критика мышления” (см. Johnson W.E. Logic, part I, London, 1921; part II, London, 1922; part III, London, 1924.). Когда при изучении логики применяются математические методы, то строятся математические системы, определенным образом связанные с логикой. Эти системы являются предметом самостоятельного исследования и рассматриваются как ветвь математики. Такие системы составляют *математическую логику*. Математической логике принадлежит задача объяснения природы математической строгости, поскольку математика является дедуктивной наукой, и понятие строгого доказательства является центральным для всех ее разделов. Более того, математическая логика включает в себя изучение оснований математики.

Логика (— математическая)

Математическая логика описывает новое направление в ма-

тематике (см. *Математика современная*), сосредоточивая внимание на используемом в ней языке, на способах определения абстрактных объектов и на законах логики, которые используются в рассуждениях об этих объектах. Такое изучение предпринято в логике, чтобы понять природу математического опыта, пополнить математику важнейшими результатами, полученными в логике, а также, чтобы найти приложения к другим разделам математики.

Логика (современная математическая —)

Современная математическая логика берет свое начало от работ Лейбница об универсальном исчислении, которое может включать в себя всю умственную деятельность и, в частности, всю математику.

Математика (— современная)

Современная математика может быть описана как наука об *абстрактных объектах* таких, как вещественные числа, функции, алгебраические системы и т.п.

Множество (— счетное)

Счетным называют всякое множество, элементы которого могут быть занумерованы, то есть расположены в виде единого списка, в котором некоторый элемент стоит первым, некоторый — вторым и т.д., так что всякий элемент этого множества рано или поздно встретится в этом списке.

Неразрешимость

Математический смысл какого-либо результата о неразрешимости состоит в том, что некоторое конкретное множество не является рекурсивным.

Нумерал

В рамках комбинаторной логики или λ -исчисления можно установить такие комбинаторы или, соответственно, термы,

которые ведут себя как числа. Эти представления чисел получили название *нумералов*. Нумералы как комбинаторы удовлетворяют всем законам комбинаторной логики. Более того, можно указать комбинаторы, представляющие арифметические операции, например, сложение чисел.

Об-система

Формальные объекты об-системы образуют индуктивный класс. Элементы этого класса называются

обами, или объектами. Начальные элементы индуктивного класса считаются *атомами*, а способы комбинации — *операциями*. Об-системы используются для поиска существенных, инвариантных образований объектов.

Оболочка Каруби

Оболочка Каруби представляет собой частный вид категории.

Объект

Объектом считается математическая сущность, которая используется в теории. Объект — это математическое *представление* реального объекта предметной области (“внешнего мира”).

Объект (— арифметический)

Арифметическими объектами считаются комбинаторные представления чисел — нумералы, а также соответствующие комбинаторные представления операций над числами (см. *нумерал*).

Объекты (система —)

См. *Система объектов*.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) представляет собой такой способ программирования, который

обеспечивает модульность программ за счет разделения памяти на области, содержащие данные и процедуры. Области могут использоваться в качестве образцов, с которых по требованию могут делаться копии.

Определение

Определение традиционно принимается как соглашение относительно использования языка. Тогда новый символ или комбинация символов, называемая *определяемым*, вводится вместе с разрешением подставлять ее вместо некоторой другой комбинации символов, называемой *определяющим*, значение которой уже известно на основе данных и предшествующих определений.

Определение (— функции)

Явные определения вводят в рассмотрение функции. Таким образом, начиная с переменной x , которая обозначает произвольный объект типа A , строится выражение $b[x]$, обозначающее объект типа $B(x)$. Далее определяется функция f типа $(\forall x \in A)B(x)$ схемой $f(x) \stackrel{def}{=} b[x]$, где квадратные скобки указывают на вхождение переменной x в выражение $b[x]$. Если $B(x)$ для каждого объекта x типа A определяет один и тот же тип B , то вместо $(\forall x \in A)B(x)$ используется сокращенная форма записи $A \rightarrow B$. Последняя запись принимается за тип функций из A в B .

Определение (— рекурсивное)

Рекурсивное определение функции — это такое определение, в котором значения функции для данных аргументов непосредственно определяются значениями той же функции для “более простых” аргументов или значениями “более простых” функций. Понятие ‘более простой’ уточняется выбором формализации — простейшими, как правило, являются все функции-константы. Такой метод формализации удобен,

поскольку рекурсивные определения можно рассматривать как алгоритм (см. *Алгоритм*).

Оценка

Оценкой считается такое соответствие, когда содержательные объекты сопоставляются с формальными объектами, причем один и тот же содержательный объект может быть поставлен в соответствие двум или более различным формальным объектам.

Переменная

Переменной считается “переменный объект”, вместо которого можно производить подстановки.

Переменная (— неопределенная)

Это (атомарный) объект, на который (в об-системе) не наложено никаких ограничений.

Переменная (— подстановочная)

Это такой объект, вместо которого допускаются подстановки по явно сформулированному правилу подстановки.

Переменная (— связанная)

Это объект, который участвует в операции, имеющей один или более формальных параметров. Связывание переменной имеет смысл относительно такого рода операции.

Постулаты

Термином “постулаты” называют правила вывода и аксиомы.

Предложение

Предложение выражает утверждение.

Представление

Представлением (системы) считается любой способ рассмотрения конкретных объектов (из предметной области)

как формальных объектов. Содержательные (конкретные) объекты сохраняют структуру формальных объектов.

Префикс

Префиксом считается функтор (оператор, “связка”), который пишется перед аргументами.

Проекция

В качестве проекции берется подмножество соответствующего декартова произведения.

Произведение

Произведение экстенционально определяется как совокупность кортежей (n -ок). В зависимости от числа элементов в кортеже произведение наделяется арностью.

Процесс (— эффективный)

Предположим, что имеются определенные преобразования, которые можно фактически производить над определенными элементами. Предположим, что имеется также предписание, определяющее последовательность преобразований, которые надо применять одно за другим к какому-то элементу. Говорят, что предписание определяет *эффективный процесс* для достижения определенной цели по отношению к элементу, если при условии, что этот элемент задан, предписание однозначно определяет такую последовательность преобразований, что цель достигается за конечное число шагов.

Реляционная система

Это система с единственным базисным предикатом, который является бинарным отношением.

Свойство

Свойством считается пропозициональная функция, определенная на (произвольном) типе A .

Система (— объектов)

В *системе объектов* формальные объекты образуют однородный индуктивный класс (см. **Класс индуктивный**). Элементы этого индуктивного класса считаются *объектами*, его начальные объекты — *атомарными объектами*, а способы комбинации — *исходными операциями*.

Суффикс

Суффиксом считается функтор, который пишется после аргументов.

Тезис Черча

Нельзя доказать гипотезу о том, что какая-либо стандартная формализация дает удовлетворительные аналоги неформального понятия *алгоритма* (см. **Алгоритм**) и *алгоритмической функции* (см. **Функция, вычисляемая алгоритмом**).

Многие математики принимают гипотезу о том, что стандартные формализации дают “разумную переделку” неизбежно расплывчатых неформальных понятий, а саму гипотезу называют *тезисом Черча*.

Теория

Теорией считают способ выбора подкласса истинных высказываний из числа высказываний, принадлежащих классу всех высказываний A .

Теория (— дедуктивная)

Теория T считается дедуктивной, если T является индуктивным классом (элементарных) высказываний (см. **Класс индуктивный**).

Теория (— моделей)

Теорией моделей считается раздел математической логики,

изучающий связи между формальным языком и его интерпретациями, или *моделями*. Основными объектами исследования являются предложения ϕ и алгебраические системы M для языка L .

Теория (— непротиворечивая)

Непротиворечивая теория определяется как такая теория, которая не охватывает всего класса A высказываний.

Теория (— полная)

Полной считается такая дедуктивная теория T , что присоединение к ее аксиомам элементарного высказывания, не являющегося элементарной теоремой, при сохранении правил неизменными делает ее противоречивой.

Теория (— полная в смысле Поста)

T полна, если каждое высказывание из класса A высказываний является следствием относительно T любого высказывания X , не входящего в T .

Теория (— рекурсии)

Теория рекурсии изучает класс рекурсивных или эффективно вычислимых функций и их применения в математике. В широком смысле теория рекурсии рассматривается как изучение общих процессов определения с помощью рекурсии, но не только на натуральных числах, а на всех типах математических структур.

Теория (— типов)

В основе этой теории лежит принцип иерархичности. Это означает, что логические понятия — высказывания, индивиды, пропозициональные функции — располагаются в иерархию типов. Существенно, что произвольная функция в качестве своих аргументов имеет лишь те понятия, которые предшествуют ей в иерархии.

Фраза (— грамматическая)

Фразами считаются *имена, предложения и функторы*.

Функтор (— грамматический)

Функтор рассматривается как средство соединения *фраз* для образования других фраз.

Функция

См. *Определение функции*.

Функция (— высшего порядка)

Функция получает название функции 'высшего порядка', если ее аргументом может в свою очередь быть функция, либо в результате ее применения получается функция.

Функция (—, вычислимая алгоритмом)

Это отображение, задаваемое процедурой, или *алгоритмом*.

Функция (— примитивнорекурсивная)

Класс примитивнорекурсивных функций — это наименьший класс \mathcal{C} всюду определенных функций такой, что:

- i) все *функции-константы* $\lambda x_1 \dots x_k. t$ содержатся в \mathcal{C} , $1 \leq k, 0 \leq t$;
- ii) функция *следования* за $\lambda x. x + 1$ содержится в \mathcal{C} ;
- iii) все *функции выбора* $\lambda x_1 \dots x_k. x_i$ содержатся в \mathcal{C} , $1 \leq i \leq k$;
- iv) если f — функция от k переменных из \mathcal{C} и g_1, g_2, \dots, g_k — функции от m из \mathcal{C} , то функция

$$\lambda x_1 \dots x_m. f(g_1(x_1, \dots, x_m), \dots, g_k(x_1, \dots, x_m))$$

содержится в \mathcal{C} , $1 \leq k, m$;

- v) если h — функция от $k + 1$ переменных из \mathcal{C} , а g — функция от $k - 1$ переменных из \mathcal{C} , то единственная функция f от k переменных, удовлетворяющая условиям

$$\begin{aligned} f(0, x_2, \dots, x_k) &= g(x_2, \dots, x_k), \\ f(y + 1, x_2, \dots, x_k) &= h(y, f(y, x_2, \dots, x_k), x_2, \dots, x_k), \end{aligned}$$

содержится в \mathcal{C} , $1 \leq k$.

Отметим, что ‘функция от нуля переменных из \mathcal{C} ’ означает фиксированное натуральное число.

Язык

Язык в широком смысле слова определяется введением в употребление *соглашений*: (1) фиксируется *алфавит*; (2) фиксируются правила образования из букв алфавита определенных комбинаций, называемых *выражениями* или словами.

Язык (— исследователя, или \mathcal{U} -язык)

\mathcal{U} -язык характеризуется следующими свойствами:

- (1) *единственностью* для каждого конкретного контекста;
- (2) наличием средств *формализации терминологии*;
- (3) изменчивостью в том смысле, что он является *процессом* относительно добавления новой символики или новых терминов, причем использование старых терминов не обязательно является неизменным;
- (4) \mathcal{U} -язык по необходимости неясен, однако пользуясь им можно достичь любой разумной степени точности.

Практикум

Источники

Первоначально практикум по аппликативным вычислениям был включен в общий практикум по курсу “Системы искусственного интеллекта”, читавшийся в МИФИ (Л.Т. Кузин, [24]). Этот практикум был подготовлен коллективом авторов по разделам дедуктивного вывода, реляционной алгебры и реляционного исчисления, понятийного представления знаний и фреймов, программирования на Lisp (К.Е. Аксенов, О.Т. Баловнев, В.Э. Вольфенгаген, О.В. Воскресенская, А.В. Ганночка, М.Ю. Чуприков, [1]; А.Г. Пантелеев, [41], М.А. Булкин, Ю.Р. Габович, А.Г. Пантелеев, [5]). Его существенной компонентой являлась реляционная СУБД Lisp/R, которая реализовывала один из путей развития вычислительных идей, изначально в него заложенных. Этот метод опирался на встроенные (погруженные) вычислительные системы и получил развитие в работе (О.В. Воскресенская, [1]), приведенной в перечне диссертаций на стр. 334.

Аппликативные вычисления

Вариант курса, излагавшегося в МИФИ на основе настоящей книги, (например, в варианте “Специальные главы дискретной математики” или “Основы компьютерных наук”) оснащен практикумом, который изложен в работе (В.Э. Вольфенгаген, Л.В. Го-

льцева, [12]). Дополнительное развитие можно найти в работах (Л.В. Гольцева, [7]) и (А.В. Гаврилов, [6]), приведенных в перечне диссертаций на стр. 334. Практикум работает на IBM PC и распространяется на машинных носителях.

Структура практикума

Практикум охватывает основные понятия и обозначения, используемые в аппликативных вычислительных системах, и позволяет проводить их изучение в полном объеме: от языка и соглашений об обозначениях до метатеорем о соответствующих формальных системах. Он дает базовые знания об использовании аппликативных вычислительных систем (АВС) в качестве основы функциональных языков, систем функционального программирования и чисто объектных языков.

Центральным используемым понятием является терм, рассматриваемый как *представление* объекта. Терм синтаксически соответствует программе на функциональном языке.

На этом основании *первый* раздел практикума посвящен технике выполнения правильных синтаксических преобразований — расстановке скобок, — для термов различных видов.

Второй раздел предназначен для изучения редукции в АВС. Результатом выполнения функциональной программы является значение, полученное по окончании процесса вычисления. В АВС результату выполнения редукции соответствует нормальная форма терма. В соответствии с теоремой Черча–Россера нормальная форма не зависит от порядка выполнения шагов редукции, что дает возможность построения различных стратегий вычисления в системах функционального программирования. Построение λ -абстракции в комбинаторной логике можно рассматривать в качестве примера для интерпретации логических систем средствами комбинаторной логики.

Так пара комбинаторов K и S образует базис для произволь-

ных λ -термов, в то время как комбинаторы I, B, C, S являются базисом только для термов, в которых отсутствуют свободные переменные. Использование этих двух базисов для разложения термов из двух разделов практикума обеспечивает достаточную полноту рассмотрения материала, поскольку будут представлены произвольные λ -термы и комбинаторные термы, которые являются λ -термами без свободных переменных.

Последний раздел практикума является наиболее творческим, так как предполагает конструирование собственных комбинаторных систем. Показано, как путем добавления к базисным комбинаторам дополнительных комбинаторов увеличить выразительные возможности реализованной среды.

В целом, каждый из разделов лабораторного практикума вырабатывает у обучаемого практические навыки по определенному кругу вопросов и может быть использован как отдельная лабораторная работа. Если практикум используется как компьютерное пособие, то разделы могут быть скомпонованы в соответствии с подготовкой обучаемого либо как это нужно преподавателю.

Независимые ресурсы

<http://www.rbjones.com> Этот электронный ресурс FACTASIA имеет своей целью “развить предвидение будущего и дать ресурсы для построения такого видения и самого будущего”, а также “сделать вклад в те ценности, которые определяют будущее, и в те технологии, которые помогают его строить”. Раздел Logic включает *комбинаторную логику и λ -исчисление*, библиографию. См. <http://www.rbjones.com/rbjpub/logic/cl/>

<http://www.cwi.nl/~tromp/cl/cl.html> Представлено руководство, библиографические ссылки и Java-апплет, позволяющий интерпретировать объекты — выражения аппликативного языка, — которые строятся по правилам комбинаторной логики.

<http://ling.ucsd.edu/~barker/Lambda/ski.html> Представлен простой учебник по комбинаторной логике в режиме on-line.

<http://tunes.org/~iepos/oldpage/lambda.html> Введение в лямбда-исчисление и комбинаторную логику.

<http://foldoc.doc.ic.ac.uk> Представлен свободно доступный словарь терминов в области вычислений FOLDOC — Free On-Line Dictionary of Computing.

<http://dmoz.org/Science/Math/> Содержит справочные сведения и библиографию. В разделе '/Logic_and_Foundations' имеется раздел '/Computational_Logic', в котором есть ссылка на '/Combinatory_Logic_and_Lambda_Calculus/'.

<http://www.cl.cam.ac.uk/Research/TSG> Представлены направления учебной и научно-исследовательской работы Компьютерной лаборатории Кембриджского университета, имеющей мировую известность.

<http://web.comlab.ox.ac/oucl> Вычислительная лаборатория Оксфордского университета, которая является кафедрой по компьютерным наукам с мировой известностью. На URL <http://web.comlab.ox.ac/oucl.strachey> размещена информация о регулярно проводимой серии лекций в честь Кристофера Стрейчи (Christopher Strachey, 1916-1975), первого профессора Оксфордского университета в области вычислений (computation). Он был первым руководителем Группы Исследований в Программировании, созданной в 1965 г. — а сэр Тони Хоар (Tony Hoare) стал в 1977 г. его преемником, — и вместе с Дана Скоттом (Dana Scott) основал *денотационную семантику* как направление, поставив языки программирования на прочную математическую основу.

Диссертации

- [1] Воскресенская О.В., *Методы разработки реляционной системы управления базой данных*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1985.
- [2] Александрова И.А., *Проектирование информационно-программного обеспечения систем организационного типа на основе концептуальных моделей*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1986.
- [3] Исмаилова Л.Ю., *Разработка программных средств реляционной обработки данных в экспертных системах*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1989.
- [5] Волков И.А., *Исследование и разработка методов анализа и обеспечения достоверности информации о НИР в области медицины*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления; 14.00.33 — Социальная гигиена и охрана здравоохранения, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1990.

- [4] Вольфенгаген В.Э., *Концептуальный метод проектирования банков данных*, Диссертация на соискание ученой степени доктора технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1990.

{*Аннотация.* Представлен концептуальный метод проектирования банков данных, который рассматривается как подход к построению, применению и управлению базами данных и базами метаданных. Подход предполагает возможность настройки на изменяющуюся предметную область и ее представление, в частности, при увеличении/уменьшении степени детализации. Исследуется управление базами данных/метаданных в условиях интегрированного использования объектов данных, объектов метаданных и программ. Унифицированная вычислительная среда сохраняет расширяемость модели объектов данных. Представления объектов данных/метаданных предполагаются встроенными в вычислительную среду. Построена концептуальная оболочка для вычислений в терминах объектов. В работе применяются и развиваются методы бестипового и типового λ -исчисления, комбинаторной логики и вычислений в категории. Круг вопросов, рассмотренных в работе, является обобщением опыта их преподавания в различных учебных курсах по компьютерным наукам.)

- [6] Гаврилов А.В., *Настраиваемая система программирования для категориальных вычислений*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1995.

- [7] Гольцева Л.В., *Аппликативная вычислительная система с интенциональными отношениями*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1995.

- [8] Зыков С.В., *Исследование и реализация интегрированной корпоративной информационной системы для решения задач управления персоналом*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-

физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 2000.

- [9] Забродин А.Л., *Исследование и реализация программного обеспечения управления данными для автоматизированных систем оперативного управления связью*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 2000.

Вячеслав Эрнстович Вольфенгаген

Комбинаторная логика в программировании

Вычисления с объектами в примерах и задачах

Макет: *Автор*

Художественное оформление обложки: *Автор*

Научный редактор: *Л. Ю. Исмаилова*

Дизайн обложки: *О. В. Мортина*

*Продукция соответствует требованиям
Министерства здравоохранения Российской Федерации.*

*Санитарно-эпидемиологическое заключение
№ 77.99.02.953.Д.003230.06.01 от 9 июня 2001 г.*

Подписано в печать 06.12.2002 Печать офсетная.

Бумага офсетная. Формат 60х84/16. Печ. л. 22

Тираж 1200 экз. Заказ № 7344.

Отпечатано в полном соответствии с качеством
предоставленных диапозитивов в ППП «Типография „Наука“».
121099, Москва, Шубинский пер., д. 6

ISBN 5-89158-101-9



9 785891 581012

Все права принадлежат АО «Центр ЮрИнфоР»
103006, Москва, Воротниковский пер., 7, тел. **(095) 956-25-12**,
<http://www.jurinform.ru>, эл. почта: info@jurinform.ru
Лицензия на издательскую деятельность
ИД № 03088 от 23 октября 2000 г.

В. Э. Вольфенгаген. Логика. Конспект лекций: техника рассуждений. 2001. ISBN 5-89158-054-3

В работе рассмотрены способы переформулирования текста фактического типа на символичный язык, допускающий применение классических логических средств. Показаны приемы и способы записи аргументации и проверки ее значимости. На большом числе примеров проиллюстрирована техника логических рассуждений, выводов и доказательств. Отмечены способы включения в вывод аннотаций (комментариев), пользуясь которыми можно проверить истинность или установить ложность приводимых доводов.

Для студентов и аспирантов гуманитарных специальностей. Может быть использована для первоначального изучения предмета, а также для самостоятельного изучения.

В. Э. Вольфенгаген. Конструкции языков программирования. Приемы описания. 2001. ISBN 5-89158-079-9

Серия: «Компьютерные науки и информационные технологии»

В работе изложены основы, касающиеся разработки, реализации и применения конструкций как императивных, так и функциональных языков программирования. Значительное внимание уделяется применению денотационной семантики, позволяющей в полной мере извлечь преимущества объектно-ориентированного подхода, что, в конечном счете, позволяет построить результирующую вычислительную модель чисто функционального типа.

Изложение материала сопровождается детально разобранными примерами, которые снабжены комментариями, помогающими уяснить реализацию конструкций различных языков.

Книга может быть рекомендована программистам, а также студентам и аспирантам, изучающим языки программирования, основы компьютерных наук и информационные технологии.

Книгу можно использовать в качестве учебника или справочного пособия. Она будет полезна как студентам и аспирантам, так и профессионалам в области компьютерных наук, информационных технологий и программирования.

В. Э. Вольфенгаген. Категориальная абстрактная машина. Конспект лекций: введение в вычисления. — 2-е изд. 2002. ISBN 5-89158-102-7

Серия: «Компьютерные науки и информационные технологии»

Работа содержит изложение базовых моделей вычислений, применяемых в компьютерных науках. Изложены основы λ исчисления и комбинаторные исчисления.

Основное внимание уделено подробному рассмотрению техники вычисления значения конструкций языков программирования, включая компилирование кода, его оптимизацию и исполнение на примере категориальной абстрактной машины. Изложение построено на примерах возрастающей сложности.

Книга может быть рекомендована студентам и аспирантам, изучающим основы компьютерных наук теорию и языки программирования, информационные технологии, информатику и дискретную математику.

■ А. С. Титов. Недоимка: Правовое регулирование взыскания., 2002. ISBN 5-89158-108-6

Издание представляет собой одно из первых комплексных исследований теоретических и практических проблем взыскания недоимки с точки зрения уяснения правовой природы и определения основополагающих понятий, таких как «недоимка», «юридическое определение состава недоимки», «налоговая обязанность», «средства погашения недоимки».

Методологической основой материалов книги являются Налоговый кодекс Российской Федерации, Федеральный закон РФ «Об исполнительном производстве», иные нормативные правовые документы, действующие по состоянию на 1 октября 2002 г. года, а также судебно-арбитражная практика.

Предназначена для судей, прокуроров, юрисконсультов, адвокатов, работников юридических фирм, студентов, аспирантов и преподавателей юридических образовательных учреждений, а также иных лиц, интересующихся этой проблемой.

■ Агарков М. М. Избранные труды по гражданскому праву. В 2-х т. 2002. ISBN 5-89158-091-8 (т. I.) ISBN 5-89158-092-6 (т. II.)

В настоящем издании собраны основные работы величайшего русского советского цивилиста Михаила Михайловича Агаркова (1890–26 июля 1947 г.), на сегодняшний день являющиеся библиографической редкостью. Среди них монографические статьи «Ценность частного права» (1920), «Ценные бумаги на предъявителя» (1926), «К вопросу о договорной ответственности» (1945), а также главное произведение ученого «Обязательство по советскому гражданскому праву» (1940). Несмотря на свой почтенный «возраст» и специфические идеологические условия их написания, все, вошедшие в настоящий сборник, публикации являются актуальными до настоящего времени. Современный российский ГК и практика его применения в ряде вопросов отталкиваются именно от научных результатов, которые в свое время впервые получил именно М. М. Агарков.

Для студентов, преподавателей, научных и практических работников, всех тех, кто изучает гражданское право и интересуется его теорией и историей.

■ А. И. Каминка. Очерки торгового права. 2002. ISBN 5-89158-089-6

Очередная книга серии «Научное наследие» «Библиотеки „ЮрИнфоР“» — переиздание работы русского цивилиста Августа Исааковича Каминки (1865–1940), представляющей собою «ряд интересных остро и живо написанных очерков по основным вопросам торгового права». Предисловие и

примечания В. А. Белова помогут читателям разобраться в вопросах о том, почему книга продолжает оставаться актуальной и как ее материал может быть использован в современных условиях.

Для студентов, аспирантов, преподавателей и ученых, а также для всех, интересующихся изучением гражданско-правового регулирования предпринимательской деятельности.

Э. Ю. Исмаилова, Ю. В. Трунцевский, Н. Е. Савич. **Экологическое право: Учебник.** 2-е изд., перераб. 2002. ISBN 5-89158-081-0

В учебнике освещаются предмет, метод, объекты и система экологического права, включая организационно-правовой и экономико-правовой механизмы охраны окружающей среды. Правовые вопросы охраны окружающей среды рассматриваются применительно к промышленной, сельскохозяйственной, энергетической и военной деятельности, а также относительно городских и сельских поселений, особо охраняемых природных территорий и объектов и территорий, находящихся в экологически опасной ситуации. Освещены также международно-правовые аспекты экологии и сотрудничества государств в сфере охраны окружающей среды.

Учебник предназначен для студентов юридических образовательных учреждений. Может оказаться полезным практикующим юристам.

В. А. Белов. **Гражданское право: Общая часть: Учебник.** 2002. ISBN 5-89158-048-9

В настоящем учебнике рассмотрены все институты Общей части российского гражданского права. Учебник отличается оригинальной авторской систематизацией правового материала: институты общей части освещаются в соответствии с системным подходом к понятию, элементам, условиям существования и динамики гражданских правоотношений. Ключевой категорией изложения избрано понятие права в субъективном смысле (субъективное право), а не норма права, как это делается традиционно.

Книга предназначена для студентов юридических вузов и факультетов, может быть использована для самообразования лиц, не имеющих юридической специальности (экономистов, финансовых и банковских работников), и может быть полезна любым читателям, интересующимся гражданским правом.

Уголовное право: Общая и особенная части. Интенсивный полный курс в алгоритмах. 2-е изд., испр. и доп. / Под ред. Л. Д. Гаухмана, А. А. Энгельгардта, 2002. ISBN 5-89158-080-2

Допущено Министерством образования Российской Федерации в качестве учебника для студентов высших учебных заведений, обучающихся по специальности «Юриспруденция»

Учебник подготовлен в соответствии с программой курса «Уголовное право» для юридических вузов и факультетов с учетом новейшего российского законодательства и сложившейся правоприменительной практики. Является методически новым, оптимальным средством изучения и освоения курса уголовного права в кратчайший срок.

Для студентов, аспирантов и преподавателей юридических вузов.



Е. В. Васьковский

Цивилистическая методология. Учение о толковании и применении гражданских законов. 2002. ISBN 5-89158-083-7

Серия: «НАУЧНОЕ НАСЛЕДИЕ»

Настоящая книга серии «Научное наследие» «Библиотеки ЮрИнфоР» включает в себе переиздание докторской диссертации великого русского цивилиста и выдающегося адвоката Евгения Владимировича Васьковского (1866–??). Данное произведение является лучшей работой в сфере теории и практики толкования законодательных норм, изучение и практическое применение которой поможет привить все знания и навыки в этой области, столь необходимые современным российским юристам.

Книга предназначена быть повседневным руководством практикующим юристам, судьям, работникам правоохранительных органов и органов исполнительной власти, сталкивающимся с необходимостью применения законодательства, а также для студентов, аспирантов, ученых, преподавателей и всех, интересующихся юридической герменевтикой.



И. И. Кучеров. Бюджетное право России: Курс лекций, 2002. ISBN 5-89158-061-6

Данное издание представляет собой курс лекций по бюджетному праву. В книге содержатся сведения по истории бюджетной системы России, раскрываются понятия и юридическая сущность бюджета, анализируется устройство бюджетной системы России и ее принципы, межбюджетные отношения, понятие бюджетного права, его источники, права и обязанности участников бюджетных правоотношений, институты государственных и муниципальных заимствований, долгов и кредитов, бюджетные процедуры, полномочия государственных органов, осуществляющих контроль за исполнением бюджета, вопросы ответственности за нарушение бюджетного законодательства. Для лучшего освоения курса слушателями и студентами материал изложен автором в простой удобочитаемой, но максимально приближенной к нормативно-правовым источникам форме.

Курс лекций предназначен для использования слушателями и аспирантами Академии налоговой полиции ФСНП России, а также студентами других вузов, обучающимися по юридической специальности.



И. И. Кучеров. Налоговое право России: Курс лекций, 2001. ISBN 5-89158-051-9

Допущено Министерством образования Российской Федерации в качестве учебника для студентов высших учебных заведений, обучающихся по специальности «Юриспруденция».

Федеральная программа книгоиздания России

Рецензенты: Козырин А. Н., доктор юридических наук, профессор.

Ялбулганов А. А., доктор юридических наук, профессор.

Данное издание представляет собой курс лекций по налоговому праву. В книге содержатся сведения по истории налогообложения, раскрывается юридическая природа налога, анализируются методы осуществления налоговой политики, устройство налоговой системы, источники налогового законодательства, права и обязанности налогоплательщиков и налоговых агентов, компетенция органов налогового администрирования и органов налоговой полиции, правовой механизм взыскания недоимок и пени, а также виды

ответственности за нарушения законодательства о налогах сборах и порядок разрешения налоговых споров.

Курс лекций рассчитан на слушателей и аспирантов Академии налоговой полиции ФСНП России, а также студентов других вузов, обучающихся по юридической специальности.

П. П. Цитович. Очерк основных понятий торгового права. 2001. ISBN 5-89158-063-2

Серия: «НАУЧНОЕ НАСЛЕДИЕ»

Переизданием настоящей работы, впервые напечатанной в 1886 г. и положившей начало систематическим сочинениям по русскому торговому праву АО «Центр ЮрИнфоР» открывает серию «Научное наследие» «Библиотеки ЮрИнфоР». Автор книги — известный русский цивилист Петр Павлович Цитович (1844–1913) — впервые в русской литературе очертил систему основных понятий дисциплины торгового (коммерческого, предпринимательского) права, обозначил ее основные проблемы и наметил тем самым путь, по которому стало развиваться научное изучение русского торгового права. Несмотря на более чем почтенный возраст книги содержательно она практически не устарела.

Издание предназначено для студентов, аспирантов, преподавателей и ученых, а также всех, интересующихся изучением гражданско-правового регулирования предпринимательской деятельности.

Российское гражданское право: Учебник.

Под ред. З. Г. Крыловой, Э. П. Гаврилова. 2-е изд., 2001. ISBN 5-89158-067-5

В работе освещаются основные институты российского гражданского права, предусматриваемые новым Гражданским кодексом Российской Федерации (первой и второй частями Гражданского кодекса). Упоминаются и другие новейшие нормативные акты. Наибольшее внимание уделяется деятельности банков, иных финансовых организаций, правовому регулированию расчетных и кредитных правоотношений.

Книга предназначена для экономистов, финансовых и банковских работников, а также для широкого круга читателей.

Гражданский Кодекс Российской Федерации. Civil Code of the Russian Federation. Параллельные русский и английский тексты. Перевод профессора Лондонского университета У. Э. Батлера, 2001. ISBN 5-89158-075-6

Готовятся к выпуску:

В. А. Белов. Гражданское право: Особенная часть: Учебник. 2003 ISBN 5-89158-105-1

Настоящий учебник является продолжением Общей части Учебника по гражданскому праву того же автора. В нем рассмотрены институты Особенной части российского гражданского права — абсолютные субъективные гражданские права (вещные, исключительные, личные, наследственные) и отно-

сительные субъективные гражданские права (обязательственные, права участия, права преимущества, деликтные, кондикционные и иные).

Книга предназначена для студентов юридических вузов и факультетов, может быть использована для самообразования лиц, не имеющих юридической специальности (экономистов, финансовых и банковских работников), и может быть полезна любым читателям, интересующимся гражданским правом.

Л. Д. Гаухман. Квалификация преступлений: закон, теория, практика. 2-е изд., перераб. и дополн. 2003. ISBN 5-89158-107-8

В книге на базе действующего уголовного законодательства России освещаются теоретические и практические проблемы квалификации преступлений и состава преступления. Развивается общее учение о составе преступления как юридическом основании квалификации преступлений. Формулируются и обосновываются общие и частные правила квалификации преступлений в следственной и судебной практике.

А. А. Соколов. Теория налогов. 2003. ISBN 5-89158-087-X

Книга выдающегося российского юриста посвящена теории налогов. В книге освещаются состояние и попытки решения базовых вопросов теории налогов в законодательстве и правовой доктрине России. Вступительная статья одного из членов редакционного совета серии посвящена описанию творческого пути ученого и расстановке акцентов, позволяющих тем, кто интересуется данной проблематикой, с успехом использовать ее положения в современных условиях.

И. И. Кучеров, А. В. Торшин. Налоговая тайна: правовой режим защиты информации. 2003. ISBN 5-89158-109-4

В книге рассматриваются общие вопросы правовой защиты конфиденциальной информации, в том числе юридическая конструкция тайны. Основное внимание уделено действующей правовой системе защиты конфиденциальной экономической информации, включая государственную, профессиональную, служебную, коммерческую, банковскую, страховую, нотариальную, аудиторскую и адвокатскую тайны. Отдельно анализируется налоговая тайна и ее соотношение с другими правовыми режимами защиты информации. Рассматриваются порядок документирования информации, составляющей налоговую тайну, доступ к конфиденциальной налоговой информации, включая международный информационный обмен, ответственность за нарушение порядка предоставления налоговозначимой информации и режима налоговой тайны. Задача освещения технических аспектов защиты информации авторами не ставилась.

Предназначено для лиц, интересующихся проблемами налогового права и налоговой деятельности, и может быть использовано слушателями Академии налоговой полиции ФСНП России, а также студентами других образовательных учреждений, обучающихся по юридической специальности.

Library of "JurInfoR"

Series: Computer Science and Information
Technologies

V. E. Wolfengagen

**Combinatory logic in programming
Computations with objects through examples
and exercises.** — 2-nd ed. — Moscow, "Center JurInfoR",
2003. — vi+334 p.

The book is intended for computer science students, programmers and professionals who have already got acquainted with the basic courses and background on discrete mathematics. It may be used as a textbook for graduate course on theoretical computer science.

The book introduces a reader to the conceptual framework for thinking about computations with the objects. The several areas of theoretical computer science are covered, including the following: type free and typed λ -calculus and combinatory logic with applications, evaluation of expressions, computations in a category. The topics, covered in the book accumulated much experience in teaching these subjects in graduate computer science courses.

A rich set of examples and exercises, including solutions, has been prepared to stimulate the self studying and to make easier the job of instructor.

ISBN 5-89158-101-9



9 785891 581012



<http://www.jurinfo.ru>