Институт Актуального Образования “ЮрИнфоР-МГУ”

Комбинаторная логика в программировании

Вычисления с объектами в примерах и задачах

доктор технических наук, профессор
В.Э.Вольфенгаген

Москва
2000

В.Э.Вольфенгаген

Комбинаторная логика в программировании

Математика • Компьютерные науки • Информационные технологии •

Вычисления с объектами в примерах и задачах

JurInfoR-MSU

JurInfoR-MSU Institute for Contemporary Education
Electronic Library of JurInfoR-MSU
FTP Service
ftp://msu.jurinfor.ru

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

Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах. - М.: Институт Актуального Образования “ЮрИнфоР-МГУ”, 2000.- 208 с.

В работе рассмотрены наиболее важные методы выработки математического представления *реальных объектов* исходной предметной области. Идеализированные сущности, посредством которых редставляются реальные объекты, носят название *математических объектов*, или просто *объектов*.

Последовательно изложен основной круг задач, сводимых к исчислению объектов. Выбор конкретного варианта исчисления определяется характером тех вычислительных задач, которые предстоит решать в ходе выполнения исследования. Принят способ изложения “от простого к сложному”, который реализуется путем последовательного подбора задач. В ходе их самостоятельного решения читателю предлагается овладеть основными методами и средствами комбинаторной логики, λ-исчисления и категориальной комбинаторной логики. Все задачи снабжены подробными решениями, выполненными вполне элементарными средствами.

Пособие ориентировано на студентов старших курсов, изучающих математические основы объектно-ориентированных вычислений, аспирантов соответствующих специальностей, начинающих и профессионально работающих над продвинутыми проектами программистов. Оно может быть использовано в курсах дискретной математики, математических основ информатики, теории программирования. От читателя не требуется предварительной математической подготовки. Материал частично или полностью может быть использован для самостоятельного изучения как книга “для первого чтения”.

Институт Актуального Образования ЮрИнфоР-МГУ
Московский государственный университет. Юридический факультет
1-ый корпус гуманитарных факультетов, комната 760
Москва, 119899 Россия

Fax: +7 (095) 939-1885
E-mail: vew@jmsuice.msk.ru
WWW: http://msu.jurinfor.ru/e-lib/

ГОСУДАРСТВЕННЫЙ КОМИТЕТ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПО ВЫСШЕМУ ОБРАЗОВАНИЮ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

В.Э. Вольфенгаген

КОМБИНАТОРНАЯ ЛОГИКА В ПРОГРАММИРОВАНИИ

(Вычисления с объектами в примерах и задачах)

PDF copy from LaTeX 2_ε output - generated January 26, 2000
Electronic Edition
Release 3.00

Утверждено
редсоветом института
в качестве учебного
пособия

Москва 1994-2000

УДК 681.3

Вольфенгаген В.Э. † *Комбинаторная логика в программировании*: учебное пособие. М.:МИФИ, 1994.- 209с.

Последовательно изложен основной круг задач, сводимых к исчислению объектов. Выбор конкретного варианта исчисления определяется характером тех вычислительных задач, которые предстоит решать в ходе выполнения исследования. Принят способ изложения “от простого к сложному”, который реализуется путем последовательного подбора задач. В ходе их самостоятельного решения читателю предлагается овладеть основными методами и средствами комбинаторной логики, λ-исчисления и категориальной комбинаторной логики. Все задачи снабжены подробными решениями, выполненными вполне элементарными средствами.

Пособие ориентировано на студентов старших курсов, изучающих математические основы объектно-ориентированных вычислений, аспирантов соответствующих специальностей, начинающих и профессионально работающих над продвинутыми проектами программистов. Оно может быть использовано в курсах дискретной математики, математических основ информатики, теории программирования. От читателя не требуется предварительной математической подготовки. Материал частично или полностью может быть использован для самостоятельного изучения как книга “для первого чтения”.

Рецензент:
Академик Международной Академии Информатизации *Я.А. Хетагуров*

(Разделы 2, 3, 7, 8 написаны совместно с *Гольцевой Л.В.*; разделы 20, 21, 22 -- совместно с *Гавриловым А.В.*; разделы 16--19 -- совместно с *Брызгаловым С.В.*, *Рукодановым А.Ю.*, *Чепурновой И.В.*, *Шориной Е.С.*)

ISBN 5-7262-0009-8

†*Исследование аппликативных вычислительных систем, по результатам которого подготовлена настоящая книга, частично поддержано Российским Фондом Фундаментальных Исследований проект 93-012-943*

Содержание

Предисловие

Кому адресована книга

Эта книга написана в помощь тем программистам, которые 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. 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 программированиия. В настоящее время с объектами работают сотни прикладных программ таких, как Windows, Designer, Corel Draw 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 требует 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*. Однако кроме него есть *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.

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.

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 удается 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* и 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.

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 необходым 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 исчислений 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.

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 удается построить такие известные формальные системы как логика высказываний, логика предикатов, арифметические системы[1] 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, имеющих 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

[1] *Более строго: системы нумералов*

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 представлют известную в программировании работу со стеком рекурсии.

• Применение принципа экстенсиональности

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 экстенсиональности (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 удается разработать арифметические операции. Другими словами, арифметические сущности встраиваются в комбинаторную логику. Эта ситуация хорошо знакома в объектно-ориентированном программировании -- приложение (арифметиче-

ские 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 удается 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 апликативной вычислительной системы

Теперь начинаем продвигаться в глубь математических абстракций и будем увязывать операторный стиль мышления с комбинаторным. Отметим, что в комбинаторной логике используется единственный оператор -- *оператор аппликациии*, 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

Показывается, 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

При динамическом формировании объектов “на лету” 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

Пересматривается техника вычисления значения выражений 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.

Раздел 1

Предварительные сведения

К настоящему времени в теоретических исследованиях в области $computer\ science$ сложился основной математический аппарат. На первый взгляд он не является однородным. Более того, каждое очередное исследование, как правило, содержит построение своего собственного математического аппарата. Очень часто применяемые автором основные математические идеи оказываются завуалированными громоздкими выкладками.

Вместе с тем при более внимательном рассмотрении часто оказывается, что работа использует те или иные идеи либо из *теории категорий*, либо из *комбинаторной логики*, либо из *исчисления λ-конверсий*, либо из всех трех этих 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 $\lambda-$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 *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.

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, в глубь 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 $\lambda-$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. Варианты задания для самостоятельной работы над материалом:

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

Для углубленного изучения разделов настоящего пособия, 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 харак-

тер 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, прорешав рекомендуемые задачи.

Пионерское для computer science исследование выполнено Дж. Маккарти в [83]. Им 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.

В работе [19] можно найти краткое и доступное неспециалисту введение в систему обозначений, принятую в $\lambda-$исчислении и комбинаторной логике. Исчерпывающее изложение всего круга математических идей содержится в [2], однако для изучения требуется предварительная математическая подготовка. Классическое, очень подробное, ясное и обстоятельное обсуждение логического аппарата аппликативных вычислительных систем в [17] 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 [38], [5], [6], [7], [8], [16], [1].

Исследование [9] содержит подробные построения различных вариантов прикладных аппликативных систем, имеющих значение для решения прикладных задач, направленных на разработку информационных систем.

Особое место занимает работа [94]. 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 [60], [61], [62] содержат разработку специальной комбинаторной логики, получившей название *категориальная комбинаторная логика*. На ее основе разработана абстрактная машина, являющаяся 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 [49], [107] можно найти применения в искусственном интеллекте.

Особо следует остановиться на исследованиях Д. Скотта [90],[91], [92],[93],[94],[95],[96],[97]. 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 работах [21],[22], [23], [24],[25], [26], [27], [28],[29]. В этих работах выполнено исследование выразительных возможностей систем с операторами аппликации и (функциональной) абстракции. Элементарные основы комбинаторной логики подробно изложены в [77].

Работы Н. Белнапа могут принести пользу при разработке теории компьютерных информационных систем ([4], [51], [52]). Эти же вопросы затронуты в [59].

Системы программирования в терминах объектов, основанные на технике аппликативных вычислений, в разной степени подробности изложены в [3], [15], [46], [47], [54], [69], [70], [75], [79], [84],

[86], [103]. В качестве основополагающей работы -- для *суперкомбинаторного* программирования -- следует обратить внимание на исследование [79] и [86].

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

1.1 Задания

Общие указания к решению типовой задачи
Формулировка задачи.
Выразить через K и S объект с комбинаторной характеристикой:

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

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

Решение

- Сформулируем постулаты, задающие отношение конвертируемости “=” :

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

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

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

- Определим комбинаторные характеристики объектов K и S:
  $v(Kxy) = vx, \qquad (K)$
  $v(Sxyz) = v(xz(yz)), \qquad (S)$
  которые выражаются в λ-исчислении посредством $K = \lambda xy.x$ и $S = \lambda xyz.xz(yz)$.

- Применяя схемы (K) и (S), убеждаемся в том, что:
  $a = Ka(Ka) \quad (K)$
  $= SKKa. \quad (S)$

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

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

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

- Применяя аналогичным образом схему (K), заключаем, что $Ka(Ka) = a$.

- По правилу транзитивности (τ), если $SKKa = Ka(Ka)$ и $Ka(Ka) = a$, то $SKKa = a$.

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

1.2 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 $B = \lambda xyz.x(yz), S = \lambda xyz.xz(yz)$, $W = \lambda xy.xyy$,
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 $Y_i a = a(Y_i a)$.)

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 объекты (доказать !):*
(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 $\neg, \Pi, \exists$ - одноместные.)

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 $\exists[a] = \exists * [a]$, $\exists = \exists[I]$.

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:

$$Babc = a(bc),\ B^2abcd = a(bcd),\ Wab = abb,$$

$$Cabc = acb,\ \Phi abcd = a(bd)(cd).$$

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

```
list1 a g f x = if null x
                then a
                else g(f(car x)) (list1 a g f(cdr x));
```

5) установить изоморфизм декартово замкнутой категории (д.з.к.) и аппликативной вычислительной системы (ABC);
6) получить отображение, соответствующее отображению каррирования функций (для n-местной функции);
7) проделать вывод основных свойств оболочки Каруби;
8) закодировать термами бестипового λ-исчисления декартово произведение n объектов $(n \geq 2)$ и получить соответствующие выражения для проекций;
9) представить основные функции (аппликативного) языка программирования $LISP$ средствами λ-исчисления и комбинаторной логики.

Формулировка задач для соответствующих целевых исследований.
1) Пусть определение терма $\lambda x.P$ дано индукцией по построению P:

1.1) $\lambda x.x = I$,
1.2) $\lambda x.P = KP$, если $x \notin FV(P)$,
1.3) $\lambda x.P'P'' = S(\lambda x.P')(\lambda x.P'')$.

Исключить все переменные из приводимых ниже λ-выражений:

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

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 ламбда-выражений:

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

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

$length(a1, a2, a3) = 3$,
$sum(1, 2, 3, 4) = 10$,
$product(1, 2, 3, 4) = 24$,
$append(1, 2)(3, 4, 5) = (1, 2, 3, 4, 5)$,
$concat((1, 2), (3, 4), ()) = (1, 2, 3, 4)$,
$mapsquare(1, 2, 3, 4) = (1, 4, 9, 16)$.

Для приведенных примеров выполнить детальную проверку вычислений.

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

(а) $length, sumsquares, reverse, identity$;
(б) $sum, product, append, concat, map$.

5) Вывести следующие равенства:

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

6) Рассматривая семейство функций h:
$h_2 : A \times B \to C$,
$h_3 : A \times B \times C \to D$,
$h_4 : A \times B \times C \times D \to E$,
$\ldots : \ldots$,
найти семейство отображений

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

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

7) Оболочкой Каруби называют следующую категорию (для $a \circ b = \lambda x.a(bx)$):
множество объектов: $\{a \mid a \circ a = a\}$,
множество морфизмов: $Hom(a, b) = \{f \mid b \circ f \circ a = f\}$,
тождественный морфизм: $id\, a = a$,
композиция морфизмов: $f \circ g$.
Пусть $[x, y] = \lambda r.rxy, < f, g >= \lambda t.[f(t), g(t)] = \lambda t.\lambda z.z(ft)(gt)$.
Проверить, что:

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

где $h : A \times B \to C, k : A \to (B \to C)$,
$\varepsilon_{BC} : (B \to C) \times B \to C, x : A, y : B$,
$\Lambda_{ABC} : (A \times B \to C) \to (A \to (B \to C))$,
$p : A \times B \to A, q : A \times B \to B$,
$\varepsilon\circ < k\circ p, q >: A \times B \to C$.
(Указание. Закодировать функции вида $f : A \to B$ термами $B \circ$

$f \circ A = \lambda x.B(f(Ax))$. Воспользоваться равенством: $A \circ A = A(= \lambda x.A(A(x)))$. Учесть, что $\Lambda h = \lambda xy.h[x, y]$.)

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

(Указание. Для случая n=2: $A_0 \times A_1 = \lambda u.[A_0(uK), A_1(u(KI))]$,
$\pi_{02} = \lambda u.(A_0 \times A_1)(u)K$,
$\pi_{12} = \lambda u.(A_0 \times A_1)(u)(KI)$.)

9) Выразить с помощью комбинаторов следующий набор функций языка $LISP$:

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

(Указание.

(1) $A \frown (B \frown C) = (A \frown B) \frown C$ $- Append$,
(2) $A \frown <> = <> \frown A = A$ $- Nil = <>$,
(3) $Null\ A = \begin{cases} 1, & \text{если} A = Nil, \\ 0, & \text{если} A \neq Nil, \end{cases}$
(4) $List\ x =< x >$,
(5) $Car < x1, x2, \ldots, xn >= x1$,
(6) $Cdr < x1, x2, \ldots, xn >=< x2, \ldots, xn >$.)

Общий 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.

Раздел 2

Синтез нового объекта

Теоретические сведения. *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 приложения, 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: тридцатилетний 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 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.

2.1.1 Понятие комбинатора

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 оператор $\mathbf{C}$:

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

Этот закон примет вид:

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

Оператор $\mathbf{C}$ может быть назван комбинатором.
Упражнение 1. Записать коммутативный закон умножения без использования переменных.

2.1.2 Простейшие комбинаторы

Начнем с того, что перечислим основные часто встречающиеся на практике комбинаторы. Простейшим комбинатором является комбинатор тождества I:

$$If = f.$$

- Элементарный *композитор* : $Bfgx = f(gx)$ выражает композицию двух функций f и g.

- Элементарный *дупликатор* W: $Wfx = fxx$ дублирует второй аргумент.
- Введенный выше комбинатор называется элементарным *перемутатором* и переобозначается как C: $Cfxy = fyx$.
- Элементарный *коннектор* S определяется правилом: $Sfgx = fx(gx)$.
- Элементарный *канцелятор* $Kcx = c$ выражает константу (константную функцию) как функцию от x.

Пример 1.

- Пусть $f = sin$ - функция “синус”, $g = exp^5$ - функция возведения в пятую степень. Тогда Bfg - синус от x в пятой степени:
  $Bfgx = B\ sin\ exp^5\ x = sin(exp^5\ x) = sin\ x^5$,
  Bgf - пятая степень синуса,
  $Bgfx = B\ exp^5\ sin\ x = exp^5(sin\ x) = sin^5 x$.
- Если Q -операция возведения в квадрат, то BQQ или WBQ - операция возведения в четвертую степень: $WBQ\,x \overset{W}{=} BQQ\,x \overset{B}{=} Q(Q\ x) = x^4$.
- 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 Основные комбинаторы

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.

Задача 2.1 *Вывод выражения для комбинатора* B .

Формулировка задачи. Выразить через K и S объект с комбинаторной характеристикой:
$Babc = a(bc),$ (B)
пользуясь постулатами $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ-конверсии.
Решение.

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

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

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

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

B--2. Определим комбинаторные характеристики объектов K и S:
$x(Kyz) = xy,$ (K)
$x(Syzw) = x(yw(zw)),$ (S)
которые выражаются в λ*-исчислении посредством:* $K = \lambda xy.x$ *и* $S = \lambda xyz.xz(yz)$.

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

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

Проверим,что B = S(KS)K.

B--1. $S(KS)Kabc = KSa(Ka)bc$, поскольку в схеме (S) можно положить $x = (KS), y = K, z = a$. Тогда, в силу действия

постулата (α): $Sxyz = S(KS)Ka, xz(yz) = (KS)a(Ka)$, 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: $S(KS)Kabc = KSa(Ka)bc$.

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

В--3. Тем же способом, последовательно применяем схемы (S) и (K), постулат (ν) и удаляя несущественные скобки, получаем: $S(Ka)bc = Kac(bc)$; $(Kac)bc = a(bc)$.

В--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:
$Cabc = acb,$ (C)
пользуясь постулатами $\alpha, \beta, \mu, \nu, \sigma, \tau, \xi$ исчисления λ-конверсии.
Решение.

С--1. Сформулируем постулаты, задающие отношение конвертируемости "=" (см. предыдущую задачу).

С--2. Напомним комбинаторные характеристики, возможно, используемых объектов:

$(K)\ Kxy = x,\ (S)\ Sxyz = xz(yz),\ (I)\ Ix = x,\ (B)\ Bxyz = x(yz).$

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

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

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:
$$Wab = abb, \qquad (W)$$
Решение.

W--1. Выпишем характиристики используемых объектов:

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

W--2. Применим эти схемы к abb:

$$\begin{array}{rcll} abb & = & ab(Ib) & (\textit{ no}(I)) \\ & = & SaIb & (\textit{ no}(S)) \\ & = & CSIab. & (\textit{ no}(C)) \end{array}$$

С учетом постулатов получаем: $CSIab = abb$, *то есть* $W = CSI$.

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

$$\begin{array}{rcl} abb & = & ab(Kba) \\ & = & ab(CKab) \\ & = & Sa(CKa)b \\ & = & SS(CK)ab \end{array} \qquad \begin{array}{rcl} abb & = & ab(Kb(Kb)) \\ & = & ab(SKKb) = Sa(SKK)b \\ & = & Sa(K(SKK)a)b \\ & = & SS(K(SKK))ab. \end{array}$$

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:

$$\Psi abcd = a(bc)(bd), \qquad (\Psi)$$

Решение.

Ψ*--1. Сформулируем постулаты, задающие отношение конвертируемости.*

Ψ*--2. Напомним комбинаторные характеристики используемых объектов:*

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

Ψ*--3. Применив эти схемы к a(bc)(bd), получим:*

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

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:

$$B^2abcd = a(bcd), \qquad (B^2)$$

Решение.

B^2*--1. Сформулируем 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:

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

Учитывая постулаты, имеем: $BBBabcd = a(bcd)$, т.е. $B^2 = BBB$.
Ответ. Объект B^2 с комбинаторной характеристикой $B^2abcd = a(bcd)$ 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:
$B^3abcde \ = \ a(bcde),$ (B^3)
Решение.

B^3--1. Сформулируем постулаты, задающие 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:

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

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:
$C^{[2]}abcd \ = \ acdb,$ $(C^{[2]})$
Решение.

$C^{[2]}$--1. Сформулируем постулаты, задающие отношение конвертируемости.

$C^{[2]}$--2. Напомним комбинаторные характеристики, возможно, используемых объектов: $(B)\ Bxyz = x(yz)$, $(C)\ Cxyz = xzy$.

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

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

Учитывая постулаты, имеем: $BC(BC)acbd = acbd$, то есть $C^{[2]} = BC(BC)$.
Ответ. Объект $C^{[2]}$ с 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:
$C_{[2]}abcd = adbc,$ $(C_{[2]})$
Решение.

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

$C_{[2]}$--2. Запишем комбинаторные характеристики, возможно, используемых объектов: $(B^2)\ B^2xyzw = x(yzw)$, $(C)\ Cxyz = xzy$.

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

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

Учитывая постулаты, имеем: $B^2CCabcd = adbc$, т.е. $C_{[2]} = B^2CC$.
Ответ. Объект $C_{[2]}$ с 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:
$C^{[3]}abcde = acdeb$, $(C^{[3]})$
Решение.

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

$C^{[3]}$--2. Запишем комбинаторные характеристики, возможно, используемых объектов: $(B)\ Bxyz = x(yz)$, $(C)\ Cxyz = xzy$, $(C^{[2]})\ Cxyzw = xzwy$.

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

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

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:
$C_{[3]}abcde = aebcd$, $(C_{[3]})$
Решение.

$C_{[3]}$--1. Сформулируем постулаты.

$C_{[3]}$--*2. Запишем комбинаторные характеристики, возможно, используемых объектов:* $(B^2)\ B^2xyzw = x(yzw),\ (C)\ Cxyz = xzy,\ (C_{[2]})\ C_{[2]}xyzw = xwyz.$
Применяя эти схемы к aebcd, получим:

$$\begin{array}{rcll} aebcd & = & Cabecd & (\textit{ по схеме } (C)) \\ & = & C_{[2]}(Cab)cde & (\textit{ по схеме } (C_{[2]})) \\ & = & B^2C_{[2]}Cabcde. & (\textit{ по схеме } (B^2)) \end{array}$$

Учитывая постулаты, имеем: $B^2C_{[2]}Cabcde = aebcd$, то есть $C_{[3]} = B^2C_{[2]}C$.
Ответ. Объект $C_{[3]}$ с комбинаторной характеристикой $C_{[3]}abcde = aebcd$ имеет вид $C_{[3]} = B^2C_{[2]}C$.

Задача 2.11 *Вывод выражения для комбинатора* Ф.

Формулировка задачи. Выразить через K и S и другие предварительно определенные объекты объект с комбинаторной характеристикой:
$$\Phi abcd \ = \ a(bd)(cd), \qquad (\Phi)$$
Решение.

Ф--*1. Сформулируем необходимые постулаты, задающие отношение конвертируемости.*

Ф--*2. Запишем комбинаторные характеристики, возможно, используемых объектов:* $(B^2)\ B^2xyzw = x(yzw),\ (B)\ Bxyz = x(yz),\ (S)\ Sxyz = xz(yz).$

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

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

Учитывая постулаты, имеем: $B^2SBabcd = a(bd)(cd)$, то есть $\Phi = B^2SB$.
Ответ. Объект Ф 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:
$Ya = a(Ya),$ (Y)
Решение.

Y--1. Сформулируем необходимые постулаты, задающие отношение конвертируемости.

Y--2. Запишем комбинаторные характеристики, возможно, используемых объектов: $(S)\ Sxyz = xz(yz),\ (W)\ Wxy = xyy,\ (B)\ Bxyz = x(yz)$.

Y--3. Докажем, что $Y = WS(BWB)$.

$$\begin{array}{rcll}
Ya & = & \underline{WS(BWB)a} & (\textit{ по предположению}) \\
 & = & \underline{S(BWB)(BWB)a} & (\textit{ по схеме } (W)) \\
 & = & \underline{BWBa(BWBa)} & (\textit{ по схеме } (S)) \\
 & = & W(Ba)(BWBa) & (\textit{ по схеме } (B)) \\
 & = & Ba(BWBa)(BWBa) & (\textit{ по схеме } (W)) \\
 & = & a(BWBa(BWBa)) & (\textit{ по схеме } (B)) \\
 & = & a\underline{(S(BWB)(BWB)a)} & (\textit{ по схеме } (S)) \\
 & = & a\underline{(WS(BWB)a)} & (\textit{ по схеме } (W)) \\
 & = & a(Ya). & (\textit{ по предположению})
\end{array}$$

Следовательно, одно из представлений объекта Y таково: $Y = WS(BWB)$.
Ответ. Объект Y с комбинаторной характеристикой $Ya = a(Ya)$ имеет вид $Y = WS(BWB)$.

Раздел 3

Неподвижная точка

Теоретические сведения. *Вычисления с неподвижной точкой являются представлением цикличности в программах. Комбинаторная логика предоставляет специальный концепт-комбинатор Y, называемый комбинатором неподвижной точки, который математически выражает цикл в вычислениях.*

3.1 Абстракция

Для каждого терма M и переменной x терм $[x]M$, называемый *абстракцией M по x*, определяется индукцией по построению терма 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 5. $[x]xy \overset{(iv)}{=} S([x]x)([x]y) \overset{(i)}{=} SI([x]y) \overset{(ii)}{=} SI(Ky)$.
Теорема 1. $\forall M, N, x: \ ([x]M)N = [N/x]M$.
Доказательство см. в [77].

3.2 Мультиабстракция

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

$$[x_1, \ldots, x_m]M = [x_1]([x_2](\ldots([x_m]M)\ldots)).$$

Пример 1. $[x, y]x = [x]([y]x) \overset{(ii)}{=} [x](Kx) \overset{(iii)}{=} K$.

3.3 Локальная рекурсия

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

$$E1\ where\ x\ =\ \ldots\ x\ \ldots$$

преобразуется в

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

где Y - комбинатор неподвижной точки, определяемый равенством:

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

Для функции f выражение Yf - это неподвижная точка f . Взаимная рекурсия в $where$-предложении вида

$$E1\ where\ f\ x\ =\ \ldots\ g\ \ldots$$
$$g\ y\ =\ \ldots\ f\ \ldots$$

транслируется сначала в выражение:

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

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

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

которое может быть скомпилировано в выражение:

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

с использованием уже известного правила.

3.4 Основные задачи

Задача 3.1 *Исследовать свойства заданного объекта.*

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

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

Решение.

Y--1. *Вид объекта* $Y:\ Y = (\lambda x.(P(xx)a))(\lambda x.(P(xx)a))$.

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

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

Итак, имеем $Y = PYa$.
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.*

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

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

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:

$$\begin{array}{rcll} Ya & = & S(BWB)(BWB)a & (\textit{ по определению}) \\ & = & BWBa(BWBa) & (\textit{ по схеме } S) \\ & = & W(Ba)(BWBa) & (\textit{ по схеме } B) \\ & = & Ba(BWBa)(BWBa) & (\textit{ по схеме } W) \\ & = & a(BWBa(BWBa)) & (\textit{ по схеме } B) \\ & = & a(S(BWB)(BWB)a) & (\textit{ по схеме } S) \\ & = & a(Ya). & (\textit{ по определению}) \end{array}$$

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

Задача 3.3 *Исследовать свойства заданного объекта.*

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

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

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:

$$(W):\ Y = WS(BWB) = S(BWB)(BWB). \qquad (3.1)$$

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

Y--4. Применим объект Y к a:

$$\begin{array}{rcll} Ya & = & S(BWB)(BWB)a & (\textit{ по определению}) \\ & = & BWBa(BWBa) & (\textit{ по схеме } S) \\ & = & W(Ba)(BWBa) & (\textit{ по схеме } B) \\ & = & Ba(BWBa)(BWBa) & (\textit{ по схеме } W) \\ & = & a(BWBa(BWBa)) & (\textit{ по схеме } B) \\ & = & a(S(BWB)(BWB)a) & (\textit{ по схеме } S) \\ & = & a(Ya) & (\textit{ по определению}). \end{array}$$

Y--5. Учитывая (3.1), получаем $Ya = a(Ya)$.

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

Задача 3.4 *Исследовать свойства заданного объекта.*

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

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

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).

$$\begin{array}{rcll} XX & = & (\lambda x.f(xx))(\lambda x.f(xx)) & (\textit{ по определению}) \\ & = & f((\lambda x.f(xx))(\lambda x.f(xx))) & (\textit{ по постулату } \beta) \\ & = & f(XX). & (\textit{ по определению}) \end{array}$$

Значит,

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

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

$$\begin{array}{rcll} Y_0a & = & (\lambda f.XX)a & (\textit{ по определению}) \\ & = & (\lambda f.f(XX))a & (\textit{ по равенству (*)}) \\ & = & (\lambda f.f((\lambda x.f(xx))(\lambda x.f(xx))))a & (\textit{ по определению } X) \\ & = & a((\lambda x.a(xx))(\lambda x.a(xx))) & (\textit{ по постулату } \beta) \\ & = & a((\lambda f.((\lambda f.f(xx))(\lambda x.f(xx))))a) & (\textit{ по постулатам } \beta,\ \xi) \\ & = & a((\lambda f.(XX))a) & (\textit{ по определению } X) \\ & = & a(Y_0a). & (\textit{ по определению } Y_0) \end{array}$$

Учитывая постулат транзитивности τ, получаем: $Y_0a = a(Y_0a)$.
Ответ. Комбинаторная характеристика объекта Y_0 имеет следующий вид: $Y_0a = a(Y_0a)$.

Задача 3.5 *Исследовать свойства заданного объекта.*

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

$$Y_1 \quad = \quad \lambda f.XX, \text{ где } X = \lambda x.f(xx). \qquad (Y_1)$$

Решение.

Y_1*--1.* *Вид объекта* $Y_1 : Y_1 = Y_0(\lambda y.\lambda f.f(yf))$, *где* $Y_0 = \lambda f.XX, X = \lambda x.f(xx)$.

Y_1*--2.* *Имеем* $Y_0 a = a(Y_0 a)$,

$$\begin{array}{rcll} Y_1 a & = & Y_0(\lambda y.\lambda f.f(yf))a & (\textit{ по определению}) \\ & = & (\lambda y.\lambda f.f(yf))(Y_0(\lambda y.\lambda f.f(yf)))a & (\textit{ по схеме } (Y_0)) \\ & = & (\lambda yf.f(yf))Y_1 a & (\textit{ по схеме } (Y_1)) \\ & = & a(Y_1 a). & (\textit{ по постулату } \beta) \end{array}$$

Итак, имеем $Y_1 a = a(Y_1 a)$.
Ответ. Комбинаторная характеристика объекта
$Y_1 = Y_0(\lambda y.\lambda f.f(yf))$ имеет вид: $Y_1 a = a(Y_1 a)$.

Раздел 4

Экстенсиональность

Теоретические сведения. *Для модели M обычно требуется выполнение некоторого свойства. В частности, пусть объект F построен, самое большее, из свободных переменных $x_0, \ldots, x_{n-1}$ - 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 $x_0, \ldots, x_{n-1}$ справедливо суждение $fx_0 \ldots x_{n-1} = F$. 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:

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

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

Для аппликативных структур справедлива более сильная форма комбинаторной полноты: существует тот единственный концепт f, что для всяких свободных в F переменных $fx_0 \ldots x_{n-1} = F(x_0, \ldots, x_{n-1})$:

$$\mathbf{I}f.\forall x_0 \ldots \forall x_{n-1}(fx_0 \ldots x_{n-1} = F(x_0, \ldots, x_{n-1})).$$

*В этой записи символ "***I***" использован как сокращение для "тот единственный ... , что ... ". Это, фактически, принцип свертывания: комбинация объектов F, среди которых в качестве свободных переменных использованы только лишь $x_0, \ldots, x_{n-1}$, сворачивается до единственного объекта (концепта) f со всеми теми и только теми свойствами, которые присущи комбинации $F(x_0, \ldots, x_{n-1})$. Это выполняется только для экстенсиональных аппликативных структур и позволяет от сложной формы записи переходить к анализу единственного объекта, снижая громоздкость в рассуждениях.*

Задача 4.1 *Доказать требуемое равенство.*

Формулировка задачи. Доказать, что равенство

$$\lambda xy.xy \quad = \quad \lambda x.x \qquad\qquad (\eta)$$

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

η--1. Приведем применяемые постулаты:

$$\lambda x.bx \quad = \quad b, \ \textit{где } x \ \textit{не имеет свободных вхождений в } b, \qquad (\eta)$$

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

η--2. Применяя эти постулаты, получим:

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

Приведем сокращенную форму изложения доказательства:

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

Раздел 5

Нумералы

Теоретические сведения. *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.

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

Формулировка задачи. Нумералы - это следующие объекты: $\bar{n} = \lambda xy.(x^n)y$, где n - натуральное число из множества $\{1, 2, 3, \dots\}$. Показать, что нумералы это объекты с характеристикой:

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

Решение.

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

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

Таким образом, $x^4 y = x(x(x(xy)))$.

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

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

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

$$\begin{array}{lll} \bar{n}ab & = & (SB)^n(KI)ab = SB((SB)^{n-1}(KI))ab \quad (\textit{ по определению}) \\ & = & Ba((SB)^{n-1}(KI)a)b \quad (\textit{ по схеме } (S)) \\ & = & a((SB)^{n-1}(KI)ab) \quad (\textit{ по схеме } (B)) \\ & = & a(\overline{n-1}ab) \quad (\textit{ по определению}) \\ & = & a(a^{n-1}b) \quad (\textit{ по определению}) \\ & = & a^n b = (\lambda xy.x^n y)ab. \quad (\textit{ по определению}) \end{array}$$

Ответ. Нумералы $(\lambda xy.x^n y)$ 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"):

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

Решение.

σ--1. Комбинаторная характеристика нумералов: $\bar{n}ab = a^n b$.

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

$$\begin{array}{rcll} \sigma\bar{n}ab & = & (\lambda xyz.xy(yz))\bar{n}ab & (\textit{ по определению}) \\ & = & \bar{n}a(ab) & (\textit{ по постулату } (\beta)) \\ & = & a^n(ab) & (\textit{ по определению } (\bar{n})) \\ & = & a^{n+1}b & (\textit{ по определению}) \\ & = & \overline{n+1}ab. & (\textit{ по определению}) \end{array}$$

Итак, $\sigma\bar{n}ab = \overline{n+1}ab$, то есть $\sigma\bar{n} = \overline{n+1}$.
Ответ. Функция $\sigma = \lambda xyz.xy(yz)$ есть функция следования для нумералов $\bar{n} = \lambda xy.x^n y$.

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

Формулировка задачи. Показать, что функция

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

есть функция определения длины списка x :

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

Решение.

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

$$Null\ x = \begin{cases} \bar{1}, & \textit{если } x = NIL\ (x - \textit{ пустой список}), \\ \bar{0}, & \textit{в противном случае;} \end{cases} \qquad (Null)$$

$$\begin{array}{rcll} \bar{1} & = & \lambda xy.xy = \lambda x.x = I, & (\bar{1}) \\ \bar{0} & = & \lambda xy.y = KI, & (\bar{0}) \\ \sigma\ \bar{n} & = & \overline{n+1}, & \end{array}$$

$$Cdr\ x = \begin{cases} NIL =<>, & \textit{для } x =< a_1 >, \\ < a_2, \ldots, a_n >, & \textit{для } x =< a_1, a_2, \ldots, a_n > . \end{cases} \qquad (Cdr)$$

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

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

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

Length--3. Приложим функцию Length к списку x, состоящему из одного элемента: $x =< a >$:

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

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 ($x \neq Nil$*), где знак* $\neq$ *означает "не конвертируется к":*

$$
\begin{array}{l}
Length\ x = \lambda y.Nullx\bar{0}(\sigma(Length(Cdr\ x))y) \\
\quad = \lambda y.\bar{0}\bar{0}(\sigma(Length(Cdr\ x))y) \\
\quad = \lambda y.\sigma(Length(Cdr\ x))y \\
\quad = \lambda y.\sigma\overline{n-1}\ y \\
\quad = \lambda y.\bar{n}y \\
\quad = \lambda y.(\lambda xz.x^n z)y = \lambda y.\lambda z.y^n z = \lambda yz.y^n z = \bar{n}.
\end{array}
$$

Ответ. Функция $Length$ действительно вычисляет длину списка.

Раздел 6

Комбинаторы с типами

Теоретические сведения. *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.*

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. Говорят, 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:

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

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

Предлагается, пользуясь аксиомами и правилом (F), приписать типы основным комбинаторам:

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

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 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:

$Exp1 = (a_1, (b_1, a_1))$
$Exp2 = a_1$
$Exp3 = ((b_1, a_1), ((a_2, b_2), (a_2, c_2)))$
$Exp4 = (b_1, a_1)$
$Exp5 = ((a_2, b_2), (a_2, c_2))$
$Exp6 = (a_2, b_2)$
$Exp7 = (a_2, c_2)$

$$\dfrac{\dfrac{\vdash \#(K) = Exp1 \qquad \vdash \#(S) = Exp2}{\vdash \#(S) = Exp3 \qquad \vdash \#(KS) = Exp4}\ (F)}{\dfrac{\vdash \#(S(KS)) = Exp5 \quad \vdash \#(K) = Exp6}{\vdash \#(S(KS)K) = Exp7}\ (F)}\ (F)$$

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* $a_1 = ((a, (b, c)), ((a, b), (a, c)))$*. 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* $b_2 = (a, (b, c)), c_2 = ((a, b), (a, c))$.
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* $b_3 = a,\ a_3 = (b, c)$*. 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:*
$Exp1 = ((a_1,(b_1,c_1)),((a_1,b_1),(a_1,c_1)))$
$Exp2 = (a_1,(b_1,c_1))$
$Exp3 = ((a_1,b_1),(a_1,c_1))$

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

$\#(SB)$*--2. По схеме* $(FB) :\vdash \#(B) = ((b,c),((a,b),(a,c)))$*, но у нас* $\vdash \#(B) = (a_1,(b_1,c_1))$*, 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.*

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

$\#(Z_0)$*--3. По схеме* $(FI) :\vdash \#(I) = a_1$*, где* $a_1 = (a,a)$*; тип* b_1 *отличен от* a_1*, то есть* $b_1 = b$ *(*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.* $(F(SB)) :\vdash \#(SB) = (((b,c),(a,b)),((b,c),(a,c)));(F(KI)) :\vdash \#(KI) = (b,(a,a))$.

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

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

$\#(Z_1)$*--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.* $Z_2 = SB(SB(KI))$.

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

  $\#(Z_2)$*--3.* $Exp1 = (((b_1, c_1), (a_1, b_1)), ((b_1, c_1), (a_1, c_1)))$
  $Exp2 = ((b_1, c_1), (a_1, b_1))$
  $Exp3 = ((b_1, c_1), (a_1, c_1))$

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

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

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

  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.* $Z_n = (SB)^n(KI) = SB((SB)^{n-1}(KI))$, *где* $n > 2$.

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

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

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

  $\#(Z_n)$--*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.* $W = CSI$.

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

$\#(W)$--*3.* $Exp1 = ((b_1,(a_1,c_1)),(a_1,(b_1,c_1)))$
$Exp2 = (b_1,(a_1,c_1))$
$Exp3 = (a_1,(b_1,c_1))$
$Exp4 = (a_1)$
$Exp5 = (b_1,c_1)$

$$\frac{\dfrac{\vdash \#(C) = Exp1 \qquad \vdash \#(S) = Exp2}{\vdash \#(CS) = Exp3}\ (F) \qquad \vdash \#(I) = Exp4}{\vdash \#(SCI) = Exp5}\ (F)$$

$\#(W)$--*4. По схеме* $(FS) : \vdash \#(S) = (b_1, (a_1, c_1))$, *но* $\vdash\ \#\,(S) = ((a, (b, c)), ((a, b), (a, c)))$. *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. $B^2 = BBB$ имеет тип $((b\ c)((d(a\ b))(d(a\ c))))$.

Задача 6.8 *Тип* $\#(B^3)$.

Решение.

$\#(B^3)$*--1.* $B^3 = BBB^2$.

$\#(B^3)$*--2.*

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

где $(a_1(b_1\ c_1)) = ((b\ c)((d(a\ b))(d(a\ c))))$, *то есть* $a_1 = (b\ c)$, $b_1 = (d(a\ b)), c_1 = (d(a\ c))$. *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.* $C^{[2]} = BC(BC)$.

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

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

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.*
$$\begin{aligned} Exp1 &= ((a_1(b_1(c_1\ d_1)))(a_1(c_1(b_1\ d_1)))) \\ Exp2 &= (a_1(b_1(c_1\ d_1))) \\ Exp3 &= (a_1(c_1(b_1\ d_1))) \end{aligned}$$

$$\frac{(BC) : Exp1 \quad (BC) : Exp2}{(BC(BC)) : Exp3},\ (F)$$

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.* $C^{[3]} = BC(BC^{[2]})$.

$\#(C^{[3]})$*--2.*
$$\begin{array}{rcl} Exp1 & = & ((b_1\ c_1)((a_1\ b_1)(a_1\ c_1))) \\ Exp2 & = & (b_1\ c_1) \\ Exp3 & = & ((a_1\ b_1)(a_1\ c_1)) \\ Exp4 & = & ((b_2\ c_2)((a_2\ b_2)(a_2\ c_2))) \\ Exp5 & = & (b_2\ c_2) \\ Exp6 & = & ((a_2\ b_2)(a_2\ c_2)) \end{array}$$

$$\dfrac{(B):\ Exp1 \quad (C):\ Exp2}{(BC):\ Exp3}\ (F)$$

$$\dfrac{\dfrac{(B):\ Exp4 \quad (C^{[2]}):\ Exp5}{(BC^{[2]}):\ Exp6}\ (F)}{(BC(BC^{[2]})):\ (a_1\ c_1)},\ (F)$$

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 есть* $a_3 = a_2 = e$, $c_2 = (b_3\ c_3)$, $b_3 = c$, $c_3 = (a(b\ c))$.

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

Задача 6.11 *Тип* $\#(C_{[2]})$.

Решение.

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

$\#(C_{[2]})$--*2.* $Exp1 = ((b_1\ c_1)((d_1(a_1\ b_1))(d_1(a_1\ c_1))))$
$Exp2 = (b_1\ c_1)$
$Exp3 = ((d_1(a_1\ b_1))(d_1(a_1\ c_1)))$
$Exp4 = (d_1(a_1\ b_1))$
$Exp5 = (d_1(a_1\ c_1))$

$$\dfrac{\dfrac{(B^2):\ Exp1 \qquad\qquad (C):\ Exp2}{(B^2C):\ Exp3}\ (F) \quad (C):\ Exp4}{(B^2CC):\ Exp5}\ (F)$$

где $(b_1\ c_1) = ((a(b\,c))(b(a\,c)))$, $(d_1(a_1\ b_1)) = ((a_2(b_2\ c_2))(b_2(a_2\ c_2)))$, *то есть* $b_1 = (a(b\,c))$, $c_1 = (b(a\,c))$, $a_1 = b_2$, $d_1 = (a_2(b_2\ c_2))$, $b_1 = (a_2\ c_2)$. *Имеем:* $d_1 = (a(b_2(b\ c)))$, $a_1 = b_2$, $c_1 = (b(a\ c))$. *Пусть* $b_2 = d$. *Тогда* $(B^2CC):\ (d_1(a_1\ c_1)) = ((a(d(b\,c)))(d(b(a\,c))))$.

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

Задача 6.12 *Тип* $\#(C_{[3]})$.

Решение.

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

$\#(C_{[3]})$--*2.* $Exp1 = ((b_1\ c_1)((d_1(a_1\ b_1))(d_1\ (a_1\ c_1))))$
$Exp2 = (b_1\ c_1)$
$Exp3 = ((d_1(a_1\ b_1))(d_1\ (a_1\ c_1)))$
$Exp4 = (d_1(a_1\ b_1))$
$Exp5 = (d_1(a_1\ c_1))$

$$\dfrac{\dfrac{(B^2):\ Exp1 \qquad\qquad (C_{[2]}):\ Exp2}{(B^2C_{[2]}):\ Exp3}\ (F) \quad (C):\ Exp4}{(B^2C_{[2]}C):\ Exp5}\ (F)$$

где $(b_1\ c_1) = ((a(b(c\,d)))(b(c(a\,d))))$, $(d_1\ (a_1\ b_1)) = ((a_2\ (b_2\ c_2))(b_2\ (a_2\ c_2)))$. *Имеем* $d_1 = (a_2(b_2\ c_2)) = (a(b_2(b(c\ d))))$, $a_1 = b_2$, $c_1 = (b(c(a\ d)))$. *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. $C_{[3]} = B^2 C_{[2]} C$ имеет тип $((a(e(b(c\ d))))(e(b(c(a\ d)))))$.

Задача 6.13 *Тип* $\#(\Phi)$.

Решение.

$\#\Phi$*--1.* $\Phi = B^2SB$.

$\#\Phi$*--2.*
$$\begin{array}{rcl} Exp1 & = & ((b_1\ c_1)((d_1(a_1\ b_1))(d_1(a_1\ c_1)))) \\ Exp2 & = & (b_1\ c_1) \\ Exp3 & = & ((d_1(a_1\ b_1))(d_1(a_1\ c_1))) \\ Exp4 & = & (d_1(a_1\ b_1)) \\ Exp5 & = & (d_1(a_1\ c_1)) \end{array}$$

$$\dfrac{\dfrac{(B^2):\ Exp1 \qquad\qquad (S):\ Exp2}{(B^2S):\ Exp3}\ (F) \qquad (B):\ Exp4}{(B^2SB):\ Exp5}\ (F)$$

где $(b_1\ c_1) = ((a(b\,c))((a\,b)(a\,c)))$, $(d_1(a_1\,b_1)) = ((b_2\,c_2)((a_2\,b_2)(a_2\,c_2)))$. *Итак,* $d_1 = (b_2\ c_2) = (b_2(b\ c))$, $a_1 = (a_2\ b_2) = (a\ b_2)$, $c_1 = ((a\ b)(a\ c))$. *Пусть* $b_2 = d$; *тогда* $(B^2SB)\ :\ (d_1(a_1\ c_1)) = ((d(b\ c))((a\ d)((a\ b)(a\ c))))$.

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

Задача 6.14 *Тип* $\#(Y)$.

Решение.

$\#Y$*--1.* $Y = WS(BWB)$.

$\#Y$*--2.*
$$\dfrac{(W):\ ((a_1(a_1\ b_1))(a_1\ b_1))\ \ (S):\ (a_1(a_1\ b_1))}{(WS):(a_1b_1)},\ (F)$$

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

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

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

где $(a_1(a_1\,b_1)) = ((a(b\,c))((a\,b)(a\,c)))$, $(b_2\,c_2) = ((a_3(a_3\,b_3))(a_3\,b_3))$, $(a_2\,b_2) = ((b_4c_4)((a_4b_4)(a_4c_4)))$, $a_1 = (a_2\ c_2)$.
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* $(F):\ \#(Y) = (a, b)$,

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

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. $D = C_{[2]}I$.

#D--2.

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

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.

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

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

$$\frac{(S):\ (a_3(b_3\ c_3))((a_3\ b_3)(a_3\ c_3))\ \ (\mathsf{BBS}):\ (a_2\ c_2)}{\mathcal{S}(\mathcal{BBS}):\ (a_3\ b_3)(a_3\ c_3)}$$

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

$$\frac{\mathcal{S}(\mathcal{BBS}):\ (a_3\ b_3)(a_3\ c_3)\quad (\mathcal{KK}):\ (b_4\ a_4)}{S(BBS)(KK):\ (a_3\ c_3)},$$

где $(a_3\,b_3)=(b_4\,a_4),\ (b_2\,c_2)=((b_4\,c_4),((a_4\,b_4),(a_4\,c_4))),\ a_4=(a_5(b_5\ a_5)),\ (a_2\ c_2)=(a_3(b_3\ c_3)),\ (a_2\ b_2)=(a\ b\ c)(a\ b)(a\ c)$.

Имеем:

$$\begin{cases} a_3 &= a_2=(a\ b\ c),\\ c_2 &= (b_3\ c_3)=(a_4\ b_4)(a_4\ c_4),\\ b_2 &= (a\ b)(a\ c)=(b_4\ c_4),\\ b_3 &= a_4=(a_5\ b_5\ a_5). \end{cases}$$

Далее, $c_4=(a\ c),\ b_4=(a\ b),\ b_3=(a_4\ b_4)=(a_4\ a\ b)=(b\ a\ b)$.

Итак, $c_3=(a_4\ c_4)=(b\ a\ c)$.
Ответ. $C=S(BBS)(KK)$ имеет тип: $((a,(b,\ c)),(b,(a,\ c)))$.

Раздел 7

Базис $I,\ K,\ S$

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.*

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:

$$\begin{array}{lll} (1) & \lambda x.x & = I, \\ (2) & \lambda x.P & = KP, \text{ если не принадлежит } FV(P), \\ (3) & \lambda x.PQ & = S(\lambda x.P)(\lambda x.Q). \end{array}$$

Исключить все переменные из приводимых λ-выражений:

$$1.\ \lambda xy.yx;\ 2.\ \lambda fx.xx;\ 3.\ f = \lambda x.B(f(Ax)).$$

Решение.

P--1.

$$\begin{array}{rcl}
\lambda xy.yx & \stackrel{def}{=} & \lambda x.(\lambda y.yx)) \\
& \stackrel{(3)}{=} & \lambda x.(S(\lambda y.y)\lambda y.x)) \\
& \stackrel{(1),\ (2)}{=} & \lambda x.SI(Kx) \\
& \stackrel{(3)}{=} & S(\lambda x.SI)(\lambda x.Kx) \\
& \stackrel{(2)}{=} & S(K(SI))(S(\lambda x.K)(\lambda x.x)) \\
& \stackrel{(1),\ (2)}{=} & S(K(SI))(S(KK)I)
\end{array}$$

P--2.

$$\begin{array}{rcl}
\lambda fx.fxx & \stackrel{def}{=} & \lambda f.(\lambda x.fxx) \\
& \stackrel{(3)}{=} & \lambda f.(S(\lambda x.fx)(\lambda x.x)) \\
& \stackrel{(1),\ (3)}{=} & \lambda f.S(S(\lambda x.f)(\lambda x.x))I \\
& \stackrel{(1),\ (2)}{=} & \lambda f.S(S(Kf)I)I \\
& \stackrel{(3)}{=} & S(\lambda f.S(S(Kf)I))(\lambda f.I) \\
& \stackrel{(2),\ (3)}{=} & S(S(\lambda f.S)(\lambda f.S(Kf)I))(KI) \\
& \stackrel{(2),\ (3)}{=} & S(S(KS)(S(\lambda f.S(Kf))(\lambda f.I)))(KI) \\
& \stackrel{(2),\ (3)}{=} & S(S(KS)(S(S(\lambda f.S)(\lambda f.Kf)))(KI)))(KI) \\
& \stackrel{(2),\ (3)}{=} & S(S(KS)(S(S(KS)(S(\lambda f.K)(\lambda f.f))(KI)))(KI) \\
& \stackrel{(1),\ (2)}{=} & S(S(KS)(S(S(KS)(S(KK)I)(KI)))(KI)
\end{array}$$

P--3.

$$\begin{array}{rcl}
f & \stackrel{def}{=} & \lambda x.b(f(ax)) \\
& \stackrel{(3)}{=} & S(\lambda x.b)(\lambda x.f(ax)) \\
& \stackrel{(2),\ (3)}{=} & S(Kb)(S(\lambda x.f)(\lambda x.ax)) \\
& \stackrel{(2),\ (3)}{=} & S(Kb)(S(Kf)(S(\lambda x.a)(\lambda x.x))) \\
& \stackrel{(2),\ (1)}{=} & S(Kb)(S(Kf)(S(Ka)I))
\end{array}$$

Раздел 8

Базис I, B, C, S

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 представление[1].

[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.

8.1 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 , W, B, K и S, K:

$$xyz = xyz, \qquad Sxyz = xz(yz),$$
$$Wxy = xyy, \qquad Kxy = x.$$
$$Bxyz = x(yz),$$
$$Kxy = x;$$

Для любого синтаксического объекта V, составленного из различных переменных $x_1, \ldots, x_n$ с помощью операции аппликации можно найти комбинатор X, составленный из базисных, такой, что: $X\ x_1, \ldots, x_n = V$. Например, если $V = xz(yz)$, то можно указать комбинатор X из базисной системы S, K, такой, что $X\ xyz = xz(yz)$. 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 переменные переупорядочиваются;

3) с помощью W устраняются дублирования переменных;

4) с помощью K вводятся переменные, отсутствующие в V.

Пример 1. Построим в первой базисной системе комбинатор X со свойством $X\ abc = ac(bc)$:

$$\underset{x}{(ac)}\ \underset{yz}{(bc)} = \underset{x}{B}\ \underset{yz}{(ac)bc} \overset{B}{=} \underset{x}{(BBa)}\ \underset{zy}{cbc} \overset{C}{=} \underset{x}{(C(BBa)b)}\ \underset{yy}{cc} \overset{W}{=}$$

$$\overset{W}{=} \underset{x}{W}\ \underset{y}{(C(BBa)}\ \underset{z}{b)c} \overset{B}{=}$$

$$\overset{B}{=} \underset{x}{(BW)}\ \underset{y}{(C}\ \underset{z}{(BBa))bc} \overset{B}{=} \underset{x}{(B(BW)C)}\ \underset{y\ z}{((BB)a)bc} \overset{B}{=}$$

$$\stackrel{B}{=} B(B(BW)C)(BB)abc.$$

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, $X = B(B(BW)C)(BB)$.

8.2 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 базиса[2].

Задача 8.1 *Представить терм* $M = \lambda x.PQ$ *комбинаторами* $I,\ B,\ C,\ S$.

Формулировка задачи. Пусть определение терма такого, что принадлежит $FV(PQ)$ дано индукцией по построению (здесь " $\in''$ означает "принадлежит", а " $\notin''$ - "не принадлежит"):

$$(1)\quad \lambda x.x = I,$$

$$(2)\quad \lambda x.PQ = \begin{cases} BP(\lambda x.Q), & \text{если } \notin FV(P) \text{ и } x \in FV(Q), \quad (a)\\ C(\lambda x.P)Q, & \text{если } \in FV(P) \text{ и } x \notin FV(Q), \quad (b)\\ S(\lambda x.P)(\lambda x.Q), & \text{если } \in FV(P) \text{ и } x \in FV(Q). \quad (c) \end{cases}$$

Исключить все переменные из приводимых ниже λ-выражений:
1. $\lambda xy.yx$; 2. $\lambda fx.fxx$.
Решение.
M--1.

$$\begin{aligned} \lambda xy.yx &= \lambda x.(\lambda y.yx)\\ &\stackrel{(2)(b)}{=} \lambda x.(C(\lambda y.y)x)\\ &\stackrel{(1)}{=} \lambda x.CIx\\ &\stackrel{(2)(a)}{=} B(CI)(\lambda x.x)\\ &\stackrel{(1)}{=} B(CI)I. \end{aligned}$$

[2]На решение задачи разложения в базисе можно взглянуть иначе. Поскольку всякие 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.

Проверка: $B(CI)Ixy = CI(Ix)y = Iy(Ix) = Iyx = yx$.

M--2.

$$\begin{array}{rcl}
\lambda fx.fxx & = & \lambda f.(\lambda x.fxx) \\
& \overset{(2)(c)}{=} & \lambda f.S(\lambda x.fx)(\lambda x.x) \\
& \overset{(1),\,(2)(a)}{=} & \lambda f.S(Bf(\lambda x.x))I \\
& \overset{(1)}{=} & \lambda f.S(BfI)) \\
& \overset{(2)(b)}{=} & C(\lambda f.S(BfI))I \\
& \overset{(2)(a)}{=} & C(BS(\lambda f.BfI))I \\
& \overset{(2)(b)}{=} & C(BS(C(\lambda f.Bf)I))I \\
& \overset{(2)(a)}{=} & C(BS(C(BB(\lambda f.f))I))I \\
& \overset{(1)}{=} & C(BS(C(BBI)I))I.
\end{array}$$

Упражнение. Доказать, что набор комбинаторов C, W, B, K проявляет свойство базисности.

Раздел 9

Применения неподвижной точки Y

Теоретические сведения. *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 $Yf = f(Yf)$. Ссылающиеся на себя определение функции f можно искать в виде $f = Ef$, где f не имеет свободного вхождения в E.

9.1 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:

$$
\begin{array}{lcl}
FAC & = & (\lambda n.IF(=\ n\ 0)1 \\
 & & \quad (\times n\ FAC(-\ n\ 1))) = \\
FAC & = & (\lambda n.IF(=\ n\ 0)1 \\
 & & \quad (\times n\ (\lambda n.IF(=\ n\ 0)1 \\
 & & \quad\quad (\times n\ FAC(-\ n\ 1)))(-\ n\ 1))) = \\
FAC & = & (\lambda n.IF(=\ n\ 0)1 \\
 & & \quad (\times n\ (\lambda n.IF(=\ n\ 0)1 \\
 & & \quad\quad (\times n(\lambda n.IF(=\ n\ 0)1 \\
 & & \quad\quad\quad (\times n\ FAC(-\ n\ 1)))(-\ n\ 1)))(-\ n\ 1))) = \\
\ldots & & \ldots .
\end{array}
$$

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.

9.2 Использование комбинатора Y

В качестве основного примера, выполнение которого иллюстрирует основные эффекты вычислений с неподвижной точкой воспользуемся функцией факториала FAC. Запишем функцию FAC сокращенно:

$$FAC = \lambda n.(\ldots FAC \ldots). \tag{9.1}$$

Применяя λ-абстракцию, получим:

$$\lambda fac.FAC\ fac = (\lambda fac.(\lambda n.(\ldots fac \ldots)))FAC. \tag{9.2}$$

По правилу (η) получаем:

$$\lambda fac.FAC\ fac = FAC,$$

и определение (9.2) запишется в виде:

$$FAC = (\lambda fac.(\lambda n.(\ldots fac \ldots)))FAC. \tag{9.3}$$

Полагая $H = \lambda fac.(\lambda n.(\ldots fac \ldots))$, получим:

$$FAC = H\ FAC. \tag{9.4}$$

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 1. Числа 0 и 1 являются неподвижными точками функции $f = \lambda x. \times x\ x$, то есть $f0 = 0$ и $f1 = 1$. Действительно,

$$\begin{array}{rcll} f0 & = & (\lambda x. \times x\ x)0 = & (\beta) \\ & = & \times\ 0\ 0 = 0, & \\ f1 & = & (\lambda x. \times x\ x)1 = & (\beta) \\ & = & \times\ 1\ 1 = 1. & \end{array}$$

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*. Фактически, пришлось дать ее частное "доказательство" для функции FAC. Полученное решение дает *нерекурсивное* определение FAC. Существо 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. Для того, чтобы убедится, что определенная таким образом функция FAC работает правильно, приложим ее к некоторому аргументу, например, к 1:

$$
\begin{array}{ll}
FAC\ 1 = Y\ H\ 1 & (FAC) \\
= H(Y\ H)1 & (Y) \\
= (\lambda fac.(\lambda n.(IF(=\ n\ 0)1)(\times n(fac(-\ n\ 1))))(Y\ H)1 & (H) \\
= (\lambda n.IF(-\ n\ ))1(\times n(Y\ H(-\ n\ 1))))1 & (\beta) \\
= IF(=\ 1\ 0)1(\times 1(Y\ H(-\ 1\ 1)))) & (\beta) \\
= \times 1(Y\ H)0 = \times 1(H(Y\ H))0 & (Y) \\
= \times 1((\lambda fac.\lambda n.IF(=\ n\ 0)1(\times n(fac(-\ n\ 1))))(Y\ H)0 & (H) \\
= \times 1((\lambda n.IF(=\ n\ 0)1(\times n(Y\ H(-\ n\ 1))))0) & (beta) \\
= \times 1(IF(=\ 0\ 0)1(\times 0(Y\ H(-\ 0\ 1)))) & (\beta) \\
= \times 11 & \\
= 1. &
\end{array}
$$

9.3 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.

Начиная работать со списками, можно оценить возможности системы программирования $LISP$[1]. Таким образом, в этой системе нет

[1]По своей основе $LISP$ имеет, практически, все возможности бестипового ламбда-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*. Ре-

различия 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.

Задача 9.1 *Пользуясь функцией поиска неподвижной точки Y, выразить определения приводимых ниже функций:*

$$
\begin{array}{lll}
2) & sum & = \lambda x.if\ null\ x \\
 & & \quad then\ 0 \\
 & & \quad else\ (car\ x) + sum(cdr\ x), \\
3) & product & = \lambda x.if\ null\ x \\
 & & \quad then\ 1 \\
 & & \quad else\ (car\ x)\ \times\ product(cdr\ x), \\
4) & append & = \lambda x.\lambda y.if\ null\ x \\
 & & \quad then\ y \\
 & & \quad else\ (list((car\ x)(append(cdr\ x)y)), \\
5) & concat & = \lambda x.if\ null\ x \\
 & & \quad then\ () \\
 & & \quad else\ append(car\ x)(concat(cdr\ x)), \\
6) & map & = \lambda f\lambda x.if\ null\ x \\
 & & \quad then\ () \\
 & & \quad else\ list((f(car\ x))(map\ f(cdr\ x))).
\end{array}
$$

$$
\begin{array}{l}
length(a_1, a_2, a_3) = 3; \\
sum(1, 2, 3, 4) = 10; \\
product(1, 2, 3, 4) = 24; \\
append(1, 2)(3, 4, 5) = (1, 2, 3, 4, 5); \\
concat((1, 2), (3, 4), ()) = (1, 2, 3, 4); \\
map\ square\ (1, 2, 3, 4) = (1, 4, 9, 16).
\end{array}
$$

Для примеров “обращения” к каждой из функций выполнить проверку.

ализованные механизмы рекурсии с математической точки зрения иллюстрируют эффект вычислений с неподвижной точкой. Обычно в языке дополнительно предлагаются нерекурсивные средства организации циклических вычислений.

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

$$\begin{aligned} length &= \lambda x.if\ null\ x \\ &\quad then\ 0 \\ &\quad else\ 1 + length(cdr\ x), \end{aligned}$$

перепишем в виде:

$$\begin{aligned} length &= (\lambda f.\lambda x.if\ null\ x \\ &\quad then\ 0 \\ &\quad else\ 1 + f(cdr\ x))length. \end{aligned}$$

length--2. Отсюда следует, что

$$\begin{aligned} length &= Y(\lambda f.\lambda x.if\ null\ x \\ &\quad then\ 0 \\ &\quad else\ 1 + f(cdr\ x)). \end{aligned}$$

Тем самым желаемая комбинаторная характеристика получена.

length--3. Произведем проверку определения для списка длины 2, то есть возьмем $x = (a_1, a_2)$:

$$\begin{aligned} &length(a_1, a_2) = Y(\lambda f\lambda x.if\ null\ x \\ &\qquad\qquad then\ 0 \\ &\qquad\qquad else\ 1 + f(cdr\ x))(a_1, a_2) \\ &= (\lambda f\lambda x.if\ null\ x \\ &\qquad then\ 0 \\ &\qquad else\ 1 + f(cdr\ x))(Y(\ldots))(a_1, a_2) \\ &= if\ null\ (a_1, a_2)\ then\ 0\ else\ 1 + (Y(\ldots))(cdr\ (a_1, a_2)) \\ &= 1 + (\lambda f\lambda x.if\ null\ x\ then\ 0\ else\ 1 + f(cdr\ x))(Y(\ldots))(a_2) \\ &= 1 + if\ null\ (a_2)\ then\ 0\ else\ 1 + (Y(\ldots))nil \\ &= 1 + 1(\lambda f\lambda x.if\ null\ x\ then\ 0\ else\ 1 + f(cdr\ x))(Y(\ldots))nil \\ &= 1 + 1 + 0 = 2. \end{aligned}$$

map--1. Для функции map исходное определение

$$map \;=\; \lambda f.\lambda x.\quad if\ null\ x \\ then\ () \\ else\ ((f(car\ x)) : (mapf(cdr\ x))$$

перепишем в виде:

$$map \;=\; (\lambda m.\lambda f.\lambda x.\quad if\ null\ x \\ then\ () \\ else\ (f(car\ x)) : (mf(cdr\ x)))\ \ map.$$

Отсюда следует, что

$$map \;=\; Y(\lambda m.\lambda f.\lambda x.\quad if\ null\ x \\ then\ () \\ else\ (f(car\ x)) : (mf(cdr\ x))).$$

Проверка для $f = square,\ x = (2,3)$*:*

$$\begin{aligned} &map\ square\ (2,3) = \\ &\quad = (\lambda m\lambda f\lambda x.if\ null\ x \\ &\qquad then\ () \\ &\qquad else\ (f(car\ x)) : (mf(cdr\ x)))(Y(\ldots))square\ (2,3) \\ &\quad = (square\ 2) : ((Y(\ldots))square\ (3)) \\ &\quad = (square\ 2) : ((\lambda m.\lambda f.\lambda x.\ldots.)(Y(\ldots))square\ (3)) \\ &\quad = (square\ 2) : ((square\ 3) : ((Y(\ldots))square\ ()) \\ &\quad = (square\ 2) : ((square\ 3) : ()) \\ &\quad = (4,9). \end{aligned}$$

В данном случае символ ":" принят для обозначения инфиксной формы функции $list$*, поэтому принимаем в качестве соглашения об обозначениях, что* $x : (y : (z : ())) = x, y, z = (x, y, z)$*.*

Раздел 10

Функция list1

Теоретические сведения. *Определим функцию* $list1$ *следующим образом:*

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

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 представители класса функций. В качестве одной из таких обобщенных функций предлагается функция $list1$.
Основная конструкция, которой будем пользоваться -- это список, 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:

$$\begin{array}{lcl} null & : & \textit{список} \rightarrow \textit{булевский}, \\ car & : & \textit{непустой список} \rightarrow \textit{(список + атом)}, \\ cdr & : & \textit{непустой список} \rightarrow \textit{список}, \\ list & : & \textit{(атом + список)} \rightarrow \textit{(список} \rightarrow \textit{список)}. \end{array}$$

Эти операции связаны друг с другом следующим образом:

$$\begin{aligned} null\ () &= true, \\ null\ (list\ x\ y) &= false, \\ car\ (list\ x\ y) &= x, \\ cdr\ (list\ x\ y) &= y, \\ list\ (car\ z)(cdr\ z) &= z. \end{aligned}$$

Кроме того, примем сокращение:

$$list\ x\ y = x : y,$$

и поэтому для $n \geq 2$ *воспользуемся соглашением об обозначении:*

$$x_1,\ x_2,\ x_3, \ldots, x_n = x_1 : (x_2 : (x_3 : (\ldots\ x_n) : \ldots () \ldots)).$$

Задача 10.1 *Исследовать свойства функции*

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

Воспользовавшись следующими определениями:

$$Ix = x,\ Kxy = x,\ postfix\ x\ y = append\ y(ux),$$

где (ux) *- обозначение списка, состоящего из единственного элемента , выразить функции:*

(а) $length$, $sumsquares$, $reverse$, $identity$;

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

Уточненная формулировка задачи. *Воспользуемся следующими опре-*

делениями функций:

(*а*) $ux = x : (),$
$length\ x = if\ null\ x$
$\quad then\ 0\ else\ 1 + length(cdrx),$
$sumsquares\ x = if\ null\ x$
$\quad then\ ()$
$else(square(car\ x))\ +\ sumsquares(cdr\ x),$
$reverse\ x = if\ null\ x\ then\ ()$
$\quad else\ append(reverse(cdr\ x))(ux),$
$identity\ x = x,$
$square\ x = if\ null\ x$
$\quad then\ 0$
$\quad else\ x\ \times\ x;$

(*б*) $sum\ x = if\ null\ x$
$\quad then\ 0$
$\quad else(car\ x)\ +\ sum(cdr\ x),$
$product\ x = if\ null\ x$
$\quad then\ 1$
$\quad else(car\ x)\ \times\ product(cdrx),$
$append\ x\ y = if\ null\ x$
$\quad then\ y$
$\quad else\ list(car\ x)(append(cdr\ x)y),$
$concat\ x = if\ null\ x$
$\quad then\ ()$
$\quad else\ append(car\ x)(concat(cdr\ x)),$
$map\ f\ x = if\ null\ x$
$\quad then\ ()$
$\quad else(f(car\ x)) : (map\ f(cdr\ x)).$

Выполнить шаги алгоритмов для следующих примеров:

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

Решение. Принимается, что $postfix\ x\ y\ =\ append\ y\ (ux)$.

$list1$--*1. Рассмотрим случай (а):*

$$\begin{aligned} length &= list1\ 0\ plus\ (K\ 1), \\ sumsquares &= list1\ 0\ plus\ square, \\ reverse &= list1\ (\ )\ postfix\ I, \\ identity &= list1\ (\ )\ list\ I. \end{aligned}$$

$list1$--*2. Рассмотрим случай (б):*

$$\begin{aligned} sum &= list1\ 0\ plus\ I, \\ product &= list1\ 1\ multiply\ I, \\ append\ x\ y &= list1\ y\ list\ I\ x, \\ concat &= list1\ (\ )\ append\ I, \\ map\ f &= list1\ (\ )\ list\ f. \end{aligned}$$

Для завершения решения требуется подставить параметры и произвести детальные вычисления. Кроме того, предполагается выполнение проверки примеров.

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 *интенсионал*. Выбирая различные значения параметров, или указывая *соотнесения*, на деле получаем целое семейство функций-индивидов. Перечислив элементы этого семейства, получаем *экстенсионал* 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.

Раздел 11

Изоморфизм д.з.к. и АВС

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:

$$[x,y] \;=\; \lambda r.rxy,$$
$$< f,g > = \; \lambda t.[f(t),g(t)] = \lambda t.\lambda z.z(f(t))(g(t)).$$

Тот факт, что f является отображением из A в B (в содержательном смысле) будем обозначать посредством $f : A \to B$. Введем в рассмотрение следующие отображения:

$$h : A \;\times\; B \to C \;\; \textit{для}\; x : A, y : B,$$
$$\Lambda_{ABC}h : A \to (B \to C),$$
$$\Lambda_{ABC} : (A \;\times\; B \to C) \to (A \to (B \to C)),$$
$$k : A \to (B \to C),$$
$$\varepsilon_{BC} : (B \to C) \;\times\; B \to C,$$
$$\varepsilon\circ < k \circ p, q >: A \;\times\; B \to C,$$
$$p : A \;\times\; B \to A, \quad q : A \;\times\; B \to B.$$

В дальнейшем эти отображения будут часто использоваться без явного упоминания их типов, если только не возникает двусмысленности в толковании.

Задача 11.1 *Поскольку в оболочке Каруби выводимо:*

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

а также выводимо:

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

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:*

$$\begin{array}{lcl}
h & = & \varepsilon\circ < (\Lambda h) \circ p,\; q >; \\
h[x,y] & = & (\varepsilon_{BC}\circ < (\Lambda h) \circ p,\; q >)[x,y] \\
 & = & \varepsilon_{BC}(< (\Lambda h) \circ p,\; q > [x,y]) \\
 & = & \varepsilon_{BC}[((\Lambda h) \circ p)[x,y],\; q[x,y]] \\
 & = & \varepsilon_{BC}[(\Lambda h)(p[x,y]),\; q[x,y]] \\
 & = & \varepsilon_{BC}[((\Lambda h)x), y] \\
 & = & (((\Lambda h)x)y) = \Lambda\; h\; x\; y \\
 & \equiv & h'\; x\; y.
\end{array}$$

Проделанные вычисления нуждаются в правильном приписывании объектам типов[1]. *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:*

$$\begin{array}{lcl} k & = & \Lambda(\varepsilon\circ < k \circ p, q >); \\ kxy & = & \Lambda(\varepsilon\circ < k \circ p, q >)xy \\ & = & (\lambda u.\lambda v.(\varepsilon\circ < k \circ p, q >)[u,\ v])xy \\ & = & \varepsilon\circ < k \circ p,\ q > [x, y] \\ & \equiv & k'[x, y]. \end{array}$$

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 придти к заключению, что

$$(\ \times\ \ \to\ ) \ \cong\ (\ \to (\ \to\ )),$$

где символ "$\cong$" читается как "изоморфно".

[1] У выражения $((\Lambda h)x)$ тип $B \to C$. Выражению y приписан тип B. Выражение $(((\Lambda h)x)y)$ получает тип C.

Раздел 12

Каррирование

Теоретические сведения. *В программировании часто приходится устанавливать различие между понятием “оператор” и понятием “функция”. В первом случае руководствуются алгебраическими идеями, когда у всякого оператора заранее предполагается арность (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 стороны. 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.*

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 (аппликировании) 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.

Между обоими видами систем можно устанавливать различные связи, проявляя потенциальные возможности и того, и другого подхода. В частности, зададимся вопросом, как средствами ABC (пользуясь операторами аппликации и абстракции) выразить содержательное представление 2-местных, 3-местных, ..., n-местных функций, трактуемых как операторы. Воспользуемся следующими соглашениями об обозначениях:

$$[x,y] = \lambda r.rxy, h : A \times B \to C,$$
$$Curry_{ABC} : (A \times B \to C) \to (A \to (B \to C)).$$

Таким образом, h считается обычным двухместным оператором, а $Curry$ - преобразованием из операторного вида в аппликативный [1]*:*

$$Curry_{ABC}\ h = \lambda xy.h[x,y],$$
$$\lambda xy.h[x,y] \ : \ A \to (B \to C).$$

Задача 12.1 *Рассмотрим семейство функций h:*

$h_2 : A \times B \to C,$
$h_3 : A \times B \times C \to D,$
$h_4 : A \times B \times C \times D \to E,$
$\ldots : \ldots .$

Найти семейство отображений:

$$Curry_{ABC},\ Curry_{(A\times B)CD},\ Curry_{(A\times B\times C)DE},\ \ldots,$$

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 $h_3 : (A \times B) \times C \to D$. Тогда $\Lambda_{(A \times B)CD}h_3 = \lambda xy.h_3[x,y] : A \times B \to (C \to D)$. Теперь можно считать, что $\Lambda_{(A \times B)CD}h_3 = h'_2$, и поэтому следующая идея

[1] В теоретических исследованиях вместо обозначения "$Curry$" часто используется обозначение "Λ"; в дальнейшем будем использовать именно это последнее обозначение.

состоит в подстановке вместо первой переменной - упорядоченной пары переменных, то есть

$$\begin{aligned}\Lambda_{AB(C\to D)}(\Lambda_{(A\times B)CD}h_3) &= \\ &= \lambda uv.(\Lambda_{(A\times B)CD}h_3)[u,v] \\ &= \lambda uv.(\lambda xy.h_3[x,y])[u,v] \\ &= \lambda uv.(\lambda y.h_3[[u,v],y]) \\ &= \lambda uvy.(h_3[[u,v],y]) : A\to\ (B\to\ (C\to\ D)) \\ &= (\Lambda_{AB(C\to D)}\ \circ\ \Lambda_{(A\times B)CD})\ h_3.\end{aligned}$$

Curry--2. Пусть теперь $h_4 : A\times B\times C\times D\to\ E$, *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:

$$\Lambda_{((A\times B)\times C)DE}h_4 = \lambda xy.h_4[x,y] : ((A\times B)\times C)\to\ (D\to\ E).$$

Шаг 2:

$$\begin{aligned}&\Lambda_{((A\times B)C(D\to E)}(\Lambda_{((A\times B)\times C)DE}h_4) = \\ &= \lambda uv.(\Lambda_{((A\times B)\times C)DE}h_4)[u,v] \\ &= \lambda uvy.h_4[[u,v],y] : A\times B\to\ (C\to\ (D\to\ E)).\end{aligned}$$

Шаг 3:

$$\begin{aligned}&\Lambda_{AB(C\to\ (D\to\ E)}(\lambda uvy.h_4[[u,v],y]) = \\ &= \lambda\overline{xy}.(\lambda uvy.h_4[[u,v],y])[\overline{x},\overline{y}] \\ &= \lambda\overline{xy}vy.h[[[\overline{x},\overline{y}],v],y].\end{aligned}$$

Обсуждая полученное в пунктах $(Curry-1)$ и $(Curry-2)$ решение, можно заметить, что функции каррирования имеют вид:

$$\begin{aligned}&\Lambda_{(A\times B\times C)DE} = \\ &\quad = \Lambda_{AB(C\to\ (D\to\ E))}\ \circ\ \Lambda_{(A\times B)C(D\to\ E)}\ \circ\ \Lambda_{(A\times B)\times C)DE}.\end{aligned}$$

Для целей отыскания решения в общем случае перепишем это равенство в виде:

$$\begin{aligned}&\Lambda_{(A_1\times A_2\times A_3)A_4B} = \\ &\quad = \Lambda_{A_1A_2(A_3\to\ (A_4\to\ B))}\circ\Lambda_{(A_1\times A_2)A_3(A_4\to\ B)}\circ\Lambda_{((A_1\times A_2)\times A_3)A_4B}.\end{aligned}$$

Остается самостоятельно получить соответствующее равенство для n-местных функций[2].

[2]Это выполняется индукцией по числу аргументных мест.

Раздел 13

Оболочка Каруби

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 $\mathcal{L}$ - некоторое исчисления λ-конверсий. Оболочкой Каруби для $\mathcal{L}$, обозначаемой через $\mathcal{C}(\mathcal{L})$, будем считать категорию, определяемую следующим образом. Положим $a \circ b = \lambda x.a(bx)$ для $a,\ b$ принадлежащих $\mathcal{L}$, где "$\circ$" - знак композиции функций.

Множество объектов категории :	$\{a \in \mathcal{L} \mid a \circ a = a\}$.
Множество морфизмов :	$Hom(a,b) = \{f \in \mathcal{L} \mid b \circ f \circ a = f\}$.
Тождественный изоморфизм :	$id\ a = a$.
Композиция морфизмов :	$f \circ g$.

Задача 13.1 *Показать, что $\mathcal{C}(\mathcal{L})$ - категория. Предлагается это проверить самостоятельно, выполнив согласование с какой-либо формой определения категории.*

Уточнение формулировки задачи. Примем следующие определения необходимых для проведения исследования объектов. Напомним, что в данном случае необходимость выполнения следующих равенств предполагается заранее:

$$A = \lambda x.A(A(x)) = A \circ A, \quad (A)$$
$$F = \lambda x.B(f(A(x))) = B \circ f \circ A. \quad (f)$$

1. Декартово произведение:

$$A \times B = \lambda u \lambda z.z(A(u(\lambda x \lambda y.x)))(B(u(\lambda xy.y))).$$

2. Проекция на первый и на второй элемент соответственно:

$$p_{AB} = \lambda u.(A \times B)(u)(\lambda x \lambda y.x),\ p_{AB} : A \times B \to A;$$
$$q_{AB} = \lambda u.(A \times B)(u)(\lambda x \lambda y.y),\ q_{AB} : A \times B \to B.$$

3. Спаривание функций:

$$< f, g >= \lambda t \lambda z.z(f(t))(g(t)) = \lambda t.[f(t), g(t)],$$
$$f : C \to A, g : C \to B, < f, g >: C \to (A \times B).$$

4. Множество отображений (функциональное пространство):

$$(A \to B) = \lambda f.B \circ f \circ A.$$

5. Аппликация (приложение функций к аргументу):

$$\varepsilon_{BC} = \lambda u.C(u(\lambda xy.x)(B(u(\lambda xy.y)))),$$
$$\varepsilon_{BC} : (B \to C) \times B \to C.$$

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”):

$$\Lambda_{ABC}h = \lambda x \lambda y.h(\lambda z.z(x)(y)),$$
$$h : (A \times B) \to C, \Lambda_{ABC}h : A \to (B \to C).$$

Требуется доказать следующее:

- Свойства проекций:

$$p_{AB} \circ < f, g >= f, \quad q_{AB} \circ < f, g >= g,$$
$$< p_{AB} \circ h, q_{AB} \circ h >= h.$$

- Пусть $h : (A \times B) \to C,\ k : A \to (B \to C)$. Тогда

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

  где $\Lambda = \Lambda_{ABC},\ p = p_{AB},\ \varepsilon = \varepsilon_{BC},\ q = q_{AB}$.

Решение. Доказательство сводится к проверке свойств введенных объектов.

$\mathcal{C}(\mathcal{L})$*--1. Заметим, что отображение* $h : (A \times B) \to C$ *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)*.*

$\mathcal{C}(\mathcal{L})$*--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, $h[x_1, x_2] = C(h[A\ x_1, B\ x_2])$.

$\mathcal{C}(\mathcal{L})$*--3. Еще раз обратим внимание на необходимость учета следующих равенств:*

$$x = \lambda z.a(x(\mathbf{1}\ z)) = A \circ x,$$
$$f = \lambda z.B(f(A\ z)) = B \circ f \circ A,$$

где $\mathbf{1} = \lambda y.y = I$.

$\mathcal{C}(\mathcal{L})$--*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:*

$$\begin{aligned}(A \times B)[u,v] &= \lambda z.z(A([u,v]K))(B([u,v](K\ I))) \\ &= \lambda z.z(A\ u)(B\ v) \\ &= [Au, Bv].\end{aligned}$$

$\mathcal{C}(\mathcal{L})$--*5. Проверка свойств проекций приводит к следующим характеристикам:*

$$p_{AB}([u,v]) = (A \times B)[u,v]K = [Au, Bv]K = A\ u,$$
$$q_{AB}([u,v]) = (A \times B)[u,v](K\ I) = [A\ u, B\ v](K\ I) = B\ v.$$

$\mathcal{C}(\mathcal{L})$--*6. Рассматривая упорядоченную пару* $[f,x]$, *покажем, как с помощью отображения* ε_{BC} *получить аппликацию* f *к* x:

$$\begin{aligned}\varepsilon_{BC}([C \circ f \circ B, B \circ x]) &= \varepsilon_{BC}([f,x]) \\ &= C([C \circ f \circ B, B \circ x]K(B([C \circ f \circ B, B \circ x](K\ I)))) \\ &= C((C \circ f \circ B)(B(B \circ x))) \\ &= C(f(B\ x)) \\ &= (C \circ f \circ B)(x) \\ &= f(x).\end{aligned}$$

Отметим необходимость учета свойств композиции.[1] *По существу, далее будем обосновывать простое равенство:* $\Lambda_{ABC}\ h\ x\ y = h([x,y])$ *(см. раздел 6, где определена функция каррирования).*

$\mathcal{C}(\mathcal{L})$--*7. Пусть* t *представимо упорядоченной парой, то есть* $t = [t_1, t_2]$. *Тогда*

$$\begin{aligned}(\varepsilon \circ < (\Lambda\ h) \circ p, q >)t &= \varepsilon[(\Lambda\ h)(p\ t), q\ t] \\ &= \varepsilon[(\lambda xy.h[x,y])(pt), qt] \\ &= \varepsilon[(\lambda xy.h[x,y])t_1, t_2] \\ &= \varepsilon[\lambda y.h[t_1,y], t_2] \\ &= (\lambda y.h[t_1,y])t_2 \\ &= h[t_1,t_2].\end{aligned}$$

Получено характеристическое равенство:

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

[1] Напомним, что $C \circ f = f,\ B \circ x = x$.

$\mathcal{C}(\mathcal{L})$--*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":*

$$\begin{aligned}\Lambda(\varepsilon \circ < k \circ p, q >)t_1\ t_2 &= (\varepsilon \circ < k \circ p, q >)([t_1, t_2]) \\ &= \varepsilon[k(p\ t), q\ t] = \varepsilon[kt_1, t_2] \\ &= (\lambda u.C(uK(B(u(K\ I)))))[k\ t_1, t_2] \\ &= C(k\ t_1(B(t_2))) \\ &= C(k\ t_1\ t_2) \\ &= k\ t_1\ t_2.\end{aligned}$$

Тем самым[2] *обосновано характеристическое равенство*

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

Все введенные равенства действительно имеют место, что завершает доказательство.

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.

[2]Следует учесть, что $B(t_2) = t_2$.

Раздел 14

Произведение и проекции

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 настоящей работы требуется небольшой запас определений: $K = \lambda xy.x,\ I = \lambda x.x,\ [x, y] = \lambda r.rxy$ *(упорядоченная пара).*
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

$$A_0 \times A_1 = \lambda u.[A_0(u\ K), A_1(u(K\ I))],$$
$$\pi_0^2 = \lambda u.(A_0 \times A_1)(u)K,$$
$$\pi_1^2 = \lambda u.(A_0 \times A_1)(u)(K\ I).)$$

Решение.

$\times$ π*--1. Прежде всего попытаемся отыскать закономерности в построении терма.*

$\times$ π*--2. Рассмотрим случай* $n = 3$*.*

$$\begin{aligned} A_0 \times A_1 \times A_2 &= (A_0 \times A_1) \times A_2 \\ &= \lambda u.[(A_0 \times A_1)(u\ K), A_2(u(K\ I))] \\ &= \lambda u.[(\lambda v.[A_0(v\ K), A_1(v(KI))])(uK), A_2(u(K\ I))] \\ &= \lambda u.[[A_0((u\ K)K), A_1((u\ K)(K\ I))], A_2(u(K\ I))]. \end{aligned}$$

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.

$$\begin{aligned}\pi_0^3 &= \lambda u.(A_0 \times A_1 \times A_2)(u)(K)(K),\\ \pi_1^3 &= \lambda u.(A_0 \times A_1 \times A_2)(u)(K)(K\ I),\\ \pi_2^3 &= \lambda u.(A_0 \times A_1 \times A_2)(u)(KI).\end{aligned}$$

Проверка.

$$\begin{aligned}\pi_0^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2]K\ K\\ &= K[x_0, x_1]x_2K\\ &= [x_0, x_1]K\\ &= x_0;\\ \pi_1^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2]K(K\ I)\\ &= K[x_0, x_1]x_2(K\ I) = x_1;\\ \pi_2^3[[x_0, x_1], x_2] &= [[x_0, x_1], x_2](K\ I)\\ &= (KI)[x_0, x_1]x_2 = I\ x_2 = x_2.\end{aligned}$$

Действительно, для определенного ранее вида произведения объекты $\pi_j^3,\ j = 0,\ 1,\ 2$ ведут себя как проекции.

$\times$ *π--3. Рассмотрим случай $n = 4$.*

$$\begin{aligned}A_0 \times A_1 \times A_2 \times A_3 &= (A_0 \times A_1 \times A_2) \times A_3\\ &= \lambda u.[(A_0 \times A_1 \times A_2)(u\ K), A_3(u(K\ I))]\\ &= \lambda u.[[[A_0(((u\ K)K)K), A_1(((u\ K)K)(K\ I))],\\ &\qquad A_2((u\ K)(K\ I))], A_3(u(K\ I))].\end{aligned}$$

Утверждение.

$$\begin{aligned}\pi_0^4 &= \lambda u.(A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K)(K),\\ \pi_1^4 &= \lambda u.(A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K)(K\ I),\\ \pi_2^4 &= \lambda u.(A_0 \times A_1 \times A_2 \times A_3)(u)(K)(K\ I),\\ \pi_3^4 &= \lambda u.(A_0 \times A_1 \times A_2 \times A_3)(u)(K\ I).\end{aligned}$$

Проверка.

$$\begin{aligned}\pi_0^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3]K\ K\ K = x_0,\\ \pi_1^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3]K\ K(K\ I) = x_1,\\ \pi_2^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3]K(K\ I) = x_2,\\ \pi_3^4[[[x_0, x_1], x_2], x_3] &= [[[x_0, x_1], x_2], x_3](K\ I) = x_3.\end{aligned}$$

Перейдем теперь к обобщению произведения.
Обобщение. *По индукции можно доказать, что:*

$$\pi_0^1 = I,\ \pi_0 = \lambda y.y\ K, \pi_1 = \lambda y.y(K\ I);$$
$$\pi_n^{n+1} = \lambda y.y(K\ I), i \leq 1;\ \pi_j^{n+1} = \pi_j^n \circ \pi_0, 0 \leq j \leq n-1.$$

Выполнить доказательство самостоятельно.
Указание 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)*.
$\pi_2^3 = \lambda y.y(KI) = \pi_1$,
$\pi_1^3 = \pi_1^2 \circ \pi_0 = \pi_1 \circ \pi_0$,
$\pi_0^3 = \pi_0^2 \circ \pi_0 = \pi_0^1 \circ \pi_0 \circ \pi_0$.
$n = 3$ *(упорядоченные 4-ки)*.
$\pi_3^4 = \pi_1$,
$\pi_2^4 = \pi_2^3 \circ \pi_0 = \pi_1 \circ \pi_0$,
$\pi_1^4 = \pi_1^3 \circ \pi_0 = \pi_1^2 \circ \pi_0 \circ \pi_0 = \pi_1 \circ \pi_0 \circ \pi_0$,
$\pi_0^4 = \pi_0^3 \circ \pi_0 = \pi_0^2 \circ \pi_0 \circ \pi_0$
$= \pi_0^1 \circ \pi_0 \circ \pi_0 \circ \pi_0 = \pi_0 \circ \pi_0 \circ \pi_0$.

Частные случаи проекций декартова произведения в преобразованном виде:

$n = 2$.
$\pi_1^2 = \lambda y.y(K\ I)$,
$\pi_0^2 = \lambda y.y\ K$.
$n = 3$.
$\pi_2^3 = \lambda y.y(KI)$,
$\pi_1^3 = \lambda y.\pi_1(\pi_0\ y)$
$= \lambda y.\pi_1(y\ K)$
$= \lambda y.(y\ K)(K\ I)$,
$\pi_0^3 = \lambda y.\pi_0(\pi_0\ y)$
$= \lambda y.\pi_0(y\ K)$
$= \lambda y.(yK)K$.
$n = 4$.
$\pi_3^4 = \lambda y.y(KI)$,
$\pi_2^4 = \lambda y.\pi_1(\pi_0\ y) = \lambda y.\pi_1(y\ K) = \lambda y.(y\ K)(K\ I)$,
$\pi_1^4 = \lambda y.\pi_1(\pi_0(\pi_0\ y)) = \lambda y.\pi_1(\pi_0(y\ K))$
$= \lambda y.\pi_1((y\ K)K) = \lambda y.y((y\ K)K)(K\ I)$,
$\pi_0^4 = \lambda y.\pi_0(\pi_0(\pi_0\ y)) = \lambda y.\pi_0(\pi_0(y\ K))$
$= \lambda y.\pi_0((y\ K)K) = \lambda y.y((y\ K)K)K$.

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 преполагает 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.

Раздел 15

Погружение $LISP$ в ABC

Теоретические сведения. *Одна из наиболее трудных проблем при разработке пользовательского (языкового) интерфейса состоит в правильном встраивании приложения в программную среду. Как известно, ее решению способствует 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, зада-

ющие отношение конвертируемости "conv" (обозначаемое знаком "="):

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

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

$$(\tau) \qquad \frac{a = b;\ \ b = c}{a = c} \qquad (\sigma) \qquad \frac{a = b}{b = a} \qquad (\rho) \qquad a = a$$

$$(\eta) \qquad \lambda x.bx = b,\ x \notin b. \qquad (\xi) \qquad \frac{a = b}{\lambda x.a = \lambda x.b}$$

Формулировка задачи. Выразить с помощью комбинаторов следующий набор функций $LISP$-системы:

$$\{Append,\ Nil,\ Null,\ List,\ Car,\ Cdr\}. \qquad (LISP)$$

Уточненная формулировка задачи.

- Рассмотрим свойства этих функций языка $LISP$.
  - ▷ Функция $Append$ осуществляет конкатенацию двух списков. Эта функция обладает свойством ассоциативности:

    $$A \frown B \frown C = (A \frown B) \frown C, \qquad (Append)$$

    где $A,\ B,\ C$ - произвольные списки, знак "$\frown$" - инфиксная форма записи функции $Append$.
  - ▷ Пустой список обозначается как $<>$ и эквивалентен объекту Nil. Очевидно, что

    $$A \frown <> = <> \frown A = A. \qquad (Nil), (<>)$$

  - ▷ Функция $Null$ распознает пустой список:

    $$\left.\begin{array}{l} Null\ A = \overline{1},\ \text{если } A = Nil, \\ Null\ A = \overline{0},\ \text{в противном случае.} \end{array}\right\} \qquad (Null)$$

  - ▷ Функция $List$ строит из атома список длины 1:

    $$Listx = < x >, \qquad (List)$$

    где x - атом, $< x_1,\ x_2,\ \ldots\ , x_n >$ - список длины n.

▷ Функция Car выбирает первый элемент списка:

$$Car < x_1,\ x_2,\ \ldots\ ,\ x_n >= x_1. \qquad (Car)$$

▷ Функция Cdr убирает первый элемент из списка:

$$Cdr < x_1,\ x_2,\ \ldots\ ,\ x_n >=< x_2,\ \ldots\ , x_n > . \qquad (Cdr)$$

- На основании этих свойств сформулируем следующие схемы аксиом:

$$\begin{aligned}
Append\ a\ (Append\ b\ c) &= Append(Append\ a\ b)c, && (15.1)\\
Append\ Nil\ a &= Append\ a\ Nil, && (15.2)\\
Null\ Nil &= \overline{1}, && (15.3)\\
Null(Append(List\ a)b) &= \overline{0}, && (15.4)\\
Car(Append(List\ a)b) &= a, && (15.5)\\
Cdr(Append(List\ a)b) &= b, && (15.6)
\end{aligned}$$

где $a,\ b,\ c$ - произвольные обьекты.

- 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 обьект B с комбинаторной характеристикой $(B):\ Babc = a(bc)$ (схема аксиом (15.1)):

$$\begin{aligned}
Ba(Bbc)x &= a(Bbcx) && \text{(по схеме } (B))\\
&= a(b(cx)) && \text{(по схеме } (B))\\
&= Bab(cx) && \text{(по схеме } (B))\\
&= B(Bab)cx. && \text{(по схеме } (B))
\end{aligned}$$

Учитывая правило транзитивности (τ), имеем:

$$Ba(Bbc)x = B(Bab)cx.$$

В $\eta - \xi$-исчислении доказуемо, что для переменной x:

$$\frac{z_1x = z_2x}{z_1 = z_2},$$

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:*

$$\begin{array}{lll}
(1) & z_1x = z_2x \Rightarrow \lambda x.z_1x = \lambda x.z_2x, & (\textit{ по } (\xi)) \\
(2) & \lambda x.z_1x = z_1, & (\textit{ по } (\eta)) \\
(3) & z_1 = \lambda x.z_1x, & (\textit{ по } (\sigma), (2)) \\
(4) & \lambda x.z_2x = z_1, & (\textit{ по } (\tau), (1), (3)) \\
(5) & z_2 = \lambda x.z_2x, & (\textit{ по } (\eta)) \\
(6) & z_1 = z_2. & (\textit{ по } (\tau), (4), (5))
\end{array}$$

Итак, схема аксиом (15.1) доказана.

$LISP$--2. *Докажем схему аксиом (15.2), принимая, что* $Nil \leftrightarrow I$ *выполняется*[1], *причем* $(I):\ Ia = a$.

$$\begin{array}{rcll}
BIax & = & I(ax) & (\textit{ по } (B)) \\
 & = & ax & (\textit{ по } (I)) \\
 & = & a(Ix) & (\textit{ по } (I)) \\
 & = & BaIx. & (\textit{ по } (B))
\end{array}$$

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 "правило обращения монотонности".*

$LISP$--3. *Перепишем схему аксиом (15.3) в виде* $\overline{1} = Null\ Nil$ *(по правилу* (σ)), *где* $Nil = I$, $\overline{1}$ *- нумерал, комбинаторная характеристика которого имеет вид* $\overline{1}ab = ab$, *или в терминах* λ *-исчисления,* $\overline{1} = \lambda xy.xy$. *Заметим, что*

$$\begin{array}{ll}
(D): & Dabc = cab, \\
(\overline{0}): & \overline{0}ab = b, \textit{ или } \overline{0} \leftrightarrow \lambda xy.y.
\end{array}$$

[1] знак "$\leftrightarrow$" обозначает взаимно однозначное соответствие.

Следует учесть, что $KI \leftrightarrow \overline{0}$ *, то есть* $KI = \overline{0}$ *. Теперь найдем обьект, соответствующий функции* $Null$*:*

$$\begin{array}{rcll}
\overline{1} & = & \lambda xy.xy & (\textit{ по определению } \overline{1}) \\
 & = & \lambda x.x & (\textit{ вывод } \lambda xy.xy = \lambda x.x \textit{ несложен}) \\
 & = & I & (\textit{ по определению } I) \\
 & = & KI(K(K\overline{0})) & (\textit{ по схеме } (K)) \\
 & = & I(KI)(K(K\overline{0})) & (\textit{ по схеме } (I)) \\
 & = & D(KI)(K(K\overline{0}))I & (\textit{ по схеме } (D)) \\
 & = & D\overline{0}(K(K\overline{0}))I. & (\textit{ по схеме } (\overline{0}))
\end{array}$$

Сравнивая полученное выражение $D\overline{0}(K(K\overline{0}))I = \overline{1}$ *и схему* $Null\ Nil = I$ *(или, более строго, схему* $Null\ Nil = \overline{1}$*), получаем что* $D\overline{0}(K(K\overline{0}))I \leftrightarrow Null\ Nil$*.*

$LISP$*--4. Проведем следующие преобразования:*

$$\begin{array}{rcll}
\overline{0} & = & K\overline{0}(b\overline{0}) & (\textit{ по схеме } (K)) \\
 & = & K(K\overline{0})a(b\overline{0}) & (\textit{ по схеме } (K)) \\
 & = & Da(b\overline{0})(K(K\overline{0})) & (\textit{ по схеме } (D)) \\
 & = & B(Da)b\overline{0}(K(K\overline{0})) & (\textit{ по схеме } (B)) \\
 & = & D\overline{0}(K(K\overline{0}))(B(Da)b). & (\textit{ по схеме } (D))
\end{array}$$

Сравним полученное выражение $D\overline{0}(K(K\overline{0}))(B(Da)b) = \overline{0}$ *со схемой аксиом (15.4):* $Null(Append(List\ a)b) = \overline{0}$ *. Если учесть, что* $D\overline{0}(K(K\overline{0})) \leftrightarrow Null,\ B \leftrightarrow Append$*, то* $D \leftrightarrow List$*.*

$LISP$*--5. Аналогично, как и в предыдущем пункте, найдем объект, соответствующий функции* Car*:*

$$\begin{array}{rcll}
a & = & Ka(bc) & (\textit{ по схеме } (K)) \\
 & = & Da(bc)K & (\textit{ по схеме } (D)) \\
 & = & B(Da)bcK & (\textit{ по схеме } (B)) \\
 & = & DcK(B(Da)b). & (\textit{ по схеме } (D)).
\end{array}$$

Очевидно, что $DcK \leftrightarrow Car$*.*

$LISP$*--6. Тем же способом получим обьект соответствующий* Cdr*:*

$$\begin{array}{rcll}
bz & = & I(bz) & (\textit{ по схеме } (I)) \\
 & = & KIa(bz) & (\textit{ по схеме } (K)) \\
 & = & Da(bz)(KI) & (\textit{ по схеме } (D)) \\
 & = & B(Da)bz(KI) & (\textit{ по схеме } (B)) \\
 & = & (\lambda xy.xy(KI))(B(Da)b)z. & (\textit{ по постулатам } (\beta), (\sigma))
\end{array}$$

Поскольку $bz = (\lambda xy.xy(K\,I))(B(Da)b)z$, *то* $(\lambda xy.xy(K\,I))(B(Da)b) = b$ *для переменной* z *(применяется правило обращения монотонности), то есть* $\lambda xy.xy\overline{0} \leftrightarrow Cdr$.

Ответ. Итоги представления основных функций системы программирования $LISP$ приведем в виде таблицы соответствия:

No п/п	Функция $LISP$	Объект комбинаторной логики или λ -исчисления
1	$Append$	B
2	Nil	I
3	$Null$	$D\overline{0}(K(K\overline{0}))$
4	$List$	D
5	Car	DcK
6	Cdr	$\lambda xy.xy\overline{0}$

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, $D\overline{0}(K(K\overline{0}))$, D, DcK, $\lambda xy.xy\overline{0}$. 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.

Раздел 16

Суперкомбинаторы

Теоретические сведения. *Имеется два подхода к применению суперкомбинаторов для реализации аппликативных языков программирования. При первом их 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 Понятие о суперкомбинаторе

Определение 16.1 *Суперкомбинатор $\$S$ арности n представляет собой ламбда-выражение (абстракцию) вида:*

$$[x_1].[x_2]....[x_n].E,$$

где E не является абстракцией. Таким образом, все "ведущие" символы абстракции $[\cdot].$ относятся только к $x_1, x_2, \ldots, x_n$, 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 су-

перкомбинатора вместо свободных вхождений соответсвующих формальных параметров называется *редукцией* суперкомбинатора. Можно вспомнить обычное определение комбинатора и произвести сравнение. Другими словами, можно сказать, что комбинатор - это такая ламбда-абстракция, которая не содержит вхождений свободных переменных. Некоторые ламбда-выражения являются комбинаторами, а некоторые комбинаторы являются суперкомбинаторами.

Пример 16.1 *Выражения*

$$3,\ +\ 2\ 5,\ [x].x,\ [x].+\ x\ x,\ [x].[y].xy$$

представляют собой суперкомбинаторы.

Пример 16.2 *Приводимые термы не являются суперкомбинаторами:*

$[x].y$ *(переменная* y *входит свободно),*
$[y].-\ y\ x$ *(переменная* x *входит свободно).*

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:*

а) 3,
б) + 4 6,
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:* $1\ 0,\ \ [x].+\ x\ 1,\ \ [f].([x].+\ x\ x)$.

Упражнение 16.2 *Объясните, почему следующие выражения не являются суперкомбинаторами:* $[x].x\ y\ z,\ \ [x].[y].+\ (+\ x\ y)z$.

Упражнение 16.3 *Привести пример комбинатора, не являющегося суперкомбинатором.*

16.2 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:*

$$\frac{\$XYx\ y\ =\ -\ y\ x}{\$XY3\ 4}$$

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:*

$$\$XY\ 5,\ \$XY\ 5\ 7,\ \$XY\ 3\ 4\ 7\ ?$$

16.3 Приведение к суперкомбинаторам

Суперкомбинаторы легко компилируются. Опишем *алгоритм* 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:

$$\frac{\begin{array}{l}\$Y\ w\ y\ =\ +\ y\ w\\ \$X\ x\ =\ \$Y\ x\ x\end{array}}{\$X\ 4}$$

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, не содержащую 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,
ВСЕ-$while$
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) $([z].+\ z(([x].([y].\times\ y\ x)x)4))2$.

16.4 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:

$$\frac{\$Y\ x\ y\ =\ -\ y\ x}{[x].\$Y\ x}$$

(2) Пусть $\$X\ =\ [x].\$Y\ x$. Тогда имеем:

$$\frac{\begin{array}{l}\$Y\ x\ y = -\ y\ x\\ \$X\ x\ =\ \$Y\ x\end{array}}{\$X}$$

(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 $\$Y$:

$$\frac{\$Y\ x\ y\ =\ -\ y\ x}{\$Y}$$

В результате оказывается, что имеется две оптимизации:

1) устранение избыточных параметров из определений с помощью η-редукции,

2) удаление избыточных определений.

16.5 Упорядочивание параметров

Во всех рассмотренных выше программах порядок вынесения переменных как экстрапараметров был произвольным. Например, рассмотрим программу:

$$
\begin{array}{l}
\ldots\ldots\ldots\ldots\ldots\ldots \\
(\ldots \\
([x].[z].+\ y(\times x\ z)) \\
\ldots),
\end{array}
$$

где "$\ldots$" указывает на объемлющий $[x]$.-абстракцию контекст. Начнем выполнять алгоритм ламбда-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:

$$
\begin{array}{l}
\$S\ x\ y\ z\ =\ +\ y(\times x\ z) \\
\hline
(\ldots \\
([x].\$S\ x\ y) \\
\ldots)
\end{array}
$$

Выражение $[x].\$S\ x\ y$ не является суперкомбинатором, поэтому к нему, в свою очередь, следует применить алгоритм ламбда-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:

$$
\begin{array}{l}
\$T\ y\ x\ =\ \$S\ x\ y \\
\hline
\$T\ y.
\end{array}
$$

Вернемся к основному алгоритму, в котором теперь получаем:
$\$S\ x\ y\ z\ =\ +\ y(\times x\ z)$
$\$T\ y\ x\ =\ \$S\ x\ y$
$(\ldots \$T\ y \ldots)$.
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:

$$\frac{\$S\ y\ x\ z\ =\ +\ y(\times x\ z)}{\begin{array}{l}(\ldots\\ ([x].\$S\ y\ x)\\ \ldots).\end{array}}$$

Выражение $[x].\$S\ y\ x$ не является суперкомбинатором, поскольку содержит свободную переменную y. Применим к $[x].\$S\ y\ x$ алгоритм подъема.

(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:

$$\frac{\$T\ y\ x\ =\ \$S\ y\ x}{\$T\ y}$$

Вернемся к основному алгоритму. Получим:

$$\frac{\begin{array}{l}\$S\ x\ y\ z\ =\ +\ y(\times x\ z)\\ \$T\ y\ x\ =\ \$S\ y\ x\end{array}}{(\ldots \$T\ y \ldots)}$$

В соответствии с правилом устранения избыточных параметров (см. подраздел 16.4) получаем: $\$T = \$S$, поэтому $\$T$ можно устранить. Тогда скомпилированный код примет вид:

$$\frac{\$S\ y\ x\ z\ =\ +\ y(\times x\ z)}{\$S\ y}$$

В первом варианте такая оптимизация невозможна. В исходной программе $(\ldots([x].[z].+\ \ y(\times x\ z))\ldots)$ 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* $([x].[y].+\ x(\times y\ y))$ *оператор* $[x]$*.-абстракции находится на уровне 1, а* $[y]$*.-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.

16.6 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 при-

влечения 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:*

$$\left.\begin{array}{l} \$F \;=\; Y\ \$F1 \\ \$F1\ E\ x \;=\; \$G(F(-\ x\ 1))0. \end{array}\right\} \qquad (*)$$

Дополнительное определение помечено символом "". Поскольку* Y *необходимо редуцировать, то такое определение* $\$F$ *потребует больше редукций, чем рекурсивная версия.*

Обозначение:

$$\begin{array}{l} \$S1\ x\ y \;=\; B1 \\ \$S2\ f \;=\; B2 \\ \ldots \\ E \end{array}$$

эквивалентно выражению:

$$\begin{array}{l} letrec \\ \qquad \$S1 \;=\; [x].[y].B1 \\ \qquad \$S2 \;=\; [f].B2 \\ \ldots. \\ in \\ \qquad E. \end{array}$$

Оно означает, что в E входят $\$S1,\ \$S2,\ \ldots$, рекурсивные определения которых приведены в $letrec$. Алгоритм ламбда-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 ламбда-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:*

$$\frac{}{\begin{array}{l} letrec\ x\ =\ cons\ 1\ x \\ in\ x \end{array}}$$

В этой программе $letrec$ находится на уровне 0, и абстракций нет, поэтому x уже является суперкомбинатором:

$$\frac{\$x\ =\ cons\ 1\ x}{\$x}$$

Пример 16.10 *Рассмотрим рекурсивную функцию вычисления факториала:*

$$\frac{}{\begin{array}{l} letrec\ fac\ =\ [n].IF(=\ n\ 0)1\ (\times\ n\ (fac(-\ n\ 1))) \\ in\ fac\ 4 \end{array}}$$

В данном случае $letrec$ имеет номер 0 и внутри $[n]$.-абстракций нет. Следовательно, fac является суперкомбинатором:

$$\frac{\begin{array}{l} \$fac\ n\ =\ IF(=\ n\ 0)1(\times n(fac(-\ n\ 1))) \\ \$Prog\ =\$fac\ 4 \end{array}}{\$Prog}$$

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.

16.7 Работа алгоритма ламбда-подъема

Рассмотрим программу, суммирующую первые 100 целых чисел:

$$
\begin{array}{l}
SumInts\ m = sum(count\ 1) \\
\qquad\qquad\quad where \\
\qquad\qquad\quad count\ n\ =\ [],\ n > m \\
\qquad\qquad\qquad\qquad\quad =\ n : count(n+1) \\
\qquad sum\ []\ =\ 0 \\
\qquad sum\ (n : ns)\ =\ n\ +\ sum\ ns
\end{array}
$$

$SumInts\ 100$

$SumInts$ 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 $count\ 2$)
$\ldots$
$= 1 : 2 : \ldots : 100 : (count\ 101)$
$= 1 : 2 : \ldots : 100 : []$ (поскольку $n = 101, m = 100,$ то $n > m,$ и $(count\ 101 = [])$)

Теперь список $1 : 2 : 3 : \ldots : 100 : []$ передается функции sum. Во втором определяющем равенстве для sum аргумент $(n : ns)$ обо-

значает список, в котором первым элементом является число n, а "хвостом" - список чисел ns :

$$sum\ 1:2:3:\ldots:100:[] \;=\; 1 \;+\; sum\ 2:3:\ldots:100:[]$$
$$\ldots$$
$$= 1+2+3+\ldots+100+sum[]$$
$$= 1+2+3+\ldots+100+0 \text{ (так как } sum\ [] = 0)$$

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 ламбда-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 $\$count\ count\ m\ n$:

$$\$count\ count\ m\ n\ =\ IF(>\ n\ m)\ NIL\ (cons\ n\ (count\ (+\ n\ 1)))$$

$$letrec\ SumInts = [m].letrec\ count\ =\ \$count\ count\ m\ in\ sum\ (count\ 1)\ sum\ =\ [ns].IF(=\ ns\ NIL)0\ (+(head\ ns)(sum(tail\ ns)))\ in\ SumInts\ 100$$

(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:*

$$
\begin{array}{l}
apply\ m\ =\ fold\ КВАДРАТ\ (constr\ 1) \\
\qquad where\ constr\ n\ =\ [],\ n > m \\
\qquad\qquad\qquad\qquad = n : constr(n+1) \\
fold\ f[]\ =\ [] \\
fold\ f(n:ns)\ =\ fn : fold\ f\ ns
\end{array}
$$

16.8 Другие способы ламбда-подъема

Представленная в предыдущих подразделах методика не является единственным способом ламбда-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:

$$
\begin{array}{l}
(\ldots \\
\qquad letrec\ f\ =\ [x].(\ldots f \ldots v \\
\ldots) \\
\qquad in\ (\ldots f \ldots) \\
\ldots)
\end{array}
$$

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:

$$
\begin{array}{l}
\$f\ v\ x\ =\ \ldots(\$f\ v)\ldots v\ldots \\
(\ldots \\
\qquad (\ldots(\$f\ v)\ldots) \\
\ldots)
\end{array}
$$

Посмотрим, как работает данный алгоритм на примере из подраздела 16.6. 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 ламбда-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 16.6 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.*

16.9 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:*

$$
\begin{array}{l}
f\ =\ g\ 4 \\
g\ x\ y\ =\ y\ +\ (sqrt\ x) \\
(f\ 1)\ +\ (f\ 2)
\end{array}
$$

Запись в виде ламбда-выражения дает:

$$
\begin{array}{l}
letrec\ f\ =\ g\ 4 \\
\qquad g\ =\ [x].[y].+\ y\ (sqrt\ x) \\
\qquad in\ +\ (f\ 1)(f\ 2)
\end{array}
$$

При вычислении этого выражения получаем следующий результат:
$+\ (f\ 1)(f\ 2) \rightarrow +\ (.\ 1)(.\ 2)$

$$\begin{array}{l}
\qquad \longrightarrow (([x].[y].+\ y\ (sqrt\ x))4) \\
\rightarrow +\ (.\ 1)(.\ 2) \\
\qquad \longrightarrow ([y].+\ y\ (sqrt\ 4)) \\
\rightarrow +\ (.\ 1)(+\ 2\ (sqrt\ 4)) \\
\qquad \longrightarrow ([y].+\ y\ (sqrt\ 4)) \\
\rightarrow +\ (.\ 1)4 \\
\qquad \longrightarrow ([y].+\ y\ (sqrt\ 4)) \\
\rightarrow +\ (+\ 1\ (sqrt\ 4))4 \\
\rightarrow +\ (+\ 1\ 2)4 \\
\rightarrow +\ 3\ 4 \\
\rightarrow 7
\end{array}$$

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:

$$\begin{array}{l}
\$g\ x\ y\ =\ +\ y\ (sqrt\ x) \\
\$f\ =\ \$g\ 4 \\
\$Prog\ =\ +\ (\$f\ 1)(\$f\ 2) \\
\hline
\$Prog
\end{array}$$

Редукция выглядит следующим образом:

$$\begin{array}{l}
\$Prog \rightarrow +\ (.\ 1)(.\ 2) \\
\qquad \longrightarrow (\$g\ 4) \\
\rightarrow +\ (.\ 1)(+\ 2(sqrt\ 4)) \\
\qquad \longrightarrow (\$g\ 4) \\
\rightarrow +\ (.\ 1)4 \\
\qquad \longrightarrow (\$g\ 4) \\
\rightarrow +\ (+\ 1(sqrt\ 4))4 \\
\rightarrow +\ (+\ 1\ 2)4 \\
\rightarrow +\ 3\ 4 \\
\rightarrow 7
\end{array}$$

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:*
$f \;=\; g\,2$
$g\;x\;y \;=\; y \times (\textit{КВАДРАТ}\;x)$
$(f\;3) \times (f\;1)$
а) Записать программу в виде абстракции и произвести вычисления.
б) Скомпилировать программу и произвести вычисления.

16.10 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 L, если все переменные E свободны в L.*

Определение 16.4 . *Максимально свободное выражение (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) $([x].\underline{sqrt\;x})$,
(2) $([x].x\underline{(sqrt\;4)})$
(3) $([y].[x].\underline{+\;(\times y\;y)}x)$.

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:
$+\ (f\ 1)(f\ 2) \rightarrow +\ (.\ 1)(.\ 2)$
$\longrightarrow (([x].[y].+\ y(sqrt\ x))4)$
$\rightarrow +\ (.\ 1)(.\ 2)$
$\longrightarrow ([y].+\ y\ (sqrt\ 4))$
(здесь: выражение $(sqrt\ 4)$ *является МСВ в* $[y]$*.-абстракции, поэтому при приложении* $[y].$ *к аргументу* $(sqrt\ 4)$ *не следует вычислять*
$\rightarrow +\ (.\ 1)(+\ 2\ .)$
$\longrightarrow ([y].+\ y\ (sqrt\ 4))\ )$
Означивание имеет указатель на $(sqrt\ 4)$ *в теле абстракции:*

$\rightarrow +\ (.\ 1)(+\ 2\ .)$
$\longrightarrow ([y].+\ y\ 2)$
$\rightarrow +\ (.\ 1)4$
$\longrightarrow ([y].+\ y\ 2)$
$\rightarrow +\ (+\ 1\ 2)4$
$\rightarrow +\ 3\ 4$
$\rightarrow 7$
В этом случае $(sqrt\ 4)$ *вычисляется только один раз.*

Упражнение 16.10 *Вычислить выражение из упражнения 16.9, воспользовавшись МСВ.*

16.11 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:

$$[x].[y].+\ y(sqrt\ x).$$

Проиллюстрируем в этой ситуации работу алгоритма.

(1) Самой внутренней абстракцией является $[y].+\ y(sqrt\ x)$.

(2) Выражение $(sqrt\ x)$ является МСВ. Вынесем МСВ в качестве экстрапараметра: $([sqrtx].[y].+\ y(sqrtx))sqrtx$, где $sqrtx$ является именем экстрапараметра. Подставим полученное выражение в исходную абстракцию: $[x].([sqrtx].[y].+\ y\ sqrtx)sqrtx$.

(3) Присвоим полученному суперкомбинатору имя:

$$\frac{\$g1\ =\ [sqrtx].[y].+\ y\ sqrtx}{[x].\$g1\ (sqrt\ x)}$$

(4) Имеющаяся $[x]$.-абстракция также является суперкомбинатором, которому дадим имя и который сопоставим скомпилированному коду:

$$\frac{\begin{array}{l}\$g1\ sqrtx\ y\ =\ +\ y\ sqrtx\\ \$g\ x\ =\ \$g1\ (sqrt\ x)\\ \$f\ =\ \$g\ 4\\ \$Prog\ =\ +\ (\$f\ 1)(\$f\ 2)\end{array}}{\$Prog}$$

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 экстрапараметров. Приведенный алгоритм считается полностью ленивым ламбда-подъемом.

Упражнение 16.11 *Выполнить полностью ленивый ламбда-подъем программы, рассмотренной в упражнении 16.9.*

16.12 Полностью ленивый ламбда-подъем с $letrec$

Рассмотрим программу:

$$
\begin{array}{l}
let\ f\ =\ [x].letrec\ fac\ =\ [n].(\ldots) \\
\qquad in\ +\ x\ (fac\ 100) \\
in \\
\qquad +\ (f\ 3)(f\ 4)
\end{array}
$$

Алгоритм из подраздела 16.6 скомпилирует ее в:

$$
\begin{array}{l}
\$fac\ fac\ n\ =\ (\ldots) \\
\$f\ x\ =\ letrec\ fac\ =\ \$fac\ fac \\
\qquad in\ +\ x\ (fac\ 100) \\
+\ (\$f\ 3)(\$f\ 4)
\end{array}
$$

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:

$$
\begin{array}{l}
letrec \\
\qquad fac\ =\ [n].(\ldots) \\
\qquad in\ let \\
\qquad f\ =\ [x].+\ x\ (fac\ 100) \\
in \\
\qquad +\ (f\ 3)(f\ 4)
\end{array}
$$

Теперь ничто не препятствует применению полностью ленивого подъема, который построит полностью ленивую программу:

$$\$fac\, n \;=\; (\ldots)$$
$$\$fac100 \;=\; \$fac\, 100$$
$$\$f\; x \;=\; +\, x\, \$fac100$$
$$\$Prog \;=\; +\, (\$f\; 3)(\$f\; 4)$$

$$\$Prog$$

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 ламбда-подъема.

Упражнение 16.12 *Выполнить полностью ленивый ламбда-подъем программы:*

$$\begin{array}{l} let \\ \quad g \;=\; [x].letrec\; el \;=\; [n].[s].(IF(=\; n\; 1)(head\; s) \\ \qquad\qquad\qquad\qquad (el(-\; n\; 1)(tail\; s))) \\ \qquad\qquad\quad in\; (cons\; x(el\; 3\; (A, B, C))) \\ in \\ \quad (cons(g\; R)(g\; L)) \end{array}$$

(*здесь:* R *и* $L -$ *константы*).

16.13 Комплексный пример

Покажем действие полностью ленивого ламбда-подъема на более детализированном примере[1]:

$$SumInts\; n \;=\; foldl\; (+)\; 0\; (count\; 1\; n)$$
$$count\; n\; m \;=\; [],\;\; n > m$$
$$count\; n\; m \;=\; n : count(n \;+\; 1)\; m$$
$$fold\; op\; base\; [] \;=\; base$$
$$foldl\; op\; base\; (x : xs) \;=\; foldl\; op\; (op\; base\; x)\; xs$$

[1] Дополнительную разработку техники ленивых вычислений с суперкомбинаторами см. в [86]. В этой же работе можно познакомиться с реализацией абстрактной машины, обеспечивающей поддержку всего спектра возможностей.

Приведенная программа в терминах абстракций записывается следующим образом:

$$
\begin{array}{l}
letrec \\
\quad SumInts\ =\ [n].foldl\ +\ 0\ (count\ 1\ n) \\
\quad count\ =\ [n].[m].IF(>\ n\ m)NIL \\
\qquad\qquad (cons\ n\ (count(+\ n\ 1)m)) \\
\quad foldl\ =\ [op].[base].[xs].IF(=\ xs\ NIL)base \\
\qquad (foldl\ op\ (op\ base\ (head\ xs))(tail\ xs)) \\
in\ SumInts\ 100
\end{array}
$$

В этой программе:

(1) внутренней абстракцией является $([xs]\ldots)$,

(2) максимально свободными выражениями являются $(fold\ op)$, $(op\ base)$ и $base$.

Вынесем их в качестве экстрапараметров p, q и $base$ соответственно:

$$
\begin{array}{l}
\$R1\ p\ q\ base\ xs\ =\ IF\ (=\ xs\ NIL)\ base \\
\qquad\qquad\qquad\qquad (p(q(head\ xs))(tail\ xs))
\end{array}
$$

$$
\begin{array}{l}
letrec \\
\quad SumInts\ =\ [n].foldl\ +\ 0\ (count\ 1\ 0) \\
\quad count\ =\ [n].[m].IF(>\ n\ m)NIL \\
\qquad (cons\ n\ (count(+\ n\ 1)m)) \\
\quad foldl = [op].[base].\$R1(foldl\ op) \\
\qquad\qquad (op\ base)base \\
in \\
\quad SumInts\ 100
\end{array}
$$

В этой программе внутренней абстракцией является “$[base].$”. Максимально свободными выражениями служат $(\$R1(foldl\ op))$ и op, которые вынесем как r и op соответственно:

$$
\begin{array}{l}
\$R1\ p\ q\ base\ xs\ =\ IF(=\ xs\ NIL)base \\
\qquad (p(q(head\ xs))(tail\ xs)) \\
\$R2\ r\ op\ base\ =\ r\ (op\ base)base
\end{array}
$$

$$
\begin{array}{l}
letrec \\
\qquad SumInts\ =\ [n].foldl\ +\ 0(count\ 1\ n) \\
\qquad count\ =\ [n].[m].IF(>\ n\ m)NIL(cons\ n \\
\qquad\qquad\qquad (count(+\ n\ 1)m)) \\
\qquad foldl\ =\ [op].\$R2(\$R1\ op)op \\
in \\
\qquad SumInts\ 100
\end{array}
$$

(3) все определения $letrec$ являются суперкомбинаторами, поскольку выполнен подъем всех внутренних абстракций. С учетом всех замечаний получаем окончательный результат:

$$
\begin{array}{l}
\$SumInts\ n\ =\ \$foldlPlus0\ (\$count1\ n) \\
\$foldlPlus0\ =\ \$foldl\ +\ 0 \\
\$count1\ =\ \$count\ 1 \\
\$count\ n\ m\ =\ IF(>\ n\ m)NIL(cons\ n(\$count(+\ n\ 1)m)) \\
\$foldl\ op\ =\ \$R2(\$R1(\$foldl\ op))op \\
\$Prog\ =\ \$SumInts\ 100 \\
\$R1\ p\ q\ base\ xs\ = \\
\qquad IF(=\ xs\ NIL)base(p(q(head\ xs))(tail\ xs)) \\
\$R2\ r\ op\ base\ =\ r\ (op\ base)base
\end{array}
$$

$$\$Prog$$

Завершая рассмотрение, отметим, сто в данном случае нельзя устранить параметр op из $\$foldl$, поскольку он используется дважды в правой части.

16.14 Задача

Задача 16.1 *Записать следующее определение исходного языка с помощью суперкомбинаторов:*

$$el\ n\ s = if\ n = 1\ then\ hd\ s$$

$$else\ el(n-1)(tl\ s).$$

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 $\lambda V.E$.

SK--1. Тело выражения преобразуется в аппликативную форму рекурсивным вызовом компилятора. Результатом является: $\lambda V.E'$.

SK--2. Свободные переменные в λ-выражении идентифицируются литерами $P,\ Q,\ \ldots,\ R$, затем λ-выражение снабжается префиксом λ , связывающим все эти переменные. Результатом служит:

$$\lambda P.\lambda Q....\lambda R.\lambda V.E'.$$

Результирующее выражение заведомо является комбинатором, поскольку E' - аппликативная форма, все свободные переменные $P,\ Q,\ \ldots,\ R$ которой связаны.

SK--3. Назовем возникающий комбинатор $alpha$ и сопоставим ему определение (определяющее равенство) :

$$alpha\ PQ\ldots\ RV \to E'.$$

Исходное λ-выражение заменяется формой $(alpha\ PR\ \ldots\ R)$. 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:*

$$\lambda s.if(=\ n\ 1)(hd\ s)(el(-\ n\ 1)(tl\ s)).$$

его свободными переменными являются n и el, поэтому комбинатор $alpha$ вводится определением:

$$alpha\, n\, el\, s \to if(=\ n\, 1)(hd\, s)(el(-\, n\, 1)(tl\, s)). \quad (alpha)$$

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:*
$$beta\ el\ n \to alpha\ n\ el, \quad (beta)$$
откуда получаем, что
$$el = Y(\lambda el.beta\ el).$$

4. *Повторим шаги (1.) и (2.) для $(\lambda el.beta\ el)$. Введем комбинатор $gamma$ определением:*
$$gamma\ el \to beta\ el, \quad (gamma)$$
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?

Упражнение. Записать с помощью суперкомбинаторов выражение:

$$fac\, n\ =\ if\ n = 0\ then\ 1\ else\ n\ \times\ (fac(n-1)).$$

16.15 Ответы к упражнениям

16.1

$1\ 0$ *не имеет свободных переменных (1) и символов абстракции (3);* $[x].+\ x\ 1$ *имеет переменную* x*, которая является связанной (1);* $+\ x\ 1$ *не содержит абстракций (2);* $[f].f([x].+\ x\ x)$ *не имеет свободных переменных (1),* $[x].+\ x\ x$ *является суперкомбинатором (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 $\$Y$*:*

$$\frac{\$Y\ v\ y\ =\ -\ y\ v}{([x].\$Y\ x\ x)5}$$

(4) $[x]$*.-абстракция также является суперкомбинатором, которому можно дать имя и который можно сопоставить скомпилированному коду:*

$$\frac{\begin{array}{l}\$Y\ v\ y\ =\ -\ y\ v\\ \$X\ x\ =\ \$Y\ x\ x\end{array}}{\$X\ 5}$$

Полученная программа выполняется следующим образом:

$$\$X\ 5\ =\ \$Y\ 5\ 5\ =\ -\ 5\ 5\ =\ 0.$$

2) $([z].+\ z\ (([x].([y].\times\ y\ x)x)4)2$ *:*
(1) внутренняя абстракция: $[y].\times\ y\ x$;
(2) вынесем x *как экстрапараметр:*
$([x].[y].\times\ y\ x)x$
$([z].+\ z(([x].([w].[y].\times\ y\ w)x\ x)4))2$*;*
(3)

$$\frac{\$Y\ w\ y\ =\ \times\ y\ w}{([z].+\ z([x].\$Y\ x\ x)4)2}$$

(4) $[x]$*.-абстракция является суперкомбинатором:*

$$\frac{\begin{array}{l}\$Y\ w\ y\ =\ \times\ y\ w\\ \$X\ x\ =\ \$Y\ x\ x\end{array}}{([z].+\ z(\$X\ 4))2}$$

Теперь видно, что $[z]$*.-абстракция является суперкомбинатором:*

$$\frac{\begin{array}{l}\$Y\ w\ y\ =\ \times\ y\ w\\ \$X\ x\ =\ \$Y\ x\ x\\ \$Z\ z\ =\ +\ z\ (\$X\ 4)\end{array}}{\$Z\ 2}$$

Выполнение программы:
$\$Z\ 2\ =\ +\ 2\ (\$X\ 4)\ =\ +\ 2\ (\$Y\ 4\ 4)\ =\ +\ 2\ (\times\ 4\ 4)\ =\ +\ 2\ 16\ =\ 18.$

16.6
inf *находится на уровне и не содержит внутренних абстракций. Поэтому* inf *уже является суперкомбинатором:*

$$\frac{\begin{array}{l}\$inf\ v\ =\ letrec\ vs\ =\ cons\ 0\ vs\ in\ vs\\ \$Prog\ =\ \$inf\ 4\end{array}}{\$Prog}$$

16.7
Запишем эту программу в терминах абстракций:

$$
\begin{array}{l}
letrec\ apply\ =\ [m].letrec\ constr\ =\ [n].(IF\ >\ n\ m)NIL \\
\qquad\qquad\qquad\qquad (cons\ n\ (constr\ (+\ n\ 1))) \\
\qquad\qquad\qquad\qquad in\ fold\ \textit{КВАДРАТ}\ (constr\ 1) \\
\qquad\quad fold\ =\ [f].[ns].IF(=\ ns\ NIL)NIL(cons(f(head\ ns)) \\
\qquad\qquad\qquad\qquad\qquad\qquad (fold(f(tail\ ns)))) \\
in\ apply\ 5
\end{array}
$$

(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* $(\$constr\ constr\ m\ n);$

$$\$constr\ constr\ m\ n\ =\ IF(>\ n\ m)NIL(cons\ n(constr(+\ n\ 1)))$$

$$
\begin{array}{l}
letrec \\
\qquad apply\ =\ [m].letrec \\
\qquad\qquad\qquad constr\ =\ \$constr\ constr\ m \\
\qquad\qquad in\ fold\ \textit{КВАДРАТ}\ (constr\ 1) \\
\qquad fold\ =\ [f].[ns].IF(=\ ns\ NIL)NIL \\
\qquad\qquad\qquad (cons(f(head\ ns))(fold\ f(tail\ ns))) \\
in\ apply\ 5
\end{array}
$$

(5) выражения $apply$ *и* $fold$ *являются суперкомбинаторами:*

$$
\begin{array}{l}
\$constr\ constr\ m\ n\ =\ IF(>\ n\ m)NIL(cons\ m(constr(+\ n\ 1))) \\
\$fold\ f\ ns\ =\ IF(=\ ns\ NIL)NIL(cons(f(head\ ns)) \\
\qquad\qquad\qquad\qquad (fold(f\ (tail\ ns)))) \\
\$apply\ m\ =\ letrec\ constr\ =\ \$constr\ constr\ m \\
\qquad\qquad in\ \$fold\ \textit{КВАДРАТ}\ (constr\ 1) \\
\$Prog\ =\ \$apply\ 5
\end{array}
$$

$$\$Prog$$

16.8

Поднимем $[n]$*.-абстракцию, абстрагируя переменную* m*, но не* $constr$*, и заменим все вхождения* $constr$ *на* $(\$constr\ m)$*:*

$$\$constr\ m\ n\ =\ IF(>\ n\ m)NIL(cons\ n(\$constr\ m(+\ n\ 1)))$$

$$\begin{array}{l} letrec \\ \quad apply\ =\ [m].fold(КВАДРАТ)(\$constr\ m\ 1) \\ \quad fold\ =\ [f].[ns].IF(=\ ns\ NIL)NIL(cons(f(head\ ns)) \\ \qquad\qquad\qquad\qquad (fold(f(tail\ ns)))) \\ in\ apply\ 5 \end{array}$$

В данном случае и apply, и fold являются суперкомбинаторами:

$$\begin{array}{l} \$constr\ m\ n\ =\ IF(>\ n\ m)NIL(cons\ n(\$constr\ m(+\ n\ 1))) \\ \$fold\ f\ ns\ =\ IF(=\ ns\ NIL)NIL(cons(f(head\ ns)) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\$fold\ f(tail\ ns))) \\ \$apply\ m\ =\ \$fold\ КВАДРАТ\ (\$constr\ m\ 1) \\ \$Prog\ =\ \$apply\ 5 \end{array}$$

$$\$Prog$$

16.9

а) Запись в виде абстракции:

$$\begin{array}{l} letrec\ f\ =\ g\ 2 \\ \qquad g\ =\ [x].[y].\times\ y\ (КВАДРАТ\ x) \\ in\ \times\ (f\ 3)(f\ 1) \end{array}$$

Вычисление выражения:

$\times\ (f\ 3)(f\ 1) \rightarrow (.\ 3)(.\ 1)$
$\quad \longrightarrow ([x].[y].\times\ y\ (КВАДРАТ\ x))2$
$\rightarrow \times\ (.\ 3)(.\ 1)$
$\quad \longrightarrow ([y].\times\ y\ (КВАДРАТ\ 2))$
$\rightarrow \times\ (.\ 3)(\times\ 1\ (КВАДРАТ\ 2))$
$\quad \longrightarrow ([y].\times\ y\ (КВАДРАТ\ 2))$
$\rightarrow \times\ (.\ 3)\ 4$
$\quad \longrightarrow ([y].\times\ y\ (КВАДРАТ\ 2))$
$\rightarrow \times\ (\times\ 3\ (КВАДРАТ\ 2))\ 4$
$\rightarrow \times\ 12\ 4$
$\rightarrow 48.$

б) Данное выражение компилируется в:

$$\begin{array}{l} \$g\ x\ y\ =\ \times\ y\ (КВАДРАТ\ x) \\ \$f\ =\ \$g\ 2 \\ \$Prog\ =\ \times\ (\$f\ 3)(\$f\ 1) \end{array}$$

$$\$Prog$$

Последовательность редукций:

$\$Prog \to (\times\ (.\ 3)(.\ 1))$
$\quad \longrightarrow (\$g\ 2)$
$\to \times\ (.\ 3)(\times\ 1\ (КВАДРАТ\ 2))$
$\quad \longrightarrow (\$g\ 2)$
$\to \times\ (.\ 3)\ 4$
$\quad \longrightarrow (\$g\ 2)$
$\to \times\ (\times\ 3\ (КВАДРАТ\ 2))\ 4$
$\to \times\ (\times\ 3\ 4)\ 4$
$\to \times\ 12\ 4$
$\to 48.$

16.10

$\times\ (f\ 3)(f\ 1) \to \times\ (.\ 3)(.\ 1)$
$\quad \longrightarrow (([x].[y].\times\ y\ (КВАДРАТ\ x))2)$
$\to \times\ (.\ 3)(\times\ 1\ .)$
$\quad \longrightarrow ([y].\times\ y\ (КВАДРАТ\ 2))$
$\to \times\ (.\ 3)\ 4\ (\times\ 1\ .)$
$\quad \longrightarrow ([y].\times\ y\ 4)$
$\to \times\ (.\ 3)\ 4$
$\quad \longrightarrow ([y].\times\ y\ 4)$
$\to \times\ (\times\ 3\ .)\ 4$
$\quad \longrightarrow 4$
$\to \times\ 12\ 4$
$\to 48$

Выражение $(КВАДРАТ\ 2)$ *вычисляется единственный раз.*

16.11

Функция g содержит абстракцию: $[x].[y].\times\ y\ (КВАДРАТ\ x)$ *:*

(1) самой внутренней абстракцией является

$$[y].\times\ y\ (КВАДРАТ\ x);$$

(2) $(КВАДРАТ\ x)$ *представляет собой МСВ, которую вынесем в качестве экстрапараметра:*

$[КВАДРАТx].[y].\times\ y\ (КВАДРАТx)КВАДРАТx.$

Подставим полученное выражение в абстракцию:

$[x].[КВАДРАТx].[y].\times\ y\ (КВАДРАТx)КВАДРАТx$ *;*

(3) присвоим полученному суперкомбинатору имя:

$$\frac{\$g1\ = [КВАДРАТx].[y].\times\ y\ КВАДРАТx}{[x].\$g1\ (КВАДРАТ\ x)}$$

(4) $[x]$*.-абстракция также является суперкомбинатором, которому дадим имя* $\$g$ *и который сопоставим скомпилированному коду:*

$$\$g1\ КВАДРАТx\ y\ =\ \times\ y\ КВАДРАТx$$
$$\$g\ x\ =\ \$g1\ (КВАДРАТ\ x)$$
$$\$f\ =\ \$g\ 2$$
$$\$Prog\ =\ \times\ (\$f\ 3)(\$f\ 1)$$

$$\$Prog$$

16.12

Данная программа компилируется в:

$$\$el\ el\ n\ s\ =\ (IF(=\ n\ 1)(head\ s)(el(-\ n\ 1)(tail\ s)))$$
$$\$g\ x\ =\ letrec\ el\ =\ \$el\ el$$
$$in\ (cons\ x\ (el\ 3\ (A,\ B,\ C)))$$

$$(cons\ (\$g\ R)(\$g\ L))$$

Поскольку определение el *не зависит от* x*, можно вынести* $letrec$ *для* el*:*

$$letrec\ el\ =\ [n].[s].(IF(=\ n\ 1)(head\ s)$$
$$(el(-n1)(tails)))$$
$$in\ let\ g\ =\ [x].cons\ x\ (el\ 3\ (A,\ B,\ C))$$
$$in\ (cons\ (g\ R)(g\ L))$$

В результате применения ламбда-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.

Раздел 17

Ленивая реализация

Задача 17.1 *Для примера определения*

$$el\ n\ s\ =\ if\ n = 1\ then\ (hd\ s)\ else\ el(n-1)(tl\ s)\ fi$$

выбрать суперкомбинаторы, дающие полностью ленивую реализацию, и рассмотреть частный случай применения el 2.

Решение. 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).

lazy--1. Рассмотрим схему трансляции. Минимальные свободные выражения каждого λ-выражения преобразуются в параметры соответствующего комбинатора. Остановимся на схеме преобразования максимальных свободных выражений в параметры соответствующего комбинатора.

lazy--2. 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. Вернемся к исходной задаче.

$lazy$--3. Пусть максимальные свободные выражения для исходного выражения

$$\lambda s.if(=\ n\ 1)(hd\ s)(el(-\ n\ 1)(tl\ s))$$

представляют собой

$$(if(=\ n\ 1))\ \ u\ (el(-\ n\ 1)).$$

Следовательно, новый комбинатор $alpha$ определяется посредством

$$alpha\ p\ q\ s \to p(hd\ s)(q(tl\ s)),$$

а определением el служит выражение

$$el = Y(\lambda el.\lambda n.alpha(if(=\ n\ 1))(el(-\ n\ 1))).$$

Продолжая этот процесс, получаем комбинаторы $beta$ и $gamma$, задаваемые определениями:

$$beta\ el\ n \to alpha(if(=\ n\ 1))(el(-\ n\ 1)),$$
$$gamma\ el \to beta\ el,$$

откуда заключаем, что выражение el эквивалентно $(Y\ gamma)$, то есть $el = Y\ gamma$, что совпадает с прежним результатом.

lazy--4. Рассмотрим частный случай, когда применяется $(el\ 2)$*:*

$$\begin{aligned} el\ 2 &\to (Y\ gamma)2 \\ &\to gamma\ el\ 2 \\ &\to beta\ el\ 2 \\ &\to alpha(if(=\ 2\ 1))(el(-\ 2\ 1)). \end{aligned}$$

Всегда при использовании $(el\ 2)$ *употребляются одни и те же копии свободных выражений, следовательно, они означиваются только один раз. Фактически получается редукция:*

$$el\ 2 \to alpha\ if - FALSE(alpha\ if - TRUE(el(-\ 1\ 1))).$$

Более подробно:

$$\begin{aligned} el\ 2 &\to alpha(if(=\ 2\ 1))(el(-\ 2\ 1)) \\ &\to alpha(if - FALSE)(el\ 1) \\ &\to alpha(if - FALSE)(alpha(if(=\ 1\ 1))(el(-\ 1\ 1))) \\ &\to alpha(if - FALSE)(alpha(if - TRUE)(el(-\ 1\ 1))) \end{aligned}$$

Рассмотренная схема приводит к субоптимальным комбинаторам.
Контрольные 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?

Упражнение. Для выражения

$$fac\ n\ =\ if\ n = 0\ then\ 1\ else\ n \times (fac(n-1))$$

осуществить полностью ленивую реализацию с помощью суперкомбинаторов и вычислить $fac\ 3$.

Раздел 18

Перестановка параметров

Задача 18.1 *Для исходного определения:*

$$el \;=\; Y(\lambda el.\lambda n\lambda s.IF(=n1)(hd\ s)(el(-\ n\ 1)(tl\ s)))$$

получить выражение с оптимальным (по двум критериям) упорядочением параметров комбинатора.

Решение. 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.

opt--1. *Для максимизации размера и минимизации числа МСВ следующего вложенного λ -выражения все МСВ подвергаемого 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 предшествет E_j. Отсюда не следует, что с необходимостью однозначно определяется оптимальное упорядочение, поскольку выражения с одним и тем же порждающим λ -выражением могут входить в любом порядке. Тем не менее всякое упорядочение, удовлетворяющее этому условию, является оптимальным, как и всякое другое.

opt--2. Рассмотрим аппликативную форму $(alpha(hd\ s)n(tl\ s))$, в которой параметры $alpha$ 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

$$alpha\ beta\ el\ hd\ tl.$$

Компилятор должен 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:

$$beta\ p\ q\ r\ s \to alpha\ \ldots\ s\ \ldots\ .$$

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 $(alpha\ E_1\ \ldots\ E_n\ s)$, 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 $(alpha\ E_1\ \ldots\ E_n\ s)$. Следовательно, фактически, если имеются какие-либо E_i, то $beta$ надо определить посредством редукции

$$beta\ p\ s \to p\ s,$$

где p соответствует $(alpha\ E_1\ \ldots\ E_n\ s)$. Если у $alpha$ единственный параметр s, то $beta$ следует определять посредством $beta\ s \to s$.
В первом случае $beta$ эквивалентен комбинатору I. Порождающее $beta$ λ -выражение замещается на аппликацию

$$(beta(alpha\ E_1\ \ldots\ E_n\ s)),$$

то есть на $(I(alpha\ E_1\ \ldots\ E_n\ s))$. Кроме того, $beta$ можно целиком опустить, и λ -выражение замещается непосредственно на $(alpha\ E_1\ \ldots\ E_n\ s)$.
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:

$$(IF(=\ n\ 1))\ \textit{и}\ (el(-\ n\ 1)).$$

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:

$$alpha\ p\ q\ s \to p(hd\ s)(q(tl\ s)),$$

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

$$beta\ el\ n \to alpha\ (IF(=\ n\ 1))(el(-\ n\ 1)),$$

по которой

$$el\ =\ Y(\lambda el.beta\ el).$$

Далее следует применять редукцию

$$gamma\ el \to beta\ el,$$

и поскольку комбинатор $gamma$ эквивалентен комбинатору $beta$, то $gamma$ порождать не следует.

Окончательно получаем, что $gamma$ эквивалентен $(Y\ beta)$.
Контрольные вопросы.

1. Какие преимущества дает использование правил оптимизации?
2. Назовите два основных критерия оптимизации.
3. На что влияет порядок параметров суперкомбинаторов?

Упражнение. Получить выражение с оптимальным порядком суперкомбинаторов для определения:

$$fac\ n\ =\ IF\ n = 0\ then\ 1\ else\ n \times (fac(n-1)).$$

Раздел 19

Непосредственные вычисления

Задача 19.1 *Для выражения:*

$$let\ x\ =\ plus\ in\ x\ (4, (x\ where\ x = 3));;$$

построить λ-выражение и выражение комбинаторной логики, а также вычислить их значение.

Решение.

dc--1. *Фиксируем выражение:*

$$let\ x\ =\ plus\ in\ x\ (4, (x\ where\ x = 3));;$$

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:

$$\begin{array}{lll} (i) & \lambda x.x & = I, \\ (ii) & \lambda x.c & = Kc,\ c \neq x, \\ (iii) & \lambda x.PQ & = S(\lambda x.P)(\lambda x.Q). \end{array}$$

Таким образом,

$$\begin{array}{rcl} N & = & (\lambda x.x\ 4((\lambda x.x)3))\oplus \\ & = & S(\lambda x.x\ 4)(\lambda x.((\lambda x.x)3))\oplus \\ & = & S(S(\lambda x.x)(\lambda x.4))(S(\lambda x.(\lambda x.x))(\lambda x.3))\oplus \\ & = & S(SI(K\ 4))(S(\lambda x.I)(K3))\oplus \\ & = & S(SI(K\ 4))(S(K\ I)(K\ 3))\oplus\ . \end{array}$$

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":

$$\begin{array}{l} S(SI(K4))(S(KI)(K3))\oplus \rightarrow \\ \quad \rightarrow (SI(K\ 4)\oplus)(S(K\ I)(K\ 3)\oplus) \\ \quad \rightarrow (I\oplus)(K\ 4\oplus)(S(K\ I)(K\ 3)\oplus) \rightarrow \oplus(K\ 4\oplus)(S(K\ I)(K\ 3)\oplus) \\ \quad \rightarrow \oplus 4(S(K\ I)(K\ 3)\oplus) \rightarrow \oplus 4(K\ I\oplus)(K3\oplus) \\ \quad \rightarrow \oplus 4(I(K\ 3\oplus)) \rightarrow \oplus 4\ 3 \rightarrow 7. \end{array}$$

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 "$\oplus$").
Контрольные вопросы.

1. Выпишите комбинаторные характеристики K, I, S, B, W, C (стандартных комбинаторов).

2. Почему для реализации β-редукции показался удобным переход к комбинаторной логике?

Упражнения.

1. Выразите через комбинаторы K и S объект I с комбинаторной характеристикой $Ia = a$.
   Указание. Воспользуйтесь тем, что $I = \lambda z.z$. Получите $\lambda z.z = (\lambda xyz.xz(yz))(\lambda xy.x)(\lambda xy.x)$ и вспомните, что $K = \lambda xy.x$, $S = \lambda xyz.xz(yz)$.
   Ответ. $I = SKK$.

2. Для выражения:

   $$let\ x\ =\ \pi/2\ in\ let\ z\ =\ sin\ in\ sqr(z\ x);;$$

   постройте выражение комбинаторной логики и вычислите его значение.
   Указания.

   (a) λ -выражение имеет вид:

   $$\begin{aligned}(\lambda x.(\lambda z.sqr(z\ x))sin)\pi/2 & \\ = (\lambda x.sqr(sin\ x))\pi/2, & \quad (\beta)\\ = sqr(sin\ \pi/2). & \quad (\beta)\end{aligned}$$

   (b) Воспользуйтесь тем, что

   $$f(gx) = (f \circ g)x = S(KS)K\ f\ g\ x.$$

   Ответ. $S(KS)K\ sqr\ sin\ \pi/2 = 1$.

Раздел 20

Код де Брейна

Задача 20.1 *Для выражения:*

$$let\ x\ =\ plus\ in\ x\ (4, (x\ where\ x = 3));;$$

построить λ-выражение и выражение кода де Брейна и вычислить последнее с помощью $SECD$-машины.

Решение.

DB--1. 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:

$$\begin{array}{l}(\lambda xy.x)y\ = \\ \quad \overset{(\alpha)}{=} (\lambda uv.u)y \overset{(\beta)}{=} (\lambda v.y) \\ \quad \neq (\lambda y.y) = I.\end{array}$$

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:

$$\begin{aligned}
&\| x \| env \;=\; env(x),\\
&\| c \| env \;=\; c,\\
&\| (MN) \| env \;= \| M \| env(\| N \| env),\\
&\| \lambda x.M \| env\ d \;= \| M \| env[x \leftarrow d],
\end{aligned}$$

где:

$env(x)$	*- значение x в среде env;*
c	*- константа, обозначающее значение, также называемое c, что соответствует обычной практике;*
$env[x \leftarrow d]$	*- среда env, где x замещен на значение d.*

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 $(\ldots((),\ w_n)\ldots,\ w_0)$, 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:

$$\begin{array}{ll} \parallel \underline{0} \parallel (env, d) & = d, \\ \parallel \underline{(n+1)} \parallel (env, d) & = \parallel \underline{n} \parallel env, \\ \parallel c \parallel env & = c, \\ \parallel MN \parallel env & = \parallel M \parallel env(\parallel N \parallel env) \\ \parallel \lambda.M \parallel env\ d & = \parallel M \parallel (env, d). \end{array}$$

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:

$$\begin{array}{ll} \mathcal{S} & \textit{с арностью } 2, \\ \Lambda & \textit{с арностью } 1, \\ ' & \textit{с арностью } 1 \end{array}$$

и бесконечно много комбинаторов $n!$ в том смысле, что:

$$\begin{array}{ll} \parallel \underline{n} \parallel & = n!, \\ \parallel c \parallel env & = c, \\ \parallel MN \parallel & = \mathcal{S}(\parallel M \parallel, \parallel N \parallel), \\ \parallel \lambda.M \parallel & = \Lambda(\parallel M \parallel). \end{array}$$

Отсюда легко устанавливается процедура перехода от семантических равенств к чисто синтаксическим:

$$\begin{array}{ll} 0!(x, y) & = y, \\ (n+1)!(x, y) & = n!x, \\ ('x)y & = x, \\ \mathcal{S}(x, y)z & = xz(yz), \\ \Lambda(x)yz & = x(y, z) \end{array}$$

Такие правила близки к SK-правилам: первые три указывают на "забывающее" аргумент свойство (наподобие комбинатора); 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 $< \cdot, \cdot >$, где

$$\parallel (M, N) \parallel = < \parallel M \parallel, \parallel N \parallel >,$$

и снабдим его выделителями (проекциями) Fst и Snd. Введем также оператор композиции "$\circ$" и новую команду ε. Рассмотрим $\mathcal{S}\ (\cdot,\cdot)$ и $n!$ как сокращения для "$\varepsilon\circ\ <\ \cdot,\cdot\ >$" и "$Snd \circ Fst^n$" соответственно, где $Fst^{n+1} = Fst \circ Fst^n$. Сведем теперь все комбинаторные равенства воедино:

$$\begin{array}{ll}(ass) & (x \circ y)z = x(yz),\\ (fst) & Fst(x,y) = x,\\ (snd) & Snd(x,y) = y,\\ (dpair) & < x,y > z = (xz,yz),\\ (ac) & \varepsilon(\Lambda(x)y,z) = x(y,z),\\ (quote) & ('x)y = x,\end{array}$$

где $(dpair)$ устанавливает связь спаривания и образования совокупности, а (acc) - композиции и аппликации. Можно заметить, что $\mathcal{S}(x,y)z = \varepsilon(xz,yz)$. Тем самым придается однородность рассмотрению операторов Fst, Snd и ε . Кроме того, справедливо равенство:

$$'M = \Lambda(M \circ Snd),$$

откуда следует, что

$$('x)yz = xz.$$

Пользуясь синтаксическими и семантическими равенствами, получаем для $M = (\lambda x.x(4,(\lambda x.x)3))+$:

$$\begin{array}{rl}M' & =\parallel M \parallel=\parallel (\lambda.\underline{0}(4,(\lambda.\underline{0})3))+ \parallel\\ & = \mathcal{S}(\parallel \lambda.\underline{0}(4,(\lambda.\underline{0})3) \parallel, \parallel + \parallel)\\ & = \mathcal{S}(\Lambda(\parallel \underline{0}(4,(\lambda.\underline{0})3) \parallel), \parallel + \parallel)\\ & = \mathcal{S}(\Lambda(\mathcal{S}(0!, \parallel (4,(\lambda.\underline{0})3) \parallel)), \parallel + \parallel)\\ & = \mathcal{S}(\Lambda(\mathcal{S}(0!, <'\ 4, \mathcal{S}(\Lambda(0!),'3) >)), \Lambda(+ \circ Snd)).\end{array}$$

DB--2. Теперь произведем вычисления по методу Лендина для $SECD$-машины, то есть следует означивать путем приложения $'$ к среде. В данном случае среда пуста, поскольку терм замкнут. При означивании $'$ применим стратегию самого левого и при том самого внутреннего выражения. Для краткости обозначим:

$$A = \mathcal{S}(0!, <'\ 4, B >), B = \mathcal{S}(\Lambda(0!), 3).$$

Приведем полную последовательность редукций:

$$\begin{array}{l}(\Lambda(A), \Lambda(+ \circ Snd))() \\ \quad \to \varepsilon(\Lambda(A)(), \Lambda(+ \circ Snd)()) \\ \quad \to A\ env\end{array}$$

(здесь: для сокращения $env = ((), \Lambda(+ \circ Snd)())$*)*

$$\begin{array}{l}\to \varepsilon(0!env, <'\ 4,\ B > env) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), ('4\ env, B\ env)) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), (4, B\ env)) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), (4,\ \varepsilon(\Lambda(0!)env,'\ 3\ env))) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), (4,\ \varepsilon(\Lambda(0!)env, 3))) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), (4, 0!(env, 3))) \\ \to \varepsilon(\Lambda(+ \circ Snd)(), (4, 3)) \\ \to (+ \circ Snd)((), (4, 3)) \\ \to +(Snd((), (4, 3))) \\ \to +(4, 3) \\ \Rightarrow 7.\end{array}$$

Видно,что результат совпадает с результатом, полученным в ходе непосредственных вычислений выражения.

Контрольные вопросы.

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. Воспользуйтесь теоретико-множественными определениями для этих понятий, положив $f : D \to E,\ g : D \to F$.

совокупность :	$h =< f, g >: D \to E \times F;$
пара :	$[f, g] = (f, g) : (D \to E) \times (D \to F).$

Упражнение. Постройте выражения де Брейна для приводимых далее λ-выражений.

1. $\lambda y.yx$.
   Ответ: $\parallel \lambda y.yx \parallel = \Lambda(\mathcal{S}(0!, 1!))$.

2. $(\lambda x.(\lambda z.zx)y)((\lambda t.t)z)$.
Указание. Обозначим исходное выражение через Q и перейдем к выражению $R = \lambda zxy.Q$. 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”, “$\circ$” на "$\mathcal{S}$", “$\underline{n}$” на “$n!$” получить код де Брейна для Q.
Ответ.

$$Q_{DB(z,x,y)} = \mathcal{S}(\Lambda(\mathcal{S}(\Lambda(\mathcal{S}(0!, 1!))1!)), \mathcal{S}(\Lambda(0!), 2!))$$

(порядок имен переменных в индексе Q соответствует порядку их связывания в промежуточном выражении R и позволяет восстановить исходное определение с соответствующими свободными переменными).

Раздел 21

Абстрактная машина: КАМ

Теоретические сведения. *Аббревиатура "КАМ" принята в качестве сокращения для "Категориальная Абстрактная Машина". Как можно видеть, такое название само по себе несет значительную смысловую нагрузку.*

- *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 $< \cdot, \cdot >$, а также проекций Fst и Snd. Декартово замкнутые категории (д.з.к.) дополнительно содержат каррирование Λ, аппликирование ε и средства экспоненцирования, то есть средства построения функциональных пространств.*
  Перечисленные в предыдущем разделе правила позволяют производить означивание категориальных термов вида M', построенных из указанных комбинаторов путем аппликации (приложения) терма к среде, которая построена из совокупности различных компонентов.
  - ▷ *Комбинаторы аппликации и построения совокупности не имеются в M'.*
  - ▷ *Аппликация возникает, когда записываем $M'()$, то есть когда категориальный терм прикладывается к (пустой) среде.*
  - ▷ *Построение совокупности происходит всякий раз, когда используется правило $(dpair)$.*

  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.

- ▷ *Рассматриваем терм как код, действующий на w, причем образован из элементарных составляющих кода.*
- ▷ *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.*
- ▷ *Для совокупностей рассматривается действие $< M, N >$ на w в соответствии с равенством:*

$$< M, N > w = (Mw, Nw).$$

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 . 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.
Исходя из рассмотренных выше соображений возникает структура машины:

T – *терм как структурированное значение, например граф*;
C – *код*;
S – *стек, или дамп (вспомогательная память)*.

Состоянием машины считается тройка $< T,\ S,\ C >$.
На основании предыдущего кодом для $< M, N >$ считается следующая последовательность инструкций:

> “$<$” , за которой следует последовательность инструкций, соответствующая коду , за которой следует “,”, за которой следует последовательность инструкций, соответствующая коду N, за которой следует “$>$”.

- *Отметим, что категориальная абстрактная машина (CAM) совершает только символьные преобразования, компиляция нигде не проводится. Достигается это с помощью следующих инструкций. 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 $< M, N >$.

- *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 соглашением.

  - ▷ *Примем xy или $x \mid y$ 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 предствляет собой среду декларации функции, описанной посредством абстракции.

- *Вернемся к операции аппликации из λ-исчисления. Ее запишем, как $\varepsilon\circ < M, N >$. В виде правила запишем равенство:*

$$(\varepsilon\circ < M, N >)w = \varepsilon(Mw, Nw) \quad (= \varepsilon[Mw, Nw]).$$

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 $\varepsilon(\Lambda(P)w'_1, w_2) = P(w'_1, w_2)$.
В терминах KAM кодом, соответствующим $\varepsilon\circ < M, N >$, 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 .

Дополнительно нужны константы. Они требуются для базисных констант, когда кодом для $'C$ является $'(C)$ со следующим действием:

> $'C$: замещает терм на инкапсулированную константу .

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".

Прежде чем приступить к построению полной таблицы инструкций, вспомним соглашения об обозначениях для списков инструкций

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

и элементов стека:

пустой список L обозначается через [];
обозначение $[e_1;\ldots;e_n]$ принято для списка с n элементами $e_1,\ldots,e_n$;
$a.L$ 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 табл. 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 обраба-

тывать (неявный) оператор неподвижной точки. Для этого снова требуются дополнительные соглашения.

Задача 21.1 *Для выражения:*

$$let\ x\ =\ plus\ in\ x(4,(x\ where\ x=3));;$$

построить его запись с помощью "категориального кода" и написать программу вычисления в терминах КАМ-инструкций.

Решение.

CAM--1. Воспользуемся математической символикой, которая подчеркивает связь с правилами перезаписи. Пусть A, B обозначают коды A и B соответственно, $+$ служит сокращением для $plus$, $\mathcal{S}(x,y)\ =\ \mathcal{S}[x,y]\ =\ \varepsilon\circ<x,y>$.
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.
Контрольные вопросы.

1. Назовите три основных подхода к реализации языков функционального программирования, лежащих в основе концепции категориальной абстрактной машины.

2. Что представляют собой базисные машинные инструкции КАМ?

3. Опишите проекции Fst и Snd применительно к структуре машины.

Упражнение. Опишите таблицу машинных инструкций КАМ для вычисления выражения:

$$let\ x = 3\ in\ (op(7, x)\ where\ op\ =\ sub).$$

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:

$$\mathcal{S}(\Lambda(\mathcal{S}(0!, <'\ 7, \mathcal{S}(\Lambda(0!), '\ 3)\ >)), \Lambda(sub \circ Snd)).$$

Теперь,переписав его в виде инструкций КАМ и выполнив необходимые преобразования, можно получить ответ.
Ответ. 4.

Таблица 21.2: Вычисление на КАМ

Терм	Код	Стек
$()$	$< \Lambda(A), \Lambda(Snd+) > \varepsilon$	$[]$
$()$	$\Lambda(A), \Lambda(Snd+) > \varepsilon$	$[()]$
$A : ()$	$, \Lambda(Snd+) > \varepsilon$	$[()]$
$()$	$\Lambda(Snd+) > \varepsilon$	$[A : ()]$
$\ldots$	$\ldots$	$\ldots$
$\oplus$	$> \varepsilon$	$[A : ()]$
$(A : (), \oplus)$	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	$[]$	$[]$

Раздел 22

Оптимизация КАМ-вычислений

Задача 22.1 *Для выражения:*

$$let\ x = 5\ in\ let\ z\ y = y + x\ in\ let\ x = 1\ in\ (zx) \times 2;;$$

составить КАМ-программу вычисления значения и по возможности оптимизировать ее.

Решение.

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:

$$P\ =\ (\lambda x.(\lambda z.(\lambda x.(zx) \times 2)1)(\lambda y.y + x))5.$$

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 - код), за которой следует "$swap$", за которой следует код 1 для N, за которым следует "$cons$" и "ε". Предполагая, что означивание 1 на терме s дает значение w, приведем основные шаги вычисления:

$$код\ ((\lambda.M)N)\ =\ push.cur\ C.swap.C1@[cons; \varepsilon],$$

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

$$C \;=\; \textit{код}(M), \quad C1 \;=\; \textit{код}(N).$$

В табл. 22.1 представлены вычисления значения.

Заметим, что

$$\parallel (\lambda.M)N \parallel = < \Lambda(\parallel M \parallel), \parallel N \parallel > \varepsilon.$$

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 аппликирование. Отправной точкой служит правило:

$$(Beta) \qquad\qquad \varepsilon\circ < \Lambda(x), y >= x\circ < Id, y > .$$

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:

$$< g > = \ \lambda t.\lambda r.r(t)(gt) \ = \ \lambda t.(t, gt) = \lambda t.(Id\, t, gt).$$

Замещая в первом равенстве f *на* Id, *получаем:*

$$< Id, g >=< g >,$$

что обосновывает употребление символа "$< \cdot >$*" для единственного выражения (вырожденный случай). Далее,*

$$< Id, g >= [push; skip; swap; g; cons], \ \ < g >= [push; g; cons].$$

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:*

$$[\oplus] \qquad \begin{aligned} \varepsilon\circ < \Lambda(+\circ Snd), < M, N >> &= \\ &= \ (+\circ Snd)\circ < Id, < M, N >> \\ &= \ +\circ < M, N >=< M, N > +. \end{aligned}$$

Более подробно:

$$\begin{aligned} (\varepsilon\circ < \Lambda(+\circ Snd), < M, N >)t \ &= \ \varepsilon(\Lambda(+\circ Snd)t, < M, N > t) \\ &= (\Lambda(+\circ Snd)t)(< M, N > t) \\ &= (+\circ Snd)(t, < M, N > t) \\ &= (+(Snd(t, < M, N > t) \\ &= (+\circ < M, N >)t, \end{aligned}$$

[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 аппликируемость 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):*

$$\lambda t.(ft, gt, kt) \;=\; \lambda t\lambda r'.r'(\lambda r.r(ft)(gt))(kt) = \\ =\; \lambda t.(< f, g > t, kt) \;=<< f, g >, k >,$$

следовательно:

$$+(M, N, O) \;=\; + << M, N >, O > .$$

Действительно,

$$\begin{aligned} &\varepsilon\circ < \Lambda(+\circ Snd), \varepsilon\circ < \Lambda(Snd), << M, N >, O >>> = \\ &\quad = \; +\circ Snd\circ < Id, \varepsilon\circ < \Lambda(Snd), << M, N >, O >>> \\ &\quad = \; +\circ (\varepsilon\circ < \Lambda(Snd), << M, N >, O >>>) \\ &\quad = \; +\circ Snd\circ < Id, << M, N >, O >>) \\ &\quad = +\circ << M, N >, O > . \end{aligned}$$

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* $x \mid y = y \circ x$ *для упрощения компилируемой записи. Введем сокращения:*

$$Fst = F,\; Snd = S,$$

Изложение сделаем замкнутым, приведя все подробности выкладок. Формулировку принципа оптимизации $(Beta)$ *возьмем в виде:*

$$(Beta) \qquad < \Lambda(X), Y >| \ \varepsilon = x \circ < Id, y > = < y >| \ x.$$

Формулировку принципа оптимизации $[\oplus]$ *возьмем в виде:*

$$[\oplus] \quad < \Lambda(+ \circ Snd), < M, N >> | \ \varepsilon = (+ \circ Snd) \circ < Id, < M, N >> .$$

Кодирование по де Брейну дает:

$$P = (\lambda x.(\lambda z.(\lambda x.(z\ x) \times 2)1)(\lambda y.y + x))5,$$

затем

$$\begin{aligned} P' = \quad & (\lambda.(\lambda.(\lambda.(\underline{1}\ \underline{0}) \times 2)1)(\lambda.\underline{0} + \underline{1}))5 \\ & = (\lambda.(\lambda.(\lambda. \times ((\underline{1}\ \underline{0}), 2))1)(\lambda. + (\underline{0}, \underline{1})))5. \end{aligned}$$

Далее вычисление значения P' *дает:*

$$\begin{aligned}
& \| \ P' \ \| = <' \ 5 >| \| \ (\lambda.(\lambda(\underline{1}\ \underline{0}) \times 2)1)(\lambda.\underline{0} + \underline{1}) \ \|; \\
& \| \ (\lambda.(\lambda(\underline{1}\ \underline{0}) \times 2)1)(\lambda.\underline{0} + \underline{1}) \ \| \\
& \qquad = <\| \ \lambda.(\lambda.(\underline{1}\ \underline{0}) \times 2)1 \ \|, \| \ \lambda.\underline{0} + \underline{1} \ \| > \varepsilon \\
& \qquad = < \Lambda \ \| \ \lambda.(\lambda.(\underline{1}\ \underline{0}) \times 2)1 \ \|, \| \ \lambda.\underline{0} + \underline{1} \ \| > \varepsilon \\
& \qquad \stackrel{(Beta)}{=} <\| \ \lambda.\underline{0} + \underline{1} \ \| >| \ \| \ (\lambda.(\underline{1}\ \underline{0}) \times 2)1 \ \|; \\
& \| \ (\lambda.(\underline{1}\ \underline{0}) \times 2)1 \ \| = <\| \ \lambda.(\underline{1}\ \underline{0}) \times 2) \ \|,' 1 > \varepsilon \\
& \qquad = < \Lambda \ \| \ (\underline{1}\ \underline{0}) \times 2 \ \|,' 1 > \varepsilon \\
& \qquad = <' \ 1 >| \ \| \ (\underline{10}) \times 2 \ \|; \\
& <\| \ \lambda.\underline{0} + \underline{1} \ \| > = < \Lambda \ \| \ +(\underline{0}, \underline{1}) \ \| > \\
& \qquad = < \Lambda <' +, < 0!, 1! >> \varepsilon > \\
& \qquad = < \Lambda < \Lambda(+ \circ S), < 0!, 1! >> \varepsilon >; \\
& \qquad \stackrel{(\oplus)}{=} < \Lambda(+ \circ < 0!, 1! >) > \\
& \qquad = < \Lambda(< 0!, 1! >| +) > \\
& \qquad = < \Lambda(< S, F \mid S >| +) >; \\
& \| \ (\underline{1}\ \underline{0}) \times 2) \ \| = <\| \ \times \ \|, <\| \ (\underline{1}\ \underline{0}) \ \|,' 2 >> \varepsilon \\
& \qquad = < \Lambda(\times \circ S), <\| \ (\underline{1}\ \underline{0}) \ \|,' 2 >> \varepsilon \\
& \qquad \stackrel{(\oplus)}{=} \times \circ <\| \ (\underline{1}\ \underline{0}) \ \|,' 2 > \\
& \qquad = \times \circ << 1!, 0! > \varepsilon,' 2 > \\
& \qquad = \times \circ << F \mid S, S > \varepsilon,' 2 > \\
& \qquad = << F \mid S, S >| \varepsilon,' 2 >| \times.
\end{aligned}$$

Для экономичной записи вычислений введены сокращения. С учетом сокращений для D, где $D = (< S, F \mid S >\mid +)$ *и для C, где* $C =< F \mid S, S >\mid \varepsilon$, *получим:*

$$\parallel P' \parallel = <' 5 >\mid < \Lambda(D) >\mid <' 1 >\mid < C', 2 >\mid \times.$$

Для сокращения громоздкости КАМ-вычислений в порядке соглашения обозначим:

$$B =< C,' 2 >\mid \times, C =< F \mid S, S >\mid \varepsilon, D =< S, F \mid S >\mid +.$$

Обращаем внимание, 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" вычислений[2] *. 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 <терм, код, стек > является в точности текущим состоянием (вычисления). Поэтому последовательность строк в таблице КАМ-вычислений задает последовательность состояний процесса вычисления*[3]. *При-*

[2]Методы оптимизации вычислений, когда удается 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".

[3]Вычисление 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.

ведем полную последовательность КАМ-вычислений:

Терм	*Код*	*Стек*
$()$	$<' 5 >< \Lambda(D) >< ' 1 > B$	$[]$
$()$	$'5 >< \Lambda(D) >< ' 1 > B$	$[()]$
(5)	$>< \Lambda(D) >< ' 1 > B$	$[()]$
$((), 5)$	$< \Lambda(D) >< ' 1 > B$	$[]$
$((), 5)$	$\Lambda(D) >< ' 1 > B$	$[((), 5)]$
$(D : ((), 5))$	$>< ' 1 > B$	$[((), 5)]$
$(((), 5), D : ((), 5))$	$<' 1 > B$	$[]$
$(((), 5), D : ((), 5))$	$'1 > B$	$[\ldots]$
(1)	$> B$	$[\ldots]$
$(((), 5), D : ((), 5)); 1$	B	$[]$
$(((), 5), D : ((), 5)); 1$	$< C,' 2 > \times$	$[]$
$(((), 5), D : ((), 5)); 1$	$C,' 2 > \times$	$[\ldots; 1]$
$(((), 5), D : ((), 5)); 1$	$< F \mid S, S >\mid \varepsilon,' 2 > \times$	$[\ldots; 1]$
$(((), 5), D : ((), 5)); 1$	$F \mid S, S >\mid \varepsilon,' 2 > \times$	$[\ldots; \ldots; 1]$
$((D : ((), 5)))$	$, S >\mid \varepsilon,' 2 > \times$	$[\ldots; \ldots; 1]$
$(((), 5), D : ((), 5)); 1$	$S >\mid \varepsilon,' 2 > \times$	$[D : ((), 5)); \ldots]$
(1)	$> \varepsilon,' 2 > \times$	$[D : ((), 5)); \ldots]$
$((D : ((), 5)); 1)$	$\varepsilon,' 2 > \times$	$[\ldots]$
$(((), 5); 1)$	$D,' 2 > \times$	$[\ldots]$
$(((), 5); 1)$	$< S, F \mid S > +,' 2 > \times$	$[\ldots]$
$(((), 5); 1)$	$S, F \mid S > +,' 2 > \times$	$[(((), 5); 1); \ldots]$
(1)	$, F \mid S > +,' 2 > \times$	$[(((), 5); 1); \ldots]$
$(((), 5); 1)$	$F \mid S > +,' 2 > \times$	$[1; \ldots]$
(5)	$> +,' 2 > \times$	$[1; \ldots]$
$(1, 5)$	$+,' 2 > \times$	$[\ldots]$
(6)	$,' 2 > \times$	$[\ldots]$
$(\ldots)$	$'2 > \times$	$[6]$
(2)	$> \times$	$[6]$
$(6, 2)$	$\times$	$[]$
(12)	$[]$	$[]$

Контрольные вопросы.

1. Сформулируйте основные правила оптимизации.

2. Какое преимущество дает использование принципов $(Beta)$ и $[\oplus]$ при написании КАМ-программ ?

Упражнение.
Напишите оптимизированную КАМ-программу для вычисления выражения:

$$let\ x = 3\ in\ let\ z = x + y\ in\ fz\ where\ y = 1\ where\ f = sqr.$$

Указание. Переходим к λ-выражению:

$$(\lambda x.(\lambda z.(\lambda f.fz)sqr)((\lambda y.+xy)1))3,$$

получаем код де Брейна:

$$(\lambda.(\lambda.(\lambda.\underline{0}\ \underline{1})sqr)((\lambda.+\ \underline{1}\ \underline{0})\ 1))3.$$

Далее,

$\mathcal{S} \parallel \lambda.(\ldots)(\cdots), 3 \parallel = \mathcal{S}(\Lambda(\parallel (\ldots)(\cdots) \parallel), '3);$
$\parallel (\ldots)(\cdots) \parallel = \mathcal{S}(\parallel (\ldots) \parallel, \parallel (\cdots) \parallel);$
$\parallel (\ldots) \parallel = \Lambda(\mathcal{S}(\Lambda(\mathcal{S}(0!, 1!)), \parallel sqr \parallel));$
$\parallel (\cdots) \parallel = \mathcal{S}(\Lambda(\mathcal{S}(\mathcal{S}(\parallel + \parallel, 1!), 0!)), '1);$
$\mathcal{S}_0(\Lambda_1(\mathcal{S}_2(\Lambda_3(\mathcal{S}_4(\Lambda_5(\mathcal{S}_6(0!, 1!)_6)_5, \parallel sqr \parallel)_4)_3,$
$\mathcal{S}_3(\Lambda_4(\mathcal{S}_5(\mathcal{S}_6(\parallel + \parallel, 1!)_6, 0!)_5)_4, '1)_3)_2)_1, '3)_0 \equiv R.$

Проделаем проверку, выполнив прямые вычисления:
$R\ \rho \equiv \mathcal{S}_0(\ldots, '3)_0\rho = (\Lambda_1(\ldots)_1\rho)('3\ \rho) =_1 (\ldots)_1(\rho, 3) \equiv_1 (\ldots)_1\rho';$
$_1(\ldots)_1\rho' \equiv \mathcal{S}_2(\ldots\ ,\ \ldots)_2\rho' = \mathcal{S}_4(\ldots\ ,\ \ldots)_4(\rho', _4(\ldots)_4(\rho', 1));$
$_4(\ldots)_4(\rho', 1) = \mathcal{S}_5(\ldots\ ,\ \ldots)_5(\rho', 1) = \mathcal{S}_6(\ldots\ ,\ \ldots)_6(\rho', 1)1 =$
$= \Lambda(+ \circ Snd)(\rho', 1)\rho'\ 1 = (+ \circ Snd)((\rho', 1), 3)1 = +\ 3\ 1 = 4;$
$\mathcal{S}_4(\ldots\ ,\ \ldots)_4(\rho', 4) = \Lambda_5(\ldots)_5(\rho', 4)(\Lambda_5(\cdots)_5(\rho', 4)) =$
$=_5 (\ldots)_5((\rho', 4), (\Lambda_5(\cdots)_5(\rho', 4))) \equiv$
$\mathcal{S}(0!, 1!)(\ldots\ ,\ \ldots) = \Lambda_5(\cdots)_5(\rho', 4)4 =_5 (\cdots)_5((\rho', 4), 4) \equiv$
$\equiv (sqr \circ Snd)((\rho', 4), 4) = sqr(4) = 16.$

Полученное выражение R следует предварительно оптимизировать по правилам $(Beta)$ и $[\oplus]$. Для этого полезно предварительно использовать равенство:

$$\mathcal{S}(x, y) = \varepsilon \circ < x, y > .$$

Ответ. $sqr(4) = 16$.

Раздел 23

Переменные объекты

Теоретические сведения. *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. Взаимодействие концептов, индивидов, состояний позволяет сформулировать основной принцип концептуализации следующем образом:*

$$\begin{gathered} \textit{значение(индивидный концепт)} = \\ = \textit{функция : соотнесения} \rightarrow \textit{индивиды} \end{gathered}$$

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:

$$\begin{gathered} \textit{значение(индивидный концепт)} \in \textit{индивид}^{\textit{соотнесение}} \succ\!\!\!- \\ \succ\!\!\!- \textit{индивид} \succ\!\!\!- \textit{состояние}^{\textit{соотнесение}} \succ\!\!\!- \textit{состояние} \end{gathered}$$

В сформулированной схеме использованы следующие обозначения:

- *знак "$\succ\!\!\!-$" отмечает переход от инварианта к семейству;*
- *запись $\textit{индивид}^{\textit{соотнесение}}$ означает множество всех отображений из соотнесений в индивиды, то есть экспоненту;*
- *запись $\textit{состояние}^{\textit{соотнесение}}$ означает множество всех отображений из соотнесений в состояния.*

.

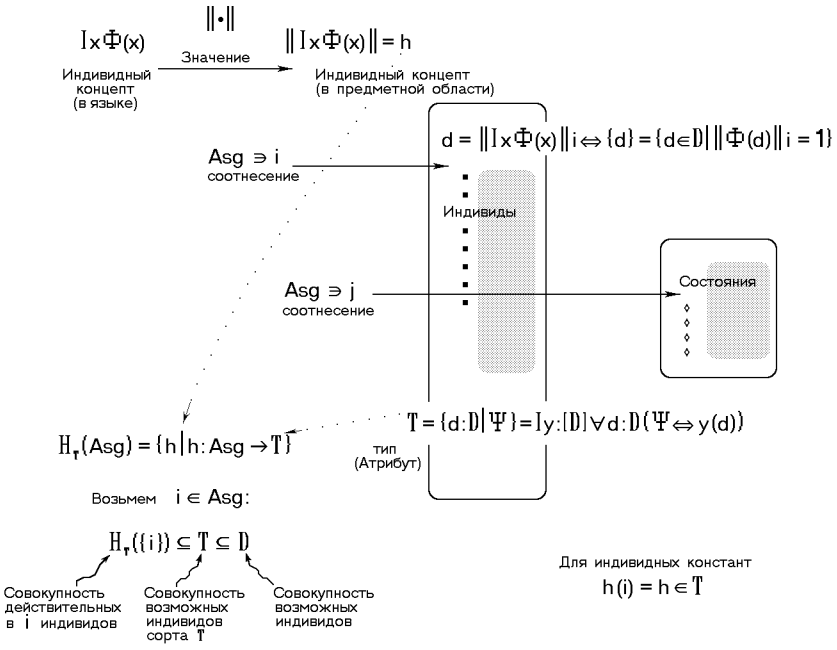

Рис. 23.1: Модель объекта данных

Основная особенность схемы заключается в переходе от индивидных концептов (в языке) к индивидным концептам в предметной области. Такой подход обеспечивается функцией вычисления значения. Далее концепт ПО[1] *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 индивидный концепт описывается формулой* Φ*, которая дополнительно снабжается оператором дескрипции* **I**. *Сперва для такого описания вычисляется значение, которое позволяет языковой конструкции по-*

[1]ПО -- **п**редметная **о**бласть.

ставить в соответствие объект предметной области. Для полученного таким способом образа языковой конструкции учтем соотнесение, что формально выражается индексом. При фиксации индивидов в МОД наиболее важно соблюдать справедливость условия

$$\| \mathbf{I}x\Phi(x) \| i \ = \ d \Leftrightarrow \{d\} \ = \ \{d \in D \mid \| \Phi(\overline{d}) \| i = 1\},$$

которое считаем основной характеристикой формализации.

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 самопременимость функции, что считается невозможным для "обычных" математических функций. В качестве основной бестиповой системы обычно используется λ*-исчисление, опирающееся на понятие связанной переменной, или комбинаторная логика, вовсе не использующая понятие переменной.*

23.1 Основная задача

Задача 23.1 *Построить объект заданного вида.*

Формулировка задачи. Установить определение объекта, применив принцип свертывания:

$$C \quad = \quad \mathbf{I}y : [D]\forall x : D(y(x) \leftrightarrow \Phi) = \{x : D \mid \Phi\}. \qquad (C)$$

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.

23.1.1 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*

$$R = \mathbf{I}z : [A, B]\forall x : A\forall y : B(z[x, y] \leftrightarrow \Psi) = \{[x : A, y : B] \mid \Psi\},$$

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.

23.1.2 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

$$\begin{aligned} C \;=\; C(I) \;&=\; \mathbf{I}z : [I,T]\forall i : I\forall hi : T(z[i,hi] \leftrightarrow \Phi) \\ &=\; \{[i,hi] \mid \Phi\} \subseteq \{h \mid h : I \to T\} \\ &=\; H_T(I). \end{aligned}$$

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

$$C = \mathbf{1}_C : C \to C,$$

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

$$\begin{aligned} \phi \;=\; \phi(I) \;&=\; \mathbf{I}z : [I,(T,\mathcal{T})].\forall i\forall ui : T\forall vi : \mathcal{T}(z[i,[ui,vi]] \leftrightarrow \Phi) \\ &=\; \{[i,[ui,vi]] \mid \Phi\} \\ &\subseteq \{< u,v > | < u,v > : I \to T \times \mathcal{T}\} \\ &=\; H_{T\times\mathcal{T}}(I). \end{aligned}$$

Существенный момент заключается в том, что u является элементом $H_T(I)$, а v -- элементом $H_{\mathcal{T}}(I)$, то есть $u \in H_T(I),\ v \in H_{\mathcal{T}}(I)$. Более того, при определении дескрипции для ϕ справедливы следующие равенства:

$$\begin{aligned}\phi = \phi(I) = & \ \mathbf{I}z : [(I,T),(I,\mathcal{T})].\forall i \forall ui \forall vi(z[[i,ui],[i,vi]] \leftrightarrow \Phi) \\ = & \ \{[[i,ui],[i,vi]] \mid \Phi\} \\ \subseteq & \ \{[u,v] \mid \Phi\} \\ = & \ H_T(I) \times H_{\mathcal{T}}(I).\end{aligned}$$

Оба выражения для ϕ оказываются изоморфными и могут быть положены в основу системы типизации.

23.1.3 Вычислительные модели

Рассмотрим типизированные вычислительные модели объектов данных. Основной целью системы типизации ОД [2] является ее использование для моделирования выполнения либо конструкций ЯМОД[3], 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

$$< \{H_B\}, \{ev_{BC}\} >$$

для произвольных типов $B,\ C$ из системы типизации ОД, где H -- домен объектов типа B, ev_{BC} -- аппликатор вычисления значения функции на аргументе из H_B.

$H_T(I)$--1. 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 $(A \to B)$;*
- *установление для его использования отображения-аппликатора $ev_{BC} : (B \to C) \times B \to C$, которое обеспечивает по $f : B \to C$ и $x : B$ вычисление $f(x) : C$, то есть $f, x \mapsto f(x)$;*
- *при вычислении $h(x,y)$ двухаргументного отображения-оператора $h : A \times B \to C$ фиксируется x, а $h(x,y)$ считается функцией*

[2] ОД -- **о**бъекты **д**анных.

[3] ЯМОД -- **я**зык **м**анипулирования **о**бъектами **д**анных.

от y;
• для связи h со значениями вводится специальная функция абстрактор Λ_{ABC}, типизируемая посредством

$$(\Lambda_{ABC}h) : A \to (B \to C);$$

• для заданного опертора h и аргумента x абстрактор имеет вид $(\Lambda_{ABC}h)(x) : B \to C$, 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. Функциональнное пространство. Этот объект представляет множество функций из типа $\mathbf{A}$ в тип $\mathbf{B}$, что записываем в виде дескрипции*

$$\mathbf{A} \to \mathbf{B} \;=\; \{z : [\mathbf{A},\mathbf{B}] \mid \forall x \in \mathbf{A} \exists y \in \mathbf{B}.z[x,y]\}$$

или в более слабой форме для сортов $A,\ B$ и $A \to B$. 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:*

$$ev_{AB} \;=\; \lambda[z,x]\mathbf{I}y.z[x,y],$$

которая по функции $z \in \mathbf{A} \to \mathbf{B}$ *и аргументу* $x \in \mathbf{A}$ *указывает значение* $y \in \mathbf{B}$*, то есть*

$$ev_{AB} : [z,x] \mapsto z(x),$$

или в более общем случае, $ev_{AB} : [z,x] \mapsto y$.

23.1.4 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 соотнесение либо разворачивание событий. Перечисленный порядок действий, нацеленный на учет динамики, предполагает введение средствами ЯООД[4] целого ряда ОМД[5], 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* $(\lambda.\Phi)\overline{h}$*, где* Φ *-- тело* λ*-абстракции,* h *- индивид (индивидная константа) из предметной области, получаем :*

$$\parallel (\lambda.\Phi)\overline{h} \parallel i = \parallel \Phi \parallel [i,hi]$$

для соотнесения i*. Тем самым увязка индивида* h *с телом* λ*-абстракции* Φ *сводится к построению концепта, соответствующего* $\parallel \Phi \parallel$ *и проверке принадлежности* $h \in \parallel \Phi \parallel$*. Проверка принадлежности производится операцией селекции*

[4] ЯООД -- **я**зык **о**пределения **о**бъектов **д**анных.
[5] ОМД -- **о**бъекты **м**ета**д**анных.

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 экстенсионала концепта $C'(i)$; для него справедливо включение

$$C'(i)\ ISA\ \mathbf{T};$$

5) все отобранные пары $[i, hi]$ идентифицируют индивиды h в мире i из совокупности I и принимаются за экстенсионал переменного концепта $C(I)$;
6) сохраняется естественное включение $C(I) \subseteq H_T(I)$ для переменного домена $H_T(I)$, задаваемого определением

$$H_T(I) \ = \ \{h \mid h : I \to \mathbf{T}\}.$$

$H_T(I)$*--2. Динамика концепта. Наиболее действенным применением 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 $f : B \to I$, то есть из мира I осуществляется переход в мир B. В этом случае применение свойства расширяемости при вычислении значений λ-абстракций имееет определенную специфику. Для аппликации $(\lambda.\Phi)\overline{h}$ получаем:*

$$\parallel (\lambda.\Phi)\overline{h} \parallel i \ = \parallel \Phi \parallel_f [b, (h \circ f)b]$$

для соотнесения $i = fb$. Таким образом, увязка индивида h с телом λ-абстракции Φ сводится не к проверке принадлежности индивида h значению $\parallel \Phi \parallel$, а к проверке принадлежности образа h при разворачивании событий по закону f в мире b для смещенной оценки $\parallel \Phi \parallel_f$. 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 экстенсионала концепта $C'_f(b)$; для него справедливо включение $C'_f(b)\ ISA\ \mathbf{T}$;
5) все отобранные пары $[b, hb]$ идентифицируют индивиды h в мире b из совокупности B и принимаются за экстенсионал переменного концепта $C_f(B)$;
6) сохраняется естественное включение $C_f(B) \subseteq H_T(B)$ для домена $H_T(B)$, задаваемого определением $H_T(B) = \{h \mid h : B \to \mathbf{T}\}$.

$H_T(I)$*--3. Статика концепта. Предельным случаем закона разворачивания событий является тождественное (единичное) преобразование. Тогда запись*

$$f = \mathbf{1}_I : I \to I$$

означает, что закон разворачивания событий f не приводит к изменениям в предметной области. При вычислении значения λ-абстракции получаем:

$$\| (\lambda.\Phi)\overline{h} \| i = \| \Phi \|_{\mathbf{1}_I}[i, hi].$$

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 $C_{\mathbf{1}_I}(I) : h \to h$, то есть индивид h переводится сам в себя и $C_{\mathbf{1}_I}(I) = C_{1_D}(D) = 1_D$. С другой стороны, получаем $C_{\mathbf{1}_I}(I) = C(I)$ и $C_I : C_I \to C_I$, откуда следует правило:

индексированный концепт ведет себя
как единичное преобразование.

Более подробно, для произвольных концептов A, B и отображения $f : A \to B$ 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 $f : B \to I$ понимается как переход от мира I к миру B, то есть от уровня знания I к уровню знания B и т.п.;
2) получаем: $f = \mathbf{1}_I \circ f \circ \mathbf{1}_B$ и $C_{\mathbf{1}B} = C(B),\ C_{\mathbf{1}I} = C(I)$;
3) для концепта $C_f = C_f(B)$ справедливо следующее включение:

$$C(f) : C(I) \to C(B);$$
$$C_f \subseteq C(B),$$

поскольку проверка типа $\mathbf{T}$ для индивида h и его образа при преобразовании f сохраняет естественные включения для переменных доменов:

$$\begin{array}{lcl} C(I) \subseteq H_T(I) & = & \{h \mid h : I \to \mathbf{T}\}; \\ C(B) \subseteq H_T(B) & = & \{h \mid h : B \to \mathbf{T}\}; \\ C_f & = & \{h \circ f \mid h \circ f : B \to \mathbf{T}\} \subseteq C(B). \end{array}$$

$H_T(I)$*--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 $(\lambda\lambda.\Phi)\overline{uv}$ для формулы*

Φ и индивидов $\overline{u}$ и $\overline{v}$. Получаем:

$$\begin{aligned} \parallel (\lambda\lambda.\Phi)\overline{uv} \parallel i &= (\parallel (\lambda\lambda.\Phi) \parallel i)(\parallel \overline{u} \parallel i)(\parallel \overline{v} \parallel i) \\ &= \Lambda \parallel \lambda.\Phi \parallel i(ui)(vi) \\ &= (\parallel \lambda.\Phi \parallel [i,ui])(vi) \\ &= \Lambda \parallel \Phi \parallel [i,ui](vi) \\ &= \parallel \Phi \parallel [[i,ui],vi] \end{aligned}$$

для соотнесения i. Возможен и иной способ рассуждения:

$$\begin{aligned} \parallel (\lambda.\Phi)[\overline{u},\overline{v}] \parallel i &= \Lambda \parallel \Phi \parallel i(\parallel [\overline{u},\overline{v}] \parallel i) \\ &= \Lambda \parallel \Phi \parallel i[ui,vi] \\ &= \parallel \Phi \parallel [i,[ui,vi]], \end{aligned}$$

при применении которого предполагается, что Φ - двухместный оператор (некаррированный). Увязка индивида $[ui,vi]$ с телом λ-абстракции Φ сводится к построению концепта, соответствующего $\parallel \Phi \parallel$, и к проверке принадлежности $[ui,vi] \in \parallel \Phi \parallel$. В данном случае индивид представляет собой упорядоченную пару, каждый элемент которой определен на соответствующем одноместном концепте:

$$\begin{aligned} \parallel (\lambda.\Psi_1)\overline{u} \parallel i &= \parallel \Psi_1 \parallel [i,ui] = \mathbf{U}(\{i\}); \\ \parallel (\lambda.\Psi_2)\overline{v} \parallel i &= \parallel \Psi_2 \parallel [i,vi] = \mathbf{V}(\{i\}). \end{aligned}$$

Проверка принадлежности индивида $wi = [ui,vi]$ концепту $\phi = \parallel \Phi \parallel$ 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 эктенсионала концепта $\phi'(i)$; для него справедливо включение

$$\phi'(i)\ ISA\ (\mathbf{T} \times \mathcal{T});$$

5) все отобранные пары $[i,wi]$ идентифицируют индивиды w в мире i из совокупности I и принимаются за экстенсионал 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

$$H_{\mathbf{T}\times\mathcal{T}}(I) = \{w \mid w : I \to (\mathbf{T} \times \mathcal{T})\}.$$

$H_T(I)$--*6. Динамика отношения. Учет изменений, происходящих в предметной области, проявляется в ассоциированных изменения базы данных. Приняв в качестве закона разворачивания событий* $f : B \to I$, *применим свойство расширяемости при вычислении значений* λ*-абстракции. Для аппликации* $(\lambda.\Phi)[\overline{u}, \overline{v}]$ *получаем:*

$$\parallel (\lambda.\Phi)[\overline{u}, \overline{v}] \parallel i \; = \parallel \Phi \parallel_f [b, (< u, v > \circ f)b].$$

Это выражение может быть записано в несколько модифицированном виде. Суть модификации состоит в следующем:

$$\begin{array}{ll} \parallel (\lambda.\Psi_1)\overline{u} \parallel (fb) & = \parallel \Psi_1 \parallel_f [b, (u \circ f)b], \\ \parallel (\lambda.\Psi_2)\overline{v} \parallel (fb) & = \parallel \Psi_2 \parallel_f [b, (v \circ f)b], \\ u \circ f \in \mathbf{U}_f, \; v \circ f \in \mathbf{V}_f. & \end{array}$$

Тогда $\phi_f \subseteq [\mathbf{U}_f, \mathbf{V}_f]$ $u < u, v > \circ f \in \phi_f$ *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* $\parallel \Phi \parallel_f$. *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 $< u, v > \circ f$*, которые кладутся в основу экстенсионала концепта* $\phi'_f(b)$*; для него справедливо включение*

$$\phi'_f(b) \; ISA \; \mathbf{T} \times \mathcal{T};$$

5) все отобранные пары $[b, (< u, v > \circ f)b]$ *идентифицируют индивиды* $< u, v > \circ f$ *в мире* b *из совокупности* B *и принимаются за экстенсионал переменного концепта* $\phi_f(B) = \phi_f$*;*
6) сохраняется естественное включение

$$\phi_f(B) \subseteq \mathbf{U}_f(B) \times \mathbf{V}_f(B) \subseteq H_T(B) \times H_{\mathcal{T}}(B)$$

для переменного домена $H_T(B) \times H_{\mathcal{T}}(B)$, *задаваемого определением*

$$H_T(B) \times H_{\mathcal{T}}(B) \; = \; \{\overline{h} \mid \overline{h} : (B \to \mathbf{T}) \times (B \to \mathcal{T})\}.$$

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. *Ламбда-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] Вольфенгаген В.Э. *Вычислительная модель реляционного исчисления, ориентированного на представление знаний.* М.: препринт МИФИ, 004 - 84, 1984.

[6] Вольфенгаген В.Э., Яцук В. Я. *Вычислительная модель реляционной алгебры.* - Программирование, No 5, М.: АН СССР, 1985. с. 64-76.

[7] 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.

[8] 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.

[9] Вольфенгаген В.Э., Яцук В.Я. *Аппликативные вычислительные системы и концептуальный метод проектирования систем знания.* МО СССР, 1987.

[10] Вольфенгаген В.Э. *Теория вычислений.* М:МИФИ, 1993.-96 с.

[11] Вольфенгаген В.Э. *Категориальная абстрактная машина.* М:МИФИ, 1993. -96 с.

[12] Вольфенгаген В.Э. *Проектирование языков программирования и теория вычислений.* М:МИФИ, 1993.-189 с.

[13] Голдблатт Р. *Топосы: категорный анализ логики.* М.:Мир, 1985.

[14] Джонстон П. Т. *Теория топосов.* М.: Наука, 1986.

[15] Захарьящев М.В., Янов Ю.И. (ред.) *Математическая логика в программировании.* М.:Мир, 1991.- 408 с.

[16] Илюхин А.А., Исмаилова Л.Ю., Шаргатова З.И. *Экспертные системы на реляционной основе.* М.: МИФИ, 1990.

[17] Карри Х.Б. *Основания математической логики.*- М.: Мир, 1969.

[18] Клини С.К. *Введение в метаматематику.* М.: ИЛ, 1957.

[19] Кузин Л.Т. *Основы кибернетики, т.2.* М: Энергия 1979, 15-9.

[20] Кузичев А.С. *Некоторые свойства комбинаторов Шейнфинкеля-Карри.* Комбинаторный анализ, вып. 1. Изд-во МГУ, 1971.

[21] Кузичев А.С. *Дедуктивно-комбинаторное построение теории функциональностей.* ДАН СССР, 1973, т. 209, N 3.

[22] Кузичев А.С. *Непротиворечивые расширения чистой комбинаторной логики.* Вестн. Моск. ун-та, матем., механ., No 3, 76-81, 1973.

[23] Кузичев А.С. *О предмете и методах комбинаторной логики.* История и методология естественных наук, М.:МГУ, вып.14, 1973, с. 131-141.

[24] Кузичев А.С. *Система ламбда-конверсии с дедуктивным оператором формальной импликации.* ДАН СССР, 212, N6, 1973, 1290-1292.

[25] Кузичев А.С. *Дедуктивные операторы комбинаторной логики.* Вестн. Моск. ун-та, матем., механ., No 3, 13-21, 1974.

[26] Кузичев А.С. *О выразительных возможностях дедуктивных систем лямбда-конверсии и комбинаторной логики.* Вестн. Моск. ун-та, матем., механ., No 6, 19-26, 1974.

[27] Кузичев А.С. *Принцип комбинаторной полноты в математической логике.* История и методология естественных наук, сб. МГУ, вып. 16, 1974. с. 106-127.

[28] Кузичев А.С. *Комбинаторно полные системы с операторами* Ξ, F, Q, $\Pi, \exists$, P, $\neg$, $\&$, $\vee$, $=$. Вестн. Моск. ун-та, сер. матем., мех., N6, 1976.

[29] Кузичев А.С. *Операция подстановки в системах с неограниченным принципом комбинаторной полноты.* Вестн. Моск. ун-та, сер. матем., мех., N5, 1976.

[30] Мальцев А.И. *Алгоритмы и рекурсивные функции.* М.:Наука, 1965.

[31] Марков А.А. *Невозможность некоторых алгорифмов в теории ассоциативных систем.* ДАН СССР, 1947, т. 55, N 7; т. 58, N 3.

[32] Марков А.А. *О логике конструктивной математики.* Вестн. Моск. ун-та, матем., механ., No 2, 7-29, 1970.

[33] Марков А.А. *О логике конструктивной математики.* М.: Знание, 1972.

[34] Мендельсон Э. *Введение в математическую логику.* М.: Наука, 1971.

[35] Скотт Д.С. *Советы по модальной логике.* Семантика модальных и интенсиональных 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).

[36] *Семантика модальных и интенсиональных логик.* Под ред. Смирнова В.А. М.: Прогресс, 1981. - 424 с.

[37] Смирнов В.А., Карпенко А.С., Сидоренко Е.А. (ред.) *Модальные и интенсиональные логики и их применение к проблемам методологии науки.* М.:Наука, 1984.- 368 с.

[38] Стогний А.А., Вольфенгаген В.Э., Кушниров В.А., Саркисян В.И., Араксян В.В., Шитиков А.В. *Проектирование интегрированных баз данных.* Киев: Техніка, 1987.

[39] Такеути Г. *Теория доказательств.* М.:Мир, 1978.

[40] Хендерсон П. *Функциональное программирование. Применение и реализация.* М.: Мир, 1983.

[41] Шабунин Л.В. *О непротиворечивости некоторых исчислений комбинаторной логики.* Вестн. Моск. ун-та, сер. матем. мех., 1971, N 6.

[42] Шенфилд Дж. *Математическая логика.* М.:Наука, 1975.

[43] Энгелер Э. *Метаматематика элементарной математики.* М.: Мир, 1986.

[44] Яновская С.А. *Основания математики и математическая логика.* Математика в СССР за тридцать лет. 1917-1947.- М.-Л.: Гостехиздат, 1948.

[45] *Nested relations and complex objects in databases.* LNCS, No 361, 1989.

[46] 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, No 8, p. 614-641.

[47] 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.

[48] Backus J.W., Williams J.H., Wimmers E.L. *FL language manual (preliminary version).* IBM research report No RJ 5339(54809), 1987.

[49] Banerji R.B. (ed.) *Formal techniques in artificial intelligence: a sourcebook.* Studies in computer science and artificial intelligence, 6, North-Holland, 1990.

[50] Beery G., Levy J-J. *Minimal and optimal computations of recursive programs.* J. Assoc. Comp. Machinery, Vol.26, No 1, 1979.

[51] 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.

[52] Belnap N.D. (Jr.) *How a computer should think.* Contemporary aspects of philosophy, Proceedings of the Oxford International Symposium, 1976.

[53] Bird R.S. *An introduction to the theory of lists.* Logic programming and calculi of descrete design (ed. Broy M.), Springer-Verlag, 1986, pp.5-42.

[54] 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.

[55] Bunder M.V.W. *Set Theory based on Combinatory Logic.* Doctoral thesis, University of Amsterdam, 1969.

[56] Bunder M.V.W. *Propositional and predicate calculuses based on combinatory logic.* Notre Dame Journal of Formal Logic, Vol. XV, 1974, pp. 25-34.

[57] 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.

[58] Church A. *The calculi of lambda-conversion.* Princeton. 1941, ed. 2, 1951.

[59] Coppo M., Dezani M., Longo G. *Applicative information systems.* LNCS, 159, 1983, pp. 35-64.

[60] Cousineau G., Curien P.-L., Mauny M. *The categorical abstract machine.* LNCS, 201, Functional programming languages computer architecture.- 1985, pp. 50-64.

[61] Curien P.-L. *Categorical combinatory logic.* LNCS, 194, 1985, pp. 139-151.

[62] Curien P.-L. *Typed categorical combinatory logic.* LNCS, 194, 1985, pp. 130-139.

[63] Curry H.B., Feys R. *Combinatory logic.* Vol. 1. Amsterdam: North-Holland, 1958.

[64] Curry H.B., Hindley J.R., Seldin J.P. *Combinatory logic.* Vol. II. Amsterdam, 1972.

[65] 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.

[66] Danforth S., Tomlison C. *Type theories and object oriented programming.* ACM Computing Surveys, 1988, v.20, N 1, pp.29-72.

[67] Darlington J., Henderson P., Turner D.A. (eds.) *Functional programming and its applications.* Cambridge Univ. Press, Cambridge, 1982.

[68] de Bruijn N.G. *Lambda-calculus notations with nameless dummies: a tool for automatic formula manipulation.* Indag. Math. 1972, N 34, pp. 381-392.

[69] Eisenbach S. (ed.) *Functional programming: languages, tools and architectures.* Chichester: Horwood, 1987.

[70] Fasel J.H., Keller R.M. (eds.) *Graph reduction.* LNCS, 279, 1986.

[71] Fenstad J.E. et al. *Situations, language and logic.* Dordrecht: D. Reidel Publ. Comp., 1987.

[72] Fitting M. *First-order logic and automated theorem proving.* Springer-Verlag, 1990.

[73] Frandsen G.S., Sturtivant C. *What is an efficient implementation of the lambda-calculus?* LNCS, 523, 1991, pp. 289-312.

[74] Gardenfors P. *Induction, Conceptual spaces and AI.* Proceedings of the workshop on inductive reasoning, Riso National Lab, Rpskilde, 1987.

[75] Henson M.S. *Elements of functional languages.* Exford:Blackwell, 1987.

[76] Hindley J.R. *The principial type-scheme of an object in combinatory logic.* Trans. Amer. Math. Soc., 1969, vol. 146.

[77] Hindley J., Lercher H., Seldin J. *Introduction to combinatory logic.* Cambridge Press, 1972.

[78] Howard W. *The folmulas-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.

[79] 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.

[80] Hughes R.J.M. *The design and implementation of programming languages.* PhD Thesis, University of Oxford, 1984.

[81] Hunt L.S. *A Hope to FLIC translator with strictness analysis.* MSc dissertation, Department of Computing, Imperial College, University of London, 1986.

[82] Kelly P.H.J. *Functional languages for loosely-coupled multiprocessors.* PhD Thesis, Imperial College, University of London, 1987.

[83] 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.

[84] Michaelson G. *An introduction to functional programming through lambda-calculus.* Addison Wesley Publ.Co, 1989, 320 p.

[85] Mycroft A. *Abstract interpretation and optimizing transformations for applicative programmes.* PhD Thesis, Department of Computer Science, University of Edinburgh, 1981.

[86] Peyton Jones S.L. *The implementation of functional programming languages.* Prentice Hall Int., 1987.

[87] Rosser J.B. *A mathematical logic without variables.* Ann. of Math., vol. 36, No 2, 1935. p. 127-150.

[88] Schönfinkel M. *Uber die Baustein der mathematischen Logik.* Math. Annalen, vol. 92, 1924, p. 305-316.

[89] Schroeder-Heister P. *Extentions of logic programming.* LNAI, 475, 1991.

[90] Scott D.S. *Advice on modal logic.* Philosophical problems in logic. Some recent developments.- Lambert K. (ed.), Dordrecht; Holland: Reidel, 1970.

[91] Scott D.S. *Outline of a mathematical theory of computation.* Proceedings of the 4-th Annual Princeton conference on information sciences and systems, 1970.

[92] 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.

[93] Scott D.S. *Identity and existence in intuitionistic logic.* In: Applications of Sheaves. Berlin: Springer, 1979, p.660-696.

[94] 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.

[95] 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.

[96] Scott D.S. *Lectures on a mathematical theory of computation.* Oxford University Computing Laboratory Technical Monograph PRG-19, 1981.-148 p.

[97] Scott D.S. *Domains for denotational semantics.* LNCS, 140, 1982, pp. 577- 613.

[98] Stoy J.E. *Denotational semantics: The Scott-Strachey approach to programming language theory.* M.I.T. Press, Cambridge, Mass., 1977.- xxx+414 p.

[99] Stoye W.R. *The implementation of functional languages using custom hardware.* PhD Thesis, University of Cambridge, 1985.

[100] Szabo M.E. *Algebra of proofs.* Studies in Logic foundations of mathematics, v. 88. North-Holland Publ. Co, 1978.-297 p.

[101] Talcott C. *Rum: An intensional theory of function and control abstractions.* Foundations of logic and functional programming, LNCS, 306, 1986, pp. 3-44.

[102] Tello E.R. *Object-Oriented Programming for Windows/Covers Windows 3.x.* Wiley and Sons, Inc., 1991.

[103] Turner D.A. *A New Implementation Technique for Applicative Languages.* Software Practice and Experience.- No 9, 1979, p.31-49.

[104] Turner D.A. *Aspects of the implementation of programming languages.* PhD Thesis, University of Oxford, 1981.

[105] Turner R. *A theory of properties.* J. Symbolic logic, v. 52, 1987, pp. 455-472.

[106] Wodsworth C.P. *Semantics and pragmatics of the lambda calculus.* PhD Thesis, University of Oxford, 1981.

[107] Wolfengagen V.E. *Frame theory and computations.* Computers and artificial intelligence. V.3, No 1, 1984, pp.1-31.

Предметный указатель

Глоссарий

Алгебра. Алгеброй часто считают систему, вовсе не использующую связанных переменных, то есть все используемые переменные являются свободными.

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, [17]): (1) пропозициональные функции рассматриваются экстенсионально (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.

Высказывание. Высказывание определено тем способом, относительно которого можно рассматривать его доказательство.

Дефинициальное (определительное) 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 многооднозначное соответствие между элементарными высказываниями теории и определенными содержательными высказываниями, относящимися к этой содержательной области.

Интерпретация адекватная. Адекватная, или относительно полная интерпретация каждому содержательному высказыванию (интерпретанту из содержательной 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, ... и *стрелки* f, g, ... Каждой стрелке f соответствует объект X, называемый *доменом* и объект Y, называемый *кодоменом*. Этот факт записывается в виде $f : X \to Y$. Дополнительно накладываются ограничения на использование композиции -- с учетом единичного (тождественного) отображения. Стрелки рассматриваются как представления отображений. Более строго, если g -- произвольная стрелка $g : Y \to Z$ с доменом Y, который совпадает с кодоменом f, то имеется стрелка $g \circ f : X \to Z$, называемая *композицией* g с f. Для каждого объекта Y существует стрелка $1 = 1_Y : Y \to Y$, называемая тождественной стрелкой для Y. Предполагается выполнение аксиом тождества и ассоциативности для всех стрелок $h : Z \to W$:
$1_Y \circ f = f,\ g \circ 1_Y = g,\ h \circ (g \circ f) = (h \circ g) \circ f : X \to W.$

Класс. Понятие класса считается интуитивно ясным. Классы объектов 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. Это концептуальный класс, порожденный из определенных исходных элементов посредством выделенных способов комбинации.

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 выразима в терминах обычных операций, то есть *без использования формальных переменных* (параметров).

λ-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.

Об-система. Формальные объекты об-системы образуют индуктивный класс. Элементы этого индуктивного 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 представлет собой частный вид категории.

Объект. Объектом считается математическая сущность, которая используется в теории. Объект -- это математическое *представление* реального объекта предметной области (“внешнего мира”).

Объектно-ориентированное программирование. Объектно-ориентированное программирование (ООП) -- это способ программирования, обеспечивающий 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. 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 определет один и тот же тип B, то вместо “$(\forall x \in A)B(x)$” используется сокращенная форма записи $A \to B$. Последняя запись принимается за тип функций из A в B.

Оценка. Оценкой считается 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 экстенсионально определяется как совокупность кортежей (n-ок). В зависимости от числа элементов в кортеже произведение наделяется арностью.

Реляционная система. Это система с единственным базисным предикатом, который является бинарным отношением.

Свойство. Свойством считается пропозициональная функция, определенная на (произвольном) типе A.

Суффикс. Суффиксом считается функтор, который пишется после аргументов.

Теория. Теорией считают способ выбора подкласса истинных высказываний из числа высказываний, принадлежащих классу всех высказываний $\mathcal{A}$.

Теория дедуктивная. Теория $\mathcal{T}$ считается дедуктивной, если $\mathcal{T}$ является индуктивным классом (элементарных) высказываний.

Теория непротиворечивая. Непротиворечивая теория определяется как такая теория, которая не охватывает всего класса $\mathcal{A}$ высказываний.

Теория полная. Полной считается такая дедуктивная теория $\mathcal{T}$, что присоединение к ее аксиомам элементарного высказывания, не являющегося элементарной теоремой, при сохранении правил неизменными делает ее противоречивой.

Теория полная в смысле Поста. $\mathcal{T}$ полна, если каждое высказывание из класса $\mathcal{A}$ высказываний является следствием относительно $\mathcal{T}$ любого высказывания X, не входящего в $\mathcal{T}$.

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.

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 (И-язык). $\mathcal{И}$-язык характеризуется: (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) $\mathcal{И}$-язык по необходимости неясен, однако пользуясь им можно достичь любой разумной степени точности.

Вячеслав Эрнстович **Вольфенгаген**

Комбинаторная логика в программировании
(Вычисления с объектами в примерах и задачах)

Редактор *Н. В. Шумакова*
Технический редактор *Е. Н. Кочубей*

Тем. план 1994 г., по письму

Лицензия ЛР-020676 от 09.12.1992 г.

Подписано в печать	С-028-1	Формат 60 × 84 1/16
Печ. л. 12,75 Уч.-изд. л. 12,75	Тираж 350 экз.	Заказ

Оригинал-макет подготовлен с помощью LaTeX
Московский государственный
инженерно-физический институт (технический университет),
Типография МИФИ.
115409, Москва, Каширское шоссе, 31