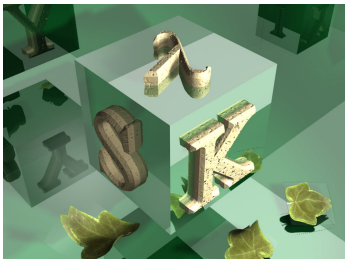
В. Э. Вольфенгаген
Л. Ю. Исмаилова
С. В. Косиков

Модели вычислений

Конспект лекций



Серия: Компьютерные науки и информационные технологии

Проект: *Аппликативные Вычислительные Системы*

В. Э. Вольфенгаген, Л. Ю. Исмаилова, С. В. Косиков

МОДЕЛИ ВЫЧИСЛЕНИЙ

Конспект лекций

Москва
МИФИ
2007

ББК 32.97
УДК 004
В721

Авторы: д. т. н., профессор **Вольфенгаген В. Э.**, к. т. н., в. н. с. **Исмаилова Л. Ю.**, с. н. с. **Косиков С. В.**,

Модели вычислений. Конспект лекций— М.: МИФИ, 2007.— IX+306 с.

Изложен основной круг задач, сводимых к исчислению объектов — “от простого к сложному”. Конкретный вариант исчисления выбирается в зависимости от решаемых вычислительных задач. В ходе последовательного решения задач читатель овладевает основными методами и средствами комбинаторной логики и λ-исчисления. Все задачи снабжены подробными и элементарными решениями.

Для студентов старших курсов и аспирантов, изучающих математические основы объектно-ориентированных вычислений, начинающих и профессионально работающих над продвинутыми проектами программистов. Может быть использована в курсах дискретной математики, информатики, теории программирования. Предварительной математической подготовки не требуется. Материал частично или полностью может быть использован для самостоятельного изучения в варианте “для первого чтения”.

E-mail: vew@jmsuice.msk.ru

Содержание

Круг вопросов

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

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

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

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

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

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

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

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

Аудитория, проблематика, постановка курса

Кому адресована книга

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

Зачем нужно исчислять объекты

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

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

Основное — адекватный способ мышления

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

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

к программированию, которые еще только зарождаются в исследовательских лабораториях.

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

Рассуждения в терминах объектов. Точно также лишь немногие и сравнительно молодые математические теории ориентированы на рассуждения в терминах *объектов*, а не в терминах *операторов*, как это следует из опыта изучения математического анализа в большинстве университетов, в том числе, и технического или компьютерного профиля. К сожалению, программисту не 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*, а также некоторые другие чрезвычайно плодотворные подходы к программированию. В частности, только по-настоящему проникнув в сам дух комбинаторной логики, можно понять в деталях и практически применить систему программирования с заранее нефиксированной системой инструкций.

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

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

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

Объекты и системный подход

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Вызывает восхищение та тщательность, трудолюбие и изобретательность, с которыми подошел К.А. Сагоян к подготовке и проверке решений всех примеров, приводимых в разделе 4, ко-

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

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

Большой труд по подбору задач для раздела 7 взяла на себя И.А. Горюнова. Проявленный ею энтузиазм и глубокое понимание механизма динамического формирования объектов позволили подобрать и подготовить упражнения, снабженные решениями.

Ряд ценных комментариев, касающихся раздела 5, был дан А.Г. Пантелеевым.

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

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

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

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

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

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

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

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 вычислений с объектами, име-

[1] *Более строго: системы нумералов*

ющих действительно принципиальное значение.

• Синтез нового объекта

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

• Характеристики комбинатора неподвижной точки

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

• Применение принципа 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 (арифметические объекты со своими правилами) встраивается в программную среду (комбинаторную логику).

• Исследование свойств комбинаторов с типами

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

• Разложение термов в базисе I, K, S

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

алгоритма разложения в базисе, который вполне аналогичен процессу компиляции.

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

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

• **Выражение определения функции с помощью оператора неподвижной точки** Y

Рассматривается случай рекурсивных определений объектов. Пользуясь фундаментальной для функционального программирования теоремой о неподвижной точке, рекурсивные определения 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 вглубь математических абстракций и будем увязывать операторный стиль мышления с комбинаторным. Отметим, что в комбинаторной логике используется единственный оператор — *оператор аппликации*, или оператор

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

• Построение отображения, каррирующего n-местную функцию

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

• Вывод основных свойств оболочки Каруби

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

• Декартово произведение и проекции: погружение в ABC

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

• Представление Lisp средствами λ-исчисления или комбинаторной логики

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

• Реализация вычисления значений выражений с помощью суперкомбинаторов

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

• Полностью ленивая реализация суперкомбинаторов

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

• Оптимизация вычислений путем перестановки параметров

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

• Реализация непосредственных вычислений выражений языков программирования

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

• Вычисление значения кода де Брейна

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

• Реализация машинных инструкций категориальной абстрактной машины (КАМ)

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

• **Возможности оптимизации при вычислении на КАМ**

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

• **Переменные объекты**

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

Краткие рекомендации по порядку изучения книги

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

К этому предварительному базису можно добавить разделы 6.5, 6–6.2, где рассматриваются нумералы, рекурсия, разложения в базисах.

Затем можно прочитать разделы 5–11.2, 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 12—15 потребует обратиться к источникам, рекомендованным в библиографии. Разделы 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 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.

Часть I

Объекты, функции, абстракции

Содержание

Тема 1

Вычисление значения

Содержание

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

Кристофер Стрейчи

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

1.1 Предварительные сведения

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

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

Комбинаторная логика. В ходе разработки этого раздела было показано, что связанные переменные можно исключить без потери выразительных возможностей формальной системы. Наиболее

важные применения комбинаторная логика нашла в построении оснований математики, а также в развитии методов и средств реализации языков программирования.

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

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

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

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

На первый взгляд круг рассматриваемых в этих разделах идей не является однородным, вызывая рост числа внешне различных формализаций и математических аппаратов. На деле происходит

даже большее: каждое очередное исследование, как правило, содержит построение своего собственного математического аппарата. Очень часто применяемые автором основные математические идеи оказываются завуалированными громоздкими выкладками.

Изучение названных разделов помогает устанавливать внутреннее единство разнородных исследований, а их систематическое применение обеспечивает исследователя или разработчика концептуально ясными и вместе с тем гибкими и мощными теоретическими средствами.

1.2 Круг основных идей

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

С синтаксической точки зрения объекты выделяются либо ис-

пользованием специальной скобочной записи, либо принятием соглашения об опускании скобок, когда их при желании можно однозначно восстановить. Другая важная отличительная особенность заключается в акцентировании свойства *базисности* некоторых заранее выделенных объектов. Всякий раз оказывается, что вновь вводимый объект может быть выражен путем *комбинирования* исходных объектов, которые в силу этого обстоятельства вполне уместно назвать *комбинаторами*. Такую разложимость произвольно введенного объекта в базисе трудно переоценить: вместо исследования свойств громоздкой прикладной теории с большим числом самых разных объектов можно заняться исследованием свойств всего нескольких объектов, нисколько не теряя при этом общности получаемого результата.

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

1.3 Структура раздела

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

Данный раздел следует воспринимать как некоторое подобие меню, в котором обозначены основные вопросы, взаимодействие

которых предстоит сначала увидеть непосредственно, а потом при детальном изучении выявить его более глубокую сущность.

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

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

1.4 Состояние исследований

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

Пионерское для computer science исследование выполнено Дж. Маккарти в (J. McCarthy, [102]). Им был разработан язык обработки списков Lisp, являющийся вполне объектно-ориентированным и к тому же функциональным языком программирования. В течение ряда лет Lisp был одной из самых популярных

Таблица 1.1: Варианты задач

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

систем программирования в области искусственного интеллекта, заслужив название ‘ассемблер знаний’. Отметим, что Lisp является компьютерной реализацией λ-исчисления, предоставляя программисту практически все средства этой мощной математической теории. Эффективная реализация Lisp-интерпретатора, выполненная А.Г. Пантелеевым (А.Г. Пантелеев, [41]; М.А. Булкин, Ю.Р. Габович, А.Г. Пантелеев, [5]), вскрыла множество деталей, касающихся реализации объектных и функциональных языков, а также путей и методов их эффективного использования. Методы реализации операций связывания переменных, структур данных, процедур, функций и механизмов рекурсии, установленные в ходе работ над этим проектом, стимулировали целое направление работ в области программирования.

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

Различную методическую помощь при решении задач, иллюстрирующих приложения ABC, можно почерпнуть из работ (А.А. Стогний, В.Э. Вольфенгаген, В.А. Кушниров, В.И. Саркисян, В.В. Араксян, А.В. Шитиков, [47]), (В.Э. Вольфенгаген, [6]), (В.Э. Вольфенгаген, В.Я. Яцук, [7]), (В.Э. Вольфенгаген, К.А. Сагоян, [8]), (В.Э. Вольфенгаген, К.Е. Аксенов, Л.Ю. Исмаилова, Т.В. Волша-

ник, [9]), (А.А. Илюхин, Л.Ю. Исмаилова, З.И. Шаргатова, [21]), (К.Е. Аксенов, О.Т. Баловнев, В.Э. Вольфенгаген, О.В. Воскресенская, А.В. Ганночка, М.Ю. Чуприков, [1]).

Исследование (В.Э. Вольфенгаген, В.Я. Яцук, [10]) отражает раннее состояния работ и содержит подробные построения различных вариантов прикладных аппликативных систем, имеющих значение для решения прикладных задач, направленных на разработку информационных систем.

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

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

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

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

ко не полный перечень направлений исследований, инициированных этим математиком.

Введение в математическую проблематику комбинаторной логики и λ-исчисления можно найти в ряде статей А.С. Кузичева (А.С. Кузичев, [27],[28], [29], [30],[31], [32], [33], [34],[35]). В этих работах выполнено исследование выразительных возможностей систем с операторами аппликации и (функциональной) абстракции. Элементарные основы комбинаторной логики подробно изложены в (J. Hindley, H. Lercher, J. Seldin, [94]).

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

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

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

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

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

1.5 Типовая задача

Общие указания к решению типовой задачи.

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

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

пользуясь постулатами α, β, μ, ν, σ, τ, ξ исчисления λ-конверсии.

Решение.

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

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

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

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

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

$$\begin{aligned} v\,(Kxy) &= v\,x, && (K)\\ v\,(Sxyz) &= v\,(xz(yz)), && (S) \end{aligned}$$

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

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

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

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

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

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

$I{-}2$. Применяя аналогичным образом схему (K), заключаем, что $Ka(Ka) = a$.

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

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

1.6 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[1] (доказать!):

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

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

Указания.

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

Воспользоваться равенствами:

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

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

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

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

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

6-1° исследовать разложение термов в базисе I, K, S;

6-2° исследовать разложение термов в базисе I, B, C, S;

6-3° выразить определение функций, пользуясь комбинатором неподвижной точки Y;

6-4° исследовать свойства функции:

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

6-5° установить изоморфизм между декартово замкнутой категорией (д.з.к.) и аппликативной вычислительной системой (ABC);

6-6° получить отображение, которое соответствует отображению каррирования функций (для n-местной функции);

6-7° проделать вывод основных свойств оболочки Каруби;

6-8° закодировать термами бестипового λ-исчисления декартово произведение n объектов ($n \geq 5$) и получить соответствующие выражения для проекций;

6-9° представить основные функции аппликативного языка программирования Lisp средствами λ-исчисления и комбинаторной логики.

Формулировки задач для соответствующих целевых исследований в *задании 6* на стр. 40.

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

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

Исключить все переменные из приводимых ниже λ-выражений:

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

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

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

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

Исключить все переменные из приводимых ниже лямбда-выражений:

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

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

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

Для приведенных примеров выполнить детальную проверку вычислений.

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

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

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

6-5° Вывести следующие равенства:

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

где:

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

6-6° Рассматривая семейство функций h:

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

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.

6-7° Оболочкой Каруби называют категорию, которая содержит

для $a \circ b = \lambda x.a(bx)$:

множества

$$\begin{array}{rl} \text{объектов:} & \{a \mid a \circ a = a\}, \\ \text{морфизмов:} & Hom(a,b) = \{f \mid b \circ f \circ a = f\}, \end{array}$$

и *морфизмы*

$$\begin{array}{rl} \text{тождества:} & id\ a = a, \\ \text{композиции:} & f \circ g. \end{array}$$

Пусть

$$\begin{array}{r} [x,y] \equiv \lambda r.rxy, \\ < f,g > \equiv \lambda t.[f(t),g(t)] \equiv \lambda t.\lambda z.z(ft)(gt). \end{array}$$

Проверить, что:

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

где

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

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

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

(**Указание**. Для случая n=2:

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

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

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

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

(1) $A \frown (B \frown C) = (A \frown B) \frown C$

(2) $A \frown <> = <> \frown A = A$

(3) $$Null\ A = \begin{cases} 1, & \text{если } A = Nil, \\ 0, & \text{если } A \neq Nil, \end{cases}$$

(4) $List\ x = < x >$,

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

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

1.7 Рекомендуемый порядок выполнения задания

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

Тема 2

Объекты и вычисления с объектами

Содержание

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

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

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

2.1 Принцип комбинаторной полноты

2.1.1 Комбинаторная характеристика

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

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

$$Ux_1 \ldots x_n = X. \qquad (U)$$

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

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

> *синтезируется* "программа", или "процедура" U, вызов которой на *фактических* параметрах $x_1, \ldots, x_n$ дает требуемый *результат* X – а это комбинация, составленная из переменных $x_1, \ldots, x_n$.

Поскольку имеем дело с аппликативной вычислительной системой, то сделаем замечание, касающееся соглашения о применяемых обозначениях. Примем по соглашению, что

$$U\,x_1x_2\ldots x_n \equiv \underbrace{((\ldots((}_{n\text{ штук}}U\,x_1)x_2)\ldots)x_n),$$

то есть опущенные скобки восстанавливаются *по ассоциации влево*.

Пример 2.1.

$$Ux_1x_2x_3 \equiv (((Ux_1)x_2)x_3).$$

2.1.2 Системы концептов

Комбинаторная логика в бестиповом варианте является основным математическим аппаратом, в рамках которого исчисляются объекты, абстрактные по самой своей сути. Фактически, комбинаторная логика представляет собой чистое исчисление *концептов*, позволяя по мере необходимости создавать или модифицировать "на лету" свою собственную систему концептов. Развитие систем "переменных" концептов имеет первостепенную роль во всех значимых 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: сорокалетний опыт практического использования системы программирования Lisp давно уже подготовил почву для переосмысления "теоретического минимума программиста", куда входят и комбинаторная логика, и λ-исчисление, и другие специальные разделы логики, по традиции называемой неклассической.

2.1.3 Комбинаторная полнота

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

Принцип комбинаторной полноты

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

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

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

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

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

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

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

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

где выражение ‘$[b_1, \ldots, b_n/x_1, \ldots, x_n]a$’ обозначает результат одновременной подстановки объектов $b_1, \ldots, b_n$ в объект a вместо соответствующих вхождений графически различных переменных $x_1, \ldots, x_n$ для $n > 0$.

Содержательная интерпретация выглядит следующим образом:

> “процедура” U с *формальными*, или *подстановочными* параметрами
>
> $$x_1, \ldots, x_n$$
>
> вызывается на *фактических* параметрах $b_1 \ldots b_n$ и дает требуемый *результат*
>
> $$[b_1, \ldots, b_n/x_1, \ldots, x_n]a.$$

2.1.4 Элементарная комбинаторная логика

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

Введение понятия комбинатором

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

Рассмотрим коммутативный закон сложения в арифметике:

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

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

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

и введя метаоператор $\mathbf{C}$:

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

Этот закон примет вид:

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

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

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

Упражнение 2.1. Записать коммутативный закон умножения без использования переменных.

Простейшие комбинаторы

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

Тождество. Простейшим комбинатором является комбинатор *тождества* I:

$$If = f.$$

Композитор. Элементарный *композитор* :

$$Bfgx = f(gx)$$

выражает композицию двух функций f и g.

Дупликатор. Элементарный *дупликатор* W:

$$Wfx = fxx$$

дублирует второй аргумент.

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

$$Cfxy = fyx.$$

Коннектор. Элементарный *коннектор* S определяется правилом:

$$Sfgx = fx(gx).$$

Канцелятор Элементарный *канцелятор*

$$Kcx = c$$

выражает константу (константную функцию) как функцию от x.

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

$$Bfgx = B\ sin\ exp^5\ x = sin(exp^5\ x) = sin\ x^5,$$
Bgf — пятая степень синуса,
$$Bgfx = B\ exp^5\ sin\ x = exp^5(sin\ x) = sin^5 x.$$

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

$$WBQ\ x \overset{W}{=} BQQ\ x \overset{B}{=} Q(Q\ x) = x^4.$$

Пример 2.4. Поскольку выполняется равенство (конверсия):

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

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

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

2.2 Синтез основных комбинаторов: задачи

Теперь сосредоточим свое внимание на выработке технического навыка установления и исследования свойств нового концепта. В качестве таких концептов избираем различные комбинаторы, широко используемые в математической практике. Общая постановка задачи состоит в синтезе концепта-комбинатора по заранее заданной *комбинаторной характеристике* (см. стр. 49).

Задача 2.1. Вывод выражения для комбинатора B .

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

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

пользуясь постулатами α, β, μ, ν, σ, τ, ξ исчисления λ-конверсии.

Решение.

B–1. Сформулируем постулаты, задающие отношение конвертируемости ‘=’ :

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

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

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

B–2. Определим комбинаторные характеристики объектов K и S:

$$\begin{array}{lcll} x(Kyz) & = & xy, & (K)\\ x(Syzw) & = & x(yw(zw)), & (S)\end{array}$$

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

Действительно, по постулату (β):

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

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

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

Проверка. Проверим, что $B = S(KS)K$.

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

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

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

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

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

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

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

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

Задача 2.2. Вывод выражения для комбинатора C.

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

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

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

Решение.

$C-1$. Сформулируем постулаты, задающие отношение конвертируемости ‘$=$’ (см. предыдущую задачу).

$C-2$. Напомним комбинаторные характеристики, возможно, используемых объектов:

$$\begin{array}{llcl@{\qquad}llcl} (K) & Kxy & = & x, & (S) & Sxyz & = & xz(yz), \\ (I) & Ix & = & x, & (B) & Bxyz & = & x(yz). \end{array}$$

Отметим, что к до сих пор известным схемам (K), (S) и (I) добавлена схема (B), полученная в результате решения задачи 2.1 на стр. 54. Как было показано, эта последняя схема выразима в терминах схем (K) и (S).

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

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

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 характеристики используемых объектов:

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

$W-2$. Применим эти схемы к abb:

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

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

$W-3$. 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.

$\Psi-1$. 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:

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

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

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

Задача 2.5. Вывод выражения для комбинатора B^2.

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

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

Решение.

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

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

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

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

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:

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

Решение.

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

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

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

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

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:

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

Решение.

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

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

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

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

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

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

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

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

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

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

Решение.

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

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

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

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

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

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

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

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

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

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

Решение.

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

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

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

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

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

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

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

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

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

$$C_{[3]}abcde = aebcd. \qquad (C_{[3]})$$

Решение.

$C_{[3]}$–1. Сформулируем постулаты.

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

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

Применяя эти схемы к $aebcd$, получим:

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

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 Φ.

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

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

Решение.

Φ−1. Сформулируем постулаты, задающие отношение конвертируемости.

Φ−2. Запишем комбинаторные характеристики, возможно, используемых объектов:

$$(B^2)\quad B^2xyzw = x(yzw), \qquad (B)\quad Bxyz = x(yz),$$
$$(S)\quad Sxyz = xz(yz).$$

Φ−3. Применяя эти схемы к $a(bd)(cd)$, получим:

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

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

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

Задача 2.12. Вывод выражения для комбинатора Y.

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

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

Решение.

$Y-1$. Сформулируем 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 $Y = WS(BWB)$.

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

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

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

2.3 Исторические замечания

Чисто хронологически комбинаторная логика как математический аппарат была введена М. Шейнфинкелем (Moses Shönfinkel) в 1920 году. Его работа была опубликована в 1924 году под названием “О строительных блоках математической логики”. В то время

привлекательной казалась задача сведения логики к наипростейшему возможному базису примитивов, хотя в настоящее время это не считается столь уж важным. В этой работе было показано, что в логике можно вовсе избавиться от связанных переменных. Использование функций высших порядков позволило свести логику до языка, содержащего один *конструктор* — апплицирование функции к аргументу, — и три примитивные *константы* — U, C (которая в настоящее время имеет обозначение K) и S. Функция получает название функции ‘высшего порядка’, если ее аргументом может в свою очередь быть функция, либо в результате ее применения получается функция. Все три константы как раз и являются функциями высшего порядка. Приведем их формальные определения.

Константа C, определяемая посредством

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

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

Константа S, определяемая посредством

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

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

Константа U, определяемая посредством

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

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

Замечание. Этих комбинаторов оказывается достаточно, чтобы выразить произвольные предикаты первого порядка без использования связанных переменных, которые возникают в логике первого порядка, как

только речь идет о применении кванторов. Тот же самый результат достигается, когда используются алгоритмы преобразования λ-выражений в комбинаторы. В последнем случае случае снимаются ограничения на первопорядковость применяемых предикатов. Однако, неограниченное применение такого преобразования при рассмотрении объектов с типами может вызвать нарушение требования непротиворечивости. В комбинаторной логике стремятся избавиться от типов и связанных с ними ограничений, чтобы не столкнуться с тяжелыми требованиями сохранения непротиворечивости.

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

Тема 3

Связи между объектами

Содержание

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

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

3.1 Теоретические сведения.

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

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

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

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

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

что устанавливает X как неподвижную точку F.

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

3.1.1 Абстракция

Для каждого терма M и переменной x терм $[x]M$, называемый *абстракцией M по x*, определяется[1] 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.

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

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

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

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

Доказательство. См. в [94].

3.1.2 Мультиабстракция

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

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

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

3.1.3 Локальная рекурсия

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

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

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 $Y \; f$ — это неподвижная точка f.

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

```
E1 where f x = ... g ...
           g y = ... f ...
```

транслируется сначала в выражение:

```
E1 where f = [x] (... g ...)
           g = [y] (... f ...),
```

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

```
E1 where (f,g) = ( [x](... g ...), [y](... f ...) ),
```

которое может быть скомпилировано в выражение:

```
([f,g]E1) (Y ([f,g]([x](... g ...), [y](... f ...)) ))
```

с использованием уже известного правила.

3.2 Основные задачи

Задача 3.1. Исследовать свойства заданного объекта

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

Формулировка задачи. Найти комбинаторную характеристику заданного объекта с помощью постулатов α, β, μ, ν, σ, τ, ξ исчисления λ-конверсии и схем (K), (S):

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

Решение.

Y−1. Вид объекта Y известен заранее:

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

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

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

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

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

Задача 3.2. Исследовать свойства заданного объекта:

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

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

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

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:

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

Таким образом, Ya является неподвижной точкой для a.

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

Задача 3.3. Исследовать свойства заданного объекта:

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

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

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

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:

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

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

Y–4. Применим объект Y к a:

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

Y–5. В итоге получаем $Ya = a(Ya)$.

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

Задача 3.4. Исследовать свойства заданного объекта:

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

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

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

Решение.

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

Y_0–2. Рассмотрим сначала объект (XX).

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

Значит,

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

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

$$\begin{array}{rcll} Y_0a & \equiv & (\lambda f.XX)a & (\text{по опр.}) \\ & = & (\lambda f.f(XX))a & (\text{по } (*)) \\ & = & (\lambda f.f((\lambda x.f(xx))(\lambda x.f(xx))))a & (\text{по опр. } X) \\ & = & a((\lambda x.a(xx))(\lambda x.a(xx))) & (\text{по } \beta) \\ & = & a((\lambda f.((\lambda f.f(xx))(\lambda x.f(xx))))a) & (\text{по } \beta, \xi) \\ & = & a((\lambda f.(XX))a) & (\text{по опр. } X) \\ & \equiv & a(Y_0a). & (\text{по опр. } Y_0) \end{array}$$

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:

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

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

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

Решение.

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

Y_1−2. Имеем $Y_0a = a(Y_0a)$,

$$\begin{array}{rcll} Y_1a & = & Y_0(\lambda y.\lambda f.f(yf))a & (\text{по опр.}) \\ & = & (\lambda y.\lambda f.f(yf))(Y_0(\lambda y.\lambda f.f(yf)))a & (\text{по } (Y_0)) \\ & = & (\lambda yf.f(yf))Y_1a & (\text{по } (Y_1)) \\ & = & a(Y_1a). & (\text{по } \beta) \end{array}$$

Итак, имеем $Y_1a = a(Y_1a)$.

Ответ. Комбинаторная характеристика объекта Y_1 имеет вид: $Y_1a = a(Y_1a)$.

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

Формулировка задачи. Циклический список L, который определяется посредством

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

дает бесконечный периодический список

$$L = 1, 2, 3, 1, 2, 3, 1, 2, 3, \ldots.$$

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

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

L–1. Выполним цепочку преобразований:

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

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

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

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

Упражнения

Упражнение 3.1. Рассмотрим циклическое определение

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

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

Указание. Поскольку, например,

```
s(4,2) = s(3,1) + 2 * s(3,2)
       = 1 + 2 * (s(2,1) + 2 * s(2,2))
       = 1 + 2 * (1 + 2 * 1) = 7,
```

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

Упражнение 3.2. Устранить цикличность в приводимых определениях:

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

факториал n =
        if zero n
        then 1
        else n × факториал (n - 1)
```

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

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

Часть II

Синтаксическая теория вычислений

Содержание

Тема 4

Системы типизации

Содержание

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.

4.1 Представление о типе

Неформальное обсуждение понятия типа иллюстрирует довольно прозрачную идею. Всякая функция имеет область определения и

область значения. Следовательно, не все аргументы представляют интерес, а только те из них, которые принадлежат выделенной области определения. Это и означает, что аргументы как объекты типизированы.

Чистые системы типов

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

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

A — множество *аксиом* вида

$$c : s,$$

где c — константа, а s — сорт;

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

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

$$\text{контекст} \vdash \text{приписывание типа}.$$

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

Построение класса типов

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

$$\delta_1,\ \delta_2,\ \delta_3,\ \ldots\ \delta_n,\ \ldots,$$

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

i) всякий базисный тип считается *типом*;

ii) если a и b — типы, то $(a \to b)$ считается *типом*:

$$\frac{a\ -\ \text{тип},\qquad b\ -\ \text{тип}}{(a \to b)\ -\ \text{тип}}.$$

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

Пример 4.1.

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

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

В аппликативных вычислительных системах в центре внимания *не* области определения функций, а сами функции как общие *законы соответствия*. Фактически, исчисляются именно законы соответствия, или, говоря иными словами, *концепты* функций, то есть в конечном счете объекты. В этом случае для учета типов

применяются более тонкие рассуждения. Действительно, комбинаторы задают функции, функции от функций, функции от функций от функций, ... , то есть возникают функции высших порядков, или функционалы. Вопрос выяснения типа у объекта становится нетривиальным, и типы приходится исчислять, пользуясь точными правилами. Соответствующие области определения становятся в сильной степени взаимозависимы. В соответствующих логических конструкциях можно прийти к противоречиям.

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

$$\text{`}a \to b\text{'}$$

применяются различные обозначения:

$$a\,b, \quad b^a, \quad F\,a\,b$$

и некоторые другие.

4.1.1 Комбинаторные термы

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

$$\text{`тип } a \text{ приписан комбинатору } X\text{'},$$

или

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

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

Схемы аксиом:

$$\vdash \#(I) = (a,a), \qquad (FI)$$
$$\vdash \#(K) = (a,(b,a)) = (a,b,a), \qquad (FK)$$
$$\vdash \#(S) = ((a,(b,c)),((a,b),(a,c))). \qquad (FS)$$

Правило:

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

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

4.1.2 λ-термы

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

Правила:

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

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

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

Отметим, что согласно правилу (F), выполняется приписывание схемы типа *аппликации* (XU) объекта X к объекту U, когда схемы типов этих последних объектов известны. Объект X рассматривается как функция из a в b, а U — как аргумент этой

Таблица 4.1: Список основных комбинаторов.

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

функции, берущийся из области определения a. Тогда значения результата (XU) приложения X к U берутся из области значений b.

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

4.2 Задачи

Предлагается, пользуясь аксиомами и правилом (F), приписать типы основным комбинаторам, которые представлены в таблице 4.1. В ходе решения этих задач предполагается уяснить, что

такое математические функции, как выполнять их композицию и как строить простейшие программы с помощью метода композиции. Каждый комбинатор дает идеализацию программы в виде “черного ящика”. Это значит, что внутренняя структура программы не уточняется, а важно установить ее поведение, ориентируясь только на вход и выход. Комбинации (композиции), составленные 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 4.1. Определить тип объекта B.

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

$$
\begin{aligned}
&Exp\,1 = (a_1, (b_1, a_1))\\
&Exp\,2 = a_1\\
&Exp\,3 = ((b_1, a_1), ((a_2, b_2), (a_2, c_2)))\\
&Exp\,4 = (b_1, a_1)\\
&Exp\,5 = ((a_2, b_2), (a_2, c_2))\\
&Exp\,6 = (a_2, b_2)\\
&Exp\,7 = (a_2, c_2)
\end{aligned}
$$

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

Решение.

$\#(B)-1$. Пусть a, b, c – заданные типы. Поскольку по схеме (FK) имеем $\vdash \#(K) = (a_1, (b_1, a_1))$ и по схеме (FS) :

$\vdash\ \#\,(S) = a_1$, то по правилу (F) : $\vdash\ \#\,(KS) = (b_1, a_1)$, где $a_1 = ((a,(b,c)),((a,b),(a,c)))$.

$\#(B)$–2. Далее по схеме (FS):

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

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

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

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

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

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

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

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

Аналогично определим типы, которые приписываются остальным комбинаторам.

Задача 4.2. Тип #(SB).

Решение.

$\#(SB)$–1. Построение дерева:

$$\begin{array}{l} Exp\,1 = ((a_1,(b_1, c_1)),((a_1, b_1),(a_1, c_1))) \\ Exp\,2 = (a_1,(b_1, c_1)) \\ Exp\,3 = ((a_1, b_1),(a_1, c_1)) \end{array}$$

$$\frac{\vdash \#(S) = Exp\,1 \qquad \vdash \#(B) = Exp\,2}{\vdash \#(SB) = Exp\,3}\ (F)$$

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

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

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

Задача 4.3. 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 $b_1 = b$ (здесь: a, b – заданные типы).

Итак, $\vdash \#(KI) = (b,(a,a))$.

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

Задача 4.4. Тип $\#(Z_1)$.

Решение.

$\#(Z_1)$–1. $Z_1 = SB(KI)$.

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

$\#(Z_1)$–3.
$$\begin{aligned} Exp\,1 &= (((b_1,c_1),(a_1,b_1)),((b_1,c_1),(a_1,c_1))) \\ Exp\,2 &= ((b_1,c_1),(a_1,b_1)) \\ Exp\,3 &= ((b_1,c_1),(a_1,c_1)) \end{aligned}$$

$$\frac{\vdash \#(SB) = Exp\,1 \qquad \vdash \#(KI) = Exp\,2}{\vdash \#(SB(KI)) = Exp\,3}\ (F)$$

$\#(Z_1)$–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 4.5. 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. $Exp\,1 = (((b_1, c_1), (a_1, b_1)), ((b_1, c_1), (a_1, c_1)))$
$Exp\,2 = ((b_1, c_1), (a_1, b_1))$
$Exp\,3 = ((b_1, c_1), (a_1, c_1))$

$$\frac{\vdash \#(SB) = Exp\,1 \qquad \vdash \#(Z_1) = Exp\,2}{\vdash \#(Z_2) = Exp\,3}\ (F)$$

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

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

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. $Exp\,1 = (((b_1,c_1),(a_1,b_1)),((b_1,c_1),(a_1,c_1)))$
$Exp\,2 = ((b_1,c_1),(a_1,b_1))$
$Exp\,3 = ((b_1,c_1),(a_1,c_1))$

$$\frac{\vdash \#(SB) = Exp\,1 \qquad \vdash \#(Z_2) = Exp\,2}{\vdash \#(Z_3) = Exp\,3}\ (F)$$

$\#(Z_n)$–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 4.6. Тип $\#(W)$.

Решение.

$\#(W)$–1. $W = CSI$.

$\#(W)$–2. $(FC) :\vdash \#(C) = ((b,(a,c)),(a,(b,c)))$. Это утверждение будет доказано далее (см. задачу 4.16).

$\#(W)$–3. $Exp\,1 = ((b_1,(a_1,c_1)),(a_1,(b_1,c_1)))$
$Exp\,2 = (b_1,(a_1,c_1))$
$Exp\,3 = (a_1,(b_1,c_1))$
$Exp\,4 = (a_1)$
$Exp\,5 = (b_1,c_1)$

$$\frac{\dfrac{\vdash \#(C) = Exp1 \qquad \vdash \#(S) = Exp\,2}{\vdash \#(CS) = Exp\,3}\ (F) \quad \vdash \#(I) = Exp\,4}{\vdash \#(SCI) = Exp\,5}\ (F)$$

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

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

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

Задача 4.7. Тип $\#(B^2)$.

Решение.

$\#(B^2)$–1. $B^2 = BBB$.

$\#(B^2)$–2. $(B) : ((bc)((ab)(ac)))$.
Здесь и далее предполагается, что запись вида:

$$\text{`}\vdash \#(X) = (a, (b, c))\text{'}$$

аналогична записи:

$$\text{`}(X) : (a(b\, c))\text{'}.$$

Договоримся в дальнейшем запятые опускать, то есть

$$\text{`}(X) : (a, (b, c))\text{'}$$

эквивалентно

$$\text{`}(X) : (a(b\, c))\text{'}.$$

$\#(B^2)$–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 4.8. Тип $\#(B^3)$.

Решение.

$\#(B^3)$–*1.* $B^3 = BBB^2$.

Поскольку

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

где $(a_1(b_1\ c_1)) = ((b\ c)((d(a\ b))(d(a\ c))))$, то $a_1 = (b\ c),\ b_1 = (d(a\ b)), c_1 = (d(a\ c))$.
Пусть $d_1 = e$. Тогда $(BBB^2) : ((b\ c)((e(d(a\ b)))(e(d(a\ c)))))$.

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

Задача 4.9. 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.
$$\begin{aligned} Exp\,1 &= ((a_1(b_1(c_1\ d_1)))(a_1(c_1(b_1\ d_1)))) \\ Exp\,2 &= (a_1(b_1(c_1\ d_1))) \\ Exp3 &= (a_1(c_1(b_1\ d_1))) \end{aligned}$$

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

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 4.10. Тип $\#(C^{[3]})$.

Решение.

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

$\#(C^{[3]})$–2.
$$\begin{aligned} Exp\,1 &= ((b_1\ c_1)((a_1\ b_1)(a_1\ c_1))) \\ Exp\,2 &= (b_1\ c_1) \\ Exp\,3 &= ((a_1\ b_1)(a_1\ c_1)) \\ Exp\,4 &= ((b_2\ c_2)((a_2\ b_2)(a_2\ c_2))) \\ Exp\,5 &= (b_2\ c_2) \\ Exp\,6 &= ((a_2\ b_2)(a_2\ c_2)) \end{aligned}$$

Поскольку

$$\dfrac{\dfrac{(B):\ Exp\,1 \quad (C):\ Exp\,2}{(BC):\ Exp\,3}\,(F) \quad \dfrac{(B):\ Exp\,4 \quad (C^{[2]}):\ Exp\,5}{(BC^{[2]}):\ Exp\,6}\,(F)}{(BC(BC^{[2]})):\ (a_1\ c_1)}\,(F),$$

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

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

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

Задача 4.11. Тип $\#(C_{[2]})$.

Решение.

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

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

Поскольку

$$\dfrac{(B^2):\ Exp\,1 \qquad\qquad (C):\ Exp\,2}{\dfrac{(B^2C):\ Exp\,3 \qquad (C):\ Exp\,4}{(B^2CC):\ Exp\,5}\,(F)}\,,\ (F)$$

где

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

то

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

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 4.12. Тип $\#(C_{[3]})$.

Решение.

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

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

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

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

Задача 4.13. Тип $\#(\Phi)$.

Решение.

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

$\#\Phi{-}2$.
$$\begin{aligned} Exp\,1 &= ((b_1\ c_1)((d_1(a_1\ b_1))(d_1(a_1\ c_1)))) \\ Exp\,2 &= (b_1\ c_1) \\ Exp\,3 &= ((d_1(a_1\ b_1))(d_1(a_1\ c_1))) \\ Exp\,4 &= (d_1(a_1\ b_1)) \\ Exp\,5 &= (d_1(a_1\ c_1)) \end{aligned}$$

Поскольку

$$\dfrac{(B^2):\ Exp\,1 \qquad \dfrac{(S):\ Exp\,2 \qquad \dfrac{(B^2S):\ Exp\,3 \qquad (B):\ Exp\,4}{(B^2SB):\ Exp\,5}\ (F)}{}}{}\ (F),$$

где

$$(b_1\ c_1) = ((a(b\ c))((a\ b)(a\ c))),$$
$$(d_1(a_1\ b_1)) = ((b_2\ c_2)((a_2\ b_2)(a_2\ c_2))),$$

то $d_1 = (b_2\ c_2) = (b_2(b\ c))$, $a_1 = (a_2\ b_2) = (a\ b_2)$, $c_1 = ((a\ b)(a\ c))$. Пусть $b_2 = d$; тогда

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

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

Задача 4.14. Тип $\#(Y)$.

Решение.

$\#Y{-}1$. Воспользуемся равенством $Y = WS(BWB)$.

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

$$\dfrac{(W):\ ((a_1(a_1\ b_1))(a_1\ b_1)) \qquad (S):\ (a_1(a_1\ b_1))}{(WS):(a_1\ b_1)}\ (F),$$

Поскольку $\#(S) = (a(b\ c))((a\ b)(a\ c))$, то $a_1 = (a(b\ c))$, $a_1 = (a\ b)$, $b_1 = (a\ c)$. Отсюда получаем, что должно быть $b = (b\ c)$, что невозможно в конечной форме. Следовательно, предположение о существовании типа $\#(WS)$ неверно. Кроме того,

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

Поскольку $\#(W) = (a_1(a_1\ b_1))(a_1\ b_1)$, то $b_2 = (a_1(a_1\ b_1))$, $c_2 = (a_1\ b_1)$. Далее,

$$\frac{(BW):\ ((a_2\ b_2)(a_2\ c_2))\qquad (B):\ (a_2\ b_2)}{(BWB):\ (a_2\ c_2)}.\ (F)$$

Но $\#(B) = (a_3\ b_3)((c_3\ a_3)(c_3\ b_3))$, откуда получаем $a_2 = (a_3\ b_3)$, $b_2 = \underline{(c_3\ a_3)}(\underline{\underline{c_3}}\ b_3)$, $b_2 = \underline{a_1}(\underline{\underline{a_1}}\ b_1)$, $c_2 = (a_1\ b_1)$. Из этих равенств вытекает, в частности, что $a_1 = (c_3\ a_3)$ и одновременно $a_1 = c_3$, то есть, что $c_3 = (c_3\ a_3)$, а это невозможно в конечной форме. Следовательно, предположение о существовании типа $\#(BWB)$ также неверно.

Получили, что выражению $WS(BWB)$ нельзя приписать тип. Таким образом, поскольку $Y = WS(BWB)$, то и комбинаторному представлению Y тип приписать не удается.

$\#Y{-}3$. Тем не менее, проведем следующие рассуждения. Известно, что $Yx = x(Yx)$, то есть $\#(Yx) = \#(x(Yx))$. Пусть $\#(x) = a$, $\#(Yx) = b$, тогда в соответствии с правилом $(F):\ \#(Y) = (a, b)$, поскольку

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

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

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

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

Ответ. Y имеет тип $((b,\ b), b)$, но $WS(BWB)$ типа не имеет.

Задача 4.15. Тип $\#(D)$.

Решение.

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

$\#D{-}2.$

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

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

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

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

Задача 4.16. Тип $\#(C)$.

Решение.

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

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

$\#C{-}3.$

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

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

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

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

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

где:

$$\begin{array}{ll}
(a_3\ b_3) = (b_4\ a_4), & \\
(b_2\ c_2) = ((b_6\ c_6), ((a_6\ b_6), (a_6\ c_6))), & \text{берется схема для } B \\
\\
a_4 = (a_5(b_5\ a_5)), & \text{берется схема для } K \\
\\
(a_2\ c_2) = (a_3(b_3\ c_3)), & \\
(a_2\ b_2) = (a\ b\ c)(a\ b)(a\ c). & \text{берется схема для } S
\end{array}$$

Имеем:

$$\left\{\begin{array}{lllllll}
\underline{a_3} & = & a_2 & = & \underline{(a\ b\ c)} & = & b_4, \\
c_2 & = & (b_3\ c_3) & = & (a_6\ b_6)(a_6\ c_6), & & \\
b_2 & = & (a\ b)(a\ c) & = & (b_6\ c_6), & & \\
b_3 & = & a_4 & = & (a_5\ b_5\ a_5). & &
\end{array}\right.$$

Далее,

$$\begin{array}{lllllll}
c_6 & = & (a\ c), & & & & \\
b_6 & = & (a\ b), & & & & \\
b_3 & = & (a_6\ b_6) & = & (a_6\ a\ b) & = & (b\ a\ b), \quad \text{поскольку } b_3 = (a_5\ b_5\ a_5).
\end{array}$$

Итак, $\underline{c_3} = (a_6\ c_6) = (b\ c_6) = \underline{(b\ a\ c)}$. Теперь остается только записать тип $(a_3,\ c_3)$.

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

Тема 5

Решение задачи синтеза структуры данных

Содержание

В аппликативную вычислительную систему встраивается нетривиальное приложение: значительный и, по-существу, полный фрагмент известной системы программирования Lisp (от англ. List processing).

5.1 Теоретические сведения

Одна из наиболее трудных проблем при разработке пользовательского (языкового) интерфейса состоит в правильном и подходящем встраивании приложения в программную среду. Успеху в ее решении, как известно, способствует применение объектного под-

хода и, естественно, объектно-ориентированное программирование. На практике определяется специальный набор классов, экспортирующий *методы* для прикладных программ.

В математике такой прием известен под названием *погружения*, когда в объемлющей теории, называемой *метатеорией*, строятся объекты, совокупность которых составляет *встроенную теорию*. Метатеория выступает в роли оболочки, а объекты встроенной теории могут адаптироваться по мере необходимости, причем не происходит выхода за рамки метатеории. Рассмотрим использование этого приема на примере. В качестве метатеории используем бестиповую комбинаторную логику. Будем считать ее оболочкой объектов. В качестве встроенной теории построим систему объектов вместе с их характеристическими равенствами, которые реализуют интерфейс Lisp, то есть образуют функционально полный универсум рассуждений о *списках* и *атомах*. Под списком объектов, как обычно, понимаем конечную последовательность объектов, среди которых могут быть другие списки.

Язык Lisp, или ЛИСП, является, по существу бестиповым языком. Его основные конструкции совпадают с конструкциями бестипового λ-исчисления. Хорошо известно, что эти конструкции выразимы средствами комбинаторной логики. Целью настоящего исследования является установление комбинаторных характеристик некоторых функций языка программирования.

Воспользуемся $\eta\xi$-исчислением λ-конверсий. Приведем постулаты, задающие отношение конвертируемости '*conv*' (обозначаемое знаком '='):

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

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

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

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

5.2 Основная задача

Задача 5.1. Выразить с помощью комбинаторов следующий набор функций Lisp-системы:

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

Формулировка задачи. Рассмотрим свойства этих функций языка Lisp.

- Посредством функции $Append$ строится конкатенация двух списков. Эта функция обладает свойством *ассоциативности*:
$$A \frown B \frown C = (A \frown B) \frown C, \qquad (Append)$$
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:
$$Null\ A = \begin{cases} \overline{1}, & \text{если } A = Nil, \\ \overline{0}, & \text{в противном случае.} \end{cases} \qquad (Null)$$

- Функция $List$ строит из атома список длины 1:
$$List\ x =< x >, \qquad (List)$$
где x – атом, а $< x_1,\ x_2,\ \ldots\ , x_n >$ – 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:

$$\begin{aligned}
Append\ a\ (Append\ b\ c) &= Append(Append\ a\ b)c, && (5.1)\\
Append\ Nil\ a &= Append\ a\ Nil, && (5.2)\\
Null\ Nil &= \overline{1}, && (5.3)\\
Null(Append(List\ a)b) &= \overline{0}, && (5.4)\\
Car(Append(List\ a)b) &= a, && (5.5)\\
Cdr(Append(List\ a)b) &= b, && (5.6)
\end{aligned}$$

где a, b, c — произвольные объекты. Докажем, что аксиомы (5.1)—(5.6) выводимы в $\eta\xi$-исчислении λ-конверсии.

Решение. Осуществим последовательный перевод содержательных равенств (5.1)—(5.6) в термы и формулы комбинаторной логики.

Lisp—1. Покажем, что функции *Append* соответствует объект B с комбинаторной характеристикой (B) : $Babc = a(bc)$ (схема аксиом (5.1)):

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

Учитывая правило транзитивности (τ), имеем:

$$Ba(Bbc)\,x = B(Bab)c\,x.$$

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:

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

Итак, схема аксиом (5.1) доказана.

Lisp–2. Докажем схему аксиом (5.2), принимая во внимание, что $Nil \leftrightarrow I$ выполняется[1], причем (I) : $Ia = a$.

$$\begin{array}{lll}
BIa\,x & = & I(ax) \qquad (\text{ по } (B)) \\
 & = & ax \qquad (\text{ по } (I)) \\
 & = & a(Ix) \qquad (\text{ по } (I)) \\
 & = & BaI\,x. \qquad (\text{ по } (B))
\end{array}$$

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:

$$(\nu^{-1}) : \qquad \frac{ux = vx}{u = v}.$$

Это правило, как оказалось, работает в случае, когда x является переменной. Если сопоставить его с одним из правил монотонности (ν), то можно заметить, что посылка и заключение в нем поменялись местами. На этом основании (в случае, когда x – переменная) это правило (ν^{-1}) может быть названо правилом *обращения монотонности*.

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

Lisp–3. Перепишем схему аксиом (5.3) в виде равенства $\overline{1} = Null\ Nil$ (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

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

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

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

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:

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

Сравним полученное выражение $D\overline{0}(K(K\overline{0}))(B(Da)b) = \overline{0}$ со схемой аксиом (5.4): $Null(Append(List\ a)b) = \overline{0}$. 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:

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

Очевидно, что $DcK \leftrightarrow Car$.

Lisp–6. Тем же способом построим объект, соответствующий функции Cdr:

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

Поскольку

$$b\,z = (\lambda xy.xy(K\ I))(B(Da)b)\,z,$$

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	$\lambda xy.xy\overline{0}$

5.3 Заключительные замечания

Решение поставленной задачи, безусловно, выполнено методом погружения. Переформулируем его в терминах объектов.

Система равенств (5.1)–(5.6) в совокупности рассматривается как *соотнесение*, для которого индивидуализируется теория-оболочка. Теория-оболочка представляет 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

$$Append, \qquad Nil, \qquad Null, \qquad List, \qquad Car, \qquad Cdr$$

в результате выполнения соотнесения даст соответствующий объект-индивид

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

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 (5.1)–(5.6) остаются в рамках одного и того же универсума.

Тема 6

Базисы

Содержание

При построении комбинаторной логики одна из изначальных задач состояла в предъявлении минимального числа наипростейших объектов, атомарных по своей сути, и способов комбинирования, которые дают достаточно богатые метаматематические средства. Эти наипростейшие объекты должны вести себя как *кон-*

стантные функции, а с точки зрения компьютерных наук давать примитивную систему-ядро для программирования, которую можно развивать на основе "самораскрутки". Это означает, что более сложные объекты строятся на основе более примитивных, а затем присоединяются к системе и в свою очередь могут быть использованы как элементы других конструкций.

Сосредоточим внимание на наипростейшей системе программирования, в которой всего только три инструкции: I, K, S. Синтезировать новый объект можно чисто механическим использованием алгоритма разложения в базисе, который вполне аналогичен процессу компиляции.

6.1 Базис I, K, S

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.

6.2 Задачи

Задача 6.1. Выразить терм $\lambda x.P$ через комбинаторы $I,\ K,\ S$.

Формулировка задачи. Пусть определение терма $\lambda x.P$ дано индукцией по построению P:

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

Исключить все переменные из приводимых λ-выражений:

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

Решение.

$$\begin{aligned}
P{-}1.\quad \lambda xy.yx &\overset{def}{=} \lambda x.(\lambda y.yx)\\
&\overset{(3)}{=} \lambda x.(S(\lambda y.y)\lambda y.x))\\
&\overset{(1),\,(2)}{=} \lambda x.SI(Kx)\\
&\overset{(3)}{=} S(\lambda x.SI)(\lambda x.Kx)\\
&\overset{(2)}{=} S(K(SI))(S(\lambda x.K)(\lambda x.x))\\
&\overset{(1),\,(2)}{=} S(K(SI))(S(KK)I)
\end{aligned}$$

$$\begin{aligned}
P{-}2.\quad \lambda fx.fxx &\overset{def}{=} \lambda f.(\lambda x.fxx)\\
&\overset{(3)}{=} \lambda f.(S(\lambda x.fx)(\lambda x.x))\\
&\overset{(1),\,(3)}{=} \lambda f.S(S(\lambda x.f)(\lambda x.x))I\\
&\overset{(1),\,(2)}{=} \lambda f.S(S(Kf)I)I\\
&\overset{(3)}{=} S(\lambda f.S(S(Kf)I))(\lambda f.I)\\
&\overset{(2),\,(3)}{=} S(S(\lambda f.S)(\lambda f.S(Kf)I))(KI)\\
&\overset{(2),\,(3)}{=} S(S(KS)(S(\lambda f.S(Kf))(\lambda f.I)))\\
&\qquad\qquad (KI)\\
&\overset{(2),\,(3)}{=} S(S(KS)(S(S(\lambda f.S)\\
&\qquad\qquad (\lambda f.Kf)))(KI)))(KI)\\
&\overset{(2),\,(3)}{=} S(S(KS)(S(S(KS)(S(\lambda f.K)\\
&\qquad\qquad (\lambda f.f))(KI)))(KI)\\
&\overset{(1),\,(2)}{=} S(S(KS)(S(S(KS)(S(KK)I)\\
&\qquad\qquad (KI)))(KI)
\end{aligned}$$

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

Упражнения

Упражнение 6.1. Выразить комбинатор $W\,mult$ возведения функции в квадрат в терминах комбинаторов I, K, S.

Упражнение 6.2. Для комбинатора Φ с характеристикой

$$\Phi fabx = f(ax)(bx)$$

выполнить следующее:

1° записать предикат

$$1 < x < 5$$

без переменной.

Указание. Для этого введите в рассмотрение подходящую конструкцию $\Phi\ and\ (greater\ 1)\ (less\ 5)$;

2° выразить комбинатор

$$\Phi\ and\ (greater\ 1)\ (less\ 5)$$

в терминах комбинаторов I, K, S.

Упражнение 6.3. Выразить комбинатор, представляющий функцию формирования суммы синусов двух чисел в терминах комбинаторов I, K, S.

Указание. Можно, например, воспользоваться комбинатором Φ и записать

$$\Phi\ plus\ sin\ sin.$$

Попытайтесь преобразовать это выражение, воспользовавшись комбинаторами для устранения двухкратного вхождения 'sin'. Например,

$$\Phi\ plus\ sin\ sin = W(\Phi\ plus)\ sin = \ldots$$

и т.д.

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.

6.3 Базис I, B, C, S

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].

6.4 Свойство базисности

Можно получить большое количество различных комбинаторов, однако оказывается, что эти комбинаторы не являются независимыми; часть их можно определить через набор комбинаторов, называемых *базисными*. Под комбинатором теперь можно понимать объект, составленный с помощью аппликаций из базисных комбинаторов.

Рассмотрим две базисные системы комбинаторов C, W, B, K и S, K:

$$\begin{aligned} Cxyz &= xyz, \\ Wxy &= xyy, \\ Bxyz &= x(yz), \\ Kxy &= x; \end{aligned} \qquad\qquad \begin{aligned} Sxyz &= xz(yz), \\ Kxy &= x. \end{aligned}$$

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 $X = S$.

[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.

В общем случае поиск комбинатора X осуществляется посредством следующих действий:

1) с помощью B из V удаляются скобки;
2) с помощью C переменные переупорядочиваются;
3) с помощью W устраняются дублирования переменных;
4) с помощью K вводятся переменные, отсутствующие в V.

Пример 6.1. Построим в первой базисной системе комбинатор X со свойством

$$ac(bc) = X\,abc:$$

$$\begin{aligned}
\underbrace{(ac)}_{x}(\underbrace{b}_{y}\ \underbrace{c}_{z}) &= \underbrace{B}_{x}(\underbrace{a}_{y}\ \underbrace{c}_{z})b\,c \\
&\overset{B}{=} \underbrace{(BBa)}_{x}\ \underbrace{c}_{z}\ \underbrace{b}_{y}\,c \\
&\overset{C}{=} \underbrace{(C(BBa)b)}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{y} \\
&\overset{W}{=} \underbrace{W}_{x}(\underbrace{C(BBa)}_{y}\ \underbrace{b}\,)_z c \\
&\overset{B}{=} \underbrace{(BW)}_{x}(\underbrace{C}_{y}\ \underbrace{(BBa)}_{z})bc \\
&\overset{B}{=} \underbrace{(B(BW)C)}_{x}(\underbrace{(BB)}_{y}\ \underbrace{a}_{z}\,)bc \\
&\overset{B}{=} B(B(BW)C)(BB)abc.
\end{aligned}$$

В этом примере под объектами, к которым можно применить заранее известную *схему*, записаны переменные — в том же порядке, что и в соответствующей комбинаторной характеристике. Таким образом,

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

6.5 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 6.2. Выразить терм $M = \lambda x.PQ$, пользуясь исключительно комбинаторами I, B, C, S.

Формулировка задачи. Пусть определение терма M такого, что переменная x входит в состав свободных переменных (PQ), то есть $x \in FV(PQ)$, дано индукцией по построению M (здесь '$\in$' означает 'принадлежит', а '$\notin$' – 'не принадлежит'):

$$
\begin{array}{lll}
(1) & \lambda x.x & = I, \\
(2) & \lambda x.PQ = \left\{ \begin{array}{lll}
(a) & BP(\lambda x.Q), & \text{если } x \notin FV(P) \text{ и } x \in FV(Q), \\
(b) & C(\lambda x.P)Q, & \text{если } x \in FV(P) \text{ и } x \notin FV(Q), \\
(c) & S(\lambda x.P)(\lambda x.Q), & \text{если } x \in FV(P) \text{ и } x \in FV(Q).
\end{array} \right.
\end{array}
$$

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

Решение.

[2] На решение задачи разложения в базисе можно взглянуть иначе. Поскольку всякий комбинатор – это *понятие* и даже *концепт* в математическом понимании, то исходный терм считается 'исследуемым' или 'познаваемым' объектом, базис – 'системой известных понятий', а процедура разложения в базисе – 'представлением знания' об исходном объекте в терминах известных понятий. Такие рассуждения в своей основе используются в приложениях объектно-ориентированного подхода.

$$
\begin{array}{lrl}
M{-}1. & \lambda xy.yx & = \lambda x.(\lambda y.yx) \\
 & & \overset{(2)(b)}{=} \lambda x.(C(\lambda y.y)x) \\
 & & \overset{(1)}{=} \lambda x.CIx \\
 & & \overset{(2)(a)}{=} B(CI)(\lambda x.x) \\
 & & \overset{(1)}{=} B(CI)I.
\end{array}
$$

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

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

Упражнения

Упражнение 6.4. Доказать, что набор комбинаторов C, W, B, K проявляет свойство *базисности*, то есть является базисом.

Указание. Воспользуйтесь определением базиса в общей форме (см. [2], с. 172):

Определение 6.1 (базис). **(i)** Пусть $\mathcal{X}$ – некоторое подмножество всех λ-термов Λ, $\mathcal{X} \subset \Lambda$. Обозначим через $\mathcal{X}^+$ множество термов, *порожденное множеством* $\mathcal{X}$. Множество $\mathcal{X}^+$ – это наименьшее множество $\mathcal{Y}$, такое что:

1) $\mathcal{X} \subseteq \mathcal{Y}$,

2) если термы $M, N \in \mathcal{Y}$, то их аппликация $(M\,N) \in Y$.

(ii) Возьмем некоторое подмножество λ-термов $\mathcal{A} \subseteq \Lambda$. Множество термов $\mathcal{X} \subseteq \Lambda$ называется *базисом* для $\mathcal{A}$, если

$$(\forall M \in \mathcal{A})(\exists N \in \mathcal{X}^+).N = M.$$

(iii) Множество $\mathcal{X}$ называется *базисом*, если $\mathcal{X}$ – базис для множества всех замкнутых термов.

Упражнение 6.5. Выразить комбинатор $W\,mult$ возведения функции в квадрат в терминах комбинаторов I, B, C, S.

Упражнение 6.6. Для комбинатора Φ с характеристикой

$$\Phi fabx = f(ax)(bx)$$

выполнить следующее:

1° записать предикат $1 < x < 5$ без переменной.

Указание. Для этого введите в рассмотрение подходящую конструкцию $\Phi\ and\ (greater\ 1)\ (less\ 5)$;

2° выразить комбинатор $\Phi\ and\ (greater\ 1)\ (less\ 5)$ в терминах комбинаторов I, B, C, S.

Упражнение 6.7. Выразить комбинатор, представляющий функцию формирования суммы синусов двух чисел в терминах комбинаторов I, B, $C\ S$.

Указание. Можно, например, воспользоваться комбинатором Φ и записать

$$\Phi\ plus\ sin\ sin.$$

Попытайтесь преобразовать это выражение, воспользовавшись комбинаторами для устранения двухкратного вхождения 'sin'. Например,

$$\Phi\ plus\ sin\ sin = W(\Phi\ plus)\ sin = \ldots$$

и т.д.

6.6 Арифметические сущности

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 (арифметические объекты со своими правилами) встраивается в программную среду (комбинаторную логику).

6.7 Числа и нумералы

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

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

Тем не менее при построении комбинаторной логики или λ-исчисления среди исходных объектов нет чисел. Нет ли в этих системах недостатка в выразительных возможностях? Оказывается, что в рамках комбинаторной логики или λ-исчисления можно установить такие комбинаторы или, соответственно, термы, которые ведут себя как числа. Эти представления чисел получили

название *нумералов*. Нумералы как комбинаторы удовлетворяют всем законам комбинаторной логики. Более того, можно указать комбинаторы, представляющие арифметические операции, например, сложение чисел. Исследования в этой области пока все еще далеки от завершения.

6.8 Комбинаторная арифметика

Построение арифметики обычно начинается с введения натуральных чисел (см. А.С. Кузичев, [29]). В комбинаторной логике для этого вводится два объекта, при записи которых использован оператор абстракции (см. пункт 3.1.1 на стр. 70 и далее):

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

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

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

i) Z_0 считается комбинаторным числом;

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

Утверждение 6.1. Можно показать, что комбинатор Z_n обладает свойством:

$$Z_n XY = \underbrace{X(X(\ldots X(X}_{n} Y)\ldots)),$$

где X и Y — объекты, а $n \geq 0$.

Доказательство. Индукцией по n.

i) $Z_0 = [xy]y$, а $Z_0 XY = \underbrace{}_{0} Y$;

ii) $Z_1 = \widehat{\sigma} Z_0$, а

$$\begin{aligned} Z_1 XY &= \widehat{\sigma} Z_0 XY \\ &= ([xyz](y(xyz)))([xy]y)XY \\ &= X(([xy]y)XY) \\ &= \underbrace{X}_{1} Y; \end{aligned}$$

и т.д.

Для введения других арифметических объектов потребуется *комбинатор пары*

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

со следующими свойствами:

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

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

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

Предшественник

Операция предшествования '-1'обозначается через π и определяется посредством

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

где $\overline{\sigma} \equiv [x](\mathcal{D}(\widehat{\sigma}(xZ_0))(xZ_0))$. Комбинатор 'предшествования' π может рассматриваться как обращение операции 'следования за' $\widehat{\sigma}$.

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

1) $\pi Z_0 = Z_0$;

2) $Z_n\overline{\sigma}(KZ_0)Z_0 = Z_n$;

3) $\pi Z_{n+1} = Z_n$.

Модифицированное вычитание

Операция *модифицированного*, или *усеченного вычитания* в обычной арифметике обозначается через '$\dot{-}$' и определяется посредством:

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

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

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

и имеющий свойства:

1) $\mathbb{L}Z_nZ_0 = Z_n$;

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

Операция минимума

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

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

где

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

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

Пример 6.2. Положим

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

Очевидно, что:

1) $\alpha Z_0 = Z_0$;

2) $\alpha Z_m = Z_1$ для $m > 0$.

Сложение

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

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

где

$$\begin{array}{rclcrcl} \mathcal{E} & \equiv & [xy](Uy[z](xxz)), & \quad & U & \equiv & [x](\mathcal{C}(\mathcal{M}xZ_1)x), \\ \mathcal{C} & \equiv & [x](xCZ_0T\mathcal{V}), & & \mathcal{V} & \equiv & [xy](xyI), \end{array}$$
$$T \equiv [xyz](\widehat{\sigma}(y(\pi x)z)).$$

Упражнение 6.8. Проверить, что комбинатор $\mathbb{A}$ обладает всеми свойствами операции сложения '+', то есть для $n, m \geq 0$:

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

Симметрическая разность

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

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

Умножение

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

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

а вместо комбинатора T используется комбинатор T':

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

Упражнение 6.9. Показать, что комбинатор $\mathbb{M}$ имеет свойства оператора умножения на арифметических объектах:

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

Комбинаторно определимые функции

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

Определение 6.2 (комбинаторная определимость)**.** Функцию ϕ от n аргументов называют *комбинаторно определимой*, если найдется объект X такой, что для любого набора из n натуральных чисел $r_1, \ldots, r_n$, для которого

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

где r — натуральное число, выполняется

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

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

Таким образом, надчеркивание рассматривается как отображение

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

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

Примитивная рекурсия

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

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

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

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

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

где:

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{E}^* & \equiv & [xy](Uy[z](xxz)), & U & \equiv & [x](C^*(\overline{\min}\; x\; \overline{1})x), \\ C^* & \equiv & [x](xC\overline{0}TV), & V & \equiv & [xy](xyg), \\ \multicolumn{6}{c}{T \equiv [xyz_1 \ldots z_n]H(\pi x)(y(\pi x)z_1 \ldots z_n)z_1 \ldots z_n.} \end{array}$$

Для такого выбора можно показать, что:

$$\begin{array}{rcl} g\,\overline{0}\,\overline{r_1}\ldots\overline{r_n} & = & \overline{\psi(r_1, \ldots, r_n)}, \\ g\,\overline{r+1}\,\overline{r_1}\ldots\overline{r_n} & = & \overline{\psi(r, \phi(r, r_1, \ldots, r_n), r_1, \ldots, r_n)}. \end{array}$$

6.9 Задачи

Задача 6.3. Определить объекты, обладающие свойствами натуральных чисел (нумералов) и исследовать их свойства.

Формулировка задачи. Нумералы – это следующие объекты:

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

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

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

Решение.

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

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

Таким образом, $x^4 y = x(x(x(xy)))$.

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

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

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

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

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

Задача 6.4. 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'):

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

Решение.

σ–1. Комбинаторная характеристика нумералов:

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

σ–2. Применим функцию σ к нумералу $\overline{n}$ в общем виде:

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

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

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

Задача 6.5. Определить объект, вычисляющий длину конечной последовательности (списка).

Формулировка задачи. Показать, что функция

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

есть функция определения длины списка x :

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

Решение.

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

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

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

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

$Length{-}2$. Приложим $Length$ к пустому списку Nil:

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

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

$Length{-}3$. Приложим функцию $Length$ к списку x, состоя-

щему из одного элемента: $x = \langle a \rangle$:

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

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

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

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

Ответ. Функция $Length$ действительно вычисляет длину списка.

Упражнения

Показать или опровергнуть следующее.

Упражнение 6.10. Сложение определяется λ-выражением

$$\lambda mnfx.mf(nfx).$$

Упражнение 6.11. Произведение определяется λ-выражением

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

Упражнение 6.12. Возведение в степень определяется λ-выражением

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

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

Часть III

Динамика вычислений

Содержание

Тема 7

Динамичные базисы

Содержание

Суперкомбинаторы устанавливают чисто объектную систему программирования, встроенную в комбинаторную логику. Тем самым непосредственно удовлетворяется потребность в денотационном вычислении инструкций языков программирования, когда объектами выражается функциональный смысл программы. Существенно, что вычисление начинается с некоторого заранее известного набора инструкций. В процессе вычисления значения программы динамически возникают заранее неизвестные, но необходимые по ходу дела инструкции, которые дополнительно фиксируются в системе программирования.

7.1 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.

7.1.1 Понятие о суперкомбинаторе

Определение 7.1. Суперкомбинатор $\$S$ арности n представляет собой ламбда-выражение

$$\$S \stackrel{\text{def}}{=} \lambda x_1.\lambda x_2.\ldots.\lambda x_n.E,$$

или, что эквивалентно, абстракцию вида:

$$\$S \stackrel{\text{def}}{=} [x_1].[x_2].\ldots.[x_n].E,$$

где E не является абстракцией. Таким образом, все "ведущие" символы абстракции $[\cdot].$ относятся только к x_1, x_2, …, x_n, при этом выполняются условия:

(1) $\$S$ не содержит свободных переменных;

(2) каждая абстракция в E является суперкомбинатором;

(3) $n \geq 0$, то есть наличие символов '$[\cdot].$' не обязательно.

Суперкомбинаторный редекс – это аппликация суперкомбинатора к n аргументам, где n – его арность. Подстановка аргументов в тело суперкомбинатора вместо свободных вхождений соответствующих формальных параметров называется *редукцией* суперкомбинатора.

Можно вспомнить обычное определение комбинатора и произвести сравнение. Другими словами, можно сказать, что комбинатор — это такая ламбда-абстракция, которая не содержит вхождений свободных переменных. Некоторые ламбда-выражения являются комбинаторами, а некоторые комбинаторы являются суперкомбинаторами.

Пример 7.1. Выражения

```
3,  + 2 5,  [x].x,  [x].+ x x,  [x].[y].xy
```

представляют собой суперкомбинаторы.

Пример 7.2. Приводимые термы не являются суперкомбинаторами:

`[x].y` — (переменная `y` входит свободно),
`[y].- y x` — (переменная `x` входит свободно).

Пример 7.3. 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 7.1. 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 7.4. 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 7.1. Показать, что следующие выражения являются суперкомбинаторами:

```
1 0,  [x].+ x 1,  [f].([x].+ x x).
```

Упражнение 7.2. Объясните, почему следующие выражения не являются суперкомбинаторами:

```
[x].x y z,  [x].[y].+ (+ x y) z.
```

Упражнение 7.3. Привести пример комбинатора, не являющегося суперкомбинатором.

7.1.2 Процесс компиляции

Реальные программы содержат значительное число абстракций. Программу следует преобразовать таким образом, чтобы она содержала только суперкомбинаторы. Согласимся с тем, что имена суперкомбинаторов будут начинаться с символа '$', например:

```
$XY = [x].[y].- y x.
```

Для того, чтобы подчеркнуть особенности суперкомбинаторов, перепишем это определение в виде:

```
$XY x y = - y x.
```

Избираемая стратегия заключается в преобразовании абстракции, которую следует откомпилировать, в:

(i) совокупность суперкомбинаторных определений,

(ii) вычисляемое выражение.

Будем изображать это посредством:

```
Определения суперкомбинаторов
.............................
.............................
-----------------------------
    Вычисляемое выражение
```

Пример 7.5. Выражение $([x].[y].-\ y\ x)3\ 4$ представимо в виде:

```
$XY x y = - y x
---------------
    $XY 3 4
```

Пример 7.6. 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 7.4. Можно ли вычислить выражения:

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

7.1.3 Приведение к суперкомбинаторам

Суперкомбинаторы легко компилируются. Дадим описание *алгоритма* преобразования абстракций в комбинаторы. Рассмотрим программу, не содержащую ни одного суперкомбинатора:

```
([x].([y].+ y x) x) 4.
```

Выберем самую внутреннюю абстракцию, то есть такую абстракцию, которая не содержит других абстракций:

```
([y].+ y x).
```

В нее входит свободная переменная `x`, поэтому эта абстракция не является суперкомбинатором.

(1) С помощью простого преобразования (обычной β-редукции) получим суперкомбинатор

```
([x].[y].+ y x)x.
```

(2) Подставим его в исходную программу:

```
([x].([w].[y].+ y w) x x)4.
```

(3) Присвоим суперкомбинатору имя `$Y`.

(4) Теперь видно, что `[x].`-абстракция тоже является суперкомбинатором. Дадим ему имя `$X` (скомпилируем абстракцию) и сопоставим его скомпилированному коду:

```
 $Y w y = + y w
   $X x = $Y x x
----------------
      $X 4
```

Можно выполнить полученную программу, осуществляя редукцию суперкомбинаторов:

```
$X 4 --> $Y 4 4 = + 4 4 = 8.
```

Таким образом, алгоритм преобразования абстракций в суперкомбинаторы приобретает следующий вид:

Цикл-*while*: есть абстракции?

(1) выбрать любую абстракцию, *без* других абстракций,

(2) вынести все свободные в этой абстракции переменные в качестве экстрапараметров,

(3) абстракции приписать некоторое имя (например, `$X34`),

(4) заменить вхождение абстракции в программу на имя суперкомбинатора, которое приложено к свободным переменным,

(5) произвести компиляцию абстракции и скомпилированному коду сопоставить имя.

Конец-*while*

В ходе преобразования объем программы возрос. Это является своеобразной платой за простоту правил редукции. Заметим, что процедура преобразования приводит исходную программу к виду:

```
... определения суперкомбинаторов ...
-------------------------------------
                  E
```

Поскольку выражение `E` является вычисляемым выражением самого высокого уровня, то оно не содержит свободных переменных. Его можно считать суперкомбинатором арности 0, то есть КАФ:

```
... определения суперкомбинаторов ...
              $Prog = E
-------------------------------------
                $Prog
```

Процесс преобразования абстракций в суперкомбинаторы называется *лямбда-подъемом*, поскольку все абстракции поднимаются на верхний уровень.

Упражнение 7.5. Преобразовать и выполнить программы:
1) `([x].([y].- y x) x) 5`,
2) `([z].+ z(([x].([y].× y x) x) 4)) 2`.

Тема 8

Использование параметров

Содержание

8.1 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:

```
$Y x y = - y x
--------------
   [x].$Y x
```

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

```
$Y x y = - y x
  $X x = $Y x
--------------
      $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`:

```
$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.

8.2 Упорядочивание параметров

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

```
..........
   (...
([x].[z].+ y (× x z))
   ...),
```

где '...' указывает на объемлющий [x].-абстракцию контекст. Начнем выполнять алгоритм ламбда-подъема.

(1) Выберем самую внутреннюю абстракцию

```
[z].+ y(× 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(× x z))x y.
```

(3) Присвоим полученному суперкомбинатору имя:

```
$S = ([x].[y].[z].+ y(× x z)).
```

(4) Подставим `$S` в исходную программу:

```
$S x y z = + y(× x z)
---------------------
        (...
    ([x].$S x y)
        ...)
```

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:

```
$T y x = $S x y
---------------
     $T y.
```

Вернемся к основному алгоритму, в котором теперь получаем:

```
$S x y z = + y(× x z)
   $T y x = $S x y
   (...  $T y ...).
```

Вариант 2.

(2) Вынесем переменные, расположив их в порядке `y, x`:

```
([y].[x].[z].+ y(× x z)) y x.
```

(3) Присвоим полученному суперкомбинатору имя:

```
$S = ([y].[x].[z].+ y(× x z)).
```

(4) Подставим $\$S$ в исходную программу:

```
$S y x z = + y(× x z)
------------------------
        (...
   ([x].$S y x)
        ...)
```

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

(1) Самая внутренняя абстракция совпадает с программой.

(2) Вынесем переменную `y`: `([y].[x].$S y x)y`.

(3) Присвоим суперкомбинатору имя `$T = [y].[x].$S y x`.

(4) Подставим комбинатор в программу:

```
$T y x = $S y x
---------------
      $T y
```

Вернемся к основному алгоритму. Получим:

```
$S x y z = + y (× x z)
   $T y x = $S y x
----------------------
    (... $T y ...)
```

В соответствии с правилом устранения избыточных параметров (см. подраздел 8.1) получаем: `$T = $S`, поэтому `$T` можно устранить. Тогда скомпилированный код примет вид:

```
$S y x z = + y(× x z)
-------------------
$S y
```

В первом варианте такая оптимизация невозможна. В исходной программе

```
(... ([x].[z].+ y(× x z)) ...)
```

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 8.1. В выражении

```
([x].[y].+ x(× 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.

Тема 9

Использование параметров (продолжение)

Содержание

9.1 Ламбда-подъем при рекурсии

Заметим, что ламбда-абстракции, как правило, не имеют имен. В отличие от них суперкомбинаторы имеют имена. Помимо этого суперкомбинаторы могут ссылаться сами на себя. Это означает, что рекурсивные суперкомбинаторы реализуются непосредственно, без привлечения комбинатора неподвижной точки Y. Конечно,

рекурсивные определения можно преобразовать в нерекурсивные, воспользовавшись Y, но это потребует введения в употребление дополнительных правил.

Пример 9.1. Для того, чтобы `$F` стал нерекурсивным, следует ввести определения:

```
$F = Y $F1
$F1 E x = $G (F(- x 1)) 0.
```
(*)

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

Обозначение:

```
$S1 x y = B1
S2 f = B2
...
E
```

эквивалентно выражению:

```
letrec
    $S1 = [x].[y].B1
    $S2 = [f].B2
...
in
    E.
```

Оно означает, что в `E` входят `$S1, $S2, ...`, рекурсивные определения которых приведены в `letrec`. Алгоритм 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 9.2. Приведем программу, дающую бесконечный список единиц:

```
-------------------
letrec x = cons 1 x
in x
```

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

```
$x = cons 1 x
-------------
x
```

Пример 9.3. Рассмотрим рекурсивную функцию вычисления факториала:

```
-----------------------------------
letrec fac = [n].IF (= n 0) 1
                    (× n (fac(- n 1)))
in fac 4
```

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

```
$fac n = IF (= n 0) 1 (× n(fac(- n 1)))
$Prog = $fac 4
----------------------------------------
$Prog
```

Упражнение 9.1. 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`.

9.2 Работа алгоритма ламбда-подъема

Рассмотрим программу, суммирующую первые 100 целых чисел:

```
SumInts m = sum (count 1)
            where
            count n = [], n > m
                    = n:count(n + 1)
     sum [] = 0
     sum (n:ns) = n + sum ns
-----------------------------------
SumInts 100
```

`SumInts` представляет собой композицию функций `sum` и `count`: сначала функция `count` применяется к 1, а затем ее результат поступает на вход функции `sum`. Функция `count` работает следующим образом:

```
count 1 = 1:count 2 (так как n = 1, m = 100,
                     и условие n > m не выполняется)
        = 1:2:(count 3) (вычисляется count 2)
...
        = 1:2: ... :100:(count 101)
        = 1:2: ... :100:[] (поскольку
                            n = 101, m = 100,
                            то n > m, (count 101 = []) )
```

Теперь список `1:2:3: ... :100:[]` передается функции `sum`. Во втором определяющем равенстве для `sum` аргумент `(n:ns)` обозначает список, в котором первым элементом является число `n`, а "хвостом" — список чисел `ns`:

```
sum 1:2:3: ... :100:[] = 1 + sum 2:3: ... :100:[]
                       ...
     = 1 + 2 + 3 + ...  + 100 + sum[]
     = 1 + 2 + 3 + ...  + 100 + 0 (так как sum [] = 0)
```

Запишем эту функцию в терминах абстракций с использованием `letrec`:

```
letrec
     SumInts
       [m].letrec
            count = [n].IF (> n m) NIL
                           (cons n (count (+ n 1) ))
            in sum (count 1)
       sum = [ns].IF (= ns NIL) 0
                     (+ (head ns)(sum (tail ns)) )
  in SumInts 100
```

В данном случае: `NIL` — пустой список `[]`, `head` — функция, возвращающая первый элемент списка, `tail` — функция, возвращающая хвост списка (список без первого элемента). Переменные `SumInts` и `sum` имеют номер 0, однако `SumInts` содержит внутреннюю `[n].`-абстракцию со свободными переменными `m` и `count`. Необходимо

выполнить алгоритм 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 конструкцию `$count count m n`:

```
$count count m n = IF (> n m) NIL
                      (cons n (count (+ n 1)))
-----------------------------------------------
letrec
    SumInts
         = [m].letrec
              count = $count count m
              in sum (count 1)
    sum = [ns].IF (= ns NIL) 0
            (+ (head ns)(sum (tail ns)))
in SumInts 100
```

(5) В выражениях `SumInts` и `sum` нет внутренних абстракций, их уровень равен 0, поэтому они являются суперкомбинаторами. Непосредственно применяя к ним подъем и добавляя суперкомбинатор `$Prog`, получим окончательный результат:

```
$count count m n = IF (> n m) NIL
                      (cons n(count (+ n 1)))
$sum ns = IF (= ns NIL) 0 (+ (head ns)
                           ($sum (tail ns)))
$SumInts m = letrec count = $count count m
         in $sum (count 1)
$Prog = $SumInts 100
-----------------------------------------
$Prog
```

Упражнение 9.2. Скомпилировать программу, которая прикладывает функцию `f = КВАДРАТ` к каждому элементу списка целых чисел от 1 до 5:

```
---------------------------------------
apply m = fold КВАДРАТ (constr 1)
         where constr n = [], n > m
                        = n:constr(n + 1)
fold f[] = []
fold f(n:ns) = fn:fold f ns
```

9.3 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`:

```
-------------------------------------
( ...
    letrec f = [x].( ...  f ... v ... )
    in ( ... f ... )
... )
```

По f строится рекурсивный суперкомбинатор $f, однако при этом абстрагируется не сама функция f, а переменная v: все вхождения v замещаются на $f v. В результате замещения получаем:

```
-------------------------------------
$f v x = ... ($f v) ... v ...
( ...
    ( ... ($f v) ... )
  ... )
```

Посмотрим, как работает данный алгоритм на примере из подраздела 9.1. 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 9.1 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 9.3. Выполнить предложенным методом компиляцию программы из упражнения 9.2.

9.4 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 9.4. Рассмотрим программу:

```
--------------------
f = g 4
g x y = y + (sqrt x)
(f 1) + (f 2)
```

Запись в виде ламбда-выражения дает:

```
----------------------------
letrec f = g 4
    g = [x].[y].+ y (sqrt x)
    in + (f 1) (f 2)
```

При вычислении этого выражения получаем следующий результат:

```
+ (f 1) (f 2) -->
--> + (. 1) (. 2)
       .-----.--->  (([x].[y].+ y (sqrt x)) 4)
-->  + (. 1)(. 2)
        .----.---> ([y].+ y (sqrt 4))
-->  + (. 1)(+ 2 (sqrt 4))
        .--------> ([y].+ y (sqrt 4))
-->  + (. 1)4
        .--------> ([y].+ y (sqrt 4))
-->  + (+ 1 (sqrt 4))4
-->  + (+ 1 2)4
-->  + 3 4
-->  7
```

В этом примере подвыражение `(sqrt 4)` вычисляется дважды, при каждом применении выражение `[y].(sqrt 4)` считается

динамически создаваемым константным подвыражением `[y].`-абстракции. Точно такой же эффект наблюдается и при переходе к суперкомбинаторам. Рассмотренное выражение компилируется следующим образом:

```
$g x y = + y (sqrt x)
$f = $g 4
$Prog = + ($f 1)($f 2)
----------------------
$Prog
```

Редукция выглядит следующим образом:

```
$Prog -->  + (. 1)(. 2)
                .----.---> ($g 4)
-->  + (. 1)(+ 2 (sqrt 4))
        .---> ($g 4)
-->  + (. 1) 4
        .---> ($g 4)
-->  + (+ 1 (sqrt 4))4
-->  + (+ 1 2)4
-->  + 3 4
  7
```

И в этом случае подвыражение `(sqrt 4)` вычисляется дважды.

После выписывания этих примеров, носящих наводящий характер, сформулируем следующую основную проблему, для преодоления которой потребуются определенные усилия:

> после связывания всех его переменных каждое выражение должно вычисляться максимум один раз.

Такое свойство вычисления выражений считается *полной ленивостью* вычислений.

Упражнение 9.4. Пусть имеется программа:

```
f = g 2
g x y = y * (КВАДРАТ x)
(f 3) * (f 1)
```

а) Записать программу в виде абстракции и произвести вычисления.

б) Скомпилировать программу и произвести вычисления.

Тема 10

Подвыражения

Содержание

10.1 Максимально свободные выражения

Для достижения полной ленивости нет необходимости производить вычисления тех выражений, которые не содержат (свободных) вхождений формального параметра.

Определение 10.1. Выражение E называется *собственным* подвыражением F, если и только если E является подвыражением F и E не совпадает с F.

Определение 10.2. Подвыражение E из абстракции считается *свободным* в ламбда-абстракции L, если все переменные E свободны в L.

Определение 10.3. . *Максимально свободное выражение*, или МСВ, в L — это такое свободное выражение, которое не является собственным подвыражением другого свободного выражения в L.

Пример 10.1. В приводимых ниже абстракциях подчеркнуты МСВ:

$$\begin{array}{ll} (1) & ([x].\underline{sqrt\ x}), \\ (2) & ([x].x\ \underline{(sqrt\ 4)}\ ) \\ (3) & ([y].[x].\underline{+(*\ y\ y)}\ x). \end{array}$$

Для обеспечения полной ленивости при выполнении β-редукции не следует означивать максимально свободные абстракции.

Пример 10.2. Вернемся к функции из примера 9.4:

```
--------------------
letrec f = g 4
    g = [x].[y].+ y (sqrt x)
    in + (f 1) (f 2)
```

Последовательность редукций начинается, как в этом примере:

```
+ (f 1)(f 2) -->
--> + (. 1)(. 2)
       .----.---> (([x].[y].+ y (sqrt x)) 4)
-->  + (. 1)(. 2)
        .----.---> ([y].+ y (sqrt 4))
```

(здесь: выражение `(sqrt 4)` является МСВ в `[y].`-абстракции, поэтому при приложении `[y].`-абстракции к аргументу `(sqrt 4)` не следует вычислять:

```
-->  + (. 1)(+ 2 .)
        .--------.---> ([y].+ y (sqrt 4)) )
```

Означивание имеет указатель на `(sqrt 4)` в теле абстракции:

```
+ (. 1)(+ 2 .)
   .---------.--->([y].+ y 2)
--> + (. 1)4
       .---> ([y].+ y 2)
--> + (+ 1 2)4
--> + 3 4
 7
```

В этом случае `(sqrt 4)` вычисляется только один раз.

Упражнение 10.1. Вычислить выражение из упражнения 9.4, воспользовавшись МСВ.

10.2 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))(sqrt x),
```

где `sqrtx` является именем экстрапараметра. Подставим полученное выражение в исходную абстракцию:

```
[x].([sqrtx].[y].+ y sqrtx)(sqrt x).
```

(3) Присвоим полученному суперкомбинатору имя:

```
$g1 = [sqrtx].[y].+ y sqrtx
---------------------------
[x].$g1 (sqrt x)
```

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

```
$g1 sqrtx y = + y sqrtx
$g x = $g1 (sqrt x)
$f = $g 4
$Prog = + ($f 1)($f 2)
----------------------
$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 10.2. Выполнить полностью ленивый ламбда-подъем программы, рассмотренной в упражнении 9.4.

10.3 Полностью ленивый ламбда-подъем с *letrec*

Рассмотрим программу:

```
-------------------------------------
let f = [x].letrec fac = [n].( ...  )
               in + x (fac 100)
in
    + (f 3)(f 4)
```

Алгоритм из подраздела 9.1 скомпилирует ее в:

```
$fac fac n = ( ...  )
$f x = letrec fac = $fac fac
          in + x (fac 100)
+ ($f 3)($f 4)
```

Функция `fac` определена локально в теле функции `f` и, следовательно, выражение `(fac 100)` не может быть поднято как МСВ из тела `f`. Это означает, что `(fac 100)` будет вычисляться заново всякий раз, когда выполняется `$f`. Таким образом, происходит потеря свойства полной ленивости.

Выход из создавшегося положения достаточно прост: достаточно заметить, что определение `fac` не зависит от `x`, поэтому можно вынести `letrec` для `fac`:

```
-------------------------
letrec
    fac = [n].( ...  )
    in let
    f = [x].+ x (fac 100)
in
    + (f 3)(f 4)
```

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

```
$fac n = ( ...  )
$fac100 = $fac 100
$f x = + x $fac100
$Prog = + ($f 3)($f 4)
----------------------
$Prog
```

В данном случае произведена некоторая оптимизация: выражение, не имеющее свободных переменных, не абстрагируется, а получает имя и становится суперкомбинатором — `$fac100`.

Таким образом, стратегия воплощается в два этапа:

1) вынесение `letrec-` (и `let-`) определений как можно "дальше",

2) выполнение полностью ленивого ламбда-подъема.

Упражнение 10.3. Выполнить полностью ленивый ламбда-подъем программы:

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

(здесь: `R` и `L` — константы).

10.4 Комплексный пример

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

[1] Дополнительную разработку техники ленивых вычислений с суперкомбинаторами см. в [106]. В этой же работе можно познакомиться с реализацией абстрактной машины, обеспечивающей поддержку всего спектра возможностей.

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

Приведенная программа в терминах абстракций записывается следующим образом:

```
-----------------------------------------------------
letrec
     SumInts = [n].foldl + 0 (count 1 n)
     count = [n].[m].IF (> n m) NIL
                        (cons n (count (+ n 1) m))
     foldl = [op].[base].[xs].IF (= xs NIL) base
               (foldl op (op base (head xs))(tail  xs))
in SumInts 100
```

В этой программе:

(1) внутренней абстракцией является `([xs]. ... )`,

(2) максимально свободными выражениями являются выражения `(fold op)`, `(op base)` и `base`.

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

```
$R1 p q base xs = IF (= xs NIL) base
                        (p (q (head xs)) (tail xs))
-----------------------------------------------------
letrec
    SumInts = [n].foldl + 0 (count 1 0)
    count = [n].[m].IF (> n m) NIL
                             (cons n (count (+ n 1) m))
    foldl = [op].[base].$R1 (foldl op) (op base) base
in
```

```
    SumInts 100
```

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

```
$R1 p q base xs = IF (= xs NIL) base
         (p (q (head xs))(tail xs))
$R2 r op base = r (op base) base
-------------------------------------------------
letrec
     SumInts = [n].foldl + 0 (count 1 n)
     count = [n].[m].IF (> n m) NIL
                        (cons n (count (+ n 1) m))
     foldl = [op].$R2($R1 op)op
in
     SumInts 100
```

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

```
$SumInts n = $foldlPlus0 ($count1 n)
$foldlPlus0 = $foldl + 0
$count1 = $count 1
$count n m = IF (> n m)NIL(cons n ($count (+ n 1) m))
$foldl op = $R2 ($R1 ($foldl op))op
$Prog = $SumInts 100
$R1 p q base xs = IF (= xs NIL) base
                     (p (q (head xs))(tail xs))
$R2 r op base = r (op base) base
-------------------------------------------------------
```

```
$Prog
```

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

10.5 Задача

Задача 10.1. Записать следующее определение исходного языка с помощью суперкомбинаторов:

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

Функция el выбирает n-й элемент из последовательности s.

Решение. В терминах λ-исчисления определение функции принимает вид:

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

Напомним алгоритм перевода λ-выражения в аппликативную форму. Возьмем произвольное λ-выражение $\lambda V.E$.

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

$SC{-}2$. Свободные переменные в λ-выражении идентифицируются литерами P, Q, …, R, затем λ-выражение снабжается префиксом ‘λ’, связывающим все эти переменные. Результатом служит:

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

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

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

$$alpha\ P\ Q\ \ldots\ R\ V \to E'.$$

Исходное λ-выражение заменяется формой

$$(alpha\ P\ R\ \ldots\ R).$$

Применительно к выражению аппликацию a можно сократить. Выражение E связано с V, а каждая свободная переменная принимает свое собственное значение в E'. 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:

$$alpha\ n\ el\ s \to if\quad \begin{array}{l}(=\ n\ 1)\ (hd\ s)\\ (el\ (-\ n\ 1)\ (tl\ s)).\end{array} \qquad (alpha)$$

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:

$$beta\ el\ n \to alpha\ n\ el, \qquad (beta)$$

откуда получаем, что

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

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

$$gamma\ el \to beta\ el, \qquad (gamma)$$

откуда получаем, что

$$el = Y\ gamma.$$

Ответ. $el = Y\ gamma$.

Контрольные вопросы.

1. Чем вызвано использование суперкомбинаторов для реализации редукции?

2. Что представляет собой язык константных аппликативных форм (КАФ)?

3. Какие специфические свойства комбинаторов S, K, I допускают их непосредственное использование в РГ-машине (машине редукции графа)?

Упражнение 10.4. Записать с помощью суперкомбинаторов выражение:

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

10.6 Ответы к упражнениям

7.1

`1 0` не имеет свободных переменных (см. Опр. 7.1, п.(1)) и символов абстракции (см. Опр. 7.1, п.(3)); `[x].+ x 1` имеет переменную `x`, которая является связанной (см. Опр. 7.1, п.(1)); `+ x 1` не содержит абстракций (см. Опр. 7.1, п.(2)); `[f].f([x].+ x x)` не имеет свободных переменных (см. Опр. 7.1, п.(1)), `[x].+ x x` является суперкомбинатором (см. Опр. 7.1, п.(2)).

7.2

`[x].x y z` содержит свободные переменные `y` и `z`, нарушается пункт (1) определения 7.1; `[x].[y].+ (+ x y) z` имеет свободную переменную `z`.

7.3

Например, `[x].[y].x y([z].z x)`.

7.4

Выражение `$XY 5` вычислить нельзя, так как оно не является редексом; `$XY 5 7` вычислимо, `$XY 3 4 7` вычислимо.

7.5

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`:

```
$Y v y = - y v
--------------
([x].$Y x x)5
```

(4) `[x].`-абстракция также является суперкомбинатором, ко-

торому можно дать имя и который можно сопоставить скомпилированному коду:

```
$Y v y = - y v
$X x = $Y x x
--------------
$X 5
```

Полученная программа выполняется следующим образом:

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

2) `([z].+ z (([x].([y]. * y x) x) 4)2`:

(1) внутренняя абстракция: `[y]. * y x`;

(2) вынесем `x` как экстрапараметр:

```
([x].[y]. * y x) x
([z].+ z(([x].([w].[y]. * y w) x x)4))2;
```

(3)

```
$Y w y = * y w
-------------------------
([z].+ z([x].$Y x x) 4) 2
```

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

```
$Y w y = * y w
$X x = $Y x x
------------------
([z].+ z ($X 4)) 2
```

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

```
$Y w y = * y w
$X x = $Y x x
$Z z = + z ($X 4)
-----------------
$Z 2
```

Выполнение программы:

```
$Z 2 = + 2 ($X 4) = + 2 ($Y 4 4) =
     = + 2 (* 4 4) = + 2 16 = 18.
```

9.1

`inf` находится на уровне 0 и не содержит внутренних абстракций. Поэтому `inf` уже является суперкомбинатором:

```
$inf v = letrec vs = cons 0 vs in vs
$Prog = $inf 4
-------------------------------------
$Prog
```

9.2

Запишем эту программу в терминах абстракций:

```
-----------------------------------------------------
letrec apply = [m].letrec constr = [n].(IF > n m) NIL
                                (cons n (constr (+ n 1)))
    in fold КВАДРАТ (constr 1)
       fold = [f].[ns].IF (= ns NIL) NIL
           (cons (f (head ns)) (fold (f (tail ns))) )
    in apply 5
```

(1) внутренняя абстракция имеет вид:

```
[n].IF (> n m) NIL (cons n (constr (+ n 1)));
```

(2) вынесем переменные `constr` и `m` в указанном порядке:

```
[constr].[m].[n].IF (> n m) NIL
              (cons n (constr (+ n 1))) constr m;
```

(3) присвоим полученному суперкомбинатору имя `$constr`:

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

(4) заменим вхождение `[n].`-абстракции в программу на выражение `($constr constr m n)`;

```
$constr constr m n = IF (> n m) NIL
                        (cons n (constr (+ n 1)))
-------------------------------------------------
letrec
     apply = [m].letrec
                        constr = $constr constr m
          in fold КВАДРАТ (constr 1)
     fold = [f].[ns].IF (= ns NIL) NIL
        (cons (f (head ns)) (fold f (tail  ns)) )
in apply 5
```

(5) выражения `apply` и `fold` являются суперкомбинаторами:

```
$constr constr m n = IF (> n m) NIL
                        (cons m (constr(+ n 1)))
$fold f ns = IF (= ns NIL) NIL
                (cons (f (head ns))
                      (fold (f (tail ns))))
$apply m = letrec constr = $constr constr m
          in $fold КВАДРАТ (constr 1)
$Prog = $apply 5
-----------------------------------------------
$Prog
```

9.3

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

```
$constr m n = IF (> n m) NIL
                         (cons n ($constr m (+ n 1)))
-----------------------------------------------------
letrec
     apply = [m].fold (КВАДРАТ) ($constr m 1)
     fold = [f].[ns].IF (= ns NIL) NIL
           (cons (f (head ns)) (fold (f (tail ns))) )
in apply 5
```

В данном случае и `apply`, и `fold` являются суперкомбинаторами:

```
$constr m n = IF (> n m) NIL
                         (cons n ($constr m (+ n 1)))
$fold f ns = IF (= ns NIL) NIL (cons
                                  (f (head ns))
                                  ($fold f (tail ns)) )
$apply m = $fold КВАДРАТ ($constr m 1)
$Prog = $apply 5
-----------------------------------------------------
$Prog
```

9.4

а) Запись в виде абстракции:

```
--------------------------------
letrec f = g 2
    g = [x].[y]. * y (КВАДРАТ x) in * (f 3)(f 1)
```

Вычисление выражения:

```
* (f 3)(f 1) -->
-->  (. 3)(. 1)
       .----.---> ([x].[y]. * y (КВАДРАТ x))2
--> * (. 3)(. 1)
        .----.---> ([y]. * y (КВАДРАТ 2))
--> * (. 3)(* 1 (КВАДРАТ 2))
        .---> ([y]. * y (КВАДРАТ 2))
--> * (. 3) 4
        .---> ([y]. * y (КВАДРАТ 2))
--> * ( * 3 (КВАДРАТ 2)) 4
--> * 12 4
--> 48.
```

б) Данное выражение компилируется в:

```
$g x y = * y (КВАДРАТ x)
$f = $g 2
$Prog = * ($f 3) ($f 1)
-----------------------
$Prog
```

Последовательность редукций:

```
$Prog -->  (* (. 3)(. 1))
                .----.---> ($g 2)
      --> * (. 3)(* 1 (КВАДРАТ 2))
             .---> ($g 2)
      --> * (. 3) 4
             .--->  ($g 2)
      -->  * (* 3 (КВАДРАТ 2)) 4
      -->  * (* 3 4) 4
      -->  * 12 4
      -->   48.
```

10.1

```
* (f 3)(f 1) -->
-->  * (. 3)(. 1)
       .----.---> (([x].[y].* y (КВАДРАТ x)) 2)
--> * (. 3)(* 1 .)
       .--------.---> ([y].* y (КВАДРАТ 2))
--> * (. 3) 4 (* 1 .)
       .-----------.---> ([y].* y 4)
--> * (. 3) 4
       .---> ([y].* y 4)
--> * (* 3 .) 4
          .--->  4
--> * 12 4
-->   48
```

Выражение `--> (КВАДРАТ 2)` вычисляется единственный раз.

10.2

Функция g содержит абстракцию:

```
[x].[y]. * y (КВАДРАТ x)
```

Далее:

(1) самой внутренней абстракцией является

```
[y]. * y (КВАДРАТ x);
```

(2) `(КВАДРАТ x)` представляет собой МСВ, которую вынесем в качестве экстрапараметра:

```
([КВАДРАТx].[y]. * y (КВАДРАТx)) (КВАДРАТ x)
```

Подставим полученное выражение в абстракцию:

```
[x].([КВАДРАТx].[y]. * y (КВАДРАТx)) (КВАДРАТ x)
```

(3) присвоим полученному суперкомбинатору имя:

```
$g1 = [КВАДРАТx].[y]. * y КВАДРАТx
---------------------------------
[x].$g1 (КВАДРАТ x)
```

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

```
$g1 КВАДРАТx y = * y КВАДРАТx
$g x = $g1 (КВАДРАТ x)
$f = $g 2
$Prog = * ($f 3)($f 1)
------------------------------
$Prog
```

10.3

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(- n 1)(tail s)))
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.

Тема 11

Оптимизации

Содержание

11.1 Ленивая реализация

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

11.2 Задачи

Задача 11.1. Для примера определения

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

где функция el возвращает n-ый элемент из списка (конечной последовательности) s, выбрать суперкомбинаторы, дающие полностью ленивую реализацию, и рассмотреть частный случай применения $el\ 2$.

Формулировка задачи. Введем некоторые определения. Оказывается, что те выражения, которые подвергаются повторным означиваниям, легко идентифицируются. Это относится и ко всякому подвыражению λ-выражения, которое не зависит от связанной переменной. Такие выражения называют свободными выражениями λ-выражения (по аналогии с введением понятия свободной переменной). Свободные выражения, которые не являются частью никакого другого большего свободного выражения, называют максимальными свободными выражениями λ-выражения (МСВ).

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

Замечание. Сначала установим, что такая схема трансляции значима, то есть получены настоящие комбинаторы и каждое λ-выражение заменено на аппликативную форму. Рассмотрим применимость новой схемы к отдельному λ-выражению, тело которого является аппликативной формой. Тот комбинатор, который при этом возникает, должен удовлетворять определению, так как его тело должно быть аппликативной формой и не должно содержать свободных переменных. Тело комбинатора заведомо будет аппликативной формой, поскольку оно в свою очередь возникло из аппликативной формы (тела исходного λ-выражения) путем

подстановки вместо некоторых выражений новых имен параметров. В нем не может быть свободных переменных, поскольку любая свободная переменная должна быть частью некоторого максимального свободного выражения и поэтому подлежит удалению как часть параметра. Все это подтверждает, что построен настоящий комбинатор. Окончательный результат, который замещает исходное λ-выражение, представляет собой новый комбинатор, приложенный к максимальным свободным выражениям, каждое из которых – уже аппликативная форма.

Решение. Вернемся к исходной задаче.

lazy–1. Пусть максимальные свободные выражения для исходного выражения

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

представляют собой

$$p \equiv (if(=\ n\ 1))\quad \text{и}\quad q \equiv (el(-\ n\ 1)).$$

Следовательно, новый комбинатор *alpha* определяется посредством

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

Отметим, что в этом случае

$$\lambda s.alpha\ p\ \ q\ s = alpha\ p\ \ q\ \ \to \lambda s.p\ (hd\ s)\ (q\ (tl\ s)).$$

Поскольку теперь

$$\begin{aligned} \lambda s.el\ n\ s &= \lambda s.if\ (=\ n\ 1)(hd\ s)(el(-\ n\ 1)(tl\ s)) \\ &= \lambda s.p\ (hd\ s)\ (q\ (tl\ s)) = alpha\ p\ \ q = el\ n, \end{aligned}$$

то

$$el\ n = alpha\ \underbrace{(if\ (=\ n\ 1))}_{p}\ \underbrace{(el\ (-\ n\ 1))}_{q}.$$

Отсюда вытекает следующий результат: определением функции el служит выражение

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

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

$$\begin{aligned} beta\ el\ n &\rightarrow \underbrace{alpha\ (if\ (=\ n\ 1))\ (el(-\ n\ 1))}_{el\ n}, \\ gamma\ el &\rightarrow beta\ el, \end{aligned}$$

откуда заключаем, что выражение el эквивалентно выражению $(Y\ gamma)$, то есть $el = Y\ gamma$, что совпадает с прежним результатом[1].

$lazy{-}2$. Рассмотрим теперь частный случай, когда применяется $(el\ 2)$:

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

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

$$el\ 2 \rightarrow alpha\ if\text{-}FALSE\ (alpha\ if\text{-}TRUE\ (el\ (-\ 1\ 1))).$$

[1] Действительно, $gamma\ el = el$, откуда $el = (\lambda f.(gamma\ f))\ el$. По теореме о неподвижной точке (см. теорему **??** на стр. **??**) $el = Y(\lambda f.(gamma\ f)) = Y\ gamma$.

Более подробно:

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

Рассмотренная схема приводит к субоптимальным комбинаторам.

Упражнения

Упражнение 11.1. Для выражения

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

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

11.3 Перестановка параметров

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

11.4 Задача

Задача 11.2. Для исходного определения:

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

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

Формулировка задачи. Напомним, что двумя основными критериями оптимальности порядка параметров комбинатора являются минимальное число MCB и максимальное устранение избыточных параметров. Оптимизируем наше определение последовательно по обоим критериям.

Замечание. Для максимизации размера и минимизации числа MCB следующего вложенного λ-выражения все MCB подвергаемого компиляции λ-выражения, которые в то же самое время являются свободными выражениями следующего вложенного λ-выражения, должны появиться прежде MCB, не обладающих указанным свойством. Оптимальное упорядочение параметров по этому критерию можно сформулировать следующим образом. Все E_i являются свободными выражениями λ-выражения, которое подлежит компилированию, однако оно может быть свободным выражением одного или большего числа вложенных λ-выражений. Назовем самое внутреннее λ-выражение, в котором выражение E_i не является свободным, *порождающим* λ-выражением. Если порождающее λ-выражение E_i включает порождающее λ-выражение E_j, то в оптимальном упорядочении E_i предшествует E_j. Отсюда не следует, что с необходимостью однозначно определяется оптимальное упорядочение, поскольку выражения с одним и тем же порождающим λ-выражением могут входить в любом порядке. Тем не менее всякое упорядочение, удовлетворяющее этому условию, является оптимальным, как и всякое другое.

Решение.

opt−1. Прежде всего примем во внимание результаты решения задачи 11.1 на стр. 186.

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 свя-

зывает n, то максимальные свободные выражения (MCB) из формы представляют собой

$$(alpha\ (hd\ s))\ \text{и}\ (tl\ s).$$

Если же выполнить переупорядочение формы вида

$$(alpha\ (hd\ s)\ (tl\ s)\ n),$$

то MCB 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 MCB представляет собой $(alpha\ n)$.

opt−4. Получив оптимальное упорядочение по одному критерию, обратимся к другому. В качестве "естественных" параметров можно принять

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

На компилятор возлагается задание так расположить параметры, чтобы происходило максимальное устранение избыточных параметров. У компилятора только тогда есть выбор, когда один комбинатор прямо определяется как вызов другого. В качестве примера рассмотрим редукцию:

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

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, суще-

ственно упрощает работу по нахождению избыточных параметров.

opt−5. Рассматриваемое выражение el определяется равенством:

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

У самого внутреннего λ-выражения имеются два МСВ:

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

Примем их за параметры p и q. Оба этих МСВ имеют одно и то же порождающее λ-выражение, поэтому их порядок несущественен, и $alpha$ можно определить редукцией:

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

поэтому

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

Теперь оказывается, что у следующего λ-выражения только одно МСВ и это el. Следовательно, $beta$ определяется редукцией

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

по которой

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

Далее следует применять редукцию

$$gamma\ el \rightarrow beta\ el,$$

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

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

Упражнения

Упражнение 11.2. Получить выражение с оптимальным порядком суперкомбинаторов для определения:

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

Вопросы для самопроверки

Попробуйте дать ответы на приводимые вопросы.

1. Определите понятия ‘ленивое означивание в λ-исчислении’ и ‘полная ленивость’ в языке КАФ и укажите связь между ними.
2. Что означает термин ‘МСВ λ-выражения’?
3. Что представляет собой окончательный результат замещения исходного λ -выражения при полностью ленивой реализации суперкомбинаторов?
4. Какие преимущества дает использование правил оптимизации?
5. Назовите два основных критерия оптимизации.
6. На что влияет порядок параметров суперкомбинаторов?

Часть IV

Абстрактные машины

Содержание

Тема 12

Механизмы вычислений

Содержание

В настоящем разделе техника вычисления значения выражений пересматривается в свете систематического построения набора синтактико-семантических равенств, которые реализуют избранную парадигму объектно-ориентированных вычислений.

12.1 Задача

Задача 12.1. 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:

$$\begin{array}{rlcl} (i) & \lambda x.x & = & I, \\ (ii) & \lambda x.c & = & Kc,\ c \not\equiv x, \\ (iii) & \lambda x.PQ & = & S(\lambda x.P)(\lambda x.Q). \end{array}$$

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’:

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

Проверить результат легко прямым выполнением β-редукции для λ-выражения:

$$M = (\lambda x.x(4,(\lambda x.x)3))+ \rightarrow +(4,(\lambda x.x)3) \rightarrow +(4,3) \rightarrow 7,$$

(еще раз обращаем внимание на использование другого символа операции сложения: ‘+’ вместо ‘$\oplus$’).

Упражнения

Упражнение 12.1. 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. $I = SKK$.

Упражнение 12.2. Для выражения:

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

постройте выражение комбинаторной логики и вычислите его значение.

Указание.

1. λ-выражение имеет вид:

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

2. Воспользуйтесь тем, что

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

Ответ. $S(KS)K\ sqr\ sin\ \pi/2 = 1$.

Вопросы для самопроверки

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

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

Тема 13

Теории вычислений

Содержание

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.

13.1 Задачи

Задача 13.1. 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.

Формулировка задачи. Известно, что в ходе выполнения λ-конверсии возникают коллизии переменных. Например, "прямое" выполнение β-редукции для $(\lambda xy.x)y$ могло бы дать $\lambda y.y$:

$$(\lambda xy.x)y = \lambda y.y,$$

что совершенно недопустимо, поскольку:

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

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 M представляет собой функцию $\|M\|$, ассоциирующую значение со средой. Представим обычные семантические равенства, где приложение

функции к аргументу представляется просто записью непосредственно вслед за символом функции символа аргумента:

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

где:

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

Вообще, формализм де Брейна может быть рассмотрен по аналогии с комбинаторной логикой с соответствующей адаптацией правил. Для осуществления перехода от обычных λ-выражений к кодировке переменных числами де Брейна рассмотрим ряд правил и соглашений.

Пусть среда env имеет вид

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

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:

$$\begin{aligned} \|\underline{0}\|(env, d) &= d, \\ \|\underline{(n+1)}\|(env, d) &= \|\underline{n}\|env, \\ \|c\|env &= c, \\ \|M\ N\|env &= \|M\|env(\|N\|env) \\ \|\lambda.M\|env\ d &= \|M\|(env, d). \end{aligned}$$

Интерес представляют не только значения сами по себе, а значения с точки зрения обеспечиваемых ими вычислений. При комбинаторном подходе подчеркивается, что значением, например, $M\ N$ служит комбинация значений M и N.
Вводится три комбинатора:

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

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

$$\begin{aligned} \|\underline{n}\| &= n!, \\ \|c\|env &= c, \\ \|M\ N\| &= \mathcal{S}(\|M\|, \|N\|), \\ \|\lambda.M\| &= \Lambda(\|M\|). \end{aligned}$$

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

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

Такие правила близки к SK-правилам: первые три указывают на “забывающее” аргумент свойство (наподобие комбинатора K);

четвертое – некаррированная форма правила S; пятое – в точности каррирование, то есть преобразование функции от двух аргументов в функцию от первого аргумента, задающую функцию от второго аргумента.

Введем также комбинатор пары $< \cdot, \cdot >$, где

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

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:

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

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

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

Решение.

$DB{-}1$. Пользуясь синтаксическими и семантическими ра-

венствами, получаем для $M = (\lambda x.x(4,(\lambda x.x)3))+$:

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

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

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

Приведем полную последовательность редукций:

$$(\Lambda(A), \Lambda(+ \circ Snd))() \to \varepsilon(\Lambda(A)(), \Lambda(+ \circ Snd)()) \to A\ env$$

здесь принято сокращение $env \equiv ((), \Lambda(+ \circ Snd)())$:
$\to \varepsilon(0!\, env, <' 4,\ B > env)$
$\to \varepsilon(\Lambda(+ \circ Snd)(), ('4\ env, B\ env))$
$\to \varepsilon(\Lambda(+ \circ Snd)(), (4, B\ env))$
$\to \varepsilon(\Lambda(+ \circ Snd)(), (4,\ \varepsilon(\Lambda(0!)env,' 3\ env)))$
$\to \varepsilon(\Lambda(+ \circ Snd)(), (4,\ \varepsilon(\Lambda(0!)env, 3)))$
$\to \varepsilon(\Lambda(+ \circ Snd)(), (4, 0!(env, 3)))$
$\to \varepsilon(\Lambda(+ \circ Snd)(), (4,3)) \to (+ \circ Snd)((), (4,3))$
$\to +(Snd((), (4,3))) \to +(4,3) \to 7.$

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

Упражнения

Упражнение 13.1. Постройте выражения де Брейна для приводимых далее λ-выражений.
1) $\lambda y.yx$. *Ответ.* $\|\lambda y.yx\| = \Lambda(\mathcal{S}(0!, 1!))$.
2) $(\lambda x.(\lambda z.zx)y)((\lambda t.t)z)$.

Указание. Обозначим исходное выражение через Q и перейдем к $R = \lambda zxy.Q$. 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 '$\lambda$' на '$\Lambda$', '$\circ$' 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).

Тема 14

Цикл работы категориальной абстрактной машины (КАМ)

Содержание

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 самостоя-

тельное 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.

14.1 Теоретические сведения

Аббревиатура ‘КАМ’ принята в качестве сокращения для ‘Категориальная Абстрактная Машина’. Как можно будет увидеть, такое название само по себе несет значительную смысловую нагрузку.

14.1.1 Структура КАМ

Сначала обоснуем введение самого термина ‘категориальная’. В *категориях* пользуются композицией и тождеством (тождественное преобразование служит целям оптимизации), в *декартовых категориях* дополнительно вводят произведение с использованием спаривания $< \cdot, \cdot >$, 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 выполнять означивание категориальных термов вида M', построенных из указанных комбинаторов путем аппликации (приложения) терма к среде, которая построена из совокупности различных компонентов:

1) комбинаторы аппликации и построения совокупности в M' не использованы;

2) аппликация возникает, когда записываем $M'()$, то есть когда категориальный терм прикладывается к (пустой) среде;

3) построение совокупности происходит всякий раз, когда используется правило ($dpair$).

Машинные инструкции

Машинные инструкции такой КАМ строятся следующим образом:

> статические операторы можно принять за базисные машинные инструкции.

Сперва отметим, что в применяемых при редукции $M'()$ правилах все редексы имеют вид Mw, где w – значение, то есть терм в нормальной форме по отношению к применяемым правилам. Терм преобразован по де Брейну, то есть является кодом де Брейна. Далее:

1. рассматриваем терм M как код, действующий на w, причем M образован из элементарных составляющих кода;

2. проекции Fst и Snd с легкостью причисляются к набору инструкций: Fst действует на значение (w_1, w_2), образуя доступ к первому порожденному элементу пары, если считать, что значение представлено в виде бинарного дерева;

3. для совокупностей рассматривается действие $< M, N >$ 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 M. Получается w_1. Перед началом обработки w его следует сохранить в памяти, чтобы иметь возможность восстановить w при означивании N, когда получается w_2. Наконец w_1 и

w_2 собираются в совокупность, но в предположении, что запомнено значение w_1, а это выполняется именно в тот момент, когда восстанавливаем w.

Структура машины

Исходя из рассмотренных соображений возникает *структура машины*:

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

Состоянием машины считается тройка $< T,\ C,\ S >$.

На основании предыдущего *кодом* для $< M, N >$ считается следующая последовательность инструкций:

‘$<$’ , за которой следует последовательность инструкций, соответствующая коду M, за которой следует ‘,’, за которой следует последовательность инструкций, соответствующая коду N, за которой следует ‘$>$’.

14.1.2 Инструкции

Отметим, что категориальная абстрактная машина (КАМ) совершает только символьные преобразования, компиляция нигде не проводится. Достигается это с помощью следующих инструкций.

Инструкции $<$, $>$

Представим действие инструкций ‘$<$’, ‘,’, ‘$>$’:

$<$: выталкивает терм на вершину стека,
, : меняет местами терм и вершину стека,
$>$: строит совокупность из вершины стека и терма, заменяет терм на эту совокупность и проталкивает стек.

Тот конкретный синтаксис, который используется для спаривания, в точности соответствует образованию управляющих инструкций. Эти инструкции должны замещать конструкцию спаривания. Тем самым достигается комбинирование означиваний M и N в означивание $< M, N >$.

Инструкции $Fst\ Snd\ \Lambda(C)$

Применительно к структуре машины проекции Fst и Snd можно описать более точно:

Fst : получает терм (s, t) и замещает его на s,
Snd : получает терм (s, t) и замещает его на t.

Код для $n!$ формируется из n и инструкции 'Fst', за которой следует инструкция 'Snd'. Для построения кода можно не предпринимать дополнительных усилий, если воспользоваться соглашениями:

примем $x\,y$ или $x|y$ для обозначения композиции, которая в обычной математической практике обозначается как $y \circ x$. При каррировании кодом $\Lambda(M)$ служит $\Lambda(C)$, где C — код M. Действие 'Λ' описывается следующим образом:

$\Lambda(C)$: замещает терм s на $C : s$, где C — код, инкапсулированный в Λ ;

обозначение $C : s$ служит сокращением для '$\Lambda(M)s$'. С точки зрения правил перезаписи действие 'Λ' пусто, поскольку $\Lambda(M)w$ является значением в силу того, что w — значение,

следовательно его нельзя перезаписать. Дадим перефразировку в терминах действий:

действием $\Lambda(M)$ на w является $\Lambda(M)w$.

Описание команды ‘Λ’ приводит к построению замыкания, как и в $SECD$-машине. Действительно, как подчеркивается самим обозначением $C : s$, в качестве значения обрабатывается совокупность, построенная из кода, соответствующего телу λ-выражения, и значения, которое представляет собой среду декларации функции, описанной посредством абстракции.

Инструкция $'$

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

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

Для конструкций типа встроенной функций, например, для символа сложения, используется кодирование, ранее уже рассмотренное:

кодом для $'(op)$ служит $\Lambda((op))$, где (op) — это инструкция Snd, за которой следует ‘op’.

Инструкция ε

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

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

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 КАМ 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 C.

Цикл работы КАМ

Прежде чем приступить к построению полной таблицы инструкций, вспомним соглашения об обозначениях для списков инструкций и элементов стека:

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

$a.L$ добавляет a в начало L;

$L1@L2$ присоединяет $L2$ к $L1$.

Для удобства дадим инструкциям мнемонические наименования в зависимости от их действия:

Fst	Snd	$<$	,	$>$	ε	Λ	$'$
car	cdr	$push$	$swap$	$cons$	app	cur	$quote$

Представим таблицу 14.1, 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.

Таблица 14.1: 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

Фактически, в этой таблице описана система программирования с небольшим числом исходных команд (инструкций). Заметим, что среди них нет инструкции условного вычисления (условного перехода). Эту инструкцию в дальнейшем приходится добавлять, пользуясь некоторыми соглашениями, касающимися представления о работе категориальной абстрактной машины. Кроме того, при вычислении рекурсивных определений приходится дополнительно обрабатывать (неявный) оператор неподвижной точки. Для этого снова требуются дополнительные соглашения.

14.2 Задачи

Задача 14.1. 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 $plus$, $\mathcal{S}(x,y) \equiv \mathcal{S}[x,y] = \varepsilon\circ < x,y >$, $\oplus \equiv (Snd+) : ()$.

Результирующие вычисления представим в табл. 14.2. 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.

Упражнения

Упражнение 14.1. Представить 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:

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

Таблица 14.2: 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	$[]$	$[]$

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

Ответ. 4.

Тема 15

Теория вычислений (продолжение)

Содержание

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.

15.1 Задача

Задача 15.1. 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 C – код M), за которой следует '$swap$', за которой следует код $C1$ для N, за которым следует '$cons$' и 'ε'. Предполагая, что означивание $C1$ на терме s дает значение w, приведем основные шаги вычисления:

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

В таблице 15.1 представлены вычисления значения.

Заметим, что

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

Рассмотрим средства получения оптимизированного кода. Кроме возможности означивания выражений относительно среды, как это уже рассматривалось, в рамках категориальной комбинаторной логики возможно весьма естественно

Таблица 15.1: 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

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:

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

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):

$$\begin{aligned}\lambda t.(ft, gt, kt) &= \lambda t \lambda r'.r'(\lambda r.r(ft)(gt))(kt) = \\ &= \lambda t.(< f, g > t, kt) = << f, g >, k >,\end{aligned}$$

следовательно:

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

Действительно,

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

Таким образом, для исходной задачи можно выделить следующие основные шаги вычисления:

s	$push.C1@cons.C$	S
s	$C1@cons.C$	$s.S$
w	$cons.C$	$s.S$
(s, w)	C	S

Такое оптимизированное вычисление использует комбинатор тождества, без которого используемую комбинаторную логику трудно назвать категориальной.

opt−4. Вернемся к вычислению на КАМ терма P. Воспользуемся обозначением $x \mid y \equiv y \circ x$ для упрощения компилируемой записи. Введем сокращения:

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

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

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

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

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

Кодирование по де Брейну дает:

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

затем

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

Далее вычисление значения P' дает:

$$\|P'\| \ = <' 5 > | \ \|(\lambda.(\lambda(\underline{1}\ \underline{0}) \times 2)1)(\lambda.\underline{0} + \underline{1})\|;$$

$$\begin{aligned} \|(\lambda.(\lambda(\underline{1}\ \underline{0}) \times 2)1)(\lambda.\underline{0} + \underline{1})\| = & \\ = \quad & < \|\lambda.(\lambda.(\underline{1}\ \underline{0}) \times 2)1\|, \|\lambda.\underline{0} + \underline{1}\| > \varepsilon \\ = \quad & < \Lambda\|\lambda.(\underline{1}\ \underline{0}) \times 2)1\|, \|\lambda.\underline{0} + \underline{1}\| > \varepsilon \\ \overset{(Beta)}{=} \quad & < \|\lambda.\underline{0} + \underline{1}\| > | \ \ \|(\lambda.(\underline{1}\ \underline{0}) \times 2)1\|; \end{aligned}$$

$$
\begin{array}{rcl}
\|(\lambda.(\underline{1}\ \underline{0}) \times 2)1\| & = & < \|\lambda.(\underline{1}\ \underline{0}) \times 2)\|,' 1 > \varepsilon \\
& = & < \Lambda\|(\underline{1}\ \underline{0}) \times 2\|,' 1 > \varepsilon \\
& = & <' 1 >|\ \|(\underline{10}) \times 2\|; \\
< \|\lambda.\underline{0} + \underline{1}\| > & = & < \Lambda\| + (\underline{0}, \underline{1})\| > \\
& = & < \Lambda <' +, < 0!, 1! >> \varepsilon > \\
& = & < \Lambda < \Lambda(+ \circ S), < 0!, 1! >> \varepsilon >; \\
& \overset{(\oplus)}{=} & < \Lambda(+ \circ < 0!, 1! >) > \\
& = & < \Lambda(< 0!, 1! >| +) > \\
& = & < \Lambda(< S, F \mid S >| +) >; \\
\|(\underline{1}\ \underline{0}) \times 2)\| & = & < \| \times \|, < \|(\underline{1}\ \underline{0})\|,' 2 >> \varepsilon \\
& = & < \Lambda(\times \circ S), < \|(\underline{1}\ \underline{0})\|,' 2 >> \varepsilon \\
& \overset{(\oplus)}{=} & \times \circ < \|(\underline{1}\ \underline{0})\|,' 2 > \\
& = & \times \circ << 1!, 0! > \varepsilon,' 2 > \\
& = & \times \circ << F \mid S, S > \varepsilon,' 2 > \\
& = & << F \mid S, S >| \varepsilon,' 2 >| \times.
\end{array}
$$

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

$$\|P'\| = <' 5 >|< \Lambda(D) >|<' 1 >|< C', 2 >|\ \times.$$

Для сокращения громоздкости KAM-вычислений в порядке соглашения обозначим:

$$B =< C,' 2 >| \times,\ C =< F \mid S, S >| \varepsilon,\ D =< S, F \mid S >| +.$$

Обращаем внимание на то обстоятельство, что приводимые переобозначения, очевидно, имеют связь с суперкомбинаторами. Действительно, каждый объект, введенный в употребление в виде математического соглашения об обозначениях, представляет собой вполне определенный набор KAM-инструкций. Этот набор может быть вычислен (заранее скомпилирован) на KAM, 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 KAM-вычислений задает последовательность состояний процесса вычисления[3].

[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.

Приведем полную последовательность КАМ-вычислений:

Терм	Код	Стек
$()$	$<' 5 >< \Lambda(D) ><' 1 > B$	$[]$
$()$	$'5 >< \Lambda(D) ><' 1 > B$	$[()]$
(5)	$>< \Lambda(D) ><' 1 > B$	$[()]$
$((), 5)$	$< \Lambda(D) ><' 1 > B$	$[]$
$((), 5)$	$\Lambda(D) ><' 1 > B$	$[((), 5)]$
$(D : ((), 5))$	$><' 1 > B$	$[((), 5)]$
$(((), 5), D : ((), 5))$	$<' 1 > B$	$[]$
$(((), 5), D : ((), 5))$	$'1 > B$	$[\ldots]$
(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)	$[]$	$[]$

Упражнения

Упражнение 15.1. Напишите оптимизированную программу для вычисления на KAM 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,

$$\begin{aligned}\mathcal{S}\|\lambda.(\ldots)(\cdots),3\| &= \mathcal{S}(\Lambda(\|(\ldots)(\cdots)\|),'3);\\ \|(\ldots)(\cdots)\| &= \mathcal{S}(\|(\ldots)\|,\|(\cdots)\|);\\ \|(\ldots)\| &= \Lambda(\mathcal{S}(\Lambda(\mathcal{S}(0!,1!)),\|sqr\|));\\ \|(\cdots)\| &= \mathcal{S}(\Lambda(\mathcal{S}(\mathcal{S}(\|+\|,1!),0!)),'1);\end{aligned}$$

В выражениях индексы будут указывать нумерацию парных скобок, например,

$$_0(\ldots)_0,\ldots,_6(\ldots)_6.$$

Запишем теперь

$$\begin{aligned}&\mathcal{S}_0(\Lambda_1(\mathcal{S}_2(\Lambda_3(\mathcal{S}_4(\Lambda_5(\mathcal{S}_6(0!,1!)_6)_5,\|sqr\|)_4)_3,\\ &\qquad\mathcal{S}_3(\Lambda_4(\mathcal{S}_5(\mathcal{S}_6(\|+\|,1!)_6,0!)_5)_4,'1)_3)_2)_1,'3)_0 \equiv R.\end{aligned}$$

Проделаем проверку, непосредственно выполнив вычисления:

$$\begin{aligned}R\rho &\equiv \mathcal{S}_0(\ldots,'3)_0\rho\\ &= (\Lambda_1(\ldots)_1\rho)('3\ \rho)\\ &= {}_1(\ldots)_1(\rho,3)\\ &\equiv {}_1(\ldots)_1\rho';\end{aligned}$$

$$
\begin{aligned}
{}_1(\ldots)_1\rho' &\equiv \mathcal{S}_2(\ldots\ ,\ \ldots)_2\rho' \\
&= \mathcal{S}_4(\ldots\ ,\ \ldots)_4(\rho',{}_4(\ldots)_4(\rho',1));
\end{aligned}
$$

$$
\begin{aligned}
{}_4(\ldots)_4(\rho',1) &= \mathcal{S}_5(\ldots\ ,\ \ldots)_5(\rho',1) \\
&= \mathcal{S}_6(\ldots\ ,\ \ldots)_6(\rho',1)1 \\
&= \Lambda(+\circ Snd)(\rho',1)\rho'\,1 \\
&= (+\circ Snd)((\rho',1),3)1 \\
&= +\,3\,1 = 4;
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{S}_4(\ldots\ ,\ \ldots)_4(\rho',4) &= \Lambda_5(\ldots)_5(\rho',4)(\Lambda_5(\cdots)_5(\rho',4)) \\
&= {}_5(\ldots)_5((\rho',4),(\Lambda_5(\cdots)_5(\rho',4))) \\
&\equiv \mathcal{S}(0!,1!)(\ldots\ ,\ \ldots) \\
&= \Lambda_5(\cdots)_5(\rho',4)4 \\
&= {}_5(\cdots)_5((\rho',4),4) \\
&\equiv (sqr\circ Snd)((\rho',4),4) \\
&= sqr(4) = 16.
\end{aligned}
$$

Полученное выражение R следует предварительно оптимизировать по правилам $(Beta)$ и $[\oplus]$. Для этого полезно предварительно использовать равенство:

$$\mathcal{S}(x,y) = \varepsilon\circ < x,y > .$$

Ответ. $sqr(4) = 16$.

Вопросы для самопроверки

1. Назовите альтернативные способы устранения коллизии переменных в λ-выражениях.

2. Обоснуйте необходимость введения оператора композиции.

3. Покажите связь и различие между понятиями ‘пара’ и ‘совокупность’.

Указание. Можно воспользоваться теоретико-множественными определениями для этих понятий, положив

$$f : D \to E, \quad g : D \to F.$$

Отсюда получаем, что

$$\begin{array}{rl} \text{совокупность} : & h =< f, g >: D \to E \times F; \\ \text{пара} : & [f, g] = (f, g) : (D \to E) \times (D \to F). \end{array}$$

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

5. Что представляют собой базисные машинные инструкции КАМ?

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

7. Сформулируйте основные правила оптимизации.

8. Какое преимущество дает использование принципов $(Beta)$ и $[\oplus]$ при написании КАМ-программ ?

Тема 16

Циклические вычисления

Содержание

16.1 Команды

16.1.1 Машинные инструкции

Машинные инструкции такой КАМ строятся следующим образом:

> статические операторы можно принять за базисные машинные инструкции.

Сперва отметим, что в применяемых при редукции $M'()$ правилах все редексы имеют вид Mw, где w — значение, то есть терм в нормальной форме по отношению к применяемым правилам. Терм преобразован по Дебрейну, то есть является кодом Дебрейна. Далее:

1. Рассматриваем терм M как код, действующий на w, причем M образован из элементарных составляющих кода;

2. Проекции Fst и Snd с легкостью причисляются к набору инструкций: Fst действует на значение (w_1, w_2), образуя доступ к первому порожденному элементу пары, если считать, что значение представлено в виде бинарного дерева;

3. Для совокупностей рассматривается действие $< M, N >$ 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 M. Получается w_1. Перед началом обработки w его следует сохранить в памяти, чтобы иметь возможность восстановить w при означивании N, когда получается w_2. Наконец w_1 и w_2 собираются в совокупность, но в предположении, что запомнено значение w_1, а это выполняется именно в тот момент, когда восстанавливаем w.

Структура машины

Исходя из рассмотренных соображений возникает *структура машины*:

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

Состоянием машины считается тройка $< T, C, S >$.

На основании предыдущего *кодом* для $< M, N >$ считается следующая последовательность инструкций:

‘$<$’ , за которой следует последовательность инструкций, соответствующая коду M, за которой следует ‘,’, за которой следует последовательность инструкций, соответствующая коду N, за которой следует ‘$>$’.

Подробности введения команд KAM и объяснение принципов их использования приведено в [**?**]. Далее для справки приводится их перечень.

$quote$: $((T), {}' c@C, [S]) \mapsto ((c), C, [S])$

${}'c$ Замещает содержимое терма на c, где c — инкапсулированная константа.

$push$: $((T), < @C, [S]) \mapsto ((T), <, [T; S])$

$<$ Эта инструкция проталкивает содержимое терма на вершину стека.

$swap$: $((T), , @C, [s_1; s_2]) \mapsto ((s_1), C, [T; s_2])$

, Меняет местами содержимое терма и вершину стека.

$cons$: $((T), > @C, [s_1; s_2]) \mapsto ((s_1, T), C, [s_2])$

$>$ Формирует пару из вершины стека и терма и замещает ею терм. Проталкивает стек.

car : $((t_1, t_2), Fst@C, [S]) \mapsto ((t_1), C, [S])$

Fst Выбирает первый элемент упорядоченной пары из терма и на него замещает терм.

cdr : $((t_1, t_2), Snd@C, [S]) \mapsto ((t_2), C, [S])$

Snd Выбирает второй элемент упорядоченной пары из терма и на него замещает терм.

cur : $((T), \Lambda(C_1)@C_2, [S]) \mapsto ((C_1 : T), C_2, [S])$

$\Lambda(C)$ Для терма T строит '$C : T$' и на него замещает терм.

app : $((C : t_1, t_2), \varepsilon@C_1, S) \mapsto ((t_1, t_2), C@C_1, S)$

ε Выбирает 'C' из терма $(C : t_1, t_2)$ и помещает его вместо ε. Замещает терм на (t_1, t_2).

$$\begin{aligned}(true, branch(C_1, C_2)@C, [s_1; s_2]) &\mapsto (s_1, C_1@C, s_2)\\ (false, branch(C_1, C_2)@C, [s_1; s_2]) &\mapsto (s_1, C_2@C, s_2)\end{aligned}$$

$branch$ Выполняет ветвление вычисления в зависимости от истинностного значения предиката.

$if \ldots then \ldots else$

Инструкция '$if\ M\ then\ P\ else\ Q$' замещается на инструкцию '$< m\ branch(p, q)$', где m, p, q – коды для M, P, Q соответственно.

rme : $((v, \ldots), \varepsilon@C_1, S]) \mapsto ((C : ((), v)), \ldots), \varepsilon@C_1, S)$

Для равенства $v = (C : ((), v))$ всякий раз, когда терм сопоставим с v, ожидается рекурсивная модификация среды – recursive modification of environment, – (rme).

16.1.2 Примеры исполнения

Для целей построения замкнутого изложения, относящегося в вычислению на KAM, приведем примеры использования команд. Их обсуждение см. в [14].

Пример 16.1.

$$\begin{aligned}\|+\ 4\ 3\| &= \varepsilon\circ < \|+4\|, \|3\| >\\ &= \varepsilon\circ < \varepsilon\circ < \|+\|, \|4\| >, \|3\| >\\ &= << ' +, ' 4 > \varepsilon, ' 3 > \varepsilon\end{aligned}$$

	1	()	$<<' +,' 4 > \varepsilon,' 3 > \varepsilon$	[]
	2	()	$<' +,' 4 > \varepsilon,' 3 > \varepsilon$	[()]
	3	()	$' +,' 4 > \varepsilon,' 3 > \varepsilon$	[(); ()]
	4	$+$	$,' 4 > \varepsilon,' 3 > \varepsilon$	[(); ()]
	5	()	$' 4 > \varepsilon,' 3 > \varepsilon$	[+; ()]
	6	4	$> \varepsilon,' 3 > \varepsilon$	[+; ()]
err	7	$(+, 4)$	$\varepsilon,' 3 > \varepsilon$	[()]
...	$7'$	$((): +, 4)$	$\varepsilon,' 3 > \varepsilon$	[()]
	8	$(+, 4)$	$,' 3 > \varepsilon$	[()]
	9	()	$' 3 > \varepsilon$	$[(+, 4)]$
	10	3	$> \varepsilon$	$[(+, 4)]$
err	11	$((+, 4), 3)$	ε	[]
...	$11'$	$((): (+, 4), 3)$	ε	[]
	12	$((+, 4), 3)$	[]	[]

Пример 16.2. В примере, который последует далее, использована каррированная версия $+$, то есть $\oplus \equiv \Lambda +$.

$$\begin{aligned} \| \oplus\ 4\ 3 \| &= \varepsilon \circ < \| \oplus 4 \|, \| 3 \| > \\ &= \varepsilon \circ < \varepsilon \circ < \| \oplus \|, \| 4 \| >, \| 3 \| > \\ &= << \Lambda(Snd\ \oplus),' 4 > \varepsilon,' 3 > \varepsilon \end{aligned}$$

Введем сокращение $(Snd \oplus) \equiv \textcircled{S}$.

1	$()$	$<< \Lambda\textcircled{S}, '4 > \varepsilon, '3 > \varepsilon$	$[]$
2	$()$	$< \Lambda\textcircled{S}, '4 > \varepsilon, '3 > \varepsilon$	$[()]$
3	$()$	$\Lambda\textcircled{S}, '4 > \varepsilon, '3 > \varepsilon$	$[(); ()]$
4	$(Snd \oplus) : ()$	$, '4 > \varepsilon, '3 > \varepsilon$	$[(); ()]$
5	$()$	$'4 > \varepsilon, '3 > \varepsilon$	$[(Snd \oplus) : (); ()]$
6	4	$> \varepsilon, '3 > \varepsilon$	$[(Snd \oplus) : (); ()]$
7	$((Snd \oplus) : (), 4)$	$\varepsilon, '3 > \varepsilon$	$[()]$
8	$((), 4)$	$Snd\ (\Lambda+), '3 > \varepsilon$	$[()]$
9	4	$(\Lambda+), '3 > \varepsilon$	$[()]$
10	$+ : 4$	$, '3 > \varepsilon$	$[()]$
11	$()$	$'3 > \varepsilon$	$[+ : 4]$
12	(3)	$> \varepsilon$	$[+ : 4]$
13	$(+ : 4, 3)$	ε	$[]$
14	$(4, 3)$	$+$	$[]$
15	7	$[]$	$[]$

Пример 16.3.

$$\begin{aligned} \| + [4, 3]| &= \varepsilon\circ < \| + \|, \|[4, 3]\| > \\ &= \varepsilon\circ < \| + \|, < \|4\|, \|3\| >> \\ &= <' +, <' 4, '3 >> \varepsilon \end{aligned}$$

	1	$()$	$<' +, <' 4, '3 >> \varepsilon$	$[]$
	2	$()$	$'+, <' 4, '3 >> \varepsilon$	$[()]$
	3	$+$	$, <' 4, '3 >> \varepsilon$	$[()]$
	4	$()$	$<' 4, '3 >> \varepsilon$	$[+]$
	5	$()$	$'4, '3 >> \varepsilon$	$[(); +]$
	6	4	$, '3 >> \varepsilon$	$[(); +]$
	7	$()$	$'3 >> \varepsilon$	$[4; +]$
	8	3	$>> \varepsilon$	$[4; +]$
	9	$(4, 3)$	$> \varepsilon$	$[+]$
err	10	$(+, (4, 3))$	ε	$[]$
...	10′	$((() : +, (4, 3))$	ε	$[]$
	11	$(+, (4, 3))$	$[]$	$[]$

Пример 16.4.

$$\begin{array}{rcl} \|+[4,3]| & = & \varepsilon\circ <\|+\|, \|[4,3]\| > \\ & = & \varepsilon\circ <\|+\|, <\|4\|, \|3\| >> \\ & = & <\Lambda(Snd+), <'\,4,'\,3 >> \varepsilon \end{array}$$

Введем сокращение $(Snd+) \equiv \textcircled{S}$.

1	$()$	$<\Lambda\textcircled{S}, <'\,4,'\,3 >> \varepsilon$	$[]$
2	$()$	$\Lambda\textcircled{S}, <'\,4,'\,3 >> \varepsilon$	$[()]$
3	$(Snd+) : ()$	$, <'\,4,'\,3 >> \varepsilon$	$[()]$
4	$()$	$<'\,4,'\,3 >> \varepsilon$	$[(Snd+) : ()]$
5	$()$	$'4,'\,3 >> \varepsilon$	$[(); (Snd+) : ()]$
6	4	$,'\,3 >> \varepsilon$	$[(); (Snd+) : ()]$
7	$()$	$'3 >> \varepsilon$	$[4; (Snd+) : ()]$
8	3	$>> \varepsilon$	$[4; (Snd+) : ()]$
9	$(4,3)$	$> \varepsilon$	$[(Snd+) : ()]$
10	$(\textcircled{S} : (), (4,3))$	ε	$[]$
11	$((), (4,3))$	$Snd+$	$[]$
12	$(4,3)$	$+$	$[]$
13	7	$[]$	$[]$

Упражнение 16.1. Выполните означивание выражения из упражнения 12.8 в [17a].

Решение. Исходное выражение можно переписать, воспользовавшись сокращениями, которые являются *подобъектами* и соответствуют *подпрограммам*:

$$\begin{array}{rcl} A & \equiv & \mathcal{S}[\Lambda(B), \Lambda(\circledast\circ Snd)] \text{ и} \\ B & \equiv & \mathcal{S}[\mathcal{S}[\underline{0}!, \underline{1}!], \underline{2}!] \end{array}$$

В данном случае $\circledast$ представляет собой каррированную версию умножения, то есть $\circledast \equiv \Lambda(*)$. Скомпилированным кодом является

$$P = \mathcal{S}[\Lambda(A),'\,3] = \varepsilon\circ <\Lambda(A),'\,3 >$$

Код для категориального вычислителя значения применительно к только что скомпилированному выражению имеет следующий вид:

$$< \Lambda(A),' 3 > |\varepsilon$$

Далее приведем пошаговое исполнение этого кода, то есть вычисление

$$(\varepsilon\circ < \Lambda(A),' 3 >)()$$

для пустой среды

$$\rho \equiv (),$$

которое записано в соответствии с соглашениями, принятыми в [?]:

1	$()$	$< \Lambda(A),' 3 > \|\varepsilon$	$[]$
2	$()$	$\Lambda(A),' 3 > \|\varepsilon$	$[()]$
3	$A : ()$	$,' 3 > \|\varepsilon$	$[()]$
4	$()$	$' 3 > \|\varepsilon$	$[A : ()]$
5	3	$> \|\varepsilon$	$[A : ()]$
6	$(A : (), 3)$	ε	$[]$
7	$((), 3)$	A	$[]$

Выражение сокращения для A использовано на шаге 7, который переписан теперь как 7':

$7'$	$((),3)$	$< \Lambda(B), \Lambda(Snd*) > \varepsilon$	$[]$
8	$((), 3)$	$\Lambda(B), \Lambda(Snd\circledast) > \varepsilon$	$[((), 3)]$
9	$B : ((), 3)$	$, \Lambda(Snd\circledast) > \varepsilon$	$[((), 3)]$
10	$((), 3)$	$\Lambda(Snd\circledast) > \varepsilon$	$[B : ((), 3)]$
11	$(Snd\circledast) : ((), 3)$	$> \varepsilon$	$[B : ((), 3)]$
12	$(B : ((), 3), s_2)$	ε	$[]$
13	$(((), 3), (Snd\circledast) : ((), 3))$	B	$[]$

В записанных далее вычислениях B является простым переписыванием шага 13, а в термах использованы следующие сокращения:

$s \equiv (((),3),(Snd\circledast) : ((),3)), s_2 \equiv (Snd\circledast) : ((),3))$:

$13'$	$(((),3),(Snd\circledast) : ((),3))$	$<< S, F \mid S > \varepsilon,' 2 > \varepsilon$	$[]$
14	$(((),3),(Snd\circledast) : ((),3))$	$S, F \mid S > \varepsilon,' 2 > \varepsilon$	$[s; s]$
15	$(Snd\circledast) : ((),3)$	$, F \mid S > \varepsilon,' 2 > \varepsilon$	$[s; s]$
16	s	$F \mid S > \varepsilon,' 2 > \varepsilon$	$[s_2; s]$
17	3	$> \varepsilon,' 2 > \varepsilon$	$[s_2; s]$
18	$((Snd\circledast) : ((),3),3)$	$\varepsilon,' 2 > \varepsilon$	$[s]$
19	$(((),3),3)$	$(Snd\circledast),' 2 > \varepsilon$	$[s]$
20	2	$> \varepsilon$	$[* : 3]$
21	$(* : 3, 2)$	ε	$[]$
22	$(3, 2)$	$*$	$[]$
23	6	$[]$	$[]$

16.2 Рекурсия

В настоящем разделе представим развернутый пример вычисления с рекурсивной модификацией среды. Дополнительные детали см. в [14].

Исходная программа:

```
letrec fact n = if n=0 then 1 else n*fact(n-1)
                in fact 1;;
let g = Y ([f].[n].if n=0 then 1 else n*f(n-1))
        in g 1
([g].g 1)(Y([f].[n].if n=0 then 1 else n*f(n-1)))
```

Скомпилированный код:

```
[f].[n].if n=0 then 1 else n*f(n-1) =
   = [f].[n].< =[n,0](branch[1,*[n,f(-[n,1])]])
   = [].[].< =[0,0](branch[1,*[0,1(-[0,1])]])
```

Сокращения:

$$\begin{aligned} B &= <= [\underline{0}, 0](branch[1, *[\underline{0}, \underline{1}(-[\underline{0}, 1])]]) \\ C &= *[\underline{0}, \underline{1}(-[\underline{0}, 1])] \end{aligned}$$

Означивание для $Snd = S, Fst = F, branch = b$:

$$\begin{array}{rcl} \|C\| & = & *\circ < \|\underline{0}\|, \varepsilon\circ < \|\underline{1}\|, -\circ < \|\underline{0}\|, '1 >>> \\ & = & < S, < F\ S, < S, '1 > - > \varepsilon > * \\ \|B\| & = & < \| = [\underline{0}, 0]\|b\|[1, C]\| \\ & = & <= \circ < S, '0 > b <' 1, \|C\| > \\ & = & << S, '0 >= b <' 1, \|C\| > \end{array}$$

Форма записи для кодогенерации:

```
if M then N else P  ⇒  <(code M)
                          branch [(code N),(code P)]
```

Генерация кода для исходной программы основывается на вычислении значения неподвижной точки для

$$\|Q\| = \|[].[].P\|$$

и оптимизации машинных инструкций:

$$\begin{array}{rcl} \|YQ\| & = & \|Q(YQ)\| = \varepsilon\circ < \|Q\|, \|YQ\| > \\ \|Y([].[].P)\| & = & \Lambda(\|P\|)\circ < Id, \|Y([].[].P)\| > \\ & = & \Lambda(\|P\|)\circ < \underline{\|Y([].[].P)\|} > \\ & = & \Lambda(\|P\|)\circ < \underline{\Lambda(\|P\|)\circ < \|Y([].[].P)\| >} > \\ & = & \Lambda(\|P\|)\circ < \Lambda(\|P\|) \\ & & \qquad \circ < \Lambda(\|P\|)\circ < \|Y([].[].P)\| > > \\ & & \qquad\qquad > \\ & = & \ldots \end{array}$$

Финальный скомпилированный код представляет собой:

$$\begin{array}{rcll} \|([].\underline{0}\ 1)(Y([].[].B))\| = & & & \\ & = & \varepsilon\circ < \Lambda(\varepsilon\circ < S, '1 >), \|Y([].[].B)\| > & \\ & = & \varepsilon\circ < S, '1 > \circ < Id, \|Y([].[].B)\| > & \text{по } (Beta) \\ & = & \varepsilon\circ < S, '1 > \circ < \underline{\|Y([].[].B)\|} > & \\ & = & \varepsilon\circ < S, '1 > \circ < \underline{\Lambda(\|B\|)\circ < \|Y([].[].B)\| >} > & \\ & = & << \|Y([].[].B)\| > \Lambda(\|B\|) >< S, '1 > \varepsilon & \end{array}$$

В записанной ранее цепочке равенств подчеркнутые выражения выведены на основании уравнения неподвижной точки. Пусть $\|B\|$ обозначено через $\mathbf{b}$, а $\|C\|$ — через $\mathbf{c}$.

Пример 16.5 (означивание инструкций на КАМ). Введем сокращение $\|Y([].[].B)\| \equiv ¥$. Означивание инструкций получается непосредственными вычислениями:

1	$()$	$<< ¥ > \Lambda(\mathbf{b}) >< S,'1 > \varepsilon$	$[]$
2	v	$> \Lambda(\mathbf{b}) >< S,'1 > \varepsilon$	$[(), ()]$
3	$((), v)$	$\Lambda(\mathbf{b}) >< S,'1 > \varepsilon$	$[()]$
4	$\mathbf{b} : ((), v)$	$>< S,'1 > \varepsilon$	$[()]$
5	$(), \mathbf{b} : ((), v)$	$< S,'1 > \varepsilon$	$[]$
6	$(), \mathbf{b} : ((), v)$	$S,'1 > \varepsilon$	$[((), \mathbf{b} : ((), v))]$
7	$\mathbf{b} : ((), v)$	$,'1 > \varepsilon$	$[((), \mathbf{b} : ((), v))]$
8	$((), \mathbf{b} : ((), v))$	$'1 > \varepsilon$	$[\mathbf{b} : ((), v)]$
9	1	$> \varepsilon$	$[\mathbf{b} : ((), v)]$
10	$(\mathbf{b} : ((), v), 1)$	ε	$[]$
11	$(((), v), 1)$	$\mathbf{b}$	$[]$

Инструкция $\mathbf{b}$ служит сокращением для $<< S,'0 >= b <' 1, \mathbf{c} >$, так что следующий шаг представляет собой непосредственное замещение:

12	$(((), v), 1)$	$<< S,'0 >= b <' 1, \mathbf{c} >$	$[]$
13	$(((), v), 1)$	$< S,'0 >= b <' 1, \mathbf{c} >$	$[(((), v), 1)]$
14	$(((), v), 1)$	$S,'0 >= b <' 1, \mathbf{c} >$	$[(((), v), 1), (((), v), 1)]$
15	1	$,'0 >= b <' 1, \mathbf{c} >$	$[(((), v), 1), (((), v), 1)]$
16	$(((), v), 1)$	$'0 >= b <' 1, \mathbf{c} >$	$[1, (((), v), 1)]$
17	0	$>= b <' 1, \mathbf{c} >$	$[1, (((), v), 1)]$
18	$(1, 0)$	$= b <' 1, \mathbf{c} >$	$[(((), v), 1)]$
19	$false$	$b <' 1, \mathbf{c} >$	$[(((), v), 1)]$

Условное выражение для $branch$ дает $false$, так что код замещается на правую часть альтернативы $\mathbf{c}$:

20	$(((), v), 1)$	$\mathbf{c}$	$[]$
21	$(((), v), 1)$	$< S, < F\ S, < S, '1 > - > \varepsilon > *$	$[]$
22	$(((), v), 1)$	$S, < F\ S, < S, '1 > - > \varepsilon > *$	$[(((), v), 1)]$
23	1	$, < F\ S, < S, '1 > - > \varepsilon > *$	$[(((), v), 1)]$
24	$(((), v), 1)$	$< F\ S, < S, '1 > - > \varepsilon > *$	$[1]$
25	$(((), v), 1)$	$F\ S, < S, '1 > - > \varepsilon > *$	$[(((), v), 1), 1]$
26	v	$, < S, '1 > - > \varepsilon > *$	$[(((), v), 1), 1]$
27	$(((), v), 1)$	$< S, '1 > - > \varepsilon > *$	$[v, 1]$
28	$(((), v), 1)$	$S, '1 > - > \varepsilon > *$	$[(((), v), 1), v, 1]$
29	1	$, '1 > - > \varepsilon > *$	$[(((), v), 1), v, 1]$
30	$(((), v), 1)$	$'1 > - > \varepsilon > *$	$[1, v, 1]$
31	1	$> - > \varepsilon > *$	$[1, v, 1]$
32	$(1, 1)$	$- > \varepsilon > *$	$[v, 1]$
33	0	$> \varepsilon > *$	$[v, 1]$
34	$(v, 0)$	$\varepsilon > *$	$[1]$

$$\ldots\ v = (\mathbf{b} : ((), v))\ \ldots$$

Введем сокращение $(((), v), 0) \equiv \emptyset$. Среда рекурсивно модифицируется:

35	$(\mathbf{b} : ((), v), 0)$	$\varepsilon > *$	$[1]$
36	$(((), v), 0)$	$\mathbf{b} > *$	$[1]$
37	$(((), v), 0)$	$<< S, '0 >= b <' 1, \mathbf{c} > > *$	$[1]$
38	$(((), v), 0)$	$< S, '0 >= b <' 1, \mathbf{c} > > *$	$[(((), v), 0), 1]$
39	$(((), v), 0)$	$S, '0 >= b <' 1, \mathbf{c} > > *$	$[\emptyset, \emptyset, 1]$
40	0	$, '0 >= b <' 1, \mathbf{c} > > *$	$[\emptyset, \emptyset, 1]$
41	$(((), v), 0)$	$'0 >= b <' 1, \mathbf{c} > > *$	$[0, \emptyset, 1]$
42	0	$>= b <' 1, \mathbf{c} > > *$	$[0, \emptyset, 1]$
43	$(0, 0)$	$= b <' 1, \mathbf{c} > > *$	$[0, \emptyset, 1]$
44	$true$	$b <' 1, \mathbf{c} > > *$	$[(((), v), 0), 1]$
45	$(((), v), 0)$	$'1\ > *$	$[1]$
46	1	$> *$	$[1]$
47	$(1, 1)$	$*$	$[]$
48	1	$[]$	$[]$

Все это и дает окончательный результат.

Для означивания $2! = 2$ будем пользоваться нумерацией строк из программы, исполненной в предыдущем примере. Отметим, что

$$([g].g\ 2)(Y(\dots)) = 2!$$

Пример 16.6. Означивание инструкций выполняется непосредственно:

$$\begin{array}{r|c|l|l}
1 & () & << \|Y([].[].B)\| > \Lambda(\mathbf{b}) >< S,'2 > \varepsilon & [] \\
 & & \dots & \\
11 & (((),v),2) & \mathbf{b} & [] \\
12 & (((),v),2) & << S,'0 >= b <' 1, \mathbf{c} > & []
\end{array}$$

Введем сокращения $(((),v),0) \equiv \emptyset$ и $(((),v),1) \equiv 1$. Это вычисление в результате последовательного означивания инструкции за инструкцией приводит к следующему условному выражению:

$$\begin{array}{r|c|l|l}
18 & (2,0) & = b <' 1, \mathbf{c} > & [((((),v),2)] \\
19 & false & b <' 1, \mathbf{c} > & [((((),v),2)] \\
20 & (((),v),2) & \mathbf{c} & [] \\
21 & (((),v),2) & < S, < F\ S, < S,'1 > - > \varepsilon > * & [] \\
 & & \dots & \\
31 & 1 & > - > \varepsilon > * & [2,v,2] \\
32 & (2,1) & - > \varepsilon > * & [v,2] \\
33 & 1 & > \varepsilon > * & [v,2] \\
34 & (v,1) & \varepsilon > * & [2]
\end{array}$$

$$\dots\ v = (\mathbf{b} : ((),v))\ \dots$$

Рекурсивная модификация среды дает следующий результат:

35	$((\mathbf{b} : ((), v)), 1)$	$\varepsilon > *$	$[2]$
36	$(((), v), 1)$	$\mathbf{b} > *$	$[2]$
37	$(((), v), 1)$	$<< S,' 0 >= b <' 1, \mathbf{c} > > *$	$[2]$
		$\ldots$	
41	$(((), v), 1)$	$'0 >= b <' 1, \mathbf{c} > > *$	$[1, \not{1}, 2]$
42	0	$>= b <' 1, \mathbf{c} > > *$	$[1, \not{1}, 2]$
43	$(1, 0)$	$= b <' 1, \mathbf{c} > > *$	$[\not{1}, 2]$

Происходит передача управления к инструкции 19:

19	$false$	$b <' 1, \mathbf{c} > > *$	$[\not{1}, 2]$
20	$(((), v), 1)$	$\mathbf{c} > *$	$[2]$
21	$(((), v), 1)$	$< S, < F\ S, < S,' 1 > - > \varepsilon > *\ > *$	$[2]$
		$\ldots$	
34	$(v, 0)$	$\varepsilon > *\ > *$	$[1, 2]$
		$\ldots\ v = (\mathbf{b} : ((), v))\ \ldots$	
35	$(\mathbf{b} : ((), v), 0)$	$\varepsilon > *\ > *$	$[1, 2]$
36	$(((), v), 0)$	$\mathbf{b} *\ > *$	$[1, 2]$
		$\ldots$	
43	$(0, 0)$	$= b <' 1, \mathbf{c} >>\ *\ > *$	$[\emptyset, 1, 2]$
44	$true$	$b <' 1, \mathbf{c} >>\ *\ > *$	$[\emptyset, 1, 2]$
45	$(((), v), 0)$	$'1 >\ *\ > *$	$[1, 2]$
46	1	$>\ *\ > *$	$[1, 2]$
47	$(1, 1)$	$*\ > *$	$[2]$
48	1	$> *$	$[2]$

Означивание оставшегося стека:

49	$(1, 2)$	$*$	$[]$
50	2	$[]$	$[]$

Таким образом, в последней строке в позиции терма — крайняя левая колонка, — оказался записанным ожидаемый финальный результат вычисления $2! = 2$.

Тема 17

CAML, F♯ и примеры программ

Содержание

17.1 Дополнительные вопросы

Структура семантических средств многоуровневой концептуализации

В настоящем разделе продолжается обсуждение формирования объяснительной системы для формализаций информационных систем (ИС), начатое в главе 8 и, в особенности, в подразделе 8.8 из [17a].

Многоуровневая концептуализация

Вычисление в категории дает возможность построения иерархии объектов разной степени детализации/агрегированности. Подробности см. в (В. Э. Вольфенгаген, [5]). Основная идея состоит в обеспечении модульности совокупностей индивидов, характеризующейся относительной независимостью каждого (мета-)уровня от других. Индивиды данного уровня являются инвариантами по отношению к нижележащему более детальному уровню. В лежащий выше более абстрактный уровень входят инварианты рассматриваемого уровня. Нижележащий уровень реализуется учетом соотнесений, а вышележащий — применением шага концептуализации в виде (неограниченного) свертывания. Для уровня j концепты строятся как индивиды уровня $j+1$:

$$T^{j+1} \equiv \{h^j : \underbrace{[\ldots[}_{j\text{ раз}} D]\ldots] \mid \Phi^j\}$$
$$\equiv \iota y^{j+1} : \underbrace{[\ldots[}_{j\text{ раз}} D]\ldots]. \forall h^j : \underbrace{[\ldots[}_{j-1\text{ раз}} D]\ldots] (\Phi^j \leftrightarrow y^{j+1}(h^j)).$$

Концептам уровня $j+1$ соответствуют предикатные символы, вводимые определениями. Базу объектов данных уровня j образуют отношения, определенные как подмножествах декартовых произведений индивидов уровня j.

Локальная реляционная полнота на уровне j обеспечивается, поскольку на этот уровень переносятся вое обычные свойства традиционной реляционной базы данных с языковыми средствами в виде реляционной алгебры или реляционного исчисления.

Глобальная реляционная полнота многоуровневой модели обеспечивается выбором подходящей системы соотнесений. При таком выборе рассматривается семейство образов предметных областей ПО^0 (индивиды), ПО^1 (концепты), ПО^2 (концепты концептов), ПО^3 (концепты концептов концептов), ...

Для уровня 0 состояния соответствуют ролям, концепты (по-

нятия) — типам (атрибутам), соотнесения по ситуациям — фреймам, соотнесения по "мирам" — (реляционной) базе данных.

Для уровня 1 состояния соответствуют индивидам уровня 0, концепты — мета^{1}понятиям, соотнесения по мета^{1}ситуациям соответствуют мета^{1}фреймам, соотнесения по "мета^{1}мирам" — базе знаний (базе мета^{1}данных).

Для уровня 2 состояния соответствуют концептам (индивидам ПО^1), концепты — мета^{2}понятиям, соотнесения по мета^{2}ситуациям — мета^{2}фреймам, соотнесения по мета^{2}мирам — базе метазнаний.

Для более высоких уровней содержательную интерпретацию выбирают, исходя из прагматических соображений, связанных с особенностями применения АИС.

В выборе структуры семантических средств с системой соотнесений основное внимание уделим выработке принципов семантики, обеспечивающих построение вычислительных моделей. Поскольку основной упор делается на АВС, то наибольший интерес представляют способы вычисления значения аппликации и упорядоченной пары. Выбор принципов производим с учетом соотнесений. Другим критерием служит согласованность стандартных средств семантики λ-исчисления типизированных объектов данных с полученным обобщением.

Принцип вычисления значения аппликации

Основная посылка состоит в следующем:

значение аппликации равно аппликации значений.

Данная формулировка нуждается в уточнении, поскольку вычисление значения можно производить как при заранее зафиксированном соотнесении, так и при нефиксированном соотнесении.

В первом случае примем:

$$\|MN\|i = (\|M\|i)(\|N\|i)$$

для произвольных термов M, N. Тогда

$$\begin{array}{rcl}(\|M\|i)(\|N\|i) & = & (\lambda r.r\|M\|\,\|N\|)Si \\ & = & CIS(\lambda r.r\|M\|\,\|N\|) \\ & \equiv & \mathcal{S}[\|M\|,\|N\|]i\end{array}$$

для $S \equiv \lambda xyz.xz(yz)$, $C \equiv \lambda xyz.xzy$, $I \equiv \lambda x.x$, $\mathcal{S} = CIS$ и упорядоченной пары $[x,y] \equiv \lambda r.rxy$. Следовательно, сформулируем правило

(правило 1) $$\|MN\| = \mathcal{S}[\|M\|,\|N\|]$$

которое выводимо в $\lambda\eta\xi$-исчислении.

Принцип вычисления значения упорядоченной пары

Основную посылку сформулируем в виде равенства

> значение упорядоченной пары равно упорядоченной паре значений.

Формально это равенство перепишем в виде

$$\|[M,N]\|i = [\|M\|i,\|N\|i]$$

для произвольных термов M, N и соотнесения i. С использованием постулатов (η), (ξ) и (τ) λ-исчисления, из исходного принципа выводимо правило:

(правило 2) $$\|[M,N]\| =< \|M\|,\|N\| >,$$

где $< f,g >\equiv \lambda t.[ft,gt]$ для произвольных f, g. Оба сформулированных принципа и выведенные из них правила позволяют получить и обосновать стандартные семантические средства вычислительных моделей. В качестве наиболее важных для средств концептуализации укажем два следствия, связанные с вычислением значений λ-выражений и аппликаций.

Вычисление значения λ-выражения

Рассмотрим аппликацию $(\lambda.M)\overline{d}$, где $M, \overline{d}$ – произвольные термы, а $(\lambda.M)$ обозначает абстракцию по (некоторой) переменной. Тогда справедлива цепочка равенств:

$$\begin{array}{rcll} \|(\lambda.M)\overline{d}\|i & = & (\|(\lambda.M)\|i)(\|\overline{d}\|i) & \text{(по принципу 1)} \\ & = & (\|\lambda.M\|i)di & \text{(полагаем } (\|\overline{d}\|i) = d) \\ & = & \Lambda\|M\|id & \text{(по определению } \Lambda) \\ & = & \|M\|\,[i,d] & \text{(в силу } \Lambda h = \lambda xy.h[x,y]). \end{array}$$

Следовательно, справедливо правило:

$$\text{(правило 3)} \qquad \|(\lambda.M)\overline{d}\|i = \|M\|\,[i,d].$$

В дальнейшем иногда будем считать, что это правило определяет производящую функцию: вычисление значения M “производит” подстановки d, удовлетворяющие M.

Второй способ вычисления значения аппликации

При вычислении значения аппликации воспользуемся определением аппликатора ε:

$$\begin{array}{rcll} \|MN\|i & = & (\|M\|i)(\|N\|i) & \text{(в силу принципа 1)} \\ & = & \varepsilon[\|M\|i, \|N\|i] & \text{(по определению } \varepsilon) \\ & = & (\varepsilon\circ < \|M\|, \|N\| >)i & \text{(по определению } < \cdot,\cdot >) \end{array}$$

Из этой цепочки равенств выводимо правило

$$\text{(правило 4)} \qquad \|MN\| = \varepsilon\circ < \|M\|, \|N\| > .$$

Наконец, выделим случай вычисления значения индивидных констант:

$$\|c\|i = c,$$

откуда выводимо (в $\lambda\eta\xi$-исчислении) правило

(правило 5) $$\|c\| = \lambda i.c.$$

Индивидные константы не зависят от конкретного соотнесения, то есть ведут себя статично.

Список правил

Для справки представим полный список полученных правил:

$$\begin{array}{lrcl}
\text{(правило 1)} & \|MN\| & = & \mathcal{S}[\|M\|, \|N\|] \\
\text{(правило 2)} & \|[M, N]\| & = & < \|M\|, \|N\| > \\
\text{(правило 3)} & \|(\lambda.M)\overline{d}\|i & = & \|M\|\,[i, d] \\
\text{(правило 4)} & \|MN\| & = & \varepsilon\circ < \|M\|, \|N\| > \\
\text{(правило 5)} & \|c\| & = & \lambda i.c.
\end{array}$$

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] Булкин М.А., Габович Ю.Р., Пантелеев А.Г. *Методические рекомендации по программированию и эксплуатации интерпретатора для алгоритмического языка ЛИСП в ОС ЕС*. – Киев : НИИАСС, 1981. – 91 с. 1.4, 17.1

[6] Вольфенгаген В.Э. *Вычислительная модель реляционного исчисления, ориентированного на представление знаний*.– М.: препринт МИФИ, 004-84, 1984. 1.4, 3.1, 6.5, 6.8

[7] Вольфенгаген В.Э., Яцук В. Я. *Вычислительная модель реляционной алгебры.* – Программирование, № 5, М.: АН СССР, 1985. – с. 64-76. 1.4

[8] Вольфенгаген В.Э., Сагоян К.А. *Методические указания к проведению практических занятий по курсу "Дискретная математика". Специальные главы дискретной математики.* – М.: МИФИ, 1987. – 56 с. 1.4

[9] Вольфенгаген В.Э., Аксенов К.Е., Исмаилова Л. Ю., Волшаник Т. В. *Лабораторный практикум по курсу "Дискретная математика. Аппликативное программирование и технология поддержки реляционных систем".*– М.: МИФИ, 1988 . – 56 с. 1.4, 17.1

[10] Вольфенгаген В.Э., Яцук В.Я. *Аппликативные вычислительные системы и концептуальный метод проектирования систем знания.* – МО СССР, 1987. 1.4

[11] Вольфенгаген В.Э., Чепурнова И.В., Гаврилов А.В. *Методические указания к проведению практических занятий по курсу "Дискретная математика". Специальные главы дискретной математики.*– М.: МИФИ, 1990. – 104 с. 1.4

[12] Вольфенгаген В.Э., Гольцева Л.В. *Аппликативные вычисления на основе комбинаторов и λ-исчисления.*– (Руководитель проекта "Аппливативные вычислительные системы" к.т.н. Л.Ю. Исмаилова.)– М.: МИФИ, 1992. – 41 с.
Основы аппликативных вычислительных систем изложены элементарными средствами, что обеспечивает студентов и аспирантов кратким и исчерпывающим руководством, которое может использоваться 'для первого чтения'. Настоящее руководство в различных вариантах течение ряда лет использовалось для проведения практический занятий и лабораторных работ по соответству-

ющим разделам курса компьютерных наук. Охватываются вопросы использования комбинаторов и λ-исчисления при реализации аппликативных вычислений. Приводится необходимый теоретический минимум, основное внимание уделено выполнению упражнений, иллюстрирующих применения основных вычислительных идей, понятий и определений. Для облегчения овладения предметом руководство снабжено простой обучающей программой, которая может использоваться в качестве вводного лабораторного практикума. При практической работе с обучающей программой следует иметь ввиду, что решение задач предполагает проведение дополнительных преобразований с целью оптимизации выражений, устранения в них переменных, упрощения целевого исполняемого выражения. Аппликативные вычисления излагаются как набор таких методов и средств.
Для студентов и аспирантов всех специальностей. Может быть использована для первоначального самостоятельного изучения предмета. 1.4

[13] Вольфенгаген В.Э. *Теория вычислений*. — М.:МИФИ, 1993. — 96 с. 1.4

[14] Вольфенгаген В.Э. *Категориальная абстрактная машина*.— М.:МИФИ, 1993. — 96 с.; — 2-е изд. — М.: АО "Центр ЮрИнфоР", 2002. — 96 с.
Основное внимание уделено подробному рассмотрению техники вычисления значения конструкций языков программирования, включая компилирование кода, его оптимизацию и исполнение на примере категориальной абстрактной машины. Изложение построено на примерах возрастающей сложности. 1.4

[15] Вольфенгаген В.Э. *Проектирование языков программирования и теория вычислений*.— М.:МИФИ, 1993. — 189 с.

[15a] Вольфенгаген В. Э. *Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах.* — М.: МИФИ, 1994. — 204 с.; 2-е изд., М.: АО «Центр ЮрИнфоР», 2003. — 336 с.

Изложен основной круг задач, сводимых к исчислению объектов — "от простого к сложному". Конкретный вариант исчисления выбирается в зависимости от решаемых вычислительных задач. В ходе последовательного решения задач читатель овладевает основными методами и средствами комбинаторной логики и λ-исчисления. Все задачи снабжены подробными и элементарными решениями.

Для студентов старших курсов и аспирантов, изучающих математические основы объектно-ориентированных вычислений, начинающих и профессионально работающих над продвинутыми проектами программистов. Может быть использована в курсах дискретной математики, информатики, теории программирования.

Предварительной математической подготовки не требуется. Материал частично или полностью может быть использован для самостоятельного изучения как книга "для первого чтения". 17.1

[16] Вольфенгаген В.Э. *Конструкции языков программирования. Приемы описания.*— М.: АО "Центр ЮрИнфоР", 2001. — 276 с.

В работе изложены основы, касающихся разработки, реализации и применения конструкций как императивных, так и функциональных языков программирования. Значительное внимание уделяется применению денотационной семантики, позволяющей в полной мере извлечь преимущества объектно-ориентированного подхода, что, в конечном счете, позволяет построить результирующую вычислительную модель чисто функционального типа.

Изложение материала сопровождается детально разобранными примерами, которые снабжены комментариями, помогающими уяснить реализацию конструкций различных языков.

Книгу можно использовать в качестве учебника или справочного

пособия. Она будет полезна как студентам и аспирантам, так и профессионалам в области компьютерных наук, информационных технологий и программирования.

[17] Вольфенгаген В.Э. *Логика. Конспект лекций: техника рассуждений.*— М.: АО "Центр ЮрИнфоР", 2001. — 137 с.; — 2-е изд., доп. и перераб. — М.: АО "Центр ЮрИнфоР", 2004. — 229 с.

★ Книга стала победителем в номинации "Лучшее учебное издание по точным наукам" на III Общероссийском конкурсе учебных изданий для высших учебных заведений "Университетская книга 2006"

Настоящее издание значительно переработано и расширено элементами техники семантических рассуждений с применением классов и отношений, что особенно важно для работы с электронными формами информации. Рассмотрены способы переформулирования текста фактического типа на символьный язык, допускающий применение классических логических средств. Показаны приемы и способы записи аргументации и проверки ее значимости. На большом числе примеров проиллюстрирована техника логических рассуждений, выводов и доказательств. Отмечены способы включения в вывод аннотаций (комментариев), пользуясь которыми можно проверить истинность или установить ложность приводимых доводов.

Для студентов и аспирантов гуманитарных специальностей. Может быть использована для первоначального изучения предмета, а также для самостоятельного изучения. 16.1.2, 16.2

[17a] Вольфенгаген В.Э. *Методы и средства вычислений с объектами. Аппликативные вычислительные системы.* — М.: JurInfoR Ltd., АО «Центр ЮрИнфоР», 2004. — xvi+789 с.

★ Книга отмечена дипломом Фонда развития отечественного образования на конкурсе 2005 г.

Систематически рассмотрены модели, методы и средства, для которых центральной сущностью является представление об объекте. Применен подход, основанный на использовании операций аппликации и абстракции, что позволило выполнить замкнутое изложений техники аппликативных вычислений, оставаясь в рамках элементарных средств. Книга основана на материале, который в различных вариантах использовался для проведения занятий по соответствующим разделам курса компьютерных наук. Приводится необходимый теоретический минимум, соответствующий мировым стандартам, иллюстрируются основные вычислительные идеи, понятия и определения.

[18] Голдблатт Р. *Топосы: категорный анализ логики.*— М.: Мир, 1983.

[19] Джонстон П. Т. *Теория топосов.*— М.: Наука, 1986. 1.4

[20] Захарьящев М.В., Янов Ю.И. (ред.) *Математическая логика в программировании.*— М.:Мир, 1991. — 408 с.

[21] Илюхин А.А., Исмаилова Л.Ю., Шаргатова З.И. *Экспертные системы на реляционной основе.*— М.: МИФИ, 1990.

[22] Карри Х.Б. *Основания математической логики.*— М.: Мир, 1969.

[23] Клини С.К. *Введение в метаматематику.*— М.: ИЛ, 1957.

[24] Кузин Л.Т. *Основы кибернетики, т.2.* — М.: Энергия, 1979, 15-9. 16.1, 17.1

[25] Кузин Л.Т. *Основы кибернетики. Т.1. Математические основы кибернетики: Учеб. пособие для вузов.* — 2-е изд., перераб. и доп. — М.: Энергоатомиздат, 1994. — 576 с.

[26] Кузичев А.С. *Некоторые свойства комбинаторов Шейнфинкеля-Карри.*— Комбинаторный анализ, вып. 1. Изд-во МГУ, 1971.

[27] Кузичев А.С. *Дедуктивно-комбинаторное построение теории функциональностей.*— ДАН СССР, 1973, т. 209, № 3.

[28] Кузичев А.С. *Непротиворечивые расширения чистой комбинаторной логики.*— Вестн. Моск. ун-та, матем., механ., №3, 76-81, 1973.

[29] Кузичев А.С. *О предмете и методах комбинаторной логики.*— История и методология естественных наук, М.:МГУ, вып.14, 1973, с. 131-141.

[30] Кузичев А.С. *Система ламбда-конверсии с дедуктивным оператором формальной импликации.*— ДАН СССР, 212, № 6, 1973, 1290-1292.

[31] Кузичев А.С. *Дедуктивные операторы комбинаторной логики.*— Вестн. Моск. ун-та, матем., механ., № 3, 13-21, 1974.

[32] Кузичев А.С. *О выразительных возможностях дедуктивных систем ламбда-конверсии и комбинаторной логики.*— Вестн. Моск. ун-та, матем., механ., № 6, 19-26, 1974.

[33] Кузичев А.С. *Принцип комбинаторной полноты в математической логике.*— История и методология естественных наук, сб. МГУ, вып. 16, 1974. — с. 106-127.

[34] Кузичев А.С. *Комбинаторно полные системы с операторами* Ξ, F, Q, Π, $\exists$, P, $\neg$, $\&$, $\vee$, $=$.— Вестн. Моск. ун-та, сер. матем., мех., № 6, 1976.

[35] Кузичев А.С. *Операция подстановки в системах с неограниченным принципом комбинаторной полноты.*— Вестн. Моск. ун-та, сер. матем., мех., № 5, 1976.

[36] Мальцев А.И. *Алгоритмы и рекурсивные функции.*— М.: Наука, 1965.

[37] Марков А.А. *Невозможность некоторых алгорифмов в теории ассоциативных систем.*— ДАН СССР, 1947, т. 55, № 7; т. 58, №3.

[38] Марков А.А. *О логике конструктивной математики.*— Вестн. Моск. ун-та, матем., механ., № 2, 7-29, 1970.

[39] Марков А.А. *О логике конструктивной математики.*— М.: Знание, 1972.

[40] Мендельсон Э. *Введение в математическую логику.*— М.: Наука, 1971.

[41] Пантелеев А.Г. *Об интерпретаторе с языка ЛИСП для ЕС ЭВМ.* — Программирование, 1980, № 3, с. 86—87 1.4

[42] Пратт Т., Зелковиц М. *Языки программирования. Разработка и реализация.* — СПб.: Питер, 2002. — 688 с. 1.4

[43] Себеста Р. *Основные концепции языков программирования.* 5-е изд.: Пер. с англ. — М.: Издательский дои “Вильямс”, 2001. — 672 с. 1.4, 2.1.3, 17.1

[44] Скотт Д.С. *Логика и языки программирования.*— Лекции лауреатов премии Тьюринга (ред.: Эшенхерст Р.). — М.: Мир, 1993. — с. 65-83.

[45] *Семантика модальных и интенсиональных логик.*— Под ред. Смирнова В.А. — М.: Прогресс, 1981. — 424 с. 1.4, 17.1

[46] 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. 1.4

[47] 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.

[48] Такеути Г. *Теория доказательств.*— М.: Мир, 1978. 1.4

[49] Фурман М. *Логика топосов.*— Справочная книга по математической логике: В 4-х частях (под. ред. Дж. Барвайса.)— Ч. IV. Теория доказательств и конструктивная математика: Пер. с англ. — М.: Наука. Главная редакция физико-математической литературы, 1983. — с. 241-277 1.4

[50] Хендерсон П. *Функциональное программирование. Применение и реализация.*— М.: Мир, 1983. 1.4, 6.8, 6.8

[51] Хопкрофт Дж., Мотвани Р., Ульман Дж. *Введение в теорию автоматов, языков и вычислений.* 2-е изд.: Пер. с англ.— М.: Издательский дом "Вильямс", 2002.— 528 с. 1.4

[52] Шабунин Л.В. *О непротиворечивости некоторых исчислений комбинаторной логики.*— Вестн. Моск. ун-та, сер. матем. мех., 1971, №6. 1.4

[53] Шенфилд Дж. *Математическая логика.*— М.: Наука, 1975. 1.4

[54] Энгелер Э. *Метаматематика элементарной математики.*— М.: Мир, 1986. 1.4

[55] Яновская С.А. *Основания математики и математическая логика.*— Математика в СССР за тридцать лет. 1917-1947.— М.-Л.: Гостехиздат, 1948.

[56] *Nested relations and complex objects in databases.*— Lecture Notes in Computer Science, 361, 1989. 1.4

[57] Amadio R.M., Curien P.-L. *Domains and lambda-calculi.*— Cambridge University Press, 1998. — 484 p.
В книге изложены математические аспекты семантики языков программирования. Основными целями являются построение формальных средств для анализа значения конструктов программирования как независимым от языка, так и независимым от машины способами, а также для обоснования свойств программ, например, их завершаемости либо возможности получения решения той задачи, для которой они предназначены.
Преследуется двойная цель: дать специалистам в компьютерных науках необходимую мотивацию для получения математических результатов, а специалистам в математике указать на возможные приложения в области компьютерных наук. 1.4

[58] Appel A. *Compiling with continuations.*— Cambridge University Press, 1992.

[59] Avron A., Honsel F., Mason I., and Pollak R. *Using typed lambda-calculus to implement formal systems on a machine.*— Journal of Automated Reasoning, 1995.

[60] Backus J.W. *Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.*— Comm. ACM, 1978, v.21, № 8, p. 614-641.

[61] Backus J.W. *The algebra of functional programs: functional level reasoning, linear equations and extended definitions.*— Int. Col. on formalization of programming concepts, LNCS, v. 107, 1981, pp. 1-43.

[62] Backus J.W., Williams J.H., Wimmers E.L. *FL language manual (preliminary version).*— IBM research report No RJ 5339(54809), 1987.

[63] Banerji R.B. (ed.) *Formal techniques in artificial intelligence: a sourcebook.*— Studies in computer science and artificial intelligence, 6, North-Holland, 1990. 1.4, 17.1

[64] Beery G., Levy J-J. *Minimal and optimal computations of recursive programs.*— J. Assoc. Comp. Machinery, Vol.26, No 1, 1979.

[65] Belnap N.D. (Jr.) *A useful four-valued logic.* Modern uses of multiple-valued logic.— Epstein G.,Dunn J.M. (eds.) Proceedings of the 1975 International Symposium of multiple-valued logic, Reidel, 1976.

[66] Belnap N.D. (Jr.) *How a computer should think.*— Contemporary aspects of philosophy, Proceedings of the Oxford International Symposium, 1976.

[67] Bird R.S. *An introduction to the theory of lists.*— Logic programming and calculi of descrete design (ed. Broy M.), Springer-Verlag, 1986, pp. 5-42.

[68] Böhm C. (ed.) *Lambda calculus and computer science theory.*— Proceedings of the Symposium held in Rome. March 25-29, LNCS, vol.37, Berlin: Springer 1975.

[69] Bonsanque M. *Topological dualities in semantics.*— PhD thesis, Vrije Universiteit Amsterdam, 1996. 1.4

[70] Bunder M.V.W. *Set Theory based on Combinatory Logic.*— Doctoral thesis, University of Amsterdam, 1969.

[71] Bunder M.V.W. *Propositional and predicate calculuses based on combinatory logic.*— Notre Dame Journal of Formal Logic, Vol. XV, 1974, pp. 25-34.

[72] Bunder M.V.W. *The naturalness of illative combinatory logic as a basis for mathematics.*— To H.B.Curry: Essays

on combinatory logic, lambda calculus and formalism.— Seldin J.P., Hindley J.R. (eds.), Academic Press, 1980, pp. 55-64. 1.4

[73] Bucciarelli A. *Logical reconstruction of bi-domains.*— In: Proc. Typed Lambda Calculi and Applications, Springer Lecture Notes in Comp. Sci., 1210, 1997.

[74] Church A. *The calculi of lambda-conversion.*— Princeton. 1941, ed. 2, 1951.

[75] Coppo M., Dezani M., Longo G. *Applicative information systems.*— LNCS, 159, 1983, pp. 35-64.

[76] Cousineau G., Curien P.-L., Mauny M. *The categorical abstract machine.*— LNCS, 201, Functional programming languages computer architecture.— 1985, pp. 50-64.

[77] Curien P.-L. *Categorical combinatory logic.*— LNCS, 194, 1985, pp. 139-151.

[78] Curien P.-L. *Typed categorical combinatory logic.*— LNCS, 194, 1985, pp. 130-139.

[79] Curry H.B., Feys R. *Combinatory logic.*— Vol. 1. Amsterdam: North-Holland, 1958.

[80] Curry H.B., Hindley J.R., Seldin J.P. *Combinatory logic.*— Vol. II. Amsterdam, 1972.

[81] Curry H.B. *Some philosophical aspects of combinatory logic.*— Barwise J., Keisler H.J., Kunen K. (eds.) The Kleene Symposium.— North-Holland Publ. Co, 1980, p. 85-101.

[82] Danforth S., Tomlison C. *Type theories and object oriented programming.*— ACM Computing Surveys, 1988, v.20, No 1, pp. 29-72.

[83] Darlington J., Henderson P., Turner D.A. (eds.) *Functional programming and its applications.*— Cambridge Univ. Press, Cambridge, 1982.

[84] de Bruijn N.G. *Lambda-calculus notations with nameless dummies: a tool for automatic formula manipulation.*— Indag. Math. 1972, №34, pp. 381-392.

[85] Eisenbach S. (ed.) *Functional programming: languages, tools and architectures.*— Chichester: Horwood, 1987.

[86] Fasel J.H., Keller R.M. (eds.) *Graph reduction.*— LNCS, 279, 1986.

[87] Fenstad J.E. et al. *Situations, language and logic.* — Dordrecht: D. Reidel Publ. Comp., 1987.

[88] Fitting M. *First-order logic and automated theorem proving.*— Springer-Verlag, 1990.

[89] Frandsen G.S., Sturtivant C. *What is an efficient implementation of the lambda-calculus?*— LNCS, 523, 1991, pp. 289-312.

[90] Friedman H. *Equality between functionals.*— Proceedings of the Symposium on Logic. Boston, 1972—1973.— Lecture Notes in Mathematics, 453, 1975, pp. 22-37.

[91] Gardenfors P. *Induction, Conceptual spaces and AI.*— Proceedings of the workshop on inductive reasoning, Riso National Lab, Rpskilde, 1987.

[92] Henson M.S. *Elements of functional languages.*— Exford: Blackwell, 1987.

[93] Hindley J.R. *The principial type-scheme of an object in combinatory logic.*— Trans. Amer. Math. Soc., 1969, vol. 146.

[94] Hindley J., Lercher H., Seldin J. *Introduction to combinatory logic.*— Cambridge University Press, 1972.

[95] Howard W. *The 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.

[96] Hughes R.J.M. *Super combinators: a new implementation method for applicative languages.*— Proceedings of the 1982 ACM symposium on LISP and functional programming, pp. 1-10.

[97] Hughes R.J.M. *The design and implementation of programming languages.*— PhD Thesis, University of Oxford, 1984.

[98] Hunt L.S. *A Hope to FLIC translator with strictness analysis.*— MSc dissertation, Department of Computing, Imperial College, University of London, 1986.

[99] Kelly P.H.J. *Functional languages for loosely-coupled multiprocessors.*— PhD Thesis, Imperial College, University of London, 1987.

[100] Lambek J., Scott P.J. *Introduction to higher order categorical logic.* — Cambridge Studies in Advanced Mathematics 7, Cambridge University Press, 1986, 1988, 1989, 1994. — 293 p.

[101] Landin P. *The next 700 programming languages.*— Communications of the ACM, 3, 1966.

[102] McCarthy J. *A basis for a mathematical theory of computation.*— Computer programming and formal systems (eds.: Braffort and Hirshberg), Amsterdam: North-Holland, 1963, pp. 33-69.

[103] Michaelson G. *An introduction to functional programming through lambda-calculus*. — Addison Wesley Publ.Co, 1989, 320 p.

[104] Milner R., Parrow J., Walker D. *A calculus of mobile process*. — Parts 1-2. — Information and Computation, 100(1), 1992, pp. 1-77 1.4

[105] Mycroft A. *Abstract interpretation and optimizing transformations for applicative programmes*.— PhD Thesis, Department of Computer Science, University of Edinburgh, 1981.

[106] Peyton Jones S.L. *The implementation of functional programming languages*.— Prentice Hall Int., 1987.

[107] Pitts A. *Relational properties of domains*. — Information and Computation, 127, 1996, pp. 66-90. 1.4

1.4

[108] Rosser J.B. *A mathematical logic without variables*.— Ann. of Math., vol. 36, No 2, 1935. pp. 127-150.

[109] Schönfinkel M. *Uber die Baustein der mathematischen Logik*.— Math. Annalen, vol. 92, 1924, pp. 305-316.

[110] Schroeder-Heister P. *Extentions of logic programming*.— LNAI, 475, 1991.

[111] Scott D.S. *Advice on modal logic*.— Philosophical problems in logic. Some recent developments.— Lambert K. (ed.), Dordrecht; Holland: Reidel, 1970.

[112] Scott D.S. *Outline of a mathematical theory of computation*.— Proceedings of the 4-th Annual Princeton conference on information sciences and systems, 1970.

[113] Scott D.S. *The lattice of flow diagrams.*— Lecture Notes in Mathematics, 188, Symposium on Semantics of Algorithmic Languages.— Berlin, Heidelberg, New York: Springer-Verlag, 1971, pp. 311-372.

[114] Scott D.S. *Identity and existence in intuitionistic logic.*— In: Applications of Sheaves. Berlin: Springer, 1979, pp. 660-696.

[115] Scott D.S. *Lambda calculus: some models, some philosophy.*— The Kleene Symposium. Barwise, J., et al.(eds.), Studies in Logic 101, North-Holland, 1980, pp. 381-421.

[116] Scott D.S. *Relating theories of the lambda calculus.*— Hindley J., Seldin J. (eds.) To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.— N.Y.& L.: Academic Press, 1980, pp. 403-450.

[117] Scott D.S. *Lectures on a mathematical theory of computation.*— Oxford University Computing Laboratory Technical Monograph PRG-19, 1981. — 148 p.

[118] Scott D.S. *Domains for denotational semantics.*— LNCS, 140, 1982, pp. 577-613.

[119] Stoy J.E. *Denotational semantics: The Scott-Strachey approach to programming language theory.*— M.I.T. Press, Cambridge, Mass., 1977.— xxx+414 p. 1.4

[120] Stoye W.R. *The implementation of functional languages using custom hardware.*— PhD Thesis, University of Cambridge, 1985.

[121] Szabo M.E. *Algebra of proofs.*— Studies in Logic foundations of mathematics, v. 88. North-Holland Publ. Co, 1978.— 297 p. 1.4

[122] Talcott C. *Rum: An intensional theory of function and control abstractions.*— Foundations of logic and functional programming, LNCS, 306, 1986, pp. 3-44. 1.4

[123] Tello E.R. *Object-Oriented Programming for Windows / Covers Windows 3.x.*— Wiley and Sons, Inc., 1991.

[124] Turner D.A. *A New Implementation Technique for Applicative Languages.*— Software Practice and Experience.— No 9, 1979, pp. 31-49. 1.4

[125] Turner D.A. *Aspects of the implementation of programming languages.*— PhD Thesis, University of Oxford, 1981.

[126] Turner R. *A theory of properties.*— J. Symbolic logic, v. 52, 1987, pp. 455-472.

[127] Wodsworth C.P. *Semantics and pragmatics of the lambda calculus.*— PhD Thesis, University of Oxford, 1981.

[128] Wolfengagen V.E. *Frame theory and computations.*— Computers and artificial intelligence. V.3, No 1, 1984, pp. 1-31.

[129] Wolfengagen V.E. *Building the access pointers to a computational environment.* — In: electronic Workshops in Computing, Berlin Heidelberg New York: Springer-Verlag, 1998. pp. 1-13
http://ewic.springer.co.uk/adbis97/

[130] Wolfengagen V.E. *Event driven objects.* — In: Proceedings of the 1st International Workshop on Computer Science and Information Technologies, Moscow, Russia, 1999, Vol. 1. pp. 88-96

[131] Wolfengagen V.E. *Functional notation for indexed concepts.* — In: Proceedings of The 9th International Workshop on

Functional and Logic Programming WFLP'2000, Benicassim, Spain, September 28-30, 2000
http://www.dsic.upv.es/~wflp2000/ 1.4

[132] Zhang *The largest cartesian closed category of stable domains*. — Theoretical Computer Science, 166, 1995, pp. 203-219. 1.4

[133] http://www.rbjones.com
Этот электронный ресурс FACTASIA имеет своей целью "развить предвидение будущего и дать ресурсы для построения такого видения и самого́ будущего", а также "сделать вклад в те ценности, которые определяют будущее, и в те технологии, которые помогают его строить". Воспользовавшись навигационными средствами в разделе Logic можно найти анализ основных разделов *комбинаторной логики* и *λ-исчисления*, библиографию по данному вопросу, а также важнейшие связи с другими разделами.
См. http://www.rbjones.com/rbjpub/logic/cl/ 1.4

[134] http://ling.ucsd.edu/~barker/Lambda/ski.html
Представлен простой учебник по комбинаторной логике в режиме on-line. 1.4

[135] http://www.cwi.nl/~tromp/cl/cl.html
Представлено руководство, необходимые библиографические ссылки и Java-апплет, позволяющий интерпретировать объекты — выражения аппликативного языка, — которые строятся по правилам комбинаторной логики.

[136] http://www.brics.dk
BRICS — Basic Research in Computer Science (Фундаментальные исследования в компьютерных науках). Иссследовательский центр и Международная школа аспирантов. В разделе "BRICS Publications" представлена "Lecture Series", в которой содержатся материалы курсов в 9-ти основных областях: *дискретной математики*, *семантики вычислений*, *логики в компьютерных*

науках, *сложности вычислений*, *построения и анализа алгоритмов*, *языков программирования*, *тестирования*, *распределенных вычислений*, *криптологии и безопасности данных*.

[137] http://www.afm.sbu.ac.uk
Formal Methods (Формальные методы). Документ содержит ряд указателей, имеющихся в Web на формальные методы, которые полезны для математического описания и обсуждения свойств компьютерных систем. Особое внимание отведено тем формальным методам, которые полезны для сокращения числа ошибок, возникающих в системах, в особенности на ранних этапах их проектирования. Эти методы являются дополнительными по отношению к такому методы выявления ошибок, как тестирование.

[138] http://liinwww.ira.uka.de/bibliography/
The Collection of Computer Science Bibliographies (Коллекция библиографий по компьютерным наукам). Представлена коллекция библиографий научной литературы в области компьютерных наук, которая собрана из разнообразных источников и покрывает многие их аспекты. Библиографии ежемесячно обновляются в всязи с их исходных размещением, поэтому поддерживаются наиболее актуальные версии. Коллекция включает более 1.2 млн. ссылок на журнальные статьи, доклады на конференциях и технические отчеты, сгруппированные в виде приблизительно 1400 библиографий. Более 150000 ссылок содержат URLы на электронные версии статей, имеющихся в доступе on-line.

[139] http://www.ncstrl.org
NCSTRL — Networked Computer Science Technical Reference Library (Сетевая библиотека технических ссылок по компьютерным наукам.) Эта библиотека создана в рамках инициативы Open Arhives Initiative (http://www.openarchives.org).

[140] http://xxx.lanl.gov/archive/cs/intro.html
CoRR — The Computing Research Repository (Исследовательский репозиторий в области компьютинга). Содержит статьи, начиная с 1993 г. в области компьютерных наук, которые классифицируются двумя способами: по предметному признаку и с применением Классификационной Системы ACM по компьютингу 1998 года. Классификационная схема ACM относительно стабильна и покрывает все разделы компьютерных наук. Предметные же признаки не являются взаимно исключающими и не претендуют на исчерпания всех разделов. Тем не менее они отражают области активного исследования в компьютерных науках.

1.4

[141] http://web.cl.cam.ac.uk/DeptInfo/CST/node13.html
Foundations of Computer Science (Основания компьютерных наук).
Курс лекций Кембриджского университета, задачей которого является закладываение базовых основ программирования. 1.4

[142] http://web.comlab.ox.ac.uk/oucl/courses/topics01-02/lc
Lambda Calculus (Ламбда-исчисление).
Курс лекций Оксфордского университета, который охватывает формальную теорию, системы перезаписи, комбинаторную логику, полноту по Тьюрингу и системы типов. Является кратким введением во многие отрасли компьютерных наук, демонстрируя их связи с ламбда-исчислением.

Предметный указатель

Глоссарий

Алгебра

Алгеброй часто считают систему, вовсе не использующую связанных переменных, то есть все используемые переменные являются свободными.

Алгоритм (неформально)

Алгоритмом считается детерминированная процедура, которую можно применять к любому элементу некоторого класса символических *входов* и которая для каждого такого входа дает в конце концов соответствующий символический *выход*. 1.4

Существенные черты алгоритма:

*1) алгоритм задается как набор инструкций конечных размеров;

*2) имеется вычислитель, который умеет обращаться с инструкциями и производить вычисления;

*3) имеются возможности для выделения, запоминания и повторения шагов вычисления;

*4) пусть P — набор инструкций в соответствии с *1), а L — вычислитель из *2). Тогда L взаимодействует с P так, что для любого данного входа вычисление происходит дискретным образом по шагам, без использования аналоговых устройств и соответствующих методов;

*5) L взаимодействует с P так, что вычисление продвигается вперед детерминированно, без обращения к случайным методам или устройствам, например к игральным костям.

Алфавит

Алфавитом считается определенный набор объектов, называемых *символами* или буквами, которые обладают свойством неограниченной воспроизводимости (на письме). 1.4, 3.1.1

Аксиоматическая теория множеств

Характеристики этой теории (см., например, [22]): (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.

Аппликативные вычислительные системы

Традиционно в состав *аппликативных вычислительных систем*, или ABC, включают системы исчислений объектов, основанные на комбинаторной логике и ламбда-исчислении. Единственное, что существенно разрабатывается в этих системах — это представление об *объекте*. В комбинаторной логике единственный метаоператор — *аппликация*, или, по иной терминологии, *приложение* одного объекта к другому.

В ламбда-исчислении два метаоператора — *аппликация* и функциональная *абстракция*, позволяющая связывать одну переменную в одном объекте. 1.4

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

- число аргументных мест, или арность объекта заранее не фиксируется, но проявляет себя постепенно, во взаимодействиях с другими объектами;
- при конструировании составного объекта один из исходных объектов — функция, — применяется к другому — аргументу, — причем в других контекстах они могут поменяться ролями, то есть функции и аргументы рассматриваются как объекты на равных правах;
- разрешается самоприменимость функций, то есть объект может применяться сам к себе.

Высказывание

Высказывание определено тем способом, относительно которого можно рассматривать его доказательство.

Возможные миры

Возможные миры будем понимать как различные собрания индивидов с дополнительной структурой или без нее. 1.4

Дескрипция

При построении логических средств в числе термов часто используются *определенные описания*, или *дескрипции* — конструкции ‘тот . . . , который . . . ’. Дескрипция соответствует терму $\mathbf{I}x\Phi$, который канонически читается как ‘тот единственный x, для которого выполняется (верно) Φ’.

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. 1.4

Интерпретация (— теории)

Под интерпретацией теории относительно содержательной области (предметной области) понимается много-однозначное соответствие между элементарными высказываниями теории и определенными содержательными высказываниями, относящимися к этой содержательной области. 1.4

Интерпретация (— терма)

Оценка, или интерпретация терма M в структуре $\mathcal{M}$ — это

отображение:

$$\|\cdot\|\cdot : \text{термы} \times \text{приписывания} \to \text{элементы из } \mathcal{M}$$

(см. также ***Оценка***).

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. 1.4, 1

Класс (— индуктивный)

Индуктивный класс — это концептуальный класс, порожденный из определенных исходных элементов посредством выделенных способов комбинации. Более строго, класс $\mathcal{K}$ индуктивен, если:

(1) класс $\mathcal{K}$ включает в себя базис;
(2) класс $\mathcal{K}$ замкнут относительно способов комбинации;
(3) класс $\mathcal{K}$ является подклассом любого класса, удовлетворяющего условиям (1) и (2).

Понятие индуктивного класса обычно используют в двух случаях:

(1) элементы являются *объектами*, а способы комбинации — *операциями*;

(2) элементы являются *высказываниями*, а способы комбинации — *связками*.

Комбинатор

Комбинатором считается объект, который относительно означивания проявляет свойство константности. С точки зрения λ-исчисления комбинатор является замкнутым термом.

Комбинаторная логика

В узкой формулировке это ветвь математической логики, изучающая комбинаторы и их свойства. В комбинаторной логике функциональная абстракция выразима в терминах обычных операций, то есть *без использования формальных переменных* (параметров).

Конструкция

Процесс получения объекта X, принадлежащего индуктивному классу $\mathcal{K}$ (см. ***Класс индуктивный***), посредством итерации способов комбинации считается *конструкцией* X относительно $\mathcal{K}$.

λ-терм (ламбда-терм)

λ-термом, или λ-выражением считается объект, полученный индукцией по построению с возможным применением операторов аппликации и абстракции. 1.4

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. 1.4

Логика (— математическая)

Математическая логика описывает новое направление в математике (см. ***Математика современная***), сосредоточивая внимание на используемом в ней языке, на способах определения абстрактных объектов и на законах логики, которые используются в рассуждениях об этих объектах. Такое изучение предпринято в логике, чтобы понять природу математического опыта, пополнить математику важнейшими результатами, полученными в логике, а также, чтобы найти приложения к другим разделам математики. 1.4

Логика (современная математическая —)

Современная математическая логика берет свое начало от работ Лейбница об универсальном исчислении, которое может включать в себя всю умственную деятельность и, в частности, всю математику. 1.4

Математика (— современная)

Современная математика может быть описана как наука об *абстрактных объектах* таких, как вещественные числа, функции, алгебраические системы и т.п. 1.4

Множество (— счетное)

Счетным называют всякое множество, элементы которого могут быть занумерованы, то есть расположены в виде единого списка, в котором некоторый элемент стоит первым,

некоторый — вторым и т.д., так что всякий элемент этого множества рано или поздно встретится в этом списке. 1.4

Неразрешимость

Математический смысл какого-либо результата о неразрешимости состоит в том, что некоторое конкретное множество не является рекурсивным. 1.4

Нумерал

В рамках комбинаторной логики или λ-исчисления можно установить такие комбинаторы или, соответственно, термы, которые ведут себя как числа. Эти представления чисел получили название *нумералов*. Нумералы как комбинаторы удовлетворяют всем законам комбинаторной логики. Более того, можно указать комбинаторы, представляющие арифметические операции, например, сложение чисел. 1.4

Об-система

Формальные объекты об-системы образуют индуктивный класс. Элементы этого класса называются *обами*, или объектами. Начальные элементы индуктивного класса считаются *атомами*, а способы комбинации — *операциями*. Об-системы используются для поиска существенных, инвариантных образований объектов.

Оболочка Каруби

Оболочка Каруби представляет собой частный вид категории.

Объект

Объектом считается математическая сущность, которая используется в теории. Объект — это математическое *представление* реального объекта предметной области (“внешнего мира”).

Объект (— арифметический)

Арифметическим объектами считаются комбинаторные представления чисел — нумералы, а также соответствующие комбинаторные представления операций над числами (см. ***Нумерал***).

Объекты (система —)

См. ***Система объектов***.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) представляет собой такой способ программирования, который обеспечивает модульность программ за счет разделения памяти на области, содержащие данные и процедуры. Области могут использоваться в качестве образцов, с которых по требованию могут делаться копии.

Определение

Определение традиционно принимается как соглашение относительно использования языка. Тогда новый символ или комбинация символов, называемая *определяемым*, вводится вместе с разрешением подставлять ее вместо некоторой другой комбинации символов, называемой *определяющим*, значение которой уже известно на основе данных и предшествующих определений. 1.4

Определение (— функции)

Явные определения вводят в рассмотрение функции. Таким образом, начиная с переменной x, которая обозначает произвольный объект типа A, строится выражение $b[x]$, обозначающее объект типа $B(x)$. Далее определяется функция f типа $(\forall x \in A)B(x)$ схемой $f(x) \stackrel{def}{=} b[x]$, где квадратные скобки указывают на вхождение переменной x в выражение $b[x]$. Если $B(x)$ для каждого объекта x типа A определяет

один и тот же тип B, то вместо '$(\forall x \in A)B(x)$' используется сокращенная форма записи $A \to B$. Последняя запись принимается за тип функций из A в B.

Определение (— рекурсивное)

Рекурсивное определение функции — это такое определение, в котором значения функции для данных аргументов непосредственно определяются значениями той же функции для "более простых" аргументов или значениями "более простых" функций. Понятие 'более простой' уточняется выбором формализации — простейшими, как правило, являются все функции-константы. Такой метод формализации удобен, поскольку рекурсивные определения можно рассматривать как алгоритм (см. ***Алгоритм***).

Оценка

Оценкой считается такое соответствие, когда содержательные объекты сопоставляются с формальными объектами, причем один и тот же содержательный объект может быть поставлен в соответствие двум или более различным формальным объектам.

Переменная

Переменной считается "переменный объект", вместо которого можно производить подстановки. 1.4

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. 1.4

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.

Процесс (— эффективный)

Предположим, что имеются определенные преобразования, которые можно фактически производить над определенными элементами. Предположим, что имеется также предписание, определяющее последовательность преобразований, которые надо применять одно за другим к какому-то элементу.

Говорят, что предписание определяет *эффективный процесс* для достижения определенной цели по отношению к элементу, если при условии, что этот элемент задан, предписание однозначно определяет такую последовательность преобразований, что цель достигается за конечное число шагов.

Реляционная система

Это система с единственным базисным предикатом, который является бинарным отношением.

Свойство

Свойством считается пропозициональная функция, определенная на (произвольном) типе A.

Система (— объектов)

В *системе объектов* формальные объекты образуют однородный индуктивный класс (см. ***Класс индуктивный***). Элементы этого индуктивного класса считаются *объектами*, его начальные объекты — *атомарными объектами*, а способы комбинации — *исходными операциями*.

Суффикс

Суффиксом считается функтор, который пишется после аргументов.

Тезис Черча

Нельзя доказать гипотезу о том, что какая-либо стандартная формализация дает удовлетворительные аналоги неформального понятия *алгоритма* (см. ***Алгоритм***) и *алгоритмической функции* (см. ***Функция, вычислимая алгоритмом***).

Многие математики принимают гипотезу о том, что стандартные формализации дают “разумную переделку” неизбежно расплывчатых неформальных понятий, а саму гипотезу называют *тезисом Черча*.

Теория

Теорией считают способ выбора подкласса истинных высказываний из числа высказываний, принадлежащих классу всех высказываний $\mathcal{A}$.

Теория (— дедуктивная)

Теория $\mathcal{T}$ считается дедуктивной, если $\mathcal{T}$ является индуктивным классом (элементарных) высказываний (см. ***Класс индуктивный***).

Теория (— моделей)

Теорией моделей считается раздел математической логики, изучающий связи между формальным языком и его интерпретациями, или *моделями*. Основными объектами исследования являются предложения ϕ и алгебраические системы $\mathcal{M}$ для языка L.

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 (— рекурсии)

Теория рекурсии изучает класс рекурсивных или эффективно вычислимых функций и их применения в математике. В

широком смысле теория рекурсии рассматривается как изучение общих процессов определения с помощью рекурсии, но не только на натуральных числах, а на всех типах математических структур.

Теория (— типов)

В основе этой теории лежит принцип иерархичности. Это означает, что логические понятия — высказывания, индивиды, пропозициональные функции, — располагаются в иерархию типов. Существенно, что произвольная функция в качестве своих аргументов имеет лишь те понятия, которые предшествуют ей в иерархии.

Фраза (— грамматическая)

Фразами считаются *имена*, *предложения* и *функторы*.

Функтор (— грамматический)

Функтор рассматривается как средство соединения *фраз* для образования других фраз.

Функция

См. ***Определение функции***.

Функция (— высшего порядка)

Функция получает название функции ‘высшего порядка’, если ее аргументом может в свою очередь быть функция, либо в результате ее применения получается функция.

Функция (—, вычислимая алгоритмом)

Это отображение, задаваемое процедурой, или *алгоритмом*.

Функция (— примитивнорекурсивная)

Класс примитивнорекурсивных функций — это наименьший класс C всюду определенных функций такой, что:

i) все *функции-константы* $\lambda x_1 \ldots x_k.m$ содержатся в $\mathcal{C}$, $1 \leq k, 0 \leq m$;

ii) функция *следования за* $\lambda x.x + 1$ содержится в $\mathcal{C}$;

iii) все *функции выбора* $\lambda x_1 \ldots x_k.x_i$ содержатся в $\mathcal{C}$, $1 \leq i \leq k$;

iv) если f — функция от k переменных из $\mathcal{C}$ и g_1, g_2, …, g_k — функции от m переменных из $\mathcal{C}$, то функция

$$\lambda x_1 \ldots x_m.f(g_1(x_1, \ldots, x_m), \ldots, g_k(x_1, \ldots, x_m))$$

содержится в $\mathcal{C}$, $1 \leq k, m$;

v) если h — функция от $k+1$ переменных из $\mathcal{C}$, а g — функция от $k-1$ переменных из $\mathcal{C}$, то единственная функция f от k переменных, удовлетворяющая условиям

$$\begin{aligned} f(0, x_2, \ldots, x_k) &= g(x_2, \ldots, x_k), \\ f(y+1, x_2, \ldots, x_k) &= h(y, f(y, x_2, \ldots, x_k), x_2, \ldots, x_k), \end{aligned}$$

содержится в $\mathcal{C}$, $1 \leq k$.

Отметим, что ‘функция от нуля переменных из $\mathcal{C}$’ означает фиксированное натуральное число.

Язык

Язык в широком смысле слова определяется введением в употребление *соглашений*: (1) фиксируется *алфавит*; (2) фиксируются правила образования из букв алфавита определенных комбинаций, называемых *выражениями* или словами.

Язык (— исследователя, или $\mathcal{U}$-язык)

$\mathcal{U}$-язык характеризуется следующими свойствами:

(1) *единственностью* для каждого конкретного контекста;

(2) наличием средств *формализации терминологии*;

(3) изменчивостью в том смысле, что он является *процессом* относительно добавления новой символики или новых терминов, причем использование старых терминов не обязательно является неизменным;

(4) $\mathcal{U}$-язык по необходимости неясен, однако пользуясь им можно достичь любой разумной степени точности.

Практикум

Источники

Первоначально практикум по аппликативным вычислениям был включен в общий практикум по курсу “Системы искусственного интеллекта”, читавшийся в МИФИ (Л.Т. Кузин, [24]). Этот практикум был подготовлен коллективом авторов по разделам дедуктивного вывода, реляционной алгебры и реляционного исчисления, понятийного представления знаний и фреймов, программирования на Lisp (К.Е. Аксенов, О.Т. Баловнев, В.Э. Вольфенгаген, О.В. Воскресенская, А.В. Ганночка, М.Ю. Чуприков, [1]; А.Г. Пантелеев, [41], М.А. Булкин, Ю.Р. Габович, А.Г. Пантелеев, [5]). Его существенной компонентой являлась реляционная СУБД Lisp/R, которая реализовывала один из путей развития вычислительных идей, изначально в него заложенных. Этот метод опирался на встроенные (погруженные) вычислительные системы и получил развитие в работе (О.В. Воскресенская, [1]), приведенной в перечне диссертаций на стр. 304.

Аппликативные вычисления

Вариант курса, излагавшегося в МИФИ на основе настоящей книги, (например, в варианте “Специальные главы дискретной математики” или “Основы компьютерных наук”) оснащен практикумом, который изложен в работе (В.Э. Вольфенгаген, Л.В. Го-

льцева, [12]). Дополнительное развитие можно найти в работах (Л.В. Гольцева, [7]) и (А.В. Гаврилов, [6]), приведенных в перечне диссертаций на стр. 304. Практикум работает на IBM PC и распространяется на машинных носителях.

Структура практикума

Практикум охватывает основные понятия и обозначения, используемые в аппликативных вычислительных системах, и позволяет проводить их изучение в полном объеме: от языка и соглашений об обозначениях до метатеорем о соответствующих формальных системах. Он дает базовые знания об использовании аппликативных вычислительных систем (ABC) в качестве основы функциональных языков, систем функционального программирования и чисто объектных языков.

Центральным используемым понятием является терм, рассматриваемый как *представление* объекта. Терм синтаксически соответствует программе на функциональном языке.

На этом основании *первый* раздел практикума посвящен технике выполнения правильных синтаксических преобразований — расстановке скобок,— для термов различных видов.

Второй раздел предназначен для изучения редукции в ABC. Результатом выполнения функциональной программы является значение, полученное по окончании процесса вычисления. В ABC результату выполнения редукции соответствует нормальная форма терма. В соответствии с теоремой Черча-Россера нормальная форма не зависит от порядка выполнения шагов редукции, что дает возможность построения различных стратегий вычисления в системах функционального программирования. Построение λ-абстракции в комбинаторной логике можно рассматривать в качестве примера для интерпретации логических систем средствами комбинаторной логики.

Так пара комбинаторов K и S образует базис для произволь-

ных λ-термов, в то время как комбинаторы I, B, C, S являются базисом только для термов, в которых отсутствуют свободные переменные. Использование этих двух базисов для разложения термов из двух разделов практикума обеспечивает достаточную полноту рассмотрения материала, поскольку будут представлены произвольные λ-термы и комбинаторные термы, которые являются λ-термами без свободных переменных.

Последний раздел практикума является наиболее творческим, так как предполагает конструирование собственных комбинаторных систем. Показано, как путем добавления к базисным комбинаторам дополнительных комбинаторов увеличить выразительные возможности реализованной среды.

В целом, каждый из разделов лабораторного практикума вырабатывает у обучаемого практические навыки по определенному кругу вопросов и может быть использован как отдельная лабораторная работа. Если практикум используется как компьютерное пособие, то разделы могут быть скомпонованы в соответствии с подготовкой обучаемого либо как это нужно преподавателю.

Независимые ресурсы

http://www.rbjones.com Этот электронный ресурс FACTASIA имеет своей целью “развить предвидение будущего и дать ресурсы для построения такого видения и самого́ будущего”, а также “сделать вклад в те ценности, которые определяют будущее, и в те технологии, которые помогают его строить”. Раздел Logic включает *комбинаторную логику* и *λ-исчисление*, библиографию. См. http://www.rbjones.com/rbjpub/logic/cl/

http://www.cwi.nl/~tromp/cl/cl.html Представлено руководство, библиографические ссылки и Java-апплет, позволяющий интерпретировать объекты — выражения аппликативного языка, — которые строятся по правилам комбинаторной логики.

http://ling.ucsd.edu/~barker/Lambda/ski.html Представлен простой учебник по комбинаторной логике в режиме on-line.

http://tunes.org/~iepos/oldpage/lambda.html Введение в ламбда-исчисление и комбинаторную логику.

http://foldoc.doc.ic.ac.uk Представлен свободно доступный словарь терминов в области вычислений FOLDOC — Free On-Line Dictionary of Computing.

http://dmoz.org/Science/Math/ Содержит справочные сведения и библиографию. В разделе '/Logic_and_Foundations' имеется раздел '/Computational_Logic', в котором есть ссылка на '/Combinatory_Logic_and_Lambda_Calculus/'.

http://www.cl.cam.ac.uk/Research/TSG Представлены направления учебной и научно-исследовательской работы Компьютерной лаборатории Кембриджского университета, имеющей мировую известность.

http://web.comlab.ox.ac.uk/oucl Вычислительная лаборатория Оксфордского университета, которая является кафедрой по компьютерным наукам с мировой известностью. На URL http://web.comlab.ox.ac.uk/oucl/strachey размещена информация о регулярно проводимой серии лекций в честь Кристофера Стрейчи (Christopher Strachey, 1916-1975), первого профессора Оксфордского университета в области вычислений (computation). Он был первым руководителем Группы Исследований в Программировании, созданной в 1965 г. — а сэр Тони Хоар (Tony Hoare) стал в 1977 г. его преемником, — и вместе с Дана Скоттом (Dana Scott) основал *денотационную семантику* как направление, поставив языки программирования на прочную математическую основу.

Диссертации

[1] Воскресенская О.В., *Методы разработки реляционной системы управления базой данных*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1985. 17.1

[2] Александрова И.А., *Проектирование информационно-программного обеспечения систем организационного типа на основе концептуальных моделей*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1986.

[3] Исмаилова Л.Ю., *Разработка программных средств реляционной обработки данных в экспертных системах*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей, Московский инженерно-физический институт, Диссертационный Совет К.053.03.04 МИФИ, Москва, 1989.

[4] Волков И.А., *Исследование и разработка методов анализа и обеспечения достоверности информации о НИР в области медицины*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.06 — Автоматизированные системы управления; 14.00.33 — Социальная гигиена и охрана здравоохранения, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1990.

[5] Вольфенгаген В.Э., *Концептуальный метод проектирования банков данных*, Диссертация на соискание ученой степени доктора технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1990.

{ *Аннотация*. Представлен концептуальный метод проектирования банков данных, который рассматривается как подход к построению, применению и управлению базами данных и базами метаданных. Подход предполагает возможность настройки на изменяющуюся предметную область и ее представление, в частности, при увеличении/уменьшении степени детализации. Исследуется управление базами данных/метаданных в условиях интегрированного использования объектов данных, объектов метаданных и программ. Унифицированная вычислительная среда сохраняет расширяемость модели объектов данных. Представления объектов данных/метаданных предполагаются встроенными в вычислительную среду. Построена концептуальная оболочка для вычислений в терминах объектов. В работе применяются и развиваются методы бестипового и типового λ-исчисления, комбинаторной логики и вычислений в категории. Круг вопросов, рассмотренных в работе, является обобщением опыта их преподавания в различных учебных курсах по компьютерным наукам. } 17.1

[6] Гаврилов А.В., *Настраиваемая система программирования для категориальных вычислений*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1995. 17.1

[7] Гольцева Л.В., *Аппликативная вычислительная система с интенсиональными отношениями*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 1995. 17.1

[8] Зыков С.В., *Исследование и реализация интегрированной корпоративной информационной системы для решения задач управления персоналом*, Диссертация на соискание ученой степени

кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 2000.

[9] Забродин А.Л., *Исследование и реализация программного обеспечения управления данными для автоматизированных систем оперативного управления связью*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-053.03.04 МИФИ, Москва, 2000.

[10] Горелов Б.Б., *Исследование и реализация распределенной информационной системы управления финансовыми данными*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-212.130.03 МИФИ, Москва, 2003.

[11] Файбисович М. Л., *Исследование и реализация программных средств выбора альтернатив в среде Web*, Диссертация на соискание ученой степени кандидата технических наук, 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, Московский инженерно-физический институт, Диссертационный Совет Д-212.130.03 МИФИ, Москва, 2006.

Вячеслав Эрнстович **Вольфенгаген**
Лариса Юсифовна **Исмаилова**
Сергей Владимирович **Косиков**

Модели вычислений

Конспект лекций

Научный редактор: *Л. Ю. Исмаилова*
Макет: *Авторы*
Корректор: *Л. М. Зинченко*

ЗАО "ЮрИнфоР"
125009, Москва, Брюсов пер., 8-10, стр. 2, тел. (495) 743-73-96,
http://www.jurinfor.ru, эл. почта: info@jurinfor.ru

LBC 32.97
UDC 004
B721

Library of "JurInfoR"
Founded in 1994

Series: Computer Science and Information Technologies

V. E. Wolfengagen et al

Models of computation. Lecture notes. — Moscow.: "JurInfoR" Ltd., 2007. — IX+306 p.

The book is intended for computer science students, programmers and professionals who have already got acquainted with the basic courses and background on discrete mathematics. It may be used as a textbook for graduate course on theoretical computer science.

The book introduces a reader to the conceptual framework for thinking about computations with the objects. The several areas of theoretical computer science are covered, including the following: type free and typed λ-calculus and combinatory logic with applications, evaluation of expressions, computations in a category. The topics, covered in the book accumulated much experience in teaching these subjects in graduate computer science courses.

A rich set of examples and exercises, including solutions, has been prepared to stimulate the self studying and to make easier the job of instructor.

Center "JurInfoR"
Institute for Contemporary Education "JurInfoR-MSU"
Fax: +7 (495) 778-87-26. E-mail: vew@jmsuice.msk.ru