И. А. Кудрявцева, М. В. Швецкий

ПРОГРАММИРОВАНИЕ
КОМБИНАТОРНАЯ ЛОГИКА

УЧЕБНОЕ ПОСОБИЕ ДЛЯ ВУЗОВ

2-е издание, переработанное и дополненное

Рекомендовано Учебно-методическим отделом высшего образования в качестве учебного пособия для студентов высших учебных заведений, обучающихся по ИТ-направлениям

Книга доступна на образовательной платформе «Юрайт» urait.ru, а также в мобильном приложении «Юрайт.Библиотека»

Москва ▪ Юрайт ▪ 2021

УДК 004.4(075.8)
ББК 32.973я73
К88

***Авторы*:**

Кудрявцева Ирина Андреевна — доцент, кандидат педагогических наук, доцент кафедры информационных систем института информационных технологий и технологического образования Российского государственного педагогического университета имени А. И. Герцена (г. Санкт-Петербург);

Швецкий Михаил Владимирович — профессор, доктор педагогических наук, профессор кафедры компьютерной инженерии и программотехники Института компьютерных наук и технологического образования Российского государственного педагогического университета имени А. И. Герцена (г. Санкт-Петербург).

***Рецензенты*:**

Пиотровская К. Р. — доцент, доктор педагогических наук, профессор кафедры методики обучения математике и информатике факультета математики Российского государственного педагогического университета имени А. И. Герцена (г. Санкт-Петербург);

Матюшичев И. Ю. — кандидат технических наук, доцент.

Кудрявцева, И. А.

К88 Программирование: комбинаторная логика : учебное пособие для вузов / И. А. Кудрявцева, М. В. Швецкий. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2021. — 524 с. — (Высшее образование). — Текст : непосредственный.

ISBN 978-5-534-10620-6

Учебное пособие представляет собой систему упражнений и лабораторных работ по курсу «Программирование» и содержит теоретические сведения по комбинаторной логике, элементам теории типов, элементам теории категорий, а также задачи для самостоятельного решения. В пособии размещены программы на языке Haskell: интерпретатор λ-функций, представленных λ-термами, и программа для вывода типовой схемы λ-терма в STT.

Издание предназначено для студентов высших учебных заведений, обучающихся по ИТ-направлениям, аспирантов и преподавателей факультетов и институтов компьютерных наук и информационных технологий.

УДК 004.4(075.8)
ББК 32.973я73

ISBN 978-5-534-10620-6

Оглавление

Часть 2
ИСЧИСЛЕНИЕ КОМБИНАТОРОВ

Часть 3
ИСЧИСЛЕНИЕ КОМБИНАТОРОВ С ТИПАМИ

Часть 4
КАТЕГОРИАЛЬНАЯ КОМБИНАТОРНАЯ ЛОГИКА

Часть 5
ТРАНСФОРМАЦИЯ ПРОГРАММ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ HASKELL

Введение

За чистую математику, которая никогда не найдет себе применения!
Тост Г. Харди

В настоящее время программирование является массовой профессией.

К сожалению, как в обучении программированию, так и в повседневной практике программирования зачастую преобладает поверхностный, рецептурный подход, при котором элементарная грамотность во владении языками программирования представляется чуть ли не исчерпывающим содержанием профессии.

Разрыв между высоким математическим зарядом, получаемым в процессе обучения будущими математиками-программистами, и обыденностью рутинной работы по написанию программ создает у некоторых ощущение девальвации их математической квалификации при занятии программированием. Более того, бытует взгляд на программиста как на мало квалифицированного ремесленника, который занимается только кодированием алгоритма для компьютера.

Распространено также мнение, что программирования как научной дисциплины не существует, что оно растворяется в таких проблемах, как создание математических моделей разнообразных явлений и процессов и разработка общей структуры и архитектуры компьютера.

Так, *Г. Р. Громов* [3] шутливо заметил, что: «…для прижизненного процветания в околокомпьютерных науках оказалось достаточно выучить два несложных приёма: 1) в беседах с инженерами снисходительно объяснять, что непонятные им формализованные описания “общей теории всего” служат будущему прогрессу “недоразвитой” пока в математическом отношении науки об ЭВМ; 2) на вопросы профессиональных математиков о назначении сомнительных, с их точки зрения, формальных упражнений на околокомпьютерные темы следует мягко и по возможности проникновенным тоном разъяснить, что им — профессиональным математикам, увы, не дано понять требований практики, их удел — чистая наука…»

Некоторые считают, что с распространением информационных технологий специальность «программист» вовсе исчезнет. При этом

забывают, что работа с компьютером может стать доступной для непрофессионалов лишь в том случае, если предварительно усилиями высококвалифицированных программистов будет создано необходимое программное обеспечение.

Очевидно, что современное программирование — это достаточно сложная область научной деятельности, в которой получен ряд фундаментальных результатов, имеющих важное значение для практики.

Далее, можно заметить, что программирование породило ряд понятий, методов и приемов работы, связанных с термином «*программа*»:

1) как *объектом* деятельности (программа — это все-таки не абстрактный, а вполне *реальный объект*, представляющий собой либо текст на фиксированном языке программирования, либо цепочку битов в памяти компьютера);

2) как *предметом* деятельности (гносеологический аспект), которым являются многочисленные математические модели понятия «*программа*». Другими словами, программирование породило обширную и сложную проблематику, тесно связанную с некоторыми разделами математики;

3) как *средством* решения прикладных задач (онтологический аспект).

Остановимся на определении понятия «*теоретическое программирование*», которое относится к гносеологическому аспекту термина «*программа*». Это словосочетание построено по аналогии с названиями таких наук, как теоретическая физика, теоретическая механика и т. д. В такой аналогии есть глубокий смысл: во всех случаях теоретическая научная дисциплина изучает фундаментальные понятия и законы основной науки и на основании обнаруженных закономерностей строит более частные математические модели исследуемых объектов, на которых ставит и решает прикладные задачи.

Приведём несколько определений.

1. *Теоретическое программирование* (при широком толковании термина) — это *вся теория алгоритмов* (например, теорему Геделя о неполноте можно рассматривать как теорему теоретического программирования [2]).

2. *Теоретическое программирование* (в узком смысле слова) — это раздел *теории алгоритмов*, изучающий взаимоотношения программы как чисто синтаксического (неинтерпретированного) объекта и содержания (смысла, значения) программы.

Общая часть теоретического программирования и теории алгоритмов велика, и классическая теория алгоритмов оказала бесспорное влияние на программирование. Это влияние, однако, состояло

не в использовании каких-либо теорем, а носило скорее идейный характер.

Конечно, для теоретического программирования характерен интерес к порождённым практикой темам, которых не касалась классическая теория алгоритмов, таким как параллельное программирование или структуры данных.

3. *Теоретическое программирование* [4, с. 6] — это раздел математической науки, чьим объектом изучения являются математические абстракции *программ* — предписаний, выраженных на специальных алгоритмических языках, обладающих определённой информационной и логической структурой и подлежащих выполнению на автоматических вычислительных машинах.

Другими словами, *теоретическое программирование* (*А. П. Ершов*, по [5]) — это математическая дисциплина, изучающая синтаксические и семантические свойства *программ*, их структуру, преобразования, процесс их составления и исполнения.

Раскрыть содержание теоретического программирования помогает перечисление сложившихся к настоящему времени (2013 г.) направлений научных исследований.

Однако вначале приведём рубрикацию (по [5, с. 5—6]), сложившуюся к концу 70-х годов XX в.:

1) *математические основы программирования*; основная цель исследований — это развитие математического аппарата, ориентированного на теоретическое программирование, разработка общей теории машинных вычислений. Эти исследования тесно соприкасаются с логикой, алгеброй, теорией алгоритмов и вычислимых функций, теорией сложности вычислений, теорией автоматов, формальных языков и грамматик;

2) *теория схем программ* (*схематология*); здесь внимание концентрируется на изучении структурных свойств и преобразований программ, а именно тех, которые отличают программы от других способов задания алгоритмов. Главным объектом исследования становится *схема программы* — математическая модель программы, в которой с той или иной степенью детализации отражено строение программы и взаимодействие составляющих её компонентов;

3) *семантическая теория программ*; этот раздел теоретического программирования изучает методы формального описания *семантики программ*, семантические методы преобразований и доказательства утверждений о программах.

Семантика программы или *семантика конструкций* языков программирования — это их математический смысл для программиста (денотационная и аксиоматическая семантика) и смысл, описание функционирования для компьютера (операционная семантика).

В частности, работы по методам проверки семантической правильности программ нацелены на автоматизацию их отладки и автоматический синтез программ;

4) *теория вычислительных процессов и структур* (*теория параллельных вычислений*); исследования в этой области направлены на разработку и обоснование новых методов и средств программирования, прежде всего методов программирования параллельных процессов.

В частности, изучаются структуры и функционирование операционных систем, методы распараллеливания алгоритмов и программ, ведётся поиск новых архитектурных принципов конструирования вычислительных машин и систем на основе результатов и рекомендаций теоретического программирования и вычислительной математики;

5) *метрическая теория программ* (*наука о программах*) [6], которая является теорией разработки программ с использованием «методов и принципов классической экспериментальной науки».

В основу этой теории положено приближённое комбинаторное соотношение, определяющее длину программы через число простых (различных) операторов и простых операндов. Это простое соотношение обладает удивительно мощным экстраполирующим свойством: достаточно знать количество обрабатываемых данных и используемые операторы языка программирования для того, чтобы предсказать, какой длины будет программа, реализующая алгоритм. В теории классифицируются все несовершенства программы, уменьшающие их эффективность, производится прогнозирование времени разработки программ и (априорно) числа ошибок программирования по заданному алгоритму.

Хотя опыт и подтвердил ценность предложенного М. Холстедом подхода, последний не является исчерпывающим, поскольку в нём игнорируются такие специфические аспекты, как выбор мнемонических имён переменных, комментарии, сложность структур управления, выбор алгоритмов или структур данных, а также такие общие метрики, как удобство переноса, гибкость и эффективность.

Однако *Б. Шнейдерман* [7, с. 112] пишет: «Быть может, было бы целесообразно при каждой компиляции подсчитывать и выдавать на печать оценки программы по Холстеду, что позволило бы осуществлять обратную связь с программистами»;

6) *прикладные задачи теоретического программирования*; к ним в первую очередь относятся разработка и обоснование алгоритмов трансляции и алгоритмов автоматической оптимизации программ.

Однако диапазон практических задач, решаемых методами теоретического программирования, постоянно расширяется.

Остановимся на современном взгляде на структуру теоретического программирования.

Конечно же, не следует забывать, что теоретическое программирование основано на фундаментальных математических дисциплинах — *алгебре*, *математической логике*, *теории алгоритмов*. В силу

этого, изучение теоретического программирования, в частности математических основ программирования, является неотъемлемой частью фундаментальной подготовки в области информационных технологий.

Далее, анализ теоретических исследований, выполненных в области компьютерных наук, показывает, что работы используют идеи либо из *теории категорий*, либо из λ-*исчисления*, либо из *исчисления комбинаторов*, либо из всех этих трёх разделов математики сразу.

О важности овладения этими разделами математики свидетельствует хотя бы тот факт, что, открыв практически наугад труды любой конференции по теоретической информатике, можно найти не только явное использование λ-обозначений, но фактическую разработку собственного математического языка, опирающегося на λ-исчисление.

Итак, можно утверждать (см. [1, с. 27—28]), что к настоящему времени в теоретических исследованиях в области теоретического программирования сложился *основной математический аппарат*, который содержит несколько взаимосвязанных разделов, обеспечивающих рост как логики, так и компьютерных наук:

1) λ-*исчисление*, в котором изучаются способы построения чистого исчисления функциональной абстракции и аппликации функций.

λ-исчисление выполняет важную роль при изучении оснований математики, является основным аппаратом исследования систем типизации и инструментом для разработки и применения языков программирования;

2) *комбинаторная логика*, в котором показано, что связанные переменные можно исключить без потери выразительных возможностей формальной системы.

Комбинаторная логика нашла применение для построения оснований математики, а также в развитии методов и средств реализации языков программирования;

3) *теория типов языков программирования*;

4) *теория категорий*, в котором рассматривается язык математики, отличный от языка теории множеств и содержащий всего две сущности (*объекты и морфизмы*), причём морфизмы в свою очередь можно считать объектами.

Перечисленные выше направления (λ-исчисление, комбинаторная логика, теория типов, теория категорий) трудно рассматривать изолированно: они взаимно переплетены, и современная точка зрения заключается в их *совместном изучении*.

Более того, именно эти четыре раздела являются основой для конструирования языков программирования.

На первый взгляд круг рассматриваемых в этих разделах идей не является однородным, вызывая рост числа внешне различимых

формализаций и математических аппаратов. На деле происходит даже большее: каждое очередное исследование, как правило, содержит построение своего собственного математического аппарата.

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

Приведём определение понятия «*теоретическое программирование*», которого мы будем придерживаться всюду ниже.

Теоретическое программирование — это раздел математической науки, в котором:

1) *объектом изучения* являются *реальные программы*, представляющие собой либо текст на фиксированном языке программирования, либо цепочку битов в памяти компьютера;

2) *предметом изучения* являются математические абстракции *программ* — предписаний, выраженных на специальных алгоритмических языках, обладающих определённой информационной и логической структурой и подлежащих выполнению на компьютерах;

3) *методом* является конструирование *исчислений*, моделирующих синтаксис и семантику программ, которые (исчисления) построены в следующих разделах математики: λ-исчисление, теория комбинаторов, теория типов, семантическая теория языков программирования;

4) *метаязыками* теоретического программирования являются два языка: язык теории множеств и язык теории категорий.

Мы будем трактовать термин «*основание*» как совокупность концепций, на базе которых строится конкретная дисциплина, причём значение этих, обычно весьма тонких, концепций удаётся оценить не начинающему, а человеку, достаточно искушённому в предмете.

Более того, обращаясь в этом смысле к «основаниям» своей дисциплины, исследователь-ученый, как правило, пользуется формальной логикой, которая в свою очередь является «основой», с помощью которой строятся содержательные теории. Так, например, основаниями математики считается математическая логика — наука о формальных математических теориях. Более конкретно, математическая логика занимается построением формальных языков, предназначенных для представления таких фундаментальных понятий, как «*функция*», «*отношение*», «*аксиома*», «*доказательство*», и изучением основанных на этих языках логических и логико-математических исчислений.

С учётом всего сказанного можно построить граф научной дисциплины «Теоретическое программирование», которая безусловно является математической:

Пособие содержит следующие разделы:

1) бестиповое λ-исчисление;
2) исчисление комбинаторов;
3) исчисление комбинаторов с типами;
4) категориальная комбинаторная логика;
5) трансформация программ в языке программирования Haskell;
6) эзотерические языки программирования.

Пособие состоит из введения, системы упражнений и лабораторных работ, а также списка литературы.

Упражнения и лабораторные работы строятся по следующей схеме:

1) выделяются обязательные результаты обучения (что нужно знать и уметь по окончании выполнения работы);

2) теоретические сведения содержат основные понятия и материал, способствующий решению практических задач;

3) далее следуют примеры решения некоторых типов упражнений или демонстрационные примеры, представляющие собой образец оформления решения задач на языке программирования;

4) заключительный раздел представляет собой задания для самостоятельного решения, где, возможно, некоторые задачи снабжены ответами или подсказками.

Список учебной, научной и научно-популярной литературы, приведенный в пособии, позволяет использовать его как справочник.

Используя материал пособия, можно построить содержание учебной дисциплины «Теоретическое программирование» для студентов факультетов информационных технологий.

Работа между авторами распределилась следующим образом (создание упражнения или лабораторной работы означает разработку содержания обучения, подбор демонстрационных примеров, построение типологии и подбор задач для самостоятельного решения):

1) доц. *И. А. Кудрявцева* написала «Введение», содержащее концепцию построения пособия;

2) лабораторные работы № 7, 13, 15, 16 и упр. 1, 2, 3, 4, 5, 6, 12, 12_1 написаны *И. А. Кудрявцевой*; причём важнейшей особенностью пособия являются программы, написанные на языке Haskell:

а) интерпретатор функций, представленных λ-термами;

б) интерпретатор функций, представленных в бесточечной форме записи с помощью комбинаторов.

3) лабораторные работы № 13, 14 и упр. № 1, 2, 3, 4, 5, 8, 9, 10, 11, 12_1 написаны *М. В. Швецким*.

Авторы выражают благодарность за любезно предоставленные материалы доценту, канд. пед. наук *А. В. Голановой* (упр. № 1—4, 8—9) и доценту, канд. пед. наук *И. И. Семёновой* (упр. № 2_1).

2000, июнь — ноябрь 2013 Кудрявцева И. А., Швецкий М. В.
Санкт-Петербург

Часть 1
БЕСТИПОВОЕ λ-ИСЧИСЛЕНИЕ

Упражнение 1
КОМБИНАТОРНАЯ ЛОГИКА: СИНТАКСИС БЕСТИПОВОГО λ-ИСЧИСЛЕНИЯ

Бог — выше логики.
П. Флоренский

Обязательные результаты обучения:

- ***знать вспомогательные понятия***

— *функция как график*;
— *частично рекурсивные функции*;

- ***знать основные понятия***

— *комбинаторная логика* (*в узком и широком смысле*);
— *λ-исчисление*;
— *алфавит бестипового λ-исчисления*;
— *λ-терм*, *соглашения об обозначениях λ-термов*;
— *длина λ-терма*, *ранг λ-терма*;
— *древесное доказательство правильности построения λ-терма*;
— *λ-формула*;
— *язык λ-исчисления*;
— *λ-подтерм λ-терма*;
— *связанное вхождение предметной переменной в λ-терм*, *свободное вхождение предметной переменной x в λ-терм*;
— *свободная предметная переменная λ-терма*, *связанная предметная переменная λ-терма*;
— *множество свободных переменных λ-терма*, *множество связанных переменных λ-терма*;
— *комбинатор* (*замкнутый λ-терм*);
— *подстановка λ-терма N вместо всех свободных вхождений предметной переменной x в λ-терм M*;
— *соглашение Барендрегта об именах переменных*;
— *лемма о подстановке*;
— *теоретико-множественная интерпретация λ-термов*;

- ***уметь***

— *распознавать λ-терм с помощью индуктивного определения*;
— *опускать и восстанавливать скобки в λ-терме*;
— *определять свободные и связанные переменные в λ-терме*;
— *выполнять операцию «подстановка» в λ-терме*;
— *интерпретировать λ-термы в математическом анализе*;

- ***владеть***

— *основными понятиями*, *представленными выше*;
— *методами решения задач*, *представленных в Упражнении.*

Теоретические сведения

Понятие «комбинаторная логика»

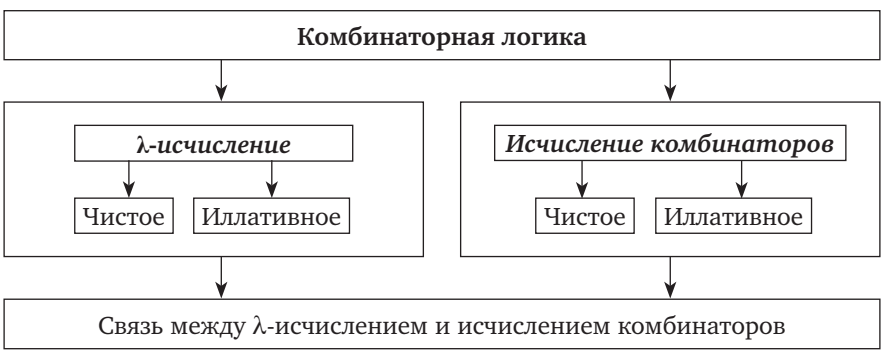
Определение.

1. *Комбинаторная логика* (от лат. *combinare* — «соединять, сочетать») — раздел математической логики, изучающий её основания.

2. [52, с. 275—276] *Комбинаторная логика* (*в широком смысле*) — это раздел *логики*, посвященный изучению и анализу таких понятий и методов, как «*переменная*», «*функция*», операция «*подстановка*», классификация предметов по типам или категориям.

3. (По [14, с. 181]) *Комбинаторная логика* (*в узком смысле*) — это ветвь *математической логики*, изучающая *комбинаторы* и их свойства. Одной из первых в комбинаторной логике была задача, которая состояла в сведении первичных логических понятий к минимальному числу достаточно простых понятий, называемых *комбинаторами*.

Общая структура содержания этого раздела логики может быть представлена следующим образом:



Лямбда-исчисление (*λ-исчисление*), разработанное А. Чёрчем, и его эквивалент без предметных переменных (*исчисление комбинаторов* М. И. Шейнфинкеля и Х. Б. Карри) были изобретены около 1930 г.

Ранее мы рассматривали понятие «функция» (по Дирихле), вводимое с помощью понятия «график», т. е. с помощью множества упорядоченных пар, содержащих значения аргумента и значения функции.

Намерением основателей комбинаторной логики было изучение понятия «функции» *как правила.* Понятие «*функция*», вводимое с помощью понятия «*правило*», описывает процесс перехода от аргумента к значению, кодируемый правилом.

Правила описываются (русскими) словами, аргументы и значения также выражены (русскими) словами (более конкретно, мы мо-

жем думать о правилах как о программах для некоторых исполнителей).

И в λ-исчислении, и в исчислении комбинаторов приходится иметь дело с *бестиповой алгебраической системой*, где объекты изучения являются одновременно и функциями, и аргументами (в частности, функция может применяться к самой себе).

Определение (*содержательное*).

1. (По [9, с. 14]) λ-*исчисление* — это бестиповая теория, рассматривающая функции как правила (а не как график). Понятие функции как закона или правила подразумевает переход от аргумента к значению, т. е. *процесс*, закодированный некоторым определением. Другими словами, λ-исчисление рассматривает функции как правила, чтобы подчеркнуть именно их *вычислительные аспекты*.

2. λ-*исчисление* — это формализм для представления функций и способов их комбинирования, который изучает функции в связи с их *аппликативным* (англ. *application* — «приложение») поведением. Поэтому *аппликация* (применение функции к аргументу) является исходной операцией λ-исчисления.

3. (По [17, с. 121]) λ-*исчисление* — это исчисление анонимных (*безымянных*) функций, которое предоставляет: 1) метод представления функций; 2) множество правил вывода для синтаксического преобразования функций.

Определение (*для знатоков частично рекурсивных функций*) (по [8, с. 278]).

λ-*исчисление* — это формализм для представления класса частичных функций на натуральных числах (λ-*определимых функций*), который оказывается в точности классом *частично рекурсивных функций*.

Эквивалентность между λ-определимыми функциями и частично рекурсивными функциями была одним из аргументов, которые А. Чёрч использовал для защиты своего *тезиса* (*тезиса Чёрча-Клини*), предлагающего отождествление интуитивного класса эффективно-вычислимых функций с классом частично рекурсивных функций.

Далее А. Тьюринг показал, что, несмотря на свой очень простой синтаксис, λ-исчисление способно *изобразить* (выразить) все механически вычислимые функции, описываемые *машиной Тьюринга*.

Замечание

Исторически λ-исчисление играло центральную роль в первых исследованиях по теории рекурсивных функций:

1) первая неразрешимая алгоритмическая проблема была построена А. Чёрчем как проблема о термах λ-исчисления (имеют ли λ-термы нормальную форму);

2) теорема о неподвижной точке для λ-исчисления вдохновила С. Клини на теорему рекурсии;

3) первое определение рекурсивных ординалов, принадлежащее А. Чёрчу и С. Клини, опиралось на λ-исчисление.

А. Чёрч первоначально строил λ-исчисление как часть общей системы функций, которая должна стать основанием логики и математики. Парадокс С. Клини и Дж. Россера показал, что эта система оказалась *противоречивой* (излагаемая ниже теория — это извлечённая из неё непротиворечивая подтеория). После этого А. Чёрч потерял интерес к использованию λ-исчисления для обоснования математики.

Бестиповое λ-исчисление как формальная система

> В то же время самостоятельно я познакомился с λ-исчислением Чёрча и Карри (которое вначале в буквальном смысле было для меня кошмаром).
>
> *Д. Скотт* (*лауреат премии Тьюринга*, 1976)

Рассмотрим один из вариантов *чистой комбинаторной логики*, который называется *бестиповым λ-исчислением*.

Определение (*с позиций математической логики*).
λ-*исчисление* — это исчисление λ-*термов* и λ-*формул*, которое определяется:

1) *языком* λ-*исчисления* как некоторым языком в алфавите;
2) *аксиомами* и *правилами вывода*;
3) понятием «*доказательство в* λ-*исчислении*»;
4) *интерпретацией* исчисления λ-термов и λ-формул.

---

1. Язык λ-исчисления

> Невозможно отделить язык от науки или науку от языка, поскольку любая естественная наука всегда использует три вещи: последовательность феноменов, на которые она опирается, краткие описания — концепции этих феноменов, которыми они представляются в мышлении, и слова, в которых выражаются концепции. Для движения к концепции необходимо слово; для описания явления необходима концепция. Все три отражают одну и ту же реальность.
>
> *А. Лавуазье*, 1789

Определим язык λ-исчисления как язык в некотором алфавите.

Определение.

Алфавит бестипового λ-исчисления (обозначается A_λ) — это множество

$$A_\lambda \overset{\text{def}}{=} G_1 \cup G_2 \cup G_3 \cup G_4,$$

1) $G_1 \overset{\text{def}}{=} \{x_1, x_2, x_3, \ldots\}$ — *множество предметных переменных*;

2) $G_2 \overset{\text{def}}{=} \{=\}$ — *множество предикатных символов*; «=» — двухместный предикатный символ (читается: «*равно*» или «*конверсия*»);

3) $G_3 \overset{\text{def}}{=} \{\lambda\}$ — *множество логических символов*; «λ» — логический символ (читается: «*лямбда*», «*абстрактор*», «*функция от*»);

4) $G_4 \overset{\text{def}}{=} \{(,), .\}$ — *множество вспомогательных символов*; символ «**(**» читается: «*открывающая скобка*», символ «**)**» читается: «*закрывающая скобка*», символ «**.**» читается: «*точка*», «*которая возвращает*».

Определим индуктивно понятие «*λ-терм*» («*лямбда-терм*» или просто «*терм*»).

В метаязыке, предназначенном для описания λ-исчисления, λ-термы будем обозначать *прописными латинскими буквами*, а предметные переменные — *строчными латинскими буквами*.

Определение.

1. Предметные переменные являются *λ-термами*.

2. Если M и N — λ-термы, то (MN) (читается: «*аппликация λ-термов M и N*») является *λ-термом*.

3. Если M — λ-терм, а x — предметная переменная, то $(\lambda x.M)$ (читается: «*абстракция λ-терма M*») является *λ-термом*.

При этом x будем называть *переменной абстракции*, а λ-терм M — *телом абстракции*.

4. Других λ-термов, кроме построенных по пп. 1—3, нет.

Работа с демонстрационными примерами

См. Пример 0, Пример 1.

Соглашения об обозначениях λ-термов.

При записи λ-термов в метаязыке часто используются следующие *соглашения об обозначениях*, позволяющие при записи λ-термов опускать скобки и повторяющиеся символы λ:

а) *соглашение о восстановлении скобок по ассоциативности влево*: слово метаязыка

$$M_1M_2\ldots M_n$$

является сокращённой записью λ-терма

$$(\ldots((M_1M_2)M_3)\ldots M_n);$$

б) *соглашение об опускании повторяющихся символов* λ *и о восстановлении скобок по ассоциативности вправо*: слово метаязыка

$$\lambda x_1 x_2 \ldots x_n.M$$

является сокращённой записью λ-терма

$$(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)))$$

(говорят, что «тела абстракций простираются *направо* максимально далеко»);

в) обобщающее *соглашение*: слово метаязыка

$$\lambda x_1 x_2 \ldots x_n.M_1 M_2 \ldots M_k$$

является сокращенной записью λ-терма

$$(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.\ (\ldots((M_1 M_2)M_3)\ldots M_k))\ldots))).$$

Примеры.

λ-терм	Сокращённая запись в метаязыке
$(((M_1M_2)M_3)M_4)$	$M_1M_2M_3M_4$
$(((xz)y)z)$	$xzyz$
$((xz)(yz))$	$xz(yz)$
$(\lambda x_1.(\lambda x_2.(\lambda x_3.M)))$	$\lambda x_1 x_2 x_3.M$
$(\lambda x_1.(\lambda x_2.(\lambda x_3.(((M_1M_2)M_3)M_4))))$	$\lambda x_1 x_2 x_3.M_1M_2M_3M_4$

Определение (*Х. Карри*).

Символом «≡» метаязыка будем обозначать бинарное отношение «*графическое равенство* λ*-термов*», устанавливающее их полное синтаксическое совпадение.

Другими словами, будем употреблять запись

$$M \equiv N$$

для обозначения *побуквенного совпадения* λ-термов M и N.

Определение (по [9, с. 35]).

Длиной λ*-терма* M (обозначается $\|M\|$) называется количество символов в записи λ-терма.

Определение.

Рангом λ*-терма* называется функция rank: $\Lambda \to \mathbb{N}$, определяемая следующим индуктивным образом:

$$\operatorname{rank}(M) \overset{\text{def}}{=} \begin{cases} 0, \text{если } M \equiv x_i, i \in \mathbb{N}, \\ \operatorname{rank}(N_1) + \operatorname{rank}(N_2), \text{если } M \equiv (N_1 N_2), N_1 N_2 \in \Lambda, \\ 1 + \operatorname{rank}(N), \text{если } M \equiv (\lambda x.N), N \in \Lambda. \end{cases}$$

Соглашение об обозначениях.
Обозначим Λ — множество λ-термов.

Определение (*для знатоков теории формальных грамматик*).
Определим понятие «множество X-термов» так:

$$\Lambda \overset{\text{def}}{=} G_1 \,|\, (\Lambda\Lambda) \,|\, (\lambda G_1 . \Lambda).$$

Определение.
Определим понятие «*доказательство правильности построения λ-терма*» индуктивно, используя следующие фигуры:

$$\frac{\text{Посылка}}{\text{Следствие}}, \quad \frac{\text{Посылка\_1} \;\; \text{Посылка\_2}}{\text{Следствие}}.$$

1) древесным доказательством аксиомы исчисления λ-термов является она сама:

$$\frac{x_i \in G_1}{x_i \in \Lambda}, i \in \mathbb{N} \setminus \{0\} \text{ (аксиома)};$$

2) если $\boldsymbol{D}_1$, $\boldsymbol{D}_2$ — древесные доказательства λ-термов M_1, M_2 соответственно, а λ-терм T является непосредственным следствием λ-термов M_1, M_2 по одному из правил вывода λ-термов:

$$\frac{M \in \Lambda \;\; N \in \Lambda}{(MN) \in \Lambda} \text{ (аппликация } \lambda\text{-термов } M \text{ и } N),$$

$$\frac{x_1 \in G_1 \;\; M \in \Lambda}{(\lambda \mathrm{x}_1 . M) \in \Lambda}, \; i \in \mathbb{N} \setminus \{0\} \text{ (абстракция } \lambda\text{-терма } M),$$

то слово вида

$$\frac{\boldsymbol{D}_1 \;\; \boldsymbol{D}_2}{T}$$

— древесное доказательство правильности построения λ-терма T.

Пусть $M, N \in \Lambda$.

Определение.
Формулой бестипового λ-исчисления (λ-*формулой* или просто *формулой*) называется слово вида

$$M = N.$$

Соглашение об обозначениях.
Обозначим F_λ — множество λ-формул.

Определение.
Языком λ-исчисления (или *λ-языком*) называется кортеж

$$L_\lambda \overset{\text{def}}{=} \langle A_\lambda, \Lambda, F_\lambda \rangle.$$

2. Свободные и связанные переменные

Определение (по [9, с. 37]).
Множество подтермов λ-терма M (обозначается Sub(*M*)) определим индуктивно следующим образом:

1) $\mathrm{Sub}(x) \overset{\text{def}}{=} \{x\}$, если $x \in G_1$;

2) $\mathrm{Sub}((M_1M_2)) \overset{\text{def}}{=} \{M_1M_2\} \cup \mathrm{Sub}(M_1) \cup \mathrm{Sub}\ (M_2)$, если $M_1 \in \Lambda$, $M_2 \in \Lambda$;

3) $\mathrm{Sub}((\lambda x.\ M)) \overset{\text{def}}{=} \{\lambda x.\ M\} \cup \mathrm{Sub}(M)$, если $M \in \Lambda$.

Работа с демонстрационными примерами

См. Пример 2.

Определение.
λ-подтермом M λ-терма N будем называть λ-терм $M \in \mathrm{Sub}(N)$.

Пусть N_1 и N_2 — λ-подтермы λ-терма M.

Определение (*важное*).
Будем говорить, что *λ-термы N_1 и N_2 дизъюнктны*, если они не имеют общих вхождений символов.

λ-термы являются *словами в алфавите бестипового λ-исчисления*, поэтому для λ-термов определено отображение «*вхождение предметной переменной в λ-терм*».

Определение.
1. *Связанным вхождением предметной переменной x в λ-терм M* называется вхождение предметной переменной *x* в λ-терм *M*, для которого *существует* такой λ-терм *N*, что $(\lambda x.N) \in \mathrm{Sub}(M)$.
2. *Свободным вхождением предметной переменной x в λ-терм M* называется вхождение предметной переменной *x* в λ-терм *M*, для которого не существует такого λ-терма *N*, что $(\lambda x.N) \in \mathrm{Sub}(M)$.

Определение (*для знатоков математической логики*).
1. Говорят, что *переменная x входит связанно в λ-терм M*, если *x* находится в *области действия* символа λ*x*.

2. Говорят, что *переменная x входит свободно в λ-терм M*, если *x* не находится в *области действия* символа λ*x*.

Определение.

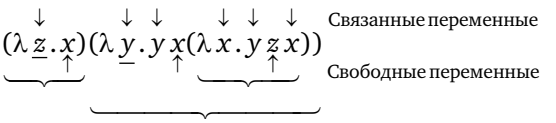
1. *Свободной предметной переменной λ-терма M* называется предметная переменная, имеющая хотя бы одно свободное вхождение в λ-терм *M*.

2. *Связанной предметной переменной λ-терма M* называется предметная переменная, имеющая хотя бы одно связанное вхождение в λ-терм *M*.

Примеры.

1. В λ-терме **(**λ*z*.*x***)(**λ*y*.*yx***(**λ*x*.*yzx***))** самое левое вхождение *z* является связанным, а самое правое — свободным; два самых левых вхождения *x* являются свободными, а два самых правых — связанными; все три вхождения *y* являются связанными.

Сказанное представим так:



2. В λ-терме (*x*(λ*x*.*x*)) предметная переменная *x* является одновременно и свободной, и связанной.

Работа с демонстрационными примерами

См. Пример 3, Пример 4.

Определение (по [9, с. 36]).

Множество свободных переменных λ-терма M (обозначается FV(*M*)) определим индуктивно:

1) $\mathrm{FV}(x) \overset{\text{def}}{=} \{x\}$;

2) $\mathrm{FV}(MN) \overset{\text{def}}{=} \mathrm{FV}(M) \cup \mathrm{FV}(N)$;

3) $\mathrm{FV}(\lambda x.M) \overset{\text{def}}{=} \mathrm{FV}(M) \setminus \{x\}$.

Определение.

Множество связанных переменных λ-терма M (обозначается CV(*M*)) определим индуктивно:

1) $\mathrm{CV}(x) \overset{\text{def}}{=} \varnothing$;

2) $\mathrm{CV}(MN) \overset{\text{def}}{=} \mathrm{CV}(M) \cup \mathrm{CV}(N)$;

3) $\mathrm{CV}(\lambda x.M) \overset{\text{def}}{=} \{x\} \cup \mathrm{CV}(M)$.

Примеры.

1. $\mathrm{FV}((\lambda z.\, x)(\lambda y.\, yx(\lambda x.\, yzx))) \overset{\text{def}}{=} \mathrm{FV}(\lambda z.\, x) \cup \mathrm{FV}(\lambda y.\, yx(\lambda x.\, yzx)) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} (\{x\} \setminus \{z\}) \cup (\mathrm{FV}(yx(\lambda x.\, yzx)) \setminus \{y\}) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} (\{x\} \setminus \{z\}) \cup (\{y, x, z\} \setminus \{y\}) = \{x,\}.$

2. $\mathrm{CV}((\lambda z.\, x)(\lambda y.\, yx(\lambda x.\, yzx))) \overset{\text{def}}{=} \mathrm{CV}(\lambda z.\, x) \cup \mathrm{CV}(\lambda y.\, yx(\lambda x.\, yzx)) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{z\} \cup \mathrm{CV}(x) \cup \{y\} \cup \mathrm{CV}(yx(\lambda x.\, yzx)) \overset{\text{def}}{=} \{z, y\} \cup \{x\} = \{x, y, z\}.$

Работа с примерами решённых упражнений

См. Пример 1.

Работа с демонстрационными примерами

См. Пример 5, Пример 6.

Определение (*важное*) (по [9, с. 36]).
Комбинатором (*замкнутым λ-термом*) назовём λ-терм M, для которого

$$\mathrm{FV}(M) = \varnothing.$$

Замечания

1. Множество связанных переменных часто обозначают BV (англ. *bounded* — «ограниченный») или CV (англ. *combined variable* — «связанная переменная»).

2. Связанная переменная играет роль своеобразного *джокера*.

Джокер (англ. *joker* — «шут») — представляет собой специальную игральную карту, имеющуюся в большинстве современных колод игральных карт, состоящих из 54 карт.

Чаще всего джокера изображают в виде шута (*синтаксис*!). Обычно в колоде идут две карты джокера, которые заметно отличаются друг от друга (например, в некоторых колодах один из джокеров выполнен в цвете, в то время как другой — чёрно-белый).

Роль (*семантика*!) джокера в карточных играх разная: 1) в основном он служит заменой любой карте; 2) иногда цветной джокер может заменять любую карту «червей» или «бубен», в то время как чёрно-белый — «пики» или «трефы».

Итак, λ выполняет функции *квантора*.

---

3. Операция «подстановка»

Я вижу слова, когда произношу их.
Я слышу фальшь в фальшивом слове…
Э. Хемингуэй. Нужна собака-поводырь

Определим фундаментальную для λ-исчисления операцию

$$(\circ)^{\circ}_{\circ} : \Lambda \times G_1 \times \Lambda \to \Lambda,$$

которую будем называть *подстановкой λ-терма N вместо всех свободных вхождений предметной переменной x в λ-терм M*.

Пусть x, y, z — буквы метаязыка, обозначающие предметные переменные, $M, M_1, M_2, N \in \Lambda$.

Определение (*по Х. Б. Карри*) [9, с. 89; с. 567; 18, с. 122].
Результат подстановки λ-терма N вместо всех свободных вхождений предметной переменной x в λ-терм M обозначается $(M)^x_N$ и определяется индуктивно следующим образом:

($\mathbf{p_1}$) $(x)^x_N \overset{\text{def}}{=} N$;

($\mathbf{p_2}$) $(M)^x_N \overset{\text{def}}{=} M$, если $x \notin \mathrm{FV}(M)$;

($\mathbf{p'_2}$) $(y)^x_N \overset{\text{def}}{=} y$, если *x графически* не совпадает с y;

($\mathbf{p_3}$) $(M_1M_2)^x_N \overset{\text{def}}{=} (M_1)^x_N(M_2)^x_N$;

($\mathbf{p_4}$) $(\lambda y.\, M)^y_N \overset{\text{def}}{=} \lambda y.\, M$;

($\mathbf{p_5}$) *подстановка без коллизии*:

$(\lambda y.\, M)^x_N \overset{\text{def}}{=} \lambda y.\, (M)^x_N$, если $y \notin \mathrm{FV}(N)$;

($\mathbf{p_6}$) *устранение коллизии имён переменных при подстановке*:

$(\lambda y.\, M)^x_N \overset{\text{def}}{=} \lambda z.\, ((M)^y_z)^x_N$, если $y \in \mathrm{FV}(N)$, $z \notin \mathrm{FV}(M) \cup \mathrm{FV}(N)$.

Примеры (выполнения операции «подстановка»).

1. $(\lambda x.\, fxy)^y_x \overset{\text{(p6) def}}{=} \lambda z.\, (fzy)^y_x \overset{\text{(p3, p1, p2, p2′) def}}{=} \lambda z.\, fzx.$

2. $(\lambda y.\, yx)^x_y \overset{\text{(p6) def}}{=} \lambda z.\, ((yx)^y_z)^x_y \overset{\text{(p3, p1, p2′) def}}{=} \lambda z.\, (zx)^x_y \overset{\text{(p3) def}}{=} \lambda z.\, (z)^x_y(x)^x_y \overset{\text{(p2) def}}{=} \lambda z.\, z(x)^x_y \overset{\text{(p1) def}}{=} \lambda z.\, zy.$

3. $(\lambda y.\, x)^x_N \overset{\text{(p5) def}}{=} \lambda y.\, (x)^x_N \overset{\text{(p1) def}}{=} \lambda y.\, N$, если $y \notin \mathrm{FV}(N)$.

4. $(\lambda y.\, x)^x_N \overset{\text{(p6) def}}{=} \lambda z.\, ((x)^y_z)^x_N \overset{\text{(p2′) def}}{=} \lambda z.\, (x)^x_N \overset{\text{(p1) def}}{=} \lambda z.\, N$, если $y \in \mathrm{FV}(N)$.

5. $(\lambda y.\, yx)^x_N \overset{\text{(p5) def}}{=} \lambda y.\, (yx)^x_N \overset{\text{(p3, p2′, p1) def}}{=} \lambda y.\, (yN)$, если $y \notin \mathrm{FV}(N)$.

6. $(\lambda y.\, yx)^x_N \overset{\text{(p6) def}}{=} \lambda z.\, ((yx)^y_z)^x_N \overset{\text{(p3, p1, p2′) def}}{=} \lambda z.\, (zx)^x_N \overset{\text{(p3, p2′, p1) def}}{=} \lambda z.\, (zN)$, если $y \in \mathrm{FV}(N)$.

7. $(\lambda y.\, \lambda x.\, fxy)^x_{2y} \overset{\text{def}}{=} \lambda z.\, (\lambda x.\, fxz)^x_{2y} \overset{\text{def}}{=} \lambda z.\, \lambda x.\, f(x)z.$

В связи с приведённым выше определением результата подстановки λ-терма N вместо всех свободных вхождений предметной переменной x в λ-терм M, можно (!) принять следующее.

Соглашение Барендрегта об именах переменных (*Barendregt convention*) [9, с. 38—39].

Если в определённом математическом контексте (определении, доказательстве) встречаются λ-термы, то подразумевается, что:

1) *связанные переменные в них выбраны отличными от свободных переменных*;

2) имена связанных переменных должны быть различными.

Используя это соглашение, можно работать с λ-термами «наивным» образом, т. е. выполнять операцию подстановки над λ-термами, не проверяя наличие коллизии переменных при подстановке.

Пример.

Пусть принимается соглашение Барендрегта и необходимо выполнить операцию «*подстановка*»

$$(\lambda y.\, M)^x_N.$$

Тогда очевидно, что $y \notin \mathrm{FV}(N)$ и применяется *подстановка без коллизии переменных.*

Работа с демонстрационными примерами

См. Пример 7.

4. Лемма о подстановке

Лемма (*о подстановке*) (по [9, с. 39—40]).

Если x графически не совпадает с y и $x \notin \mathrm{FV}(L)$, то

$$((M)^x_N)^y_L = ((M)^y_L)^x_{(N)^y_L}.$$

Доказательство. Воспользуемся индукцией по построению терма M.

$1_а$) Пусть $M \equiv x$. Тогда

$$((x)^x_N)^y_L \overset{(\mathrm{p1})}{\equiv} (N)^y_L,\ ((x)^y_L)^x_{(N)^y_L} \overset{(\mathrm{p2'})}{\equiv} (x)^x_{(N)^y_L} \overset{(\mathrm{p1})}{\equiv} (N)^y_L.$$

$1_б$) Пусть $M \equiv y$. Тогда

$$((y)^x_N)^y_L \overset{(\mathrm{p2'})}{\equiv} (y)^y_L \overset{(\mathrm{p1})}{\equiv} L,\ ((y)^y_L)^x_{(N)^y_L} \overset{(\mathrm{p1})}{\equiv} (L)^x_{(N)^y_L} \overset{(\mathrm{p2})}{\equiv} L.$$

$1_в$) Пусть $M \equiv y$ (z графически не совпадает с x и y). Тогда

$$((z)^x_N)^y_L \overset{(\mathrm{p2'})}{\equiv} (z)^y_L \overset{(\mathrm{p2})}{\equiv} z,\ ((z)^y_L)^x_{(N)^y_L} \overset{(\mathrm{p2'})}{\equiv} (z)^x_{(N)^y_L} \overset{(\mathrm{p2'})}{\equiv} z.$$

2) Пусть $M \equiv (PQ)$. Предположим, что для любого λ-терма P («меньшего» λ-терма M) выполняется:

$$((P)^x_N)^y_L = ((P)^y_L)^x_{(N)^y_L}.$$

Тогда

$$((PQ)^x_N)^y_L \overset{(p3)}{\equiv} ((P)^x_N (Q)^x_N)^y_L \overset{(p3)}{\equiv} ((P)^x_N)^y_L ((Q)^x_N)^y_L,$$

$$((PQ)^y_L)^x_{(N)^y_L} \overset{(p3)}{\equiv} ((P)^y_L (Q)^y_L)^x_{(N)^y_L} \overset{(p3)}{\equiv} ((P)^y_L)^x_{(N)^y_L} ((Q)^y_L)^x_{(N)^y_L}.$$

3) Пусть $M \equiv \lambda z.\, P$. Предположим, что z графически не совпадает с x и y, $z \notin \mathrm{FV}(N)$ и $z \notin \mathrm{FV}(L)$.

$$((\lambda z.\, P)^x_N)^y_L \underset{\substack{x \in \mathrm{FV}(\lambda z.\, P),\\ z \notin \mathrm{FV}(N)}}{\overset{(p5)}{\equiv}} (\lambda z.\, (P)^x_N)^y_L \underset{\substack{x \in \mathrm{FV}(\lambda z.\, (P)^x_N),\\ z \notin \mathrm{FV}(L)}}{\overset{(p5)}{\equiv}} \lambda z.\, ((P)^x_N)^y_L,$$

$$((\lambda z.P)^y_L)^x_{(N)^y_L} \underset{\substack{y \in \mathrm{FV}(\lambda z.\, P),\\ z \notin \mathrm{FV}(L)}}{\overset{(p5)}{\equiv}} (\lambda z.\, (P)^y_L)^x_{(N)^y_L} \underset{\substack{x \in \mathrm{FV}(\lambda z.\, (P)^x_N),\\ z \notin \mathrm{FV}((N)^y_L)}}{\overset{(p5)}{\equiv}} \lambda z.\, ((P)^y_L)^x_{(N)^y_L}.$$

4) Пусть $M \equiv \lambda z.\, P$. Предположим, что z графически не совпадает с x и y, $z \in \mathrm{FV}(N) \cup \mathrm{FV}(L)$.

Рассмотрите этот случай *самостоятельно.*

Лемма подстановки доказана.

5. Представление об интерпретации λ-термов

> Школу я, разумеется, закончил и научился писать на русском языке разные слова и смог бы, пожалуй, соединять их е предложения и даже в абзацы, каждый новый абзац начинать с красной строки, а между главами оставлять пробелы в две строчки, и так страница за страницей. Но я ведь понимаю, что дело не в этом.
>
> *Э. Кочергин. Прочтения*

Напомним, что для задания функции достаточно указать множества X, Y и *правило сопоставления*, которое записывается в виде:

$$x \mapsto f(x),$$

и по которому каждый $x \in X$ объединяется в пару с некоторым $y \in Y$ (в частности, эти пары могут быть просто перечислены).

Определение.
Теоретико-множественной интерпретацией λ-терма

$$\lambda x.(fx)$$

называется *правило сопоставления* $x \mapsto f(x)$.

Приведём *содержательную теоретико-множественную интерпретацию* λ-термов с помощью нескольких примеров:

1) $(\lambda x.x)$ интерпретируется тождественной функцией $y = x$ со значением, равным её аргументу;

2) $(\lambda x.f(gx))$ интерпретируется композицией функций f и g, т. е. функцией со значением, равным $f(g(x))$ для аргумента x;

3) $(\lambda x.(fg)x)$ интерпретируется применением *функционала* f к двум аргументам: функциональному — g и «обычному» — x. Этот способ интерпретации позволяет описывать *функции высших порядков*;

(4) $(\lambda f.\lambda x.fx)$ интерпретируется функцией высшего порядка, значением которой для функции f и аргумента x является значение $f(x)$.

λ-термы можно интерпретировать с помощью *бинарных деревьев* (см. [9, с. 568]).

Работа с примерами решённых упражнений

См. Пример 2.

Замечания

1. [21, с. 112] «…лямбда-запись преодолевает нерегулярность обозначений при обычном функциональном способе выражения».

2. В C++11 включена поддержка λ-функций. Это анонимные функции, которые используются только один раз (часто в качестве аргументов) другой функции. Например,

```
int main()
{
int a = fun([=] (int x) { return x*x*x; });
…
}
```

Синтаксис λ-функций аналогичен синтаксису любой другой функции, только место имени занимает выражение [=], а тип возвращаемого значения обычно задавать не надо — компилятор выведет его самостоятельно.

6. Кванторы в содержательной математике и математической логике

> Теперь, если я приду к вам, братия, и стану говорить на незнакомых языках, то какую принесу вам пользу, когда не изъяснюсь вам или откровением, или познанием, или пророчеством, или учением?
>
> *Первое послание к Коринфянам*, 14:6

Нетрудно видеть, что слово λx имеет такие же «связывающие» свойства, что и слово $\forall x$ в языке формальной системы первого порядка или $\int f(x)dx$ в математическом анализе.

В связи с этим представляет интерес перечисление *кванторов*, встречающихся в языках содержательной математики и математической логики (по [35, с. 66; с. 158; 36, с. 46; с.145]):

1) *квантор образования множества*:

$$\{x|P(x)\},$$

где $P(x)$ — предикат. Этот квантор конструирует множество всех x, обладающих свойством $P(x)$;

2) операции *произведения*, *суммы*, *вычисление предела*, *вычисление интеграла*, *объединение множеств*:

$$\prod_{x=0}^{n}(x^2+y),\ \sum_{x=0}^{n}(x^2+y),\ \lim_{x\to 1}(x^2+y),\ \int_{0}^{y}(x^2+z)dx,\ \bigcup_{k=1}^{\infty}A_k;$$

3) *квантор минимизации*

$$(\mu x \in X)\ P(x),$$

обозначающий наименьшее число в множестве натуральных чисел X, обладающее данным свойством $P(x)$. Этот квантор появляется в силу *принципа наименьшего числа* (поскольку это наименьшее число одно и только одно в каждом непустом множестве натуральных чисел);

4) *квантор «тот самый x»*:

$$\iota x P(x),$$

где $P(x)$ — формула языка первого порядка, которая должна принимать значение «*истина*» для единственного x, которое и является значением терма $\iota x P(x)$;

5) *квантор функциональности*:

$$\lambda x.\ t(x), \text{ либо } x \mapsto t(x),$$

где $t(x)$ — терм. Этот квантор конструирует по терму $t(x)$ функцию, вычисляющую значения этого терма. Квантор часто встречается в информатике; в программировании ему соответствует описание функции (t — тело функции, x — её формальный параметр).

Замечание (*важное*)

В λ-*исчислении* функция может быть определена без предварительного указания её области определения, поэтому функции часто представляются *бестиповыми* λ-*термами*.

В математике функция не является полностью определённой, пока не указаны её область определения и множество значений.

Поэтому структура λ-терма, представляющего функцию, должна содержать информацию о её области определения и множестве значений, а типизированные λ-термы соответствуют стандартному понятию «*функция*» в математике.

Биографические сведения

Чёрч Алонзо (14.06.1903—11.08.1995) — американский логик и математик, член Американской академии искусств.

Родился в Вашингтоне. Окончил Принстонский университет (1927).

В 1927—1928 был стипендиатом по математике в Гарвардском университете, в 1928—1929 слушал лекции в Геттингенском, в 1929 — в Амстердамском университетах. С 1929 работал в Принстонском университете (с 1939 — профессор математики, с 1961 — профессор философии), с 1967 — профессор Калифорнийского университета в Лос-Анжелесе.

Основные исследования относятся к математической логике. Разработал (1932—1933) систему аксиом для общей теории логики и дал первую формулировку теории, названо позже исчислением λ-преобразования. Исследовал (1936) класс вычислимых функций, выделив класс λ-определимых функций. В том же году сформулировал *тезис Чёрча*, в котором утверждается, что для логики предикатов нельзя дать никакого общего метода решения вопроса относительно общезначимости некоторой формулы. Ряд работ по математической и символической логике.

Примеры решения некоторых типов упражнений

Только некоторые наши сообщения только для некоторых людей имеют только некоторый смысл.

Английская мудрость

Пример 1.
Укажите множества связанных и свободных переменных в λ-терме

$$(\lambda x.\, xy)\ (\lambda y.\, y).$$

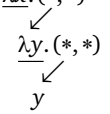
Решение. Воспользуемся индуктивными определениями функций FV() и CV():

$$\mathrm{FV}((\lambda x.xy)(\lambda y.y)) \overset{\text{def}}{=} \mathrm{FV}(\lambda x.xy) \cup \mathrm{FV}(\lambda y.y) \overset{\text{def}}{=}$$
$$\overset{\text{def}}{=} (\mathrm{FV}(xy) \setminus \{x\}) \cup (\mathrm{FV}(y) \setminus \{y\}) \overset{\text{def}}{=} ((\mathrm{FV}(x) \cup \mathrm{FV}(y)) \setminus \{x\}) \cup (\{y\} \setminus \{y\}) \overset{\text{def}}{=}$$
$$\overset{\text{def}}{=} (\{x\} \cup \{y\}) \setminus \{x\}) \cup \varnothing \overset{\text{def}}{=} \{y\};$$

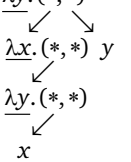
$$\mathrm{CV}((\lambda x.xy)(\lambda y.y)) \overset{\text{def}}{=} \mathrm{CV}(\lambda x.xy) \cup \mathrm{CV}(\lambda y.y) \overset{\text{def}}{=}$$
$$\overset{\text{def}}{=} \{x\} \cup \mathrm{CV}(xy) \cup \{y\} \cup \mathrm{CV}(y) \overset{\text{def}}{=} \{x, y\} \cup \mathrm{CV}(x) \cup \mathrm{CV}(y) \overset{\text{def}}{=} \{x, y\}.$$

Пример 2 (*интерпретации λ-термов с помощью бинарных деревьев*)

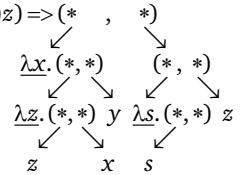
1. $\lambda x.\lambda y.\, y \Rightarrow \underline{\lambda x}.(*,*)$



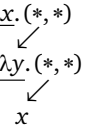
2. $\lambda y.(\lambda x.(\lambda y.\, x))y \Rightarrow \underline{\lambda y}.(*,*)$



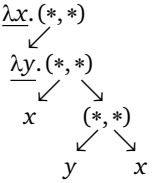
3. $(\lambda x.(\lambda z.\, zx)y)((\lambda s.\, s)z) \Rightarrow (* \quad , \quad *)$



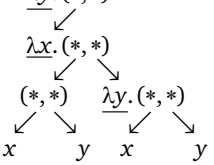
4. $\lambda x.\lambda y.\, x \Rightarrow \underline{\lambda x}.(*,*)$



5. $\lambda x.\lambda y.(x(yx)) \Rightarrow \underline{\lambda x}.(*,*)$



6. $\lambda y.\lambda x.\, xy(\lambda y.\, xy) \Rightarrow \underline{\lambda y}.(*,*)$



Демонстрационные примеры

Пример 0.

```
-- Демонстрация модуля, содержащего описание типа данных
-- "Лямбда-терм" и связывание его с классом Show для
-- визуализации лямбда-термов в виде строк.
--
-- [29, с. 279–280]
-- ****************
```

```
module L
   (Lambda(..), povto, povto1)
where
-----------------------------------------------
-- Описание типа для представления лямбда-терма
-----------------------------------------------
-- Лямбда-терм:
-- (1) предметная переменная, имя которой задаётся
-- в виде строки (конструктор Var);
-- (2) аппликация лямбда-термов, т. е. приложение
-- лямбда-термов друг к другу (конструктор App);
-- (3) лямбда-абстракция, в которой указывается связь
-- предметной переменной (имя которой задаётся в виде
-- строки) с лямбда-термом (конструктор Lam)
-------------------------------------------------------
data Lambda =   Var String        --  Предметная переменная
              | App Lambda Lambda --  Аппликация
              | Lam String Lambda --  Абстракция
  deriving Eq
-------------------------------------------------------
-- Тип Lambda является экземпляром класса Eq, т.е. значения
-- типа Lambda могут сравниваться друг с другом с помощью
-- операций сравнения (==) и (/=)
-- *****************************************************
-- Реализация типа Lambda для класса Show для представления
-- лямбда-термов в виде строк, т. е. представление типа Lambda
-- в качестве экземпляра класса Show.
--  Представление лямбда-терма будет следующим:
--  (1) символ "лямбда" — знак "\";
--  (2) предметная переменная — её имя;
--  (3) аппликация — два лямбда-терма, заключённые в скобки
--  (или без скобок, если лямбда-термы простые);
--  (4) абстракции, участвующие в аппликации, —
--  лямбда-абстракции,
--  заключённые в круглые скобки;
--  (5) абстракция — строка вида "\x. TERM",
--  где x — имя предметной переменной
-----------------------------------
instance Show Lambda where
   show (Var x)   = x
   show (App x y) = case y of
                      App _ _ -> showLam x ++ "("++ show y ++")"
                      _       -> showLam x ++ showLam y
         where showLam l@(Lam _ _) = "("++ show l ++ ")"
               showLam x           = show x
   show (Lam x e) = "\\" ++ x ++ "." ++ show e
--*****************************************
-- Функция, удаляющая из непустого одноуровневого
-- списка повторные вхождения элементов.
--
-- Автор: Н. Степушина (2009, июнь)
-----------------------------------
povto []  = []
povto [x] = [x]
```

```
povto (x: y: lst) | x==y = povto1 [x] []  lst
                  | True = povto1 [x] [y] lst
-----------------------------------------
povto1 lst1        []      []    = reverse lst1
povto1 lst1     (y: lst2)  []    = povto1 ([y] ++ lst1) [] lst2
povto1 (x: lst1)  lst2  (y: lst3)
                  | x==y = povto1 (x: lst1) lst2          lst3
                  | True = povto1 (x: lst1) (lst2 ++ [y]) lst3
--************************************************************
-- Неудачные тестовые примеры:
test1 = povto [1, 2, 3, 4, 1, 2, 3, 4]
test2 = povto [1, 2, 3, 4]
```

Пример 1.

```
-- Демонстрация функции, распознающей, является ли заданное
-- слово лямбда-термом,  с помощью  следующего индуктивного
-- определения лямбда-терма:
-- (1) предметная  переменная,  имя  которой  задаётся  в
-- виде строки (конструктор Var), является лямбда-термом;
-- (2) если  e1  и  e2  –  лямбда-термы,  то (e1e2),
-- т. е. аппликация лямбда-термов (конструктор App),
-- является лямбда-термом;
-- (3) если y – предметная переменная, e – лямбда-терм,
-- то  лямбда-абстракция,  в которой  указывается связь
-- предметной  переменной  (имя которой задаётся в виде
-- строки)  с лямбда-термом (конструктор Lam), является
-- лямбда-термом
-- ****************
module P01_00 where
import L
import Char
-- ********************
lambda:: Lambda -> Bool
lambda (Var y)     = isAlpha (head y)
lambda (App e1 e2) = lambda e1 && lambda e2
lambda (Lam y e)   = isAlpha (head y) && lambda e
-- ******************************************
-- Функции, определяющие CL-термы:
-----------------------------------
lam01 = Lam "x" (App (Var "x")
                     (App (App (Var "y") (Var "x"))
                          (Var "z")))
lam02 = Lam "x" (Lam "y" (Var "x"))
lam03 = Lam "x" (Lam "y" (App (Var "y") (Var "x")))
lam04 = App (App (Var "x1") (Var "x2")) (Var "x3")
lam05 = App (Var "x1") (App (Var "x2") (Var "x3"))
-- ******************************************
-- Неудачные тестовые примеры:
-----------------------------------------------------
test0 = lambda lam0l && lambda lam02 && lambda lam03
                     && lambda lam04 && lambda lam05
```

Пример 2.

```
-- Демонстрация функции, возвращающей множество
-- подтермов заданного лямбда-терма.
```

```
--
-- Автор: Н. Степушина (2009, июнь)
-- ******************************
module P01_01 where
import L
------------------------
sub:: Lambda -> [Lambda]
sub (Var y)     = povto [(Var y)]
sub (App e1 e2) = povto ([(App e1 e2)] ++ sub e2 ++ sub e1)
sub (Lam y e)   = povto ([(Lam y e)] ++ sub e)
-- ****************************************
-- Функции для представления лямбда-термов:
------------------------------------------
lam11 = Lam "x" (App (App (Var "x") (Var "y"))
(Lam "z" (Var "y")))
lam12 = Lam "x" (App (Var "y") (Var "x"))
-- ************************************
-- Неудачные тестовые примеры:
-----------------------------
test1 =  show (sub lam11)
         == "[\\x. xy (\\z. y), xy (\\z. y), \\z. y, y, xy,
              x]"
      && show (sub lam12)
         == "[\\x. yx, yx, x, y]"
```

Пример 3.

```
-- Демонстрация предиката, устанавливающего, является ли
-- предметная переменная x свободной в лямбда-терме l
-- ************************************************
module P01_02 where
import L
------------------------------
free:: [Char] -> Lambda -> Bool
free x (Var y)     = x==y
free x (App e1 e2) = free x e1 | | free x e2
free x (Lam ye)    = x/=y && free x e
-- ************************************
-- Функции, представляющие лямбда-термы:
----------------------------------------
lam21 = Lam "x" (Lam "y" (Lam "z"
                (App (Var "x") (App (Var "y") (Var "z")))))
lam22 = Lam "x" (Lam "y" (Lam "z"
                (App (App (Var "x") (Var "z")) (Var "y"))))
lam23 = Lam "x" (Lam "y"
                (App (App (Var "x") (Var "y")) (Var "y")))
Iam24 = App (Lam "x" (App (Var "x") (Var "y")))
                     (Lam "y" (Var "y"))
lam25 = Lam "x" (App (Var "b") (App (Var "f")
                (App (Var "a") (Var "x"))))
-- ************************************
-- Неудачные тестовые примеры:
------------------------------------------------
test2 =   not (free "x" lam21) && not (free "y" lam21)
```

```
       && not (free "z" lam21) && not (free "x" lam22)
       && not (free "y" lam22) && not (free "z" lam22)
       && not (free "x" lam23) && not (free "y" lam23)
       && not (free "x" lam24) &&      free "y" lam24
       && not (free "x" lam25) &&      free "b" lam25
       &&      free "f" lam25  &&      free "a" lam25
```

Пример 4.

```
-- Демонстрация предиката, устанавливающего, является ли
-- предметная переменная x связанной в лямбда-терме
--
-- Автор: Н. Степушина (2009, июнь)
-- ********************************
module P01_03 where
import L
-- ******************************
notfree:: [Char] -> Lambda -> Bool
notfree x (Var y)     = False
notfree x (App e1 e2) = notfree x e1 | | notfree x e2
notfree x (Lam y e)   = x==y | | notfree x e
-- ************************************
-- Функции для представления лямбда-термов:
-----------------------------------------
lam31 = App(Lam "v" (Var "x"))
           (Lam "y" (App (App (Var "y") (Var "x"))
                         (Lam "x" (App (App (Var "y")(Var "v"))
                                       (Var "x")))))
-------------------------------------------------
lam32 = Lam "x" (Lam "y" (App (Var "z")
                         (Lam "z" (App (Var "z")
                                       (Lam "x" (Var "y"))))))
-- ********************************************************
-- Неудачные тестовые примеры:
----------------------------------------------------------
test3 =   notfree "v" lam31 -- lam1 = (\v. x)(\y. yx(\x. yvx))
       && notfree "y" lam31
       && notfree "x" lam31
       && notfree "x" lam32 -- lam2 = \x. \y. z(\z. z(\x. y))
       && notfree "y" lam32
       && notfree "z" lam32
```

Пример 5.

```
-- Демонстрация функции, возвращающей множество
--  свободных переменных заданного терма
--
-- Автор: Н. Степушина (2009, июнь)
-- ********************************
module P01_04 where
import L
----------------------------
mn_free:: Lambda -> [[Char]]
mn_free (Var y)     = [y]
mn_free (Lam ye)    = povto (filter (y/=) (mn_free e))
```

```
mn_free (App e1 e2) = povto (mn_free e1 + + mn_free e2)
-- ************************************************
— Функции для представления лямбда-термов:
-----------------------------------------------
lam41 = App (Lam "x" (App (Var "x") (Var "y")))
            (Lam "y" (Var "y"))
-------------------------------
lam42 = Lam "x" (Lam "y"
                (App (Var "z")
                     (Lam "z" (App (Var "z")
                              (Lam "x" (Var "y"))))))
-----------------------------------------------------------------
lam43 = App (Lam "y" (App (App (Var "x") (Var "y")) (Var "z")))
            (Lam "z" (Lam "x" (App (Var "x") (Var "y"))))
-----------------------------------------------------------
lam44 = App (Lam "x" (Lam "y" (App (Var "x")
                                   (Lam "z" (App (Var "y")
                                                 (Var "z"))))))
            (Lam "x" (Lam "y" (Var "z")))
-- ***************************************
-- Неудачные тестовые примеры:
-----------------------------
test4 =   mn_free lam41 == ["y"]
       && mn_free lam42 == ["z"]
       && mn_free lam43 == ["x", "z", "y"]
       && mn_free lam44 == ["z"]
```

Пример 6.

```
-- Демонстрация функции, возвращающей множество
-- связанных переменных заданного терма
--
-- Автор: Н. Степушина (2009, июнь)
-- *********************************
module P01_05 where
import L
-------------------------------
mn_notfree:: Lambda - > [[Char]]
mn_notfree (Var y)     = []
mn_notfree (App e1 e2) = povto (mn_notfree e1 ++ mn_notfree e2)
mn_notfree (Lam ye)    = povto ([y] ++ mn_notfree e)
-- ************************************************
— Функции для представления лямбда-термов:
-----------------------------------------------
lam51 = App (Lam "x" (App (Var "x") (Var "y")))
            (Lam "y" (Var "y"))
-------------------------------------
lam52 = Lam "x" (Lam "y" (App (Var "z")
                              (Lam "z" (App (Var "z")
                                            (Lam "x" (Var "y"))))))
-------------------------------------------------------------
lam53 = App (Lam "y" (App (App (Var "x") (Var "y")) (Var "z")))
            (Lam "z" (Lam "x" (App (Var "x") (Var "y"))))
-----------------------------------------------------------
```

```
lam54 = App (Lam "x" (Lam "y" (App (Var "x")
                                   (Lam "z" (App (Var "y")
                                                 (Var "z"))))))
            (Lam "x" (Lam"y" (Var "z")))
-- ***************************************
-- Неудачные тестовые примеры:
-------------------------------------
test5 =   mn_notfree lam51 == ["x", "y"]
       && mn_notfree lam52 == ["x", "y", "z"]
       && mn_notfree lam53 == ["y", "z", "x"]
       && mn_notfree lam54 == ["x", "y", "z"]
```

Пример 7.

```
-- Демонстрация функции, реализующей операцию "подстановка
-- лямбда-терма n вместо всех свободных вхождений предмет-
-- ной переменной x в лямбда-терм"
--
-- Автор: Н. Степушина (2009, июнь)
-- *******************************
module P01_06 where
import L
import P01_02
import P01_04
-- ***********************
frees:: [String] -> [Char]
frees lst | notElem ("a") lst = "a"
          | notElem ("b") lst = "b"
          | notElem ("c") lst = "c"
          | notElem ("d") lst = "d"
          | notElem ("e") lst = "e"
          | notElem ("f") lst = "f"
          | notElem ("g") lst = "g"
          | notElem ("h") lst = "h"
          | notElem ("i") lst = "i"
          | notElem ("j") lst = "J"
          | notElem ("k") lst = "k"
          | notElem ("l") lst = "l"
          | notElem ("m") lst = "m"
          | notElem ("n") lst = "n"
          | notElem ("o") lst = "o"
          | notElem ("p") lst = "p"
          | notElem ("q") lst = "q"
          | notElem ("r") lst = "r"
          | notElem ("s") lst = "s"
          | notElem ("t") lst = "t"
          | notElem ("u") lst = "u"
          | notElem ("v") lst = "v"
          | notElem ("w") lst = "w"
          | notElem ("x") lst = "x"
          | notElem ("y") lst = "y"
          | notElem ("z") lst = "z"
          | True              = "0"
-- ****************************
```

```
podctanovka:: Lambda -> [Char] -> Lambda -> Lambda
podctanovka (Var y) x n | free x (Var y) == False = Var y
                        | True                    = n
podctanovka (App e1 e2) x n = App (podctanovka e1 x n)
                                  (podctanovka e2 x n)
podctanovka (Lam y e) x n
   | x==y                         = Lam y e
   | free x e == False            = Lam y e
   | free x e && free y n==False = Lam y (podctanovka e x n)
   | free x e && free y n &&
     (frees (mn_free e ++ mn_free n)/="0")
                                  =
        (Lam (frees (mn_free e ++ mn_free n))
                 (podctanovka (podctanovka e y
                                  (Var (frees (mn_free e ++
                                               mn_free n)))) x n))
   |True = error "В терме использованы все символы алфавита"
-- *******************************************************
-- Неудачные тестовые примеры:
-------------------------------------------------------
test6 =  podctanovka (Lam "x" (Var "x")) "y" (Var "z")
              == Lam "x" (Var "x")
      ---------------------------
      && podctanovka
            (Lam "y" (App (Var "z") (App (Var "x") (Var "y"))))
            "x"
            (Lam "x" (Var "x"))
               == Lam "y" (App (Var "z")
                               (App (Lam "x" (Var "x"))
                                    (Var "y")))
      ----------------------------------------
      && podctanovka
            (Lam "x" (App (Var "y") (Var "x")))
            "x"
            (App (Var "v") (Var "x"))
              == Lam "x" (App (Var "y") (Var "x"))
      ----------------------------------------
      && podctanovka
            (Lam "y" (App (Var "y") (Var "x")))
            "x"
            (App (Var "v") (Var "x"))
              == (Lam "y" (App (Var "y")
                               (App (Var "v") (Var "x"))))
```

Упражнения для самостоятельного решения

Решать задачи — долг математиков.
П. Рибенбойм

1. Язык λ-термов как язык в алфавите

1. Докажите, что следующие слова являются λ-термами:
а) (x_1x_1);
б) $(\lambda x_2.\, x_2)$;

в) $(\lambda x_1.\ (x_1 x_1))$;
г) $(\lambda x_1.\ (x_1 (x_2 x_2)))$;
д) $(x_1\ (\lambda x_1.\ (\lambda x_2.\ x_3)))$;
е) $((x_2 x_1)(\lambda x_1.\ (\lambda x_1.\ x_1)))$;
ж) $((\lambda x_2.\ x_2)(\ \lambda x_3.(x_3 x_2)))$;
з) $(\lambda x_1.\ (\lambda x_2.\ ((x_2 x_1)(\lambda x_3.\ x_3))))$.

Решение.

а) $\dfrac{x_1\ x_1}{(x_1 x_1)}$ б) $\dfrac{x_2\ \ x_2}{(\lambda x_2.x_2)}$ в) $\dfrac{\dfrac{x_1\ x_1}{x_1\ (x_1 x_1)}}{(\lambda x_1.(x_1 x_1))}$ г) $\dfrac{\dfrac{x_2\ x_2}{x_1\ \ (x_2 x_2)}}{\dfrac{x_1\ \ (x_1(x_2 x_2))}{(\lambda x_1.(x_1(x_2 x_2)))}}$

д) $\dfrac{\dfrac{x_2\ \ x_3}{x_1\ (\lambda x_2.x_3)}}{(x_1(\lambda x_1.(\lambda x_2.x_3)))}$ е) $\dfrac{\dfrac{x_2\ \ x_1}{(x_2 x_1)}\ \ \dfrac{\dfrac{x_1\ \ x_1}{x_1\ (\lambda x_1.x_1)}}{(\lambda x_1.(\lambda x_1.x_1))}}{((x_2 x_1)(\lambda x_1.(\lambda x_1.x_1)))}$

ж) $\dfrac{\dfrac{x_2\ x_2}{(\lambda x_2.x_2)}\ \ \dfrac{\dfrac{x_3\ x_2}{x_3\ (x_3 x_2)}}{(\lambda x_3.(x_3 x_2))}}{((\lambda x_2.x_2)(\lambda x_3.(x_3 x_2)))}$ з) $\dfrac{\dfrac{\dfrac{x_2\ x_1}{(x_2 x_1)}\ \ \dfrac{x_3\ \ x_3}{(\lambda x_3.x_3)}}{x_2\ ((x_2 x_1)(\lambda x_3.x_3))}}{x_1\ (\lambda x_2.((x_2 x_1)(\lambda x_3.x_3)))}$
$\overline{(\lambda x_1.(\lambda x_2.((x_2 x_1)(\lambda x_3.x_3))))}$

2. Докажите, что следующие слова не являются λ-термами:
а) $((x_2)(x_1 x_3))$;
б) $(\lambda x_2.\lambda x_2.(x_2 x_2))$;
в) $((\lambda x_3.x_3 x_2)(\lambda x_2.x_2))$;
г) $((x_3 x_2 x_3)(\lambda x_1.(\lambda x_1.x_1)))$;
д) $((x_1 x_2).(\lambda x_1.x_1))$;
е) $(\lambda x_1.(\lambda x_2((x_2 x_1)(\lambda x_3.x_3))))$.

Решение.

а) $\dfrac{\overset{\text{Не является } \lambda\text{-термом}}{\overset{\downarrow}{(x_2)}}\ (x_1 x_3)}{((x_2)(x_1 x_3))}$ б) $\dfrac{x_2\ \ \overset{\text{Отсутствуют внешние скобки}}{\overset{\downarrow}{\lambda x_2.(x_2 x_2)}}}{(\lambda x_2.\ \lambda x_2.(x_2 x_2))}$ в) $\dfrac{x_3\ \ \overset{\text{Отсутствуют внешние скобки}}{\overset{\downarrow}{x_3 x_2}}}{(\lambda x_3.\ x_3 x_2)\ \ (\lambda x_2.\ x_2)}$
$\overline{((\lambda x_3.\ x_3 x_2)(\lambda x_2.\ x_2))}$

г) $\dfrac{\overset{\text{Отсутствуют внешние скобки}}{\overset{\downarrow}{x_3 x_2}}\ \ \ x_3}{(x_3 x_2 x_3)\ \ \ (\lambda x_1.(\lambda x_1.\ x_1))}$
$\overline{((x_3 x_2 x_3)(\lambda x_1.(\lambda x_1.\ x_1)))}$ д) $\overset{\text{Лишняя точка}}{\overset{\downarrow}{((x_1 x_2).(\lambda x_1.\ x_1))}}$

е) $\dfrac{x_1\ \ \overset{\text{Отсутствует точка}}{\overset{\downarrow}{(\lambda x_2((x_2 x_1)(\lambda x_3.\ x_3)))}}}{(\lambda x_1.(\lambda x_2((x_2 x_1)(\lambda x_3.\ x_3))))}$

3. Опустите скобки в следующих λ-термах, пользуясь *соглашениями об обозначениях*:

а) $((\lambda x_1.(\lambda x_2.(x_2 x_1)))(\lambda x_3.\, x_1))$;
б) $((x_1 x_2)(\lambda x_3.(x_3(x_4 x_2))))$;
в) $(\lambda x_3.((\lambda x_1.((\lambda x_2.x_2)x_3))x_6))$;
г) $((\lambda x_5.(x_1 x_1))(\lambda x_3.(\lambda x_4.x_4)))$;
д) $(((x_2 x_2)(x_3(\lambda x_1.x_1)))(x_4 x_3))$;
е) $((((x_2(\lambda x_3.\, x_3))(\lambda x_5.(\lambda x_6.x_5)))x_1)x_1)$.

Решение. Скобки, которые можно опустить, выделены жирным шрифтом.

а) $\mathbf{((}\lambda x_1.\mathbf{(}\lambda x_2.\mathbf{(}x_2 x_1\mathbf{)))(}\lambda x_3.x_1\mathbf{))} \equiv (\lambda x_1 x_2.x_2 x_1)\lambda x_3.x_1$;
б) $\mathbf{((}x_1 x_2\mathbf{)(}\lambda x_3.\mathbf{(}x_3(x_4 x_2)\mathbf{))))} \equiv x_1 x_2(\lambda x_3.x_3(x_4 x_2))$;
в) $\mathbf{(}\lambda x_3.\mathbf{((}\lambda x_1.\mathbf{((}\lambda x_2.x_2\mathbf{)}x_3\mathbf{))}x_6\mathbf{))} \equiv \lambda x_3.(\lambda x_1.(\lambda x_2.x_2)x_3)x_6$;
г) $\mathbf{((}\lambda x_5.\mathbf{(}x_1 x_1\mathbf{))(}\lambda x_3.\mathbf{(}\lambda x_4.\, x_4\mathbf{)))} \equiv (\lambda x_5.x_1 x_1)\lambda x_3 x_4.x_4$;
д) $\mathbf{(((}x_2 x_2\mathbf{)(}x_3(\lambda x_1.x_1)\mathbf{)))(}x_4 x_3\mathbf{))} \equiv x_2 x_2(x_3(\lambda x_1.x_1))(x_4 x_3)$;
е) $\mathbf{(((}(x_2(\lambda x_3.x_3))\ (\lambda x_5.\mathbf{(}\lambda x_6.x_5\mathbf{)))}x_1\mathbf{)}x_1\mathbf{)} \equiv (x_2(\lambda x_3.x_3))(\lambda x_5\lambda x_6.x_5)x_1 x_1$.

4. Восстановите скобки в следующих λ-термах, пользуясь *соглашениями об обозначениях*:

а) $(\lambda x_1 x_2 x_3.x_3 x_3)x_2 x_4(x_5 x_5)$;
б) $(x_2 x_3 \lambda x_2.x_2)x_5(\lambda x_6.x_6 x_7)$;
в) $(\lambda x_3 x_4.(x_6 x_7)\lambda x_8.x_8)$;
г) $x_1 x_1(x_3 x_3)x_5(\lambda x_6.x_6)$;
д) $(\lambda x_2.x_3 x_2)\ (x_4 x_4)(\lambda x_5.x_6 x_5 x_4)$;
е) $x_1(x_2 \lambda x_3 x_5.x_5)x_6 x_8 x_9$.

Решение. Восстановленные скобки выделены жирным шрифтом.

а) $(\lambda x_1 x_2 x_3.\ x_3 x_3)x_2 x_4(x_5 x_5) \equiv \mathbf{((((}(\lambda x_1.\mathbf{(}\lambda x_2.\mathbf{(}\lambda x_3.\mathbf{(}x_3 x_3\mathbf{))))}x_2\mathbf{)}x_4\mathbf{)(}x_5 x_5\mathbf{))}$;
б) $(x_2 x_3 \lambda x_2.x_2)x_5(\lambda x_6.x_6 x_7) \equiv \mathbf{((((}x_2 x_3\mathbf{)(}\lambda x_2.x_2\mathbf{))}x_5\mathbf{)(}\lambda x_6.\mathbf{(}x_6 x_7\mathbf{)))}$;
в) $(\lambda x_3 x_4.(x_6 x_7)\lambda x_8.x_8) \equiv (\lambda x_3.\mathbf{(}\lambda x_4.\mathbf{((}x_6 x_7\mathbf{)(}\lambda x_8.x_8\mathbf{))))}$;
г) $x_1 x_1(x_3 x_3)x_5(\lambda x_6.x_6) \equiv \mathbf{((((}x_1 x_1\mathbf{)(}x_3 x_3\mathbf{))}x_5\mathbf{)(}\lambda x_6.x_6\mathbf{))}$;
д) $(\lambda x_2.x_3 x_2)(x_4 x_4)(\lambda x_5.x_6 x_5 x_4) \equiv \mathbf{(((}\lambda x_2.\mathbf{(}x_3 x_2\mathbf{))(}x_4 x_4\mathbf{))(}\lambda x_5.\mathbf{((}x_6 x_5\mathbf{)}x_4\mathbf{)))}$;
е) $x_1(x_2 \lambda x_3 x_5.x_5)x_6 x_8 x_9 \equiv \mathbf{(((}x_1(x_2\mathbf{(}\lambda x_3(\lambda x_5.x_5)\mathbf{)))}x_6\mathbf{)}x_8\mathbf{)}x_9\mathbf{)}$.

2. Свободные и связанные переменные

1. (По [9, с. 37])

Пусть $M \overset{\text{def}}{=} \lambda.\, xy\mathbf{(}\lambda z.\, y\mathbf{)}$. Установите верно ли, что $\mathbf{(}xy\mathbf{)} \in \mathrm{Sub}(M)$, $z \in \mathrm{Sub}(M)$ $\mathbf{(}y\mathbf{(}\lambda z.\, y\mathbf{))} \in \mathrm{Sub}(M)$?

Решение.

$$\mathrm{Sub}(M) \overset{\text{def}}{=} \mathrm{Sub}(\lambda x.xy(\lambda z.y)) \overset{\text{def}}{=} \{\lambda x.xy(\lambda z.y)\} \cup \mathrm{Sub}(xy(\lambda z.y)) \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \{\lambda x.xy(\lambda z.y)\} \cup \mathrm{Sub}(xy) \cup \mathrm{Sub}(\lambda z.y)) \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \{\lambda x.xy(\lambda z.y)\} \cup \{xy\} \cup \mathrm{Sub}(x) \cup \mathrm{Sub}(y) \cup \{\lambda z.y\} \cup \mathrm{Sub}(y) \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \{\lambda x.xy(\lambda z.y)\} \cup \{xy\} \cup \{x\} \cup \{y\} \cup \{\lambda z.y\} \cup \{y\} \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \{\lambda x.xy(\lambda z.y)\} \cup \{xy\} \cup \{x\} \cup \{y\} \cup \{\lambda z.y\} \Rightarrow$$
$$\Rightarrow (xy) \in \mathrm{Sub}(M), z \notin \mathrm{Sub}(M), (y(\lambda z.y)) \notin \mathrm{Sub}(M).$$

2. Укажите множества свободных и связанных переменных для следующих λ-термов:

а) $(\lambda x.xy)(\lambda y.y)$;
б) $\lambda x.\lambda y.z(\lambda z.z(\lambda x.y))$;
в) $(\lambda y.xyz)(\lambda z.\lambda x.xy)$;
г) $y(\lambda y.yx)(\lambda x.x)$;
д) $(\lambda x.\lambda y.xz(yz))(\lambda x.y(\lambda y.y))$;
е) $(\lambda x.\lambda y.x(\lambda z.yz))(\lambda x.\lambda y.z)$.

Решение.

а) $\mathrm{FV}((\lambda x.\,xy)(\lambda y.\,y)) \overset{\text{def}}{=} \mathrm{FV}((\lambda x.\,xy) \cup \mathrm{FV}(\lambda y.\,y) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} (\mathrm{FV}(xy) \setminus \{x\}) \cup (\mathrm{FV}(y) \setminus \{y\}) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} ((\mathrm{FV}(x) \cup \mathrm{FV}(y)) \setminus \{x\}) \cup (\{y\} \setminus \{y\}) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} (\{x\} \cup \{y\}) \setminus \{x\}) \cup \varnothing \overset{\text{def}}{=} \{y\}$;

$\mathrm{CV}((\lambda x.\,xy)(\lambda y.\,y)) \overset{\text{def}}{=} \mathrm{CV}((\lambda x.\,xy) \cup \mathrm{CV}(\lambda y.\,y) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{x\} \cup \mathrm{CV}(xy) \cup \{y\}\mathrm{CV}(y) \overset{\text{def}}{=} \{x, y\} \cup \mathrm{CV}(x) \cup \mathrm{CV}(y) \overset{\text{def}}{=} \{x, y\}$;

б) $\mathrm{FV}(\lambda x.\,\lambda y.\,z(\lambda z.\,z(\lambda x.\,y))) \overset{\text{def}}{=} \mathrm{FV}(\lambda y.\,z(\lambda z.\,z(\lambda x.\,y))) \setminus \{x\} \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \mathrm{FV}(z(\lambda z.\,z(\lambda x.\,y))) \setminus \{x\} \setminus \{y\} \overset{\text{def}}{=} \mathrm{FV}(z) \cup \mathrm{FV}(\lambda z.\,z(\lambda x.\,y)) \setminus \{x, y\} \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{z\} \cup \mathrm{FV}(\lambda z.\,z(\lambda x.\,y)) \setminus \{x, y\} \overset{\text{def}}{=} \{z\} \cup \mathrm{FV}(z(\lambda x.\,y)) \setminus \{\mathrm{z}\} \setminus \{x, y\} \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \mathrm{FV}(z) \cup \mathrm{FV}(\lambda x.\,y)) \setminus \{x, y\} \overset{\text{def}}{=} \{z\} \cup \mathrm{FV}(y) \setminus \{x\} \setminus \{x, y\} \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{z\} \cup \{y\} \setminus \{x\} \setminus \{x, y\} \overset{\text{def}}{=} \{z\}$;

$\mathrm{CV}(\lambda x.\,\lambda y.\,z(\lambda z.\,z(\lambda x.\,y))) \overset{\text{def}}{=} \{x\} \cup \mathrm{CV}(\lambda y.\,z(\lambda z.\,z(\lambda x.\,y))) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{x\} \cup \{y\} \cup \mathrm{CV}(z(\lambda z.\,z(\lambda x.\,y))) \overset{\text{def}}{=} \{x, y\} \cup \mathrm{CV}(z) \cup \mathrm{CV}(\lambda z.\,z(\lambda x.\,y)) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{x, y\} \cup \{z\} \cup \mathrm{CV}(z(\lambda x.\,y)) \overset{\text{def}}{=} \{x, y, z\} \cup \mathrm{CV}(z) \cup \mathrm{CV}(\lambda x.\,y) \overset{\text{def}}{=}$
$\overset{\text{def}}{=} \{x, y, z\} \cup \{x\} \cup \mathrm{CV}(y) \overset{\text{def}}{=} \{x, y, z\}$;

в) $\mathrm{FV}((\lambda y.\,xyz)(\lambda z.\,\lambda x.\,xy)) \overset{\text{def}}{=} \{x, y, z\}$; $\mathrm{CV}((\lambda y.\,xyz)(\lambda z.\,\lambda x.\,xy)) \overset{\text{def}}{=} \{x, y, z\}$;

г) $\mathrm{FV}(y(\lambda y.\,yx)(\lambda x.\,x)) \overset{\text{def}}{=} \{x, y\}$; $\mathrm{CV}(y(\lambda y.\,yx)(\lambda x.\,x)) \overset{\text{def}}{=} \{x, y\}$;

д) $\mathrm{FV}((\lambda x.\,\lambda y.\,xz(yz))(\lambda x.\,y(\lambda y.\,y))) \overset{\text{def}}{=} \{y, z\}$;

$\mathrm{CV}((\lambda x.\,\lambda y.\,xz(yz))(\lambda x.\,y(\lambda y.\,y))) \overset{\text{def}}{=} \{x, y\}$;

е) $\mathrm{FV}((\lambda x.\,\lambda y.\,x(\lambda z.\,yz))(\lambda x.\,\lambda y.\,z)) \overset{\text{def}}{=} \{z\}$;

$\mathrm{CV}((\lambda x.\,\lambda y.\,x(\lambda z.\,yz))(\lambda x.\,\lambda y.\,z)) \overset{\text{def}}{=} \{x, y, z\}$.

3. Синтаксическая однозначность λ-термов

1$_*$**.** Докажите, что всякий λ-терм представим в одном и только одном из следующих видов:

1) x_i, где x_i ($i \in \mathbb{N}$) — однозначно определенная предметная переменная;

2) **(***MN***)**, где M и N — однозначно определённые λ-термы;

3) **(**$\lambda x.\ M$**)**, где x — однозначно определённая предметная переменная, M — однозначно определённый λ-терм.

2$_*$**.** Докажите, что каждое вхождение в λ-терм M предметной переменной и открывающей скобки является началом вхождения в M однозначно определённого λ-терма.

3$_*$**.** Пусть M и N — λ-термы. Докажите, что если один из них является началом другого, то $M = N$.

4. Операция «подстановка» в λ-исчислении

1$_*$**.** Выполните операцию «подстановка»:

а) $(\lambda y.\,yx)^x_{zx}$;

б) $(\lambda y.\,yx)^x_{yx}$;

в) $((\lambda z.\,x)(xy))^x_z$;

г) $(\lambda y.\,z(xy))^x_{\lambda x.x}$.

Решение.

а) $(\lambda y.\,yx)^x_{(zx)} \overset{\text{(p5) def}}{=} \lambda y.\,(yx)^x_{(zx)} \overset{\text{(p3,p2′,p1) def}}{=} \lambda y.\,y(zx)$;

б) $(\lambda y.\,yx)^x_{(yx)} \overset{\text{(p6) def}}{=} \lambda z.\,((yx)^y_z)^x_{(yx)} \overset{\text{(p3,p2′,p1) def}}{=} \lambda z.\,(zx)^x_{(yx)} \overset{\text{(p3,p2′,p1) def}}{=} \lambda z.\,z(yx)$;

в) $((\lambda z.\,x)(xy))^x_z \overset{\text{(p3) def}}{=} ((\lambda z.\,x)^x_z)((xy)^x_z) \overset{\text{(p6,p3) def}}{=} (\lambda w.\,((x)^z_w)^x_z)((x)^x_z(y)^x_z) \overset{\text{(p1) def}}{=}$

$\overset{\text{(p1) def}}{=} (\lambda w.\,((x)^z_w)^x_z)(z(y)^x_z) \overset{\text{(p2′) def}}{=} (\lambda w.\,(x)^x_z)(zy) \overset{\text{(p1) def}}{=} (\lambda w.\,z)(zy)$;

г) $(\lambda y.\,z(xy))^x_{(\lambda x.x)} \overset{\text{(p5) def}}{=} \lambda y.\,(z(xy))^x_{(\lambda x.x)} \overset{\text{(p3) def}}{=} \lambda y.\,(z)^x_{(\lambda x.x)}((xy)^x_{(\lambda x.x)}) \overset{\text{(p2′,p3) def}}{=}$

$\overset{\text{def}}{=} \lambda y.\,z((x)^x_{(\lambda x.x)}(y)^x_{(\lambda x.x)}) \overset{\text{(p1,p2′) def}}{=} \lambda y.\,z((\lambda x.\,x)y)$.

2$_*$**.** Выполните операцию «*подстановка*»:

а) $(\lambda x.\,\lambda y.\,\lambda z.\,x)^x_{\lambda x.zx}$;

б) $(\lambda y.\,\lambda z.\,yx)^x_{\lambda y.x}$;

в) $(\lambda z.\,\lambda x.\,x)^x_y$;

г) $(\lambda z.\,\lambda y.\,xyz)^x_{\lambda y.xz}$.

Решение.

а) $(\lambda x.\,\lambda y.\,\lambda z.\,x)^{x}_{(\lambda x.zx)} \overset{\text{(p4) def}}{=} \lambda x.\,\lambda y.\,\lambda z.\,x$;

б) $(\lambda yz.\,yx)^{x}_{(\lambda y.x)} \overset{\text{(p5) def}}{=} \lambda y.\,(\lambda z.\,yx)^{x}_{(\lambda y.x)} \overset{\text{(p5) def}}{=} \lambda yz.\,(yx)^{x}_{(\lambda y.x)} \overset{\text{(p3) def}}{=}$
$\overset{\text{(p3) def}}{=} \lambda yz.\,(y)^{x}_{(\lambda y.x)}(x)^{x}_{(\lambda y.x)} \overset{\text{(p1,p2′) def}}{=} \lambda y.\,\lambda z.\,y(\lambda y.\,x)$;

в) $(\lambda z.\,\lambda x.\,x)^{x}_{y} \overset{\text{(p2) def}}{=} \lambda z.\,\lambda x.\,x$;

г) $(\lambda zy.\,xyz)^{x}_{(\lambda y.xz)} \overset{\text{(p6) def}}{=} \lambda m.\,((\lambda y.\,xyz)^{z}_{m})^{x}_{(\lambda y.xz)} \overset{\text{(p5) def}}{=} \lambda m.\,(\lambda y.\,(xyz)^{z}_{m})^{x}_{(\lambda y.xz)} \overset{\text{(p3) def}}{=}$
$\overset{\text{def}}{=} \lambda m.\,(\lambda y.\,(x)^{z}_{m}(y)^{z}_{m}(z)^{z}_{m})^{x}_{(\lambda y.xz)} \overset{\text{(p1,p2′) def}}{=} \lambda m.\,(\lambda y.\,xym)^{x}_{(\lambda y.xz)} \overset{\text{(p5) def}}{=}$
$\overset{\text{(p5) def}}{=} \lambda my.\,(xym)^{x}_{(\lambda y.xz)} \overset{\text{(p3,p1,p2) def}}{=} \lambda m.\,\lambda y.\,((\lambda y.\,xz)ym)$.

3. Докажите, что приведённое ниже правило является «лишним» в определении операции «*подстановка*»:

(p′$_5$) $(\lambda y.\,M)^{x}_{N} \overset{\text{def}}{=} \lambda y.\,(M)^{x}_{N}$, если $y \notin \mathrm{FV}(N)$ или $x \notin \mathrm{FV}(M)$.

Решение (*для знатоков бинарного отношения* «$\equiv_{\alpha}$»).

$$(\lambda y.\,M)^{x}_{N} \overset{\text{def}}{=} \{x \notin \mathrm{FV}(M)\} \overset{\text{(p2) def}}{=} \lambda y.\,M;$$

$$(\lambda y.\,M)^{x}_{N} \overset{\text{def}}{=} \{x \in \mathrm{FV}(M),\, y \notin \mathrm{FV}(N)\} \overset{\text{(p5) def}}{=} \lambda y.\,(M)^{x}_{N}.$$

5. Интерпретация λ-термов в математическом анализе

Вопрос. Что это — большое, зелёное, живёт на глубине трёх метров под землей и ест камни?
Ответ. Большой Зелёный Камнеед.

Из фольклора МФТИ

1$_*$. [36, с. 383, 14.1.1]
Пусть $z = f(x, y)$ — функция, вычисляющая расстояние между двумя действительными числами. Что такое $\lambda x.f(0, x)$?

2$_*$. (По [36, с. 384, 14.1.6])
Приведите возможный смысл утверждения $\lambda x.x^2+1 > \lambda x.1/(2+x^2)$?

3. (По [36, с. 384, 14.1.3])
Что такое $\lambda x.\lambda y.f(y, \pi/2 - x)$, если $f(x, y) = e^x \sin(y)$?
Ответ: $\lambda x.\lambda y.e^y \cos(x)$.

4$_*$**.** (По [36, с. 384, 14.1.4])
Пусть $\boldsymbol{D}$ — оператор дифференцирования, т. е.
$\boldsymbol{D}(\lambda x.f(x))(x_0) = f'(x_0)$.
Вычислите:
1) $\boldsymbol{D}(\lambda x.e^x)(2)$; 2) $(\lambda x.(\boldsymbol{D}(\lambda y.(y^3 x + y))(1)))(3)$.

Решение.
1) $\lambda(\lambda x.e^x)(2) \overset{\text{def}}{=} e^2$;
2) $\lambda x.(\boldsymbol{D}(\lambda y.(y^3 x + y))(1))(3) \overset{\text{def}}{=} (\lambda x.((3y^2 x + 1)(1)))(3) \to_\beta$
$\to_\beta (\lambda x.(3x + 1))(3) \to_\beta 10$.

5. (По [36, с. 384, 14.1.2])
Запишите на «λ-языке» определенный интеграл

$$\int_a^b f(x)dx.$$

Решение. $\lambda f.\lambda a.\lambda b.Sfab$.

6. Компьютерное моделирование языка λ-исчисления как языка в алфавите

1. Проведите тестирование программ из раздела «Демонстрационные примеры» по полученным результатам решения задач.

2. Составьте программу для построения древесного доказательства того, что данное слово является λ-термом.

3. Составьте программу для определения ранга λ-терма.

4. Составьте программу для определения того факта, являются ли два терма дизъюнктными.

5. Составьте программу, реализующую операцию «*подстановка λ-терма N вместо всех свободных вхождений предметной переменной x в λ-терм M*».

6. Составьте программу-транслятор, переводящий:

1) сокращенную запись λ-терма в запись λ-терма со всеми скобками и абстракциями по соглашению об обозначениях λ-термов;

2) λ-терм в сокращенную запись, опуская скобки и абстракции по соглашению об обозначениях λ-термов.

7. Составьте программу, моделирующую квантор функциональности, конструирующего по терму функцию, вычисляющую значения этого терма.

Упражнение 2
БЕСТИПОВОЕ λ-ИСЧИСЛЕНИЕ: ДОКАЗАТЕЛЬСТВО λ-ФОРМУЛ

Я проникся глубоким уважением к математике, которую в её тонкости по своему слабоумию до сих пор считал роскошью.

А. Эйнштейн

Обязательные результаты обучения:

- ***знать основные понятия***:

— *бинарное отношение, совместимое с операциями λ-исчисления*;

— *отношение равенства на множестве λ-термов* (*отношение конгруэнтности*), *отношение редукции на множестве λ-термов*;

— *α-конверсия*, *β-конверсия*, *η-конверсия*;

— *схемы аксиом бестипового λ-исчисления*;

— *правила вывода бестипового λ-исчисления*;

— *линейное доказательство λ-формулы*, *доказуемая λ-формула*, *отношение доказуемости*, *древесное доказательство λ-формулы*;

— *β-конвертируемые λ-термы*;

— *бестиповое λ-исчисление*;

— *правило экстенсиональности* (*в λ-исчислении*);

— *разновидности бестипового λ-исчисления*: *λω-исчисление*, *λη-исчисление*, *λ+ext-исчисление*;

- ***уметь***

— *конструировать доказательства* (*линейные и древесные*) *λ-формул в бестиповом λ-исчислении*;

— *конструировать доказательства утверждений о β-конвертируемости λ-термов в бестиповом λ-исчислении*;

— *конструировать доказательства утверждений в бестиповом λη- и λ+ext-исчислениях*;

- ***владеть***

— *основными понятиями*, *представленными выше*;

— *методами решения задач*, *представленных в Упражнении*.

Теоретические сведения

Бинарные отношения, связанные с λ-исчислением

Определение (по [9, с. 61—62]).
Бинарным отношением, совместимым с операциями λ-исчисления (или *монотонным бинарным отношением*), называется бинарное отношение $\Re$ на множестве Λ, если для всех значений M_1, M_2, $Z \in \Lambda$ выполнены условия (приведены два варианта обозначений):

1) $(M_1, M_2) \in \Re \Rightarrow (ZM_1, ZM_2) \in \Re$; $\quad \dfrac{(M_1, M_2) \in \Re}{(ZM_1, ZM_2) \in \Re}$;

2) $(M_1, M_2) \in \Re \Rightarrow (M_1Z, M_2Z) \in \Re$; $\quad \dfrac{(M_1, M_2) \in \Re}{(M_1Z, M_2Z) \in \Re}$;

3) $(M_1, M_2) \in \Re \Rightarrow (\lambda x.\, M_1, \lambda x.\, M_2) \in \Re$. $\quad \dfrac{(M_1, M_2) \in \Re}{(\lambda x.\, M_1, \lambda x.\, M_2) \in \Re}$.

Определение (по [9, с. 61—62]).
Отношением равенства (*отношением конгруэнтности*) *на* Λ называется бинарное отношение на множестве Λ, обладающее свойствами:
1) совместимость с операциями λ-исчисления;
2) рефлексивность;
3) симметричность;
4) транзитивность.

Другими словами, *отношение равенства на* Λ является бинарным отношением эквивалентности, совместимым с операциями λ-исчисления.

Отношение равенства на Λ используется для определения λ-исчисления как *исчисления с равенством*, изучаемого средствами *математической логики*.

Определение (по [9, с. 61—62]).
Отношением редукции на Λ называется бинарное отношение на множестве Λ, обладающее следующими свойствами:
1) совместимость с операциями λ-исчисления;
2) рефлексивность;
3) транзитивность.

Отношение редукции на Λ используется для определения λ-исчисления как *алгоритмической системы*, изучаемого средствами *математической логики*, *теории алгоритмов* и *теоретического программирования*.

Схемы аксиом и правила вывода бестипового λ-исчисления

Аналитическая машина сможет ткать алгебраические формулы, как станок Жаккара — цветы и листья.

Ада Лавлейс

Построим *отношение равенства* (*отношение конгруэнтности*) *на* Λ.

Для этого вначале определим λ-формулы, которым (в силу их важности) присвоены специальные названия.

Пусть $x, y \in G_1, M, N \in \Lambda$.

Определение (по [9, с. 43]).

1) $\lambda x.M = \lambda y.(M)_y^x,\ y \notin \mathrm{FV}(M) \cup \mathrm{CV}(M)$ (α-*конверсия*);

2) $(\lambda x.M)N = (M)_N^x$ (β-*конверсия*);

3) $\lambda x.(Mx) = M,\ x \notin \mathrm{FV}(M)$ (η-*конверсия*).

Замечание **[46]**

Конверсия (лат. *conversio* — «превращение, изменение») — это способ образования слова посредством изменения его грамматических характеристик; переход слова из одной части речи в другую.

Пусть $x \in G_1, M, N, L \in \Lambda, Z \in F_\lambda$.

Определение (по [8, с. 284]).

Схемами аксиом бестипового λ-*исчисления* называются λ-формулы:

(Ax$_1$) $M = M$ (*рефлексивность*);

(Ax$_2$) $(\lambda x.M)N = (M)_N^x$ (β-*конверсия*);

(Ax$_3$) $\lambda x.M = \lambda y.(M)_y^x,\ y \notin \mathrm{FV}(M) \cup \mathrm{CV}(M)$ (α-*конверсия*).

Иногда будем пользоваться схемой аксиом

(Ax$_4$) $\lambda x.(Mx) = M,\ x \notin \mathrm{FV}(M)$ (η-*конверсия*).

Определение (по [8, с. 284]).

Правилами вывода бестипового λ-*исчисления* называются *фигуры*:

1) $\dfrac{N = M}{M = N}$ (*симметричность*);

2) $\dfrac{M = N \quad N = L}{M = L}$ (*транзитивность*, *сечение*);

3) $\dfrac{M = N}{LM = LN}$ (*преобразование аргумента*);

4) $\dfrac{M = N}{ML = NL}$ (*преобразование функции*);

5) $\dfrac{M = N}{\lambda x.\, M = \lambda x.\, N}$ (*правило* ξ);

6) $\dfrac{Z \overset{\text{def}}{M} = N}{Z(M / N)}$ (*общелогический принцип замены равного на равное*).

Правила вывода разделены на три группы:

а) правила 1, 2 и аксиома (Ах1) описывают свойства предиката «*равенство*»;

б) правила 3, 4, 5 связаны со свойствами *отношения равенства на множестве* Λ;

в) фигура 6 не является правилом вывода, поэтому она названа *принципом.*

Бинарное отношение «доказуемость λ-формулы»

Что такое логика, как не искусство доказывать?

Ж. Пиаже

Определение (по [21, с. 99]).

Линейным доказательством λ-формулы α_n будем называть последовательность λ-формул

$$\alpha_1, \alpha_2, \ldots, \alpha_n,$$

в которой λ-формула α_i для любого $i = 1, 2, \ldots, n$:

а) либо является аксиомой λ-исчисления;

б) либо найдутся $j, k < i$, такие, что α_i получается из α_j и α_k по одному из правил вывода λ-исчисления.

Определение.

Доказуемой λ-формулой (или λ-*теоремой*) называется λ-формула α_n, т. е. последняя формула в линейном доказательстве.

Определение.

Отношением доказуемости назовем (и обозначим «$\vdash_\lambda$» или «$\vdash$») бинарное отношение между множествами $\varnothing$ и F_λ, которое определим так: