Министерство образования и науки
Российской Федерации

Тверской государственный
университет

С. М. Дудаков
Б. Н. Карлов

МАТЕМАТИЧЕСКОЕ ВВЕДЕНИЕ В ИНФОРМАТИКУ

Издание второе, исправленное и дополненное

Учебник по дисциплине «Теоретические основы информатики»

Тверь — 2017

УДК 519.681
ББК З97-018я73-1
Д 81

Тверской государственный университет
Факультет прикладной математики и кибернетики

Рецензенты первого издания:
профессор доктор физ.-мат. наук Чагров А. В.
профессор доктор техн. наук Сухомлин В. А.

Дудаков С. М. Карлов Б. Н. Математическое введение в информатику: Учебник. Тверь: Твер. гос. ун-т, 2017. — 320 с.

В пособии освещаются теоретические вопросы программирования: связь и эквивалентность различных языков программирования, доказательство корректности программ, вычислительная сложность алгоритмов.

Учебник адресован, прежде всего, студентам младших курсов, обучающихся по направлениям укрупненных групп 01.03.00 «Математика и механика», 02.03.00 «Компьютерные и информационные науки», 09.03.00 «Информатика и вычислительная техника».

Выражаем глубокую благодарность доктору физико-математических наук профессору кафедры общей и прикладной алгебры и геометрии Александу Васильевичу Чагрову за внимательное ознакомление с рукописью и ценные замечания и доктору физико-математических наук декану факультета прикладной математики и кибернетики профессору Александу Васильевичу Язенину за помощь в издании книги.

УДК 519.681
ББК З81я731-1

© Дудаков Сергей Михайлович, Карлов Борис Николаевич, 2017

Оглавление

Предисловие **5**
Ко второму изданию 5
К первому изданию 6

Глава 1. Введение **11**
§ 1.1. Основные понятия 11
§ 1.2. Исторические сведения 13
§ 1.3. Свойства алгоритмов и языков программирования . . 16
§ 1.4. Примеры алгоритмов 19

Глава 2. Некоторые математические сведения **23**
§ 2.1. Алгебра и теория множеств 23
§ 2.2. Графы 30
§ 2.3. Математическая логика 34

Глава 3. Структурированные программы **39**
§ 3.1. Синтаксис 39
§ 3.2. Семантика 52
§ 3.3. *Свойства структурированных программ 74
§ 3.4. *Простые программы 86
§ 3.5. *Подстановка 97

Глава 4. Программы с метками 106
§ 4.1. Синтаксис 106
§ 4.2. Семантика 108
§ 4.3. Построение программ с метками 117
§ 4.4. Построение структурированных программ 127
§ 4.5. Блок-схемы 134

Глава 5. Корректность структурированных программ . 147
§ 5.1. Исчисления 147
§ 5.2. *Исчисление Хоара 154
§ 5.3. Корректность циклов 164
§ 5.4. *Примеры формальных выводов 181
§ 5.5. *Исчисление предусловий 202
§ 5.6. **Существование слабейших предусловий 216
§ 5.7. *Невозможность автоматической верификации 221

Глава 6. Подпрограммы, функциональное программирование 226
§ 6.1. Подпрограммы 226
§ 6.2. Графы зависимости, списки и деревья вызовов 238
§ 6.3. *Функциональное программирование 249
§ 6.4. **Удаление подпрограмм 259
§ 6.5. *Корректность подпрограмм 272

Глава 7. Вычислительная сложность 285
§ 7.1. Хранение чисел 285
§ 7.2. *Вычисления 288
§ 7.3. Время и память вычисления 296

Предметный указатель 310

Список литературы 317

Предисловие

Ко второму изданию

По сравнению с первым изданием материал был значительно переработан.

В главе о структурированных программах используемый формальный язык описания алгоритмов стал более «С-образным», что во многих случаях упростило изложение. Несколько изменен способ определения результата алгоритма. Как показал опыт, используемый в первом издании способ введения формальной переменной для результата не всегда оказывался легким для восприятия обучающимися.

В главе о программах с метками был добавлен параграф о блок-схемах. Добавление тоже продиктовано опытом, так как во многих случаях школьный курс информатики ориентируется на блок-схемы, поэтому проведение параллелей между применяемыми формальными описаниями и блок-схемами должно упростить восприятие соответствующего материала.

В следующей главе по предложению коллег был добавлен параграф о невозможности эффективной верификации.

Глава о функциональном программировании подверглась самой большой переработке, в связи с изменением формального языка, о чем сказано выше. В ряде случаев это значительно упростило изложение. Практически заново написаны последние параграфы. За счет всех этих изменений некоторые доказательства удалось значительно сократить.

Во многих параграфах было увеличено количество разобранных примеров и задач для самостоятельного решения.

Исправлено значительное число опечаток, неточностей и технических недостатков.

К первому изданию

Книга написана, прежде всего, для студентов, обучающихся по направлению «Прикладная математика и информатика» и специальности «Прикладная математика». Поводом к написанию стало то, что автор не смог найти другого издания, где весь необходимый материал был бы изложен на достаточно высоком уровне.

В ходе написания было принято решение включить в это пособие не только материал для студентов первого курса, но и некоторые темы, которые могут быть изучены на старших курсах, а также в ходе факультативных или самостоятельных занятий. Это продиктовано желанием дать полное изложение соответствующих тем. Для удобства читателя мы указываем уровень сложности каждого параграфа. Параграфы, не имеющие никаких пометок, являются обязательными для изучения в полном объёме. Параграфы, помеченные одной звёздочкой, имеют более высокий уровень сложности, на младших курсах обязательным является изучение только основных понятий из изложенного в них материала. Параграфы, отмеченные двумя звёздочками, не содержат материала, который был бы необходимым для младших курсов, и рекомендуются для самостоятельного изучения, факультативных занятий или спецкурсов.

При написании автор ставил перед собой задачу не впасть в одну из двух крайностей. С одной стороны он не хотел бы, чтобы это было очередным пособием по теории алгоритмов. По этой дисциплине имеется уже много прекрасных изданий. Кроме того, курс «Информатика» предназначен, прежде всего, не для изучения теоретических основ, а для получения знаний, которые были бы полезны при практическом программировании.

С другой стороны было бы неправильно свести эту книгу к изучению какого-то конкретного языка программирования или к перечню некоторых алгоритмов. Такого рода издания тоже не редкость, а

соперничать в этом вопросе с Д. Э. Кнутом было бы просто бессмысленно.

В этой книге выбран промежуточный вариант. Мы предлагаем несколько вычислительных моделей, которые основаны на реальных языках программирования, и ограничиваемся изучением свойств этих моделей. Мы не вдаёмся в такие понятия как «разрешимость» или «рекурсивная перечислимость», свойственные теории алгоритмов. С другой стороны мы предельно абстрактизировали и упростили выбранные модели, в связи с чем имеем возможность аксиоматически определять, что такое программа и её семантика, а также доказывать их свойства. Из-за близости вводимых моделей к реальным языкам программирования эти свойства легко переносятся на практически используемые языки.

При изложении материала мы придерживаемся следующего порядка. В первой главе мы пытаемся дать краткие исторические сведения о том, как развивались представления об алгоритмах, а так же привести несколько примеров, которые проиллюстрировали бы понятие «алгоритм» на хорошо известных примерах.

Вторая глава является кратким справочным пособием по тем разделам математики, которые потребуются при изучении материала, но которые могут быть ещё не известны студентам первого курса. Поскольку изложение этих сведений не есть основная цель книги, то мы даём эти определения на «полуформальном» уровне, прибегая к некоторым интуитивным примерам.

Изложение основного материала начинается с третьей главы. Мы определяем некоторый простой язык, на котором в дальнейшем будем писать программы. В качестве базовых синтаксических конструкций мы берём стандартный набор для построения структурированных программ: присваивание, следование, ветвление и цикл. Такой набор, с одной стороны, является достаточным для того, чтобы построенный язык был универсальным и чтобы доказанные резуль-

таты можно было распространить на все алголоподобные языки, с другой стороны, он достаточно прост, и можно легко доказывать утверждения о построенных программах.

Мы определяем семантику программы как некоторое преобразование конечных функций, определённых на натуральных числах. Это даёт возможность излагать материал на уровне доступном как для тех, кто имел опыт программирования, так и для тех, у кого его нет. Остаток этой главы посвящён изучению свойств построенного языка и его семантики. Мы получаем некоторые хорошо известные в программировании факты как теоремы, доказанные исходя из определений алгоритма и его семантики

В четвёртой главе мы рассматриваем другую концепцию программирования, ориентированную на метки и операторы перехода. Мы приводим необходимые определения и формулируем простую теорему, которая используется в дальнейшем. Затем мы доказываем эквивалентность языка структурного программирования и языка программирования с метками, и в качестве следствия получаем все нужные свойства программ с метками, а также теорему о цикле для структурированных программ.

Пятая глава посвящена доказательству корректности структурированных программ методом Хоара. Вначале мы кратко описываем понятие исчисления и вывода, затем формулируем аксиомы и правила вывода Хоара и семантику этого исчисления. Для каждого правила мы доказываем его непротиворечивость. Также мы формулируем некоторые правила, которые допустимы в исчислении Хоара, с доказательством допустимости. В следующих параграфах мы рассматриваем слабейшие предусловия, формулируем и доказываем правила их нахождения. В качестве завершения этой главы мы доказываем существование слабейших предусловий и полноту обоих исчислений.

В шестой главе мы изучаем вопросы, связанные с подпрограммами. Мы определяем их синтаксис, семантику, а также приводим

некоторые определения, связанные с рекурсивными алгоритмами. Дальше мы изучаем метод построения программ без использования циклов и операторов перехода (функциональное программирования), доказывая, что любая программа эквивалентна функциональной. Затем мы доказываем обратное утверждение о том, что от рекурсии всегда можно избавиться, и, как следствие, от подпрограмм вообще. Завершается глава изучением методов доказательства корректности подпрограмм.

В седьмой главе мы изучаем некоторые вопросы, связанные с ресурсоёмкостью вычислительного процесса. Мы доказываем нижнюю границу памяти, необходимую для хранения натуральных чисел. Затем мы определяем, что такое время и память вычисления при работе алгоритма на конкретных входных значениях, а затем, что такое максимальные и средние время и память. Основная задача этой главы — дать общие представления об эффективности того или иного алгоритма и некоторые приёмы нахождения времени и памяти.

При изложении всего материала мы приводим достаточно большое число примеров. Детальный разбор примеров должен помочь читателю понять формальные определения на интуитивном уровне и показать, что все они имеют простой практический смысл.

Часто после примеров даётся некоторое количество задач. Как правило, эти задачи того же типа, что и рассмотренный пример, и решение задач позволяет глубже вникнуть в суть рассматриваемых вопросов и получить практический опыт работы с изучаемыми объектами.

Излагая доказательства тех или иных утверждений, мы старались по мере возможности избегать оборотов типа «ясно что», за которыми следует не вполне очевидный факт. Тем не менее, доказательства не всех утверждений приведены целиком. Отдельные части оставлены в качестве задач, которые сформулированы после соответствующих доказательств. Например, мы почти всегда, доказывая

утверждение для неполного ветвления, доказательство для полного оставляем в качестве задачи. Для некоторых утверждений мы вообще не приводим доказательств, оставляя их на самостоятельное изучение.

Все задачи в данной книге разбиты на три уровня сложности. Задачи без пометок — самые простые. Эти задачи составляют необходимый минимум для того, чтобы получить на экзамене положительную оценку. Задачи, отмеченные одной звёздочкой, имеют более высокую сложность. В эту категорию отнесены многие из задач на доказательство. Третий уровень трудности — задачи с двумя звёздочками. Эти задачи адресованы тем, кто желает глубже разобраться в изучаемом предмете, а также для факультативных занятий. Для решения некоторых из этих задач математического материала, изложенного во второй главе, может оказаться недостаточно. В этом случае следует обратиться к другим пособиям, где более подробно изложена соответствующая тема. Прежде всего, это относится к пятой главе, где мы делаем некоторые ссылки на утверждения, которые доказываются в курсе математической логики.

В конце книги приводится предметный указатель, в который включены все понятия, определяемые в данной книге. В начале указателя приведён перечень условных символов и обозначений. Если точного соответствия какого-либо обозначения не найдено, следует искать аналогичные, возможно, с другими буквами.

После предметного указателя приводится список литературы, рекомендованной для более глубокого изучения вопросов, упоминаемых в этой книге.

Все замечания и предложения следует отправлять авторам

sergeydudakov@yandex.ru
bnkarlov@gmail.com

Глава 1

Введение

§ 1.1. Основные понятия

С некоторым допущением можно сказать, что информатика — это наука о способах автоматической обработки данных. Это далеко не единственная дисциплина, изучающая подобного рода вопросы. Например, математическая логика, дискретная математика, теория вероятностей тоже тем или иным способом затрагивают эту тему. Можно сказать, что информатика, в отличие от перечисленных выше дисциплин, изучает, прежде всего, вопрос практического использования этих методов. Разумеется, широкое практическое использование невозможно без теоретических основ, поэтому можно считать, что теоретические дисциплины образуют фундамент для информатики.

Основные понятия информатики — алгоритм и исполнитель. Алгоритм — некоторый набор инструкций, который предназначен для решения какой-либо задачи. Исполнитель — тот, для кого этот набор инструкций предназначен. Естественно, что исполнитель должен понимать, как выполнять инструкции алгоритма. Следовательно, между алгоритмом и исполнителем должна быть связь —

семантика. Семантика — это то, как исполнитель выполняет инструкции алгоритма.

Один и тот же алгоритм может быть записан разными способами. Например, алгоритм решения квадратного уравнения можно записать в виде формулы, а можно — в виде последовательности действий вида: «Возвести b в квадрат», «Умножить a на c», и т. д. Поэтому удобно иметь некоторый способ единообразной записи алгоритмов.

Самый очевидный путь — использовать для записи алгоритмов естественные языки (например, русский или английский). Однако этот путь имеет множество недостатков, основные из которых следующие:

- Сложность изучения. Для изучения естественного языка необходимо потратить довольно существенное время.
- Сложность трансляции. Как правило, алгоритмы пишутся для их выполнения каким-либо вычислительным устройством. Следовательно, это вычислительное устройство должно тем или иным способом «понимать» инструкции алгоритма. Однако, восприятие естественных языков вычислительными устройствами — крайне сложная проблема, которая до сих пор не решена окончательно.
- Возможная неоднозначность. Многие конструкции естественных языков имеют больше одного толкования. Следовательно, исполнитель в ряде случаев не в состоянии решить, что же именно требуется выполнить.

Из-за этих недостатков, появились специальные языки — языки программирования (ЯП), которые их лишены. Как правило, ЯП строятся на основе небольшого количества слов естественного языка (обычно, английского) и их сокращений. Таким образом, ЯП просты в изучении и легко транслируются. Кроме того, имеется

жёсткий перечень способов построения программ, каждый из которых снабжён однозначным способом толкования. Поэтому все ЯП однозначны и не допускают различных толкований[1]. Алгоритм, записанный на каком-либо ЯП, называют п р о г р а м м о й. Как правило, ЯП являются у н и в е р с а л ь н ы м и, то есть на ЯП можно написать программу для решения любой теоретически разрешимой задачи. Однако, существуют примеры ЯП, которые таковыми не являются. Обычно, это — специализированные языки для решения достаточно узкого круга задач (например, старые версии языка запросов к базам данных — SQL, языки потоковой обработки текстовых файлов SED и AWK).

§ 1.2. Исторические сведения

Слово «алгоритм», пришло в русский язык из английского (algorithm), а в западных языках оно произошло от имени средневекового учёного Аль-Хорезми (лат. Algorismus, Algorithmi). Первоначально оно означало повсеместную сейчас десятичную систему счисления и способы действий над числами, в ней записанными. Впоследствии словом «алгоритм» стали называть произвольную последовательность действий.

Хотя современное значение слова «алгоритм» весьма недавнего происхождения, но алгоритмы используются людьми с самых давних времён. Например, ещё в древнем Вавилоне знали, что треугольник со сторонами 3, 4 и 5 является прямоугольным и использовали алгоритм построения прямого угла, основанный на построении таких треугольников. Там же был известен алгоритм решения квадратного уравнения. Древним грекам был известен алгоритм нахождения

[1]Это условие не всегда выполняется полностью, то есть существуют некоторые синтаксические конструкции, которые допускают двоякий смысл. Однако число таких конструкций невелико, и они специально оговорены в описании языка.

простых чисел (решето Эратосфена) и многие алгоритмы, предназначенные для геометрических построений. Естественно, что с развитием математики количество задач накапливалось, и для многих из них были придуманы те или иные алгоритмы решения. Хотя не для всех задач алгоритмы были придуманы, но большинство математиков верило, что рано или поздно они будут найдены. Ещё Г. В. Лейбниц мечтал придумать алгоритм, который решал бы любую математическую задачу.

Первые удары по этой мечте были нанесены в 19 веке, когда было доказано, что многие из задач (например, классические задачи древности — трисекция угла, удвоение куба, квадратура круга) не могут быть решены традиционными средствами (в случае приведённых выше задач, построение невозможно с помощью циркуля и линейки). Поэтому круг задач, для решения которых искались алгоритмы, заметно сузился. Постепенно всё свелось к задаче разрешения арифметики натуральных чисел: по произвольному утверждению о натуральных чисел (записанному, естественно, некоторым специальным образом) нужно определить, истинно оно или ложно.

В первой половине 20 века были предложены некоторые математические модели алгоритмов. Разные модели были предложены независимо многими людьми (А. М. Тьюринг, А. Чёрч, С. Клини, А. А. Марков, В. А. Успенский и т. д.). Однако возник вопрос: насколько универсальными являются эти модели? Тот факт, что задача неразрешима с их использованием, может быть истолкован так, что модели слишком упрощённые, то есть можно в определение алгоритма добавить какие-то средства, и с их использованием задача может быть решена. Оказалось, что все предложенные модели алгоритма эквивалентны, то есть то, что можно формализовать в рамках одной из этих моделей, можно формализовать и в любой другой. Это дало повод для того, чтобы выдвинуть гипотезу (тезис Чёрча), что эти модели в точности описывают всевозможные алгоритмы. В частно-

сти, все современные ЯП, допускают трансляцию в любую из этих моделей, что является ещё одним подтверждением тезиса Чёрча.

Точная математическая формулировка понятий «алгоритм» и «разрешимость» (то есть возможность решить задачу с помощью какого-либо алгоритма) дала возможность определять, разрешима та или иная задача или нет. В частности, было доказано, что арифметика натуральных чисел неразрешима.

В связи с появлением понятия «алгоритм» возникли многие задачи, которые непосредственно с ними связаны. Например, определить, останавливается ли алгоритм при некоторых входных данных, эквивалентны ли два алгоритма между собой и т. д. Большинство таких задач тоже оказались неразрешимыми.

Исторические представления об исполнителях тоже менялись. Исполнителем для первых алгоритмов был человек. Но уже в 17 веке Б. Паскаль построил первое механическое устройство[2] для выполнения сложения и вычитания многозначных чисел. В 19 веке появились арифмометры, которые помогали в выполнении более сложных операций. Однако, эти устройства являлись всего лишь средствами, исполнителем оставался человек.

Ч. Бэббидж в 19 веке дал теоретическое описание «аналитической машины». Эта машина должна имела запоминающее, операционное и управляющее устройства. В связи с наличием памяти и устройства управления, она уже могла работать по программе, то есть сама выступать исполнителем.

Первые электронные вычислительные машины появились в середине 20 века. Единственным способом написания программ была ручная запись чисел в ячейки памяти. Программы представляли собой последовательность чисел, которые обозначали те или иные машинные команды (ЯП первого уровня).

[2]Существуют сведения, что аналогичные устройства были сконструированы ранее, но ни их самих, ни их точных описаний не сохранилось.

В дальнейшем с увеличением скорости работы машин и объёма их памяти был изобретён более удобный способ записи программ — каждая машинная команда записывалась с помощью того или иного, легко запоминаемого обозначения. Были написаны программы, которые автоматически переводили эти языки в ЯП первого уровня — первые трансляторы. Такие ЯП, каждая инструкция которых соответствует одной машинной команде, называют ассемблерами (ЯП второго уровня).

Позже были построены ещё более удобные ЯП (третьего уровня), которые, в основном, используются и до сих пор с теми или иными дополнениями. В них уже имелись такие конструкции, которые реализовывались целым набором (иногда довольно большим) машинных команд (ЯП третьего уровня). Языки, которые изучаются в данной книге, являются математическими моделями ЯП третьего уровня.

В связи с распространением вычислительной техники появились новые разделы теории алгоритмов, например, сложность вычислений, которая изучает вопросы ресурсоёмкости решения задач с помощью вычислительных устройств.

§ 1.3. Свойства алгоритмов и языков программирования

В математике рассматривают два вида алгоритмов — детерминированные и недетерминированные. Алгоритм называют детерминированным, если каждая инструкция алгоритма однозначно определяет действие исполнителя. Алгоритм недетерминированный, если такой однозначности нет. Очевидно, что в реальных вычислительных устройствах недетерминированные алгоритмы неприменимы.

Алгоритм называется у н и в е р с а л ь н ы м, если он приводит к правильному решению задачи при любых начальных условиях. Алгоритм называют к о н е ч н ы м, если он приводит к решению задачи за конечное число шагов. Естественно, что самый «хороший» случай — когда для решения задачи существует универсальный и конечный алгоритм. Однако, имеется ряд задач, для которых эти условия оказываются взаимоисключающими — каждый конечный алгоритм не универсален, а каждый универсальный не конечен. Но большинство задач, которые приходится решать с помощью вычислительных устройств являются «хорошими», то есть для них существуют универсальные конечные алгоритмы.

Если для решения задачи существует несколько алгоритмов, то естественно возникает вопрос, какой из них наиболее целесообразно использовать. Можно сравнивать алгоритмы по их размеру — чем меньше алгоритм, тем более простым он является, тем меньше вероятность ошибки. Кроме того, для хранения самого алгоритма тоже необходима память, которая при небольших размерах алгоритма может быть использована под другие нужды.

Другой способ сравнения алгоритмов — по их вычислительной сложности, то есть по тому, сколько времени, памяти или других ресурсов требуется для выполнения этого алгоритма. Чем меньше эти требования, тем более приемлемым является алгоритм. Если, например, алгоритм существует, но для завершения работы требуется не меньше 10^{100} шагов, то его следует считать неприемлемым, так как мы никогда не дождёмся результата. Можно доказать, что существуют задачи, для решения которых существуют алгоритмы, но каждый из них требует колоссальных ресурсов уже для небольших входных данных. Подобного рода задачи следует считать практически неразрешимыми.

В противоположность универсальным алгоритмам выделяют алгоритмы ч а с т н ы е, то есть, предназначенные для решения кон-

кретной задачи, и не рассчитанные на решение иных аналогичных задач. Такие алгоритмы менее полезны, однако, часто прибегают именно к ним. Основная причина — универсальный алгоритм или отсутствует, или является чрезмерно сложным, или требует много ресурсов (времени, памяти), которые предоставить невозможно. В частном же случае почти всегда можно прибегнуть к тем или иным упрощениям, которые делают алгоритм приемлемым.

Промежуточное положение между универсальными и частными алгоритмами занимают алгоритмы полууниверсальные, то есть рассчитанные на решение задачи не при любых начальных условиях, а при некоторых из них.

Языки программирования можно разделить на две большие группы — декларативные и императивные. Программы на императивных языках программирования содержат точную последовательность инструкций, выполняя которую, исполнитель получает результат. К этой группе относится большинство языков.

Программы на декларативных языках (SQL, Prolog) содержать не точные последовательности инструкций, а цели, которые должны быть достигнуты, и описание средств, с помощью которых это следует делать. Точную последовательность действий в этом случае определяет сам исполнитель, анализируя эти сведения.

Другой способ деления ЯП — в зависимости от задач, на решение которых ориентированы ЯП. Выделяют универсальные ЯП — ЯП, которые не предназначены для решения какой-то специальной группы задач, а годятся для решения их широкого круга[3] (C, C++, Python, Java). Другая группа — специализированные ЯП, то есть, ориентированные на решение задач из достаточно узкой

[3]Термин универсальный по отношению к ЯП, следовательно, используется в двух различных смыслах. С одной стороны, универсальный ЯП — ЯП, на котором можно решить любую разрешимую проблему, с другой — ЯП, который предназначен для решения широкого круга задач.

области (Refal, Lisp, Prolog, SQL, языки разных интегрированных систем, систем управления базами данных).

Императивные языки делятся по способу записи инструкций. Выделяют структурные ЯП (C, C++, Python), ЯП ориентированные на метки (ранние версии Basic'a, Fortran, Focal, ассемблеры), функциональные (Lisp), объектно-ориентированные (C++, Smalltalk) и т. д.

§ 1.4. Примеры алгоритмов

Рассмотрим несколько примеров математических задач и построим алгоритмы (пока на естественном языке, с привлечением математических формул) для их решения. Мы предполагаем, что исполнитель умеет выполнять арифметические действия с действительными числами.

Пример 1.1 (Решение квадратного уравнения)**.** *Пусть необходимо решить квадратное уравнение*

$$x^2 - 5x + 4 = 0.$$

Естественно, что корни находятся по формулам:

$$x_1 = \frac{5 + \sqrt{5^2 - 4 \times 4}}{2}; \quad x_2 = \frac{5 - \sqrt{5^2 - 4 \times 4}}{2}.$$

Это пример частного алгоритма.

Пример 1.2. *Усложним предыдущий пример и построим более универсальный алгоритм. Пусть дано произвольное уравнение вида*

$$ax^2 + bx + c = 0.$$

Простейший способ обобщения — это подставить в предыдущие формулы вместо конкретных чисел буквенные обозначения:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}; \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Такой алгоритм уже будет полууниверсальным, так как рассчитан на решение некоторой группы исходных задач. Более точно, он применим в случае, когда $a \neq 0$ *и* $b^2 - 4ac \geqslant 0$.

Пример 1.3. *Сформулируем алгоритм, который решает уравнение*

$$ax^2 + bx + c = 0$$

для любых коэффициентов a, b *и* c.

1) *Если* $a = b = c = 0$, *то ответ — «*x *— любое число».*

2) *Если* $a = b = 0$, $c \neq 0$, *то ответ — «решений нет».*

3) *Если* $a = 0$, $b \neq 0$, *то*
$$x = \frac{-c}{b}.$$

Если три предыдущих пункта не выполнены, то $a \neq 0$.

4) *Вычислить дискриминант:*
$$D = b^2 - 4ac.$$

5) *Если* $D = 0$, *то*
$$x = \frac{-b}{2a}.$$

6) *Если* $D > 0$, *то*
$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}; \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

7) *Если* $D < 0$, *то пусть*
$$u = \frac{-b}{2a}; \quad v = \frac{\sqrt{-D}}{2a}.$$
Тогда
$$x_1 = u + vi; \quad x_2 = u - vi.$$
Здесь $i = \sqrt{-1}$.

Пример 1.4 (Нахождение НОД)**.** *Простейший алгоритм нахождения наибольшего общего делителя x и y следует из его определения: можно перебирать все натуральные числа от 1 до x и найти наибольшее из них, которое делит и x, и y.*

1) *Пусть $z = 1$, $i = 1$.*
2) *Если $i = x$, то ответ — z.*
3) *Если i делит x и y, то пусть $z = i$.*
4) *Увеличить i на 1 и перейти к п. 2.*

В отличие от предыдущего алгоритма, данный алгоритм является итерационным, то есть при его выполнении будет многократно повторяться одна и та же последовательность действий (пункты 2–4).

Данный алгоритм является полууниверсальным, например, он даёт неправильный ответ в случае, когда $x = 0$.

Задача 1.1. Доработайте алгоритм из предыдущего примера так, чтобы он стал универсальным.

Приведённый алгоритм является достаточно медленным из-за того, что перебираются все натуральные числа некоторого диапазона, хотя заранее можно сказать, что большинство из них не могут равняться НОД(x, y). Более совершенным является алгоритм Евклида, который основан на следующем утверждении:

Утверждение 1.1. *Если x, y — натуральные числа и $x > y$, то* НОД$(x, y) =$ НОД$(x - y, y)$.

Пример 1.5 (Алгоритм Евклида)**.**

1) *Если $x = y$, то ответ — x.*
2) *Если $x < y$, то ответ —* НОД$(x, y - x)$.
3) *Если $x > y$, то ответ —* НОД$(x - y, y)$.

Этот алгоритм является рекурсивным, то есть для достижения результата используется тот же самый алгоритм, но с другими входными данными. Например, для вычисления НОД$(6, 4)$*, нужно будет последовательно вычислить* НОД$(2, 4)$ *и* НОД$(2, 2)$.

Задача 1.2. Определите, является ли этот алгоритм универсальным. Если нет, измените его так, чтобы он стал им.

Можно превратить этот алгоритм в нерекурсивный:

Пример 1.6.

1) *Если* $x = y$, *то ответ —* x.
2) *Если* $x < y$, *то вычесть* x *из* y *и перейти к п. 1.*
3) *Если* $x > y$, *то вычесть* y *из* x *и перейти к п. 1.*

Рассмотрим ещё одну задачу.

Пример 1.7. *Пусть даны длины сторон треугольника* a, b, c, *и требуется определить вид треугольника — остро-, прямо- или тупоугольный. Для решения данной задачи можно воспользоваться теоремой косинусов: если угол* α *лежит против стороны* a, *то*

$$\cos\alpha = \frac{b^2 + c^2 - a^2}{2bc}.$$

Используя знак косинуса, а так же тот факт, что наибольший угол лежит против наибольшей стороны можно предложить такой алгоритм:

1) *Если* $a < b$, *поменять* a *и* b *между собой.*
2) *Если* $a < c$, *поменять* a *и* c *между собой.*
3) *Если* $a \geqslant b + c$, *то ответ — «такого треугольника не существует».*
4) *Вычислить*
$$A = \frac{b^2 + c^2 - a^2}{2bc}.$$
5) *Если* $A = 0$, *то ответ — «прямоугольный».*
6) *Если* $A < 0$, *то ответ — «тупоугольный».*
7) *Если* $A > 0$, *то ответ — «остроугольный».*

Глава 2

Некоторые математические сведения

В этой книге мы будем предполагать, что исполнителем является некоторое устройство, которое может оперировать некоторыми конечными математическими объектами: натуральными числами, конечными отношениями, конечными множествами и т. д. Мы считаем, что читатель знаком с понятиями «натуральное число» и «множество» и умеет выполнять действия с ними. В этой главе мы постараемся кратко описать более сложные понятия для тех, кто с ними не знаком. Подготовленные читатели могут пропустить эту часть, и вернуться к ней, если встретят какой-либо незнакомый термин.

§ 2.1. Алгебра и теория множеств

Прежде всего определим термин функция (отображение).

Определение 2.1 (Функция, отображение)**.** *Пусть даны множества A и B. Будем называть f n-местной функцией,*

действующей из A в B, если f — это множество упорядоченных наборов из $n+1$ элемента вида $(a_1, a_2, \ldots, a_n, b)$, где $a_1, \ldots, a_n \in A$, $b \in B$, которое удовлетворяет следующему условию: для всяких $a_1, \ldots, a_n \in A$ существует в точности один элемент $b \in B$ такой, что $(a_1, \ldots, a_n, b) \in f$.

Рассмотрим пример.

Пример 2.1. *Пусть $A = \{a_1, a_2\}, B = \{b_1, b_2\}$. Тогда*

- *$f = \{(a_1, b_1), (a_2, b_1)\}$ — функция.*
- *$g = \{(a_1, b_1), (a_1, b_2), (a_2, b_1)\}$ — не функция, потому что для элемента $a_1 \in A$ существуют два элемента из B: b_1 и b_2 такие, что $(a_1, b_1) \in g$ и $(a_1, b_2) \in g$.*
- *$h = \{(a_1, b_2)\}$ — не функция, потому что для элемента $a_2 \in A$ не существует элемента $b \in B$ такого, что $(a_2, b) \in h$.*

Задача 2.1. Выпишите всевозможные одно и двухместные функции из множества $A = \{a_1, a_2\}$ в множество $B = \{b_1, b_2\}$.

Определение 2.2 (Значение функции)**.** *Значением функции f на элементах $a_1, \ldots, a_n$ называется элемент $b \in B$ такой, что $(a_1, \ldots, a_n, b) \in f$. Значение функции f на $a_1, \ldots, a_n$ записывается в виде $f(a_1, \ldots, a_n)$. Для одноместных функций мы часто будем опускать скобки и писать fa_1 вместо $f(a_1)$.*

Пример 2.2. *Рассмотрим функцию f из примера 2.1. Для нее $fa_1 = f(a_1) = b_1$ и $fa_2 = f(a_2) = b_1$.*

Ещё один способ записи значений функции: $f : a \mapsto b$, что означает $fa = b$.

Определение 2.3 (Ограничение функции)**.** *Пусть $A_1 \subseteq A$ и f — n-местная функция, действующая из A в B. Тогда f_1 называется ограничением f на множество A_1 (записывается в виде $f \restriction A_1$), если f_1 — n-местная функция, действующая из A_1 в B,*

и $f(a_1, \ldots, a_n) = f_1(a_1, \ldots, a_n)$ *выполняется для любых элементов* $a_1, \ldots, a_n \in A_1$.

Пример 2.3. *Рассмотрим множества* A, B *и функцию* f *из примера 2.1 на предыдущей странице. Пусть* $A_1 = \{a_1\}$. *Тогда* $A_1 \subseteq A$. *Функция* $f_1 = \{(a_1, b_1)\}$ *будет ограничением функции* f *на* A_1. *Если* $A_2 = \{a_2\}$, *то* $f_2 = \{(a_2, b_1)\}$ *будет ограничением* f *на* A_2.

Таким образом, ограничение — это функция, которая получается из исходной сужением области определения.

Задача 2.2. Выпишите всевозможные ограничения функций из задачи 2.1 на противоположной странице.

Замечание 2.1. Для двухместных функций вместо рассмотренной выше записи значения (п р е ф и к с н а я ф о р м а — так как символ функции стоит до её аргументов), используется и н ф и к с н а я ф о р м а записи, когда символ функции ставится между аргументами. Например, мы пишем $a + b$ вместо $+(a, b)$.

Кроме этого, для некоторых функций используются и другие формы записи, например, постфиксная: $x!$, с использованием индексов: x^y, $\log_x y$ и т. д.

Задача 2.3. Запишите формулы для нахождения корней квадратного уравнения в префиксной форме.

Определение 2.4 (Частичная функция)**.** *Пусть даны множества* A *и* B. *Говорим, что* f *—* n*-*м е с т н а я ч а с т и ч н а я ф у н к ц и я *(*и л и ч а с т и ч н о е о т о б р а ж е н и е*), действующая из* A *в* B, *если* f *— это множество упорядоченных наборов из* $n+1$ *элементов вида* $(a_1, a_2, \ldots, a_n, b)$, *где* $a_1, \ldots, a_n \in A$, *а* $b \in B$. *Кроме того, для всяких* $a_1, \ldots, a_n \in A$ *существует не более одного элемента* $b \in B$ *такого, что* $(a_1, \ldots, a_n, b) \in f$.

Если для $a_1, \ldots, a_n$ *не существует элемента* $b \in B$ *такого, что* $(a_1, \ldots, a_n, b) \in f$, *то говорим, что на элементах* $a_1, \ldots, a_n$ *частичная функция* f *н е о п р е д е л е н а. Если такой элемент* $b \in B$ *существует, то он называется з н а ч е н и е м* f *на элементах* $a_1, \ldots, a_n$ *и обозначается* $f(a_1, \ldots, a_n)$.

Пример 2.4. *Рассмотрим те же множества:* $A = \{a_1, a_2\}$ *и* $B = \{b_1, b_2\}$. *Пусть* $f = \{(a_1, b_2)\}$. *Тогда* f *— частичная функция.* $f(a_1)$ *определено и* $f(a_1) = b_2$. $f(a_2)$ *не определено. Иногда последний факт записывают в виде* $f(a_2) = \infty$.

Задача 2.4. Напишите всевозможные одноместные частичные функции из $A = \{a_1, a_2\}$ в $B = \{b_1, b_2\}$.

Замечание 2.2. Пусть f и g — некоторые частичные функции из A в B. Нам часто будет требоваться оборот вида: «для всякого $a \in A$ либо $f(a)$ и $g(a)$ одновременно неопределены, либо одновременно определены и при этом $f(a) = g(a)$». Мы часто будем заменять это более кратким: «$f(a) = g(a)$ для всякого $a \in A$». Таким образом, если ни $f(a)$, ни $g(a)$ неопределено, то мы считаем, что $f(a) = g(a)$. Это можно рассматривать, как некоторое обозначение.

Нам понадобится ещё одно определение, связанное с действиями уже над самими функциями.

Определение 2.5 (Композиция отображений)**.** *Пусть даны два (частичных) одноместных отображения:* $f : A \to B$ *и* $g : B \to C$. *Произведение отображений (композиция отображений)* f *и* g *(именно в таком порядке) — это (частичное) отображение* $h : A \to C$ *определённое следующим образом:*

$$h = \{(a, c) : a \in A, c \in C \text{ существует } b \in B \text{ такой, что } (a, b) \in f \text{ и } (b, c) \in g\}.$$

Очевидно, что $h(a)$ *определено тогда и только тогда, когда определены* $f(a)$ *и* $g(f(a))$, *и в этом случае* $h(a) = g(f(a))$. *Композиция отображений* f *и* g *обозначается так:* $h = gf$.

Если $f : A \to A$, *то точно так же как и для чисел можно определить степень отображения:*

$$f^n = \underbrace{ff \dots f}_{n \text{ раз}}$$

n — натуральное число. Считаем, что f^0 — тождественное отображение, то есть $f^0(a) = a$ для всех $a \in A$.

Пример 2.5. *Пусть $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$, $C = \{c_1, c_2\}$. Пусть $f = \{(a_1, b_1), (a_2, b_1)\}$ и $g = \{(b_1, c_2)\}$. Тогда, очевидно,*

$$gf = \{(a_1, c_2), (a_2, c_2)\}.$$

Заметим, что хотя g и является частичной функцией в результате произведения получилась функция не частичная.

Замечание 2.3. Иногда, чтобы подчеркнуть, что функция всюду определена, её называют тотальной.

Задача 2.5. Напишите всевозможные одноместные частичные функции из множества $A = \{a_1, a_2\}$ в A и их композиции.

Определение 2.6 (Декартова степень)**.** *Пусть A — множество, n — натуральное число. С помощью A^n мы обозначаем множество всевозможных упорядоченных наборов длины n, элементами которых являются элементы A (декартова степень A):*

$$A^n = \{(a_1, \ldots, a_n) : a_1, \ldots, a_n \in A\}.$$

В частности, $A^0 = \{()\}$ для любого A. Если отождествить набор из одного элемента (a) с самим элементом a, то $A^1 = A$. Множество $A^{<\omega}$ — множество всевозможных конечных наборов из элементов A, то есть объединение всех декартовых степеней:

$$A^{<\omega} = \bigcup\{A^i : i \in \omega\}.$$

Пример 2.6. *Если $A = \{a_1, a_2\}$, то*

$$A^2 = \{(a_1, a_1), (a_1, a_2), (a_2, a_1), (a_2, a_2)\}.$$

Замечание 2.4. Если f — n-местная (частичная) функция на множестве A, то f можно рассматривать как одноместную (частичную) функцию на

множестве A^n. В самом деле, каждый набор из $n+1$ элемента $(a_1, \ldots, a_n, b)$ можно рассматривать как набор из двух элементов: первый — $(a_1, \ldots, a_n) \in A^n$, второй — $b \in B$.

Заметим, что если множество A рассматривать как множество символов (букв), то $A^{<\omega}$ — это множество всевозможных конечных наборов букв из A, то есть множество *слов* алфавита A.

Определение 2.7 (Область определения)**.** *Если f — одноместная (частичная) функция, действующая из A в B, то с помощью* $\operatorname{dom} f$ *мы обозначим область определения f:*

$$\operatorname{dom} f = \{a \in A : f(a) \text{ определено}\}.$$

Если функция n-местная, то $\operatorname{dom} f$ *— подмножество A^n (см. предыдущее замечание).*

Определение 2.8 (Множество значений)**.** *Если f — (частичная) функция, то с помощью* $\operatorname{rng} f$ *мы обозначим множество значений f:*

$$\operatorname{rng} f = \{f(a) : a \in \operatorname{dom} f\}.$$

Пример 2.7. *Если* $f = \{(a, b), (c, b)\}$, *то* $\operatorname{dom} f = \{a, c\}$, $\operatorname{rng} f = \{b\}$.

Определение 2.9 (Множество подмножеств)**.** $\mathcal{P}(A)$ *— множество всех подмножеств множества A:*

$$\mathcal{P}(A) = \{A' : A' \subseteq A\}.$$

Пример 2.8. *Если* $A = \{a_1, a_2\}$,*то*

$$\mathcal{P}(A) = \{\varnothing, \{a_1\}, \{a_2\}, \{a_1, a_2\}\}.$$

Определение 2.10 (Последовательность)**.** *Последовательность элементов множества A — это отображение множества натуральных чисел или его начального отрезка в A. Последовательность можно считать конечной или бесконечной строкой, элементами которой являются элементы A. Последовательности мы будем записывать в виде $(a_i)_i$,что означает, что a_i является i-м элементом последовательности. Если последовательность конечна, то её мы будем записывать также в виде $(a_i)_{i=0}^{y}$, что означает $a_0, a_1, \ldots, a_{y-1}, a_y$.*

Часто бывает удобно начинать нумерацию элементов не с нуля, а с какой-то другой границы x, тогда записываем $(a_i)_{i=x}^{y}$, что означает $a_x, a_{x+1}, \ldots, a_{y-1}, a_y$.

Конечную последовательность

$$(a_i)_{i=x}^{y},$$

можно отождествить с упорядоченным набором:

$$(a_x, a_{x+1}, \ldots, a_{y-1}, a_y).$$

Иногда последовательности называют списками.

Важное свойство натуральных чисел — отсутствие на нём бесконечных убывающих цепей, то есть таких бесконечных последовательностей $(a_i)_i$, что $a_i > a_{i+1}$ для всех i.

Теорема 2.1 (Об убывающих цепях)**.** *Пусть дана последовательность натуральных чисел $(x_i)_i$ такая, что $x_{i+1} < x_i$. Тогда последовательность содержит конечное количество элементов.*

Доказательство. По индукции легко доказывается, что $x_i \leqslant x_0 - i$.

Базис индукции. $x_0 = x_0 - 0$.

Шаг индукции. Пусть для i доказано.

$$x_{i+1} < x_i \leqslant x_0 - i,$$

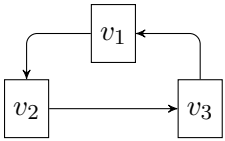


Рис. 2.1: Граф G из примера 2.9.

следовательно,

$$x_{i+1} \leqslant x_0 - (i+1).$$

Получаем, что

$$x_{x_0} \leqslant x_0 - x_0 = 0$$

и в последовательности не может быть больше $x_0 + 1$ элемента, поскольку нельзя найти натурального x_{x_0+1} для которого было бы выполнено $x_{x_0+1} < 0$. □

§ 2.2. Графы

Определим, что такое граф и дерево.

Определение 2.11 (Граф). *Граф — пара $G = (V, E)$, где V — произвольное множество, а $E \subseteq V^2$. V называется множеством вершин графа G, E — множество рёбер графа G.*

Граф можно представлять как множество точек V (вершин), и из точки v_1 в точку v_2 ($v_1, v_2 \in V$) идёт стрелка (ребро), если $(v_1, v_2) \in E$.

Пример 2.9. *Рассмотрим пример. Пусть $V = \{v_1, v_2, v_3\}$,*

$$E = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$$

и $G = (V, E)$. Этот граф состоит из трёх вершин v_1, v_2, v_3 и из v_1 в v_2, из v_2 в v_3, из v_3 в v_1 идут рёбра (рис. 2.1).

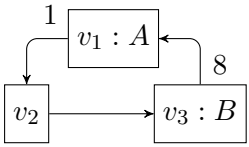


Рис. 2.2: Нагруженный граф (G, f_V, f_E) (пример 2.10).

Задача 2.6. На рисунке 6.8 на стр. 239 изображён граф. Выпишите явным образом множество вершин и множество рёбер этого графа.

Обычно необходимо с вершинами и рёбрами графа связать некоторую информацию.

Определение 2.12 (Нагруженный граф)**.** *Нагруженный (или размеченный) граф — это тройка (G, f_V, f_E), в которой $G = (V, E)$ — граф, а f_V и f_E — некоторые (возможно, частичные) функции, определённые на множествах V и E соответственно. f_V — функция нагрузки (разметки) вершин, которая по вершине v определяет объект $f_V(v)$, который приписан этой вершине, f_E — функция нагрузки (разметки) рёбер, которая по каждому ребру e определяет объект $f_E(e)$, сопоставленный ему. Если $f_V(v)$ не определено для некоторой вершины v, то считаем, что в этой вершине нет никакой дополнительной информации. Аналогично для рёбер.*

Пример 2.10. *Нагрузим граф из предыдущего примера. Пусть*

$$f_V = \{(v_1, A), (v_3, B)\};$$
$$f_E = \{(v_1, v_2, 1), (v_3, v_1, 8)\}.$$

Нагруженный граф (G, f_V, f_E) можно представить рисунком 2.2.

Определение 2.13 (Путь, цикл)**.** *Путь на графе $G = (V, E)$ из вершины v_1 в вершину v_n — последовательность вершин $(v_1, v_2, \ldots, v_n)$ такая, что $(v_i, v_{i+1}) \in E$ для $i = 1, \ldots, n-1$. Число n — это длина пути. Если $v_1 = v_n$, то есть, путь начинается там же, где и заканчивается, то он называется цикл ом.*

Пример 2.11. *В предыдущих примерах (v_1, v_2, v_3) — путь из v_1 в v_3, так как из v_1 в v_2 идёт ребро и из v_2 в v_3 идёт ребро. (v_1, v_2, v_3, v_1) — цикл.*

Задача 2.7. Выпишите все пути длины 4 и менее на графе на рис. 6.8 на стр. 239.

Важную роль в математике играют специального вида графы — деревья.

Определение 2.14 (Дерево, корень, лист)**.** *Граф G называется дерев ом, если существует единственная вершина r (корень дерева), из которой в любую другую вершину имеется путь и при том — единственный. Вершины дерева, из которых не выходит ни одно ребро, называются лист ьями дерева, остальные вершины — внутренними.*

Пример 2.12. *Рассмотрим нагруженное дерево на рис. 2.3 на следующей странице:*

$$V = \{v_1, v_2, v_3, v_4\},$$
$$E = \{(v_1, v_2), (v_1, v_3), (v_2, v_4)\},$$
$$f_V = \{(v_1, 4), (v_2, 5), (v_3, 2), (v_4, 8)\},$$
$$f_E = \varnothing.$$

В этом дереве v_1 — корень.

Задача 2.8. На рисунке 6.10 на стр. 246 изображён некоторый граф. Проверьте, что он является деревом. Является ли деревом граф на рисунке 6.8 на стр. 239?

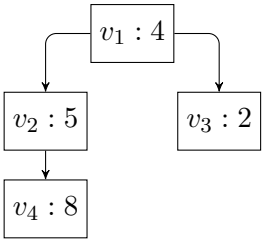


Рис. 2.3: Нагруженное дерево $((V, E), f_V, f_E)$ из примера 2.12.

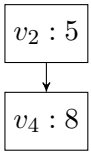


Рис. 2.4: Поддерево из примера 2.13.

Определение 2.15 (Поддерево)**.** *Для каждой вершины v дерева $D = (V, E)$ можно определить множество вершин V_v — тех, в которые ведут пути из v. Граф $(V_v, E \cap V_v^2)$ называется* п о д д е р е в о м *дерева D с корнем v.*

Задача 2.9. Докажите, что каждое поддерево — дерево.

Пример 2.13. *В предыдущем примере можно выделить, например, поддерево $(V_{v_2}, \{(v_2, v_4)\})$ с корнем v_2, которое изображено на рис. 2.4.*

Задача 2.10. Выделите из дерева на рис. 6.10 на стр. 246 всевозможные поддеревья.

Следующая теорема даёт необходимые и достаточные условия конечности (бесконечности) дерева:

Теорема 2.2 (О конечных деревьях)**.** *Дерево конечно тогда и только тогда, когда количество сыновей каждой вершины конечно и каждая ветвь дерева конечна.*

§ 2.3. Математическая логика

Из логических формул мы будем рассматривать только формулы элементарной арифметики, и все определения будут относиться только к элементарной арифметике и её стандартной модели $(\omega, +, \times)$ — множеству натуральных чисел с обычными операциями сложения и умножения. При необходимости мы будем применять и другие общеизвестные арифметические операции и отношения: факториал $x!$, порядок $x < y$ и т. д. Мы также будем избегать точных формальных определений, если они слишком громоздки, и дадим только сведения, которые необходимы для прочтения книги. Более общие и точные определения можно найти, например, в [2].

Определение 2.16 (Формула первого порядка)**.** *Пусть зафиксировано некоторое множество V для обозначения переменных. Атомные формулы — строки вида «$P(e_1, \ldots, e_n)$», где P — наименование какого-либо отношения, $e_1, \ldots, e_n$ — выражения (иначе называемые термами)*[1]*. Кавычки здесь и далее в этом определении использованы только для того, чтобы отделить вводимые понятия от основного текста. Мы будем записывать отношения в инфиксной форме для стандартных отношений ($=$ или $<$).*

Также атомными формулами будем считать специальные строки: «$\top$» и «$\bot$».

Если φ, ψ — формулы, $x \in V$ — переменная то строки «$(\varphi \wedge \psi)$», «$(\varphi \vee \psi)$», «$(\varphi \to \psi)$», «$(\neg\varphi)$», «$(\exists x)\varphi$» и «$(\forall x)\varphi$» тоже являются формулами. Мы будем часто опускать скобки, считая, что унарные

операции ($\neg$, $\exists$, $\forall$) имеют приоритет над бинарными, $\wedge$ — над $\vee$, а $\vee$ — над $\to$.

Знаки $\wedge$ — логическое «И», $\vee$ — логическое «ИЛИ», $\to$ — импликация, и $\neg$ — логическое «НЕ» называют булевыми связками, знаки $\exists$ и $\forall$ — кванторными символами.

Пример 2.14. *$x + 5z = xy$ и $x + z^2 < y + zy$ — атомные формулы. Из них можно составить, например, такие формулы:*

$$x + 5z = xy \wedge x + z^2 < y + zy$$

$$x + 5z = xy \vee x + z^2 < y + zy$$

$$x + 5z = xy \to x + z^2 < y + zy$$

$$\neg x + 5z = xy$$

$$(\exists z) x + 5z = xy$$

$$(\forall y) x + z^2 < y + zy$$

Задача 2.11. Приведите другие примеры формул.

Определение 2.17 (Значение формулы)**.** *Пусть дано некоторое состояние σ — отображение множества переменных V в множество натуральных чисел. Значением выражения e на состоянии σ будем называть результат вычисления выражения, которое получается из e заменой всех переменных x на $\sigma(x)$.*

Значением формулы на состоянии будет 1 *или* 0*. Единицу можно мыслить как истинность утверждения, обозначаемого формулой, а ноль — как его ложность, поэтому оборот «формула имеет значение* 1*» часто заменяют на «формула истинна», а «формула имеет значение* 0*» — на «формула ложна».*

Для атомной формулы значением является 1*, если соответствующее отношение выполняется, и* 0 *— если нет.*

[1] Пока мы не уточняем, что такое выражение. Мы полагаем, что для понимания этого материала достаточно считать e_i арифметическими выражениями в интуитивном смысле, например, $(x + 3) \times y$ или $xy + z$.

Для формул $\top$ *и* $\bot$ *значения всегда равны* 1 *и* 0 *соответственно.*

Значения формул, построенных с помощью булевых связок, определяются по следующей таблице:

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \to \psi$	$\neg\varphi$
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	1	0

Пример 2.15. *Рассмотрим формулы из предыдущего примера. Возьмем состояие*

$$\sigma = \{(x, 4), (y, 6), (z, 3)\}.$$

Тогда значениями выражений $x+5z$, xy, $x+z^2$ *и* $y+zy$ *будут соответственно* 19, 24, 13 *и* 24. *Значение формулы* $x + 5z = xy$ *равно* 0, *потому что* $19 \neq 24$. *Значение формулы* $x + z^2 < y + zy$ *равно* 1, *потому что* $13 < 24$. *По таблице находим значение следующих четырёх формул:*

Формула	*Значение*
$x + 5z = xy \wedge x + z^2 < y + zy$	0
$x + 5z = xy \vee x + z^2 < y + zy$	1
$x + 5z = xy \to x + z^2 < y + zy$	1
$\neg x + 5z = xy$	1

Определение 2.18 (Значение формулы)**.** *Формула* $(\exists x)\varphi$ *имеет значение* 1 *на состоянии* σ, *если значение формулы* φ *равно* 1 *на некотором состоянии* τ, *которое отличается от* σ *только значением на* x *(или не отличается вообще). Значение формулы* $(\forall x)\varphi$ *равно* 1 *на состоянии* σ, *если значение формулы* φ *равно* 1 *на любом состоянии* τ, *которое отличается от* σ *только значением на* x *(или не отличается вообще). Можно сказать, что формула* $(\exists x)\varphi$ *истинна, если «существует* x *такое, что* φ *истинна», а формула* $(\forall x)\varphi$ *истинна, если «для всякого* x *формула* φ *истинна»,. В связи с этим строку вида* $(\exists x)$ *называют квантором существова-*

ния по переменной x, а строку вида $(\forall x)$ — квантором всеобщности по переменной x.

Продолжим рассматривать тот же пример.

Пример 2.16. *Определим значение формул с кванторами. Формула $(\exists z)\ x+5z=xy$ утверждает, что «существует z, для которого $x+5z=xy$». Легко проверить, что, взяв $z=4$ и оставив значение остальных переменных без изменений, мы получим $24=24$. Следовательно, формула $(\exists z)\ x+5z=xy$ истинна на σ.*

$(\forall y)\ x+z^2<y+zy$ говорит «для всякого y выполняется $x+z^2<y+zy$». Взяв $y=3$ и не изменяя значений x и z получим $13<12$, то есть 0. Значит формула $(\forall y)\ x+z^2<y+zy$ на состоянии σ ложна.

Рассмотрим более сложную формулу:

$$(\forall x)(\exists y)(\forall u)(\forall v)(uv=x+y\to(u=1\vee v=1)).$$

Покажем, что она истинна на любом состоянии σ. Пусть состояние τ, если и отличается от σ, то только значением x. Поскольку существуют сколь угодно большие простые числа, то пусть p — произвольное простое число не меньшее τx. Пусть состояние ρ отличается от τ только тем, что $\rho y=p-\tau x$. Рассмотрим теперь произвольное состояние π, в котором по сравнению с ρ могут быть изменены только значения переменных u и v. Если $\pi u\times\pi v\neq\pi x+\pi y$, то по таблице определяем, что импликация должна быть истинна. Если $\pi u\times\pi v=\pi x+\pi y$, то поскольку $\pi x+\pi y=p$ — простое число, делаем вывод, что или πu, или πv равно 1. В этом случае импликация также оказывается истинной.

Поскольку мы убедились что для произвольного состояния π указанного вида значение импликации равно 1, то формула

$$(\forall u)(\forall v)(uv=x+y\to(u=1\vee v=1))$$

истинна на состоянии ρ. Так как ρ отличается от τ только значением переменной y, то на τ истинна формула

$$(\exists y)(\forall u)(\forall v)(uv=x+y\to(u=1\vee v=1)).$$

А поскольку мы в качестве τ взяли произвольное состояние, отличающееся от σ значением x, то на σ истинна исходная формула.

Задача 2.12. Для каждой формулы из примера 2.14 на стр. 35 постройте пример состояния, когда её значение отличается от того, которое было найдено в примере 2.15 на стр. 36. Для всех ли формул это возможно?

Задача 2.13. Запишите формулы, которые были бы истинны на состоянии σ тогда и только тогда, когда

1) σx — простое число,
2) σx — степень простого числа,
3) σx — полный квадрат,
4) Существует треугольник со сторонами σx, σy и σz.

Определение 2.19. *Истинность формулы φ на состоянии σ обозначается с помощью $\sigma \models \varphi$, а ложность — $\sigma \not\models \varphi$.*

Определение 2.20 (Эквивалентные формулы)**.** *Пусть φ, ψ — формулы. Если $\sigma \models \varphi$ тогда и только тогда, когда $\sigma \models \psi$ для любого состояния σ, то мы называем формулы φ и ψ эквивалентными (записывается $\varphi \equiv \psi$).*

Глава 3

Структурированные программы

§ 3.1. Синтаксис

Мы определим некоторый универсальный язык программирования и будем подробно изучать свойства ЯП на его примере. Позднее мы введём и другие ЯП.

Определение 3.1 (Предметная область)**.** *Предметная область ЯП — это множество объектов, с которыми ЯП позволяет работать без дополнительных программных частей.*

В большинстве ЯП предметная область включает в себя:

1) целые числа некоторого диапазона;

2) рациональные числа (с некоторыми ограничениями);

3) символы;

4) логические значения (истина, ложь, неизвестно).

То есть практически все ЯП позволяют оперировать этими объектами. Кроме того, в большинстве ЯП есть и другие объекты предметной области, например, комплексные числа, векторы, функции, графы, списки и так далее.

Для нашего ЯП мы введём предметную область в виде множества натуральных чисел ω. Это некоторая абстракция, так как чисел бесконечно много, а память любого реального вычислительного устройства конечна. Но мы не будем рассматривать ограничения, присущие реальным ЯП.

Определение 3.2 (Константа)**.** *Константа в ЯП — это некоторое имя, которое представляет некоторый зафиксированный элемент предметной области.*

Мы будем считать, что в нашем ЯП есть константа 0, представляющая число 0.

Определение 3.3 (Операция)**.** *Операция в ЯП — это некоторое (может быть частичное) отображение предметной области в себя, которое имеет фиксированное имя в ЯП.*

Практически во всех ЯП есть операции, соответствующие арифметическим функциям (сложение, вычитание, умножение, деление, сравнения). Если предметная область включает в себя текстовые строки, то как правило присутствуют конкатенация и лексикографические сравнения.

В нашем ЯП мы введем две операции

- succ — одноместная операция, которая будет означать прибавление единицы: $\mathsf{succ}(0) = 1$, $\mathsf{succ}(3) = 4$;
- $<$ — двухместная функция, принимающая значение 1, если первый аргумент меньше второго, и 0 в противном случае: $<(4, 6) = 1$, $<(10, 8) = 0$. Для простоты и удобства мы будем

обычно в дальнейшем использовать традиционную инфиксную запись: вместо $<(x, y)$ писать $x < y$.

Как мы увидим, все остальные операции можно определить через эти две, и можно написать любую программу, используя только их. Заметим, что обе операции succ и $<$ определены для любых элементов нашей предметной области. Поэтому случай, когда результат операции неопределен (например, в случае деления на ноль) мы пока рассматривать не будем.

При использовании инфиксной формы для операций возникает неоднозначность интерпретации при многократном их использовании. Например, строку вида $1 < 0 < 1$ можно интерпретировать двумя способами: $(1 < 0) < 1$ или $1 < (0 < 1)$. Результатом в первом случае будет 1, а во втором — 0. В подобных случаях мы всегда будем использовать скобки, указывая, какая из операций должна выполняться первой.

Замечание 3.1. В реальных языках программирования для каждой операции однозначно определены приоритет и ассоциативность. Приоритет — натуральное число, при возникновении неоднозначности в первую очередь выполняются операция, имеющая наибольший приоритет. Ассоциативность определяет порядок, в котором выполняются операции, имеющие одинаковый приоритет: левоассоциативные выполняются слева направо, а правоассоциативные — справа налево. Например, в языках C, C++ и им подобных операция + имеет приоритет больше, чем $<$, и обе они являются левоассоциативными. Следовательно, запись $1+2 < 2+3+4 < 7$ интерпретируется так: $((1 + 2) < ((2 + 3) + 4)) < 7$.

Кроме того, нам потребуется какой-то способ для запоминания промежуточных результатов вычисления. Для этого используется следующий элемент языка программирования — переменные. Каждая переменная имеет некоторое имя. Мы будем считать, что всевозможные имена переменных составляют некоторое счётное множество $\mathfrak{V}$. Также будем для удобства полагать, что

1) разные переменные имеют разные имена;

2) разные имена обозначают разные переменные.

Эти ограничения введены только для удобства изложения, в реальных языках программирования они могут не выполняться. Обычно мы будем в качестве имен переменных использовать буквы конца латинского алфавита, возможно, с индексами: x, y_1, v_u и т. д. Подчеркнем, что имя переменной является единым словом и никак не интерпретируется, например, переменные v, u и v_u никак между собой не связаны.

Мы будем считать, что имена переменных не совпадают ни с именами констант, ни со специальными словами (algorithm, if и т. д.), которые нам встретятся в дальнейшем.

В семантике реальных языков программирования переменной обычно считают некоторую область памяти, связанную с её именем. Мы же будем отождествлять переменную и её имя.

Часто нам будут встречаться последовательности однотипных переменных (а так же и других объектов), например, $x_1, x_2, \ldots, x_n$ или $y_1, y_2, \ldots, y_m$. Для краткости такие последовательности будем обозначать как $\bar{x}$ или $\bar{y}$ соответственно, предполагая, что количество таких объектов заранее определено.

Знаком $\dot{=}$ мы будем обозначать графическое совпадение строк.

Пример 3.1. *Пусть*

$$m \dot{=} abc = def$$

$$n \dot{=} xyz \circ abd$$

Тогда

$$mn \dot{=} abc = defxyz \circ abd$$

Если a, b, c — строки, $a \dot{=} bc$, то b называют префиксом строки a, c — суффиксом строки a. Если префикс (суффикс) строки a отличается от самой a, то такой префикс (суффикс) называют собственным.

Чтобы развить синтаксис нашего языка программирования введём понятие выражения.

Определение 3.4 (Выражение)**.** *Определим, что такое выражение, по индукции.*

1) *Если x — имя переменной, то x — выражение.*
2) *Если c — константа, то c — выражение.*
3) *Если $e_1, \ldots, e_n$ — выражения, а o — это n-местная операция, то $o(e_1, \ldots, e_n)$ — выражение.*

Других выражений нет.

Рассмотрим пример.

Пример 3.2. 0, x, $\mathsf{succ}(0)$, $\mathsf{succ}(x) < \mathsf{succ}(\mathsf{succ}(y))$ — *выражения.*

Минимальным элементом языка программирования, который означает выполнение каких-либо действий, является оператор. Почти в любом языке программирования имеется оператор, который позволяет сохранить в переменной элемент предметной области. Для сохранения промежуточных результатов в нашем языка программирования будет служить оператор присваивания.

Определение 3.5 (Оператор присваивания)**.** *Оператором присваивания называется строка вида*

$$x = e;$$

где x — имя переменной (левая часть оператора присваивания), e — некоторое выражение (правая часть оператора присваивания).

Оператор присваивания в том или ином виде присутствует почти во всех языках программирования. Этот оператор будет служить

нам «кирпичиком», из которого мы будем строить все наши более сложные программы.

Пример 3.3. *Примеры операторов присваивания:*

$$x = \mathsf{succ}(x);$$
$$y = \mathsf{succ}(\mathsf{succ}(0));$$
$$z = x < y;$$

Теперь мы можем определить, что такое структурированная программа.

Определение 3.6 (Структурированная программа)**.** *Как и в случае с выражениями мы будем давать определение структурированной программы по индукции.*

1) *Точка с запятой «;» является программой (пустой оператор).*

2) *Оператор присваивания $x = e$; является программой.*

3) *Следование. Если Π_1 и Π_2 — программы, то $\Pi_1\Pi_2$ — программа.*

4) *Неполное ветвление. Если Π_1 — программа, а e — выражение, то*

$$\begin{array}{l}\mathsf{if}\ e\ \mathsf{then}\\ \quad \Pi_1\\ \mathsf{end};\end{array}$$

является программой.

5) *Полное ветвление. Если* Π_1 *и* Π_2 *— программы, а* e *— выражение, то*

$$\begin{array}{l}\mathsf{if}\ e\ \mathsf{then}\\ \quad \Pi_1\\ \mathsf{else}\\ \quad \Pi_2\\ \mathsf{end};\end{array}$$

является программой.

6) *Цикл. Если* Π_1 *— программа, а* e *— выражение, то*

$$\begin{array}{l}\mathsf{while}\ e\ \mathsf{do}\\ \quad \Pi_1\\ \mathsf{end};\end{array}$$

является программой.

В определении некоторые операторы были записаны в несколько строк исключительно для удобства восприятия. Мы не будем отличать запись в одну строку и в несколько строк:

$$x = y; y = z; \quad \text{и} \quad \begin{array}{l} x = y;\\ y = z;\end{array}$$

будут означать одно и то же.

Рассмотрим несколько примеров.

Пример 3.4. *Операторы присваивания*

$$x = 0; \qquad \text{и} \qquad y = \mathsf{succ}(x);$$

являются программами. Поэтому их соединение:

$$\begin{array}{l} x = 0;\\ y = \mathsf{succ}(x);\end{array}$$

тоже является программой. Также программами являются и следующие строки Π_1, Π_2 *и* Π_3, *которые получены с помощью пунктов 3, 4 и 5*

соответственно:

$$\left.\begin{array}{l}\textsf{if } 0 < x \textsf{ then}\\ \quad x = 0;\\ \quad y = \textsf{succ}(x);\\ \textsf{end;}\end{array}\right\}\Pi_1 \qquad \left.\begin{array}{l}\textsf{if } x < \textsf{succ}(y) \textsf{ then}\\ \quad x = \textsf{succ}(y);\\ \textsf{else}\\ \quad y = 0;\\ \textsf{end;}\end{array}\right\}\Pi_2 \qquad \left.\begin{array}{l}\textsf{while succ}(x) \textsf{ do}\\ \quad z = \textsf{succ}(z);\\ \quad y = \textsf{succ}(x);\\ \textsf{end;}\end{array}\right\}\Pi_3$$

Определение 3.7 (Множество переменных)**.** *Определим множество* $\mathrm{Var}(\alpha)$ *— множество переменных объекта* α*. Для выражений:*

1) $\mathrm{Var}(x) = \{x\}$, *если* x *— переменная;*
2) $\mathrm{Var}(c) = \varnothing$, *если* c *— константа;*
3) $\mathrm{Var}(o(e_1, \ldots, e_n)) = \mathrm{Var}(e_1) \cup \cdots \cup \mathrm{Var}(e_n)$, *если* o *— n-местная операция,* $e_1, \ldots, e_n$ *— выражения.*

Для программы Π *кроме* $\mathrm{Var}(\Pi)$ *мы определим* $\mathrm{LVar}(\Pi)$ *— множество изменяемых переменных:*

1) *если* $\Pi \Rightarrow$;. $\mathrm{Var}(\Pi) = \mathrm{LVar}(\Pi) = \varnothing$.
2) *если* $\Pi \Rightarrow x = e$;. $\mathrm{Var}(\Pi) = \{x\} \cup \mathrm{Var}(e)$, $\mathrm{LVar}(\Pi) = \{x\}$.
3) *если* $\Pi \Rightarrow \Pi_1\Pi_2$

$$\mathrm{Var}(\Pi) = \mathrm{Var}(\Pi_1) \cup \mathrm{Var}(\Pi_2),$$

$$\mathrm{LVar}(\Pi) = \mathrm{LVar}(\Pi_1) \cup \mathrm{LVar}(\Pi_2).$$

4) *если* $\Pi \Rightarrow$ $\left\{\begin{array}{l}\textsf{if } e \textsf{ then}\\ \quad \Pi_1\\ \textsf{end;}\end{array}\right.$

$$\mathrm{Var}(\Pi) = \mathrm{Var}(e) \cup \mathrm{Var}(\Pi_1), \quad \mathrm{LVar}(\Pi) = \mathrm{LVar}(\Pi_1).$$

5) *если* $\Pi \mathrel{\overset{\circ}{\circ}} \begin{cases} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end}; \end{cases}$

$$\mathrm{Var}(\Pi) = \mathrm{Var}(e) \cup \mathrm{Var}(\Pi_1) \cup \mathrm{Var}(\Pi_2),$$

$$\mathrm{LVar}(\Pi) = \mathrm{LVar}(\Pi_1) \cup \mathrm{LVar}(\Pi_2).$$

6) *если* $\Pi \mathrel{\overset{\circ}{\circ}} \begin{cases} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$

$$\mathrm{Var}(\Pi) = \mathrm{Var}(e) \cup \mathrm{Var}(\Pi_1), \quad \mathrm{LVar}(\Pi) = \mathrm{LVar}(\Pi_1).$$

Таким образом, $\mathrm{Var}(a)$ — это множество переменных, которые встречаются в a, $\mathrm{LVar}(\Pi)$ — множество переменных программы Π, которые встречаются в левой части операторов присваивания.

Пример 3.5. *Для программ из примера 3.4 на стр. 45:*

1) $\mathrm{Var}(\Pi) = \{x, y\}$, $\mathrm{LVar}(\Pi) = \{x, y\}$.
2) $\mathrm{Var}(\Pi) = \{x, y\}$, $\mathrm{LVar}(\Pi) = \{x, y\}$.
3) $\mathrm{Var}(\Pi) = \{x, y, z\}$, $\mathrm{LVar}(\Pi) = \{y, z\}$.

Некоторые переменные программы используются для того, чтобы передать входные данные, другие — чтобы программа могла сообщить результат. Чтобы выделить эти переменные, мы будем записывать наши алгоритмы в специальном виде.

```
algorithm Max;
arguments x, y;
returns z;
    if x < y then
        z = y;
    else
        z = x;
    end;
end;
```

Рис. 3.1: Наибольшее из двух чисел.

Определение 3.8 (Алгоритм)**.** *Будем называть* алгоритмом *строку следующего вида:*

algorithm *Имя алгоритма*;
arguments *Список входных переменных*;
returns *Выражение-результат*;
Тело алгоритма
end;

После algorithm *указывается имя, которое мы даём алгоритму (*имя алгоритма*), после* arguments *— список разделенных запятыми переменных, в которых задаются начальные условия (*входные переменные *или* параметры*). Каждая переменная может появляться в этом списке не более одного раза. Тело алгоритма — программа, которую мы будем обозначать* Π_A*, где* A *— имя алгоритма. После слова* returns *указывается выражение, определяющее результат.*

Пример 3.6. *Рассмотрим алгоритм на рис. 3.1. В данном случае, имя*

алгоритма — Max, входные переменные — x и y. Тело алгоритма:

```
if x < y then
    z = y;
else
    z = x;
end;
```

Выражение-результат — z.

Лемма 3.1 (О сбалансированности)**.** *В любой программе* П *сумма количеств слов* if *и* while *равна количеству слов* end*. В любом префиксе программы сумма количества слов* if *и* while *не меньше количества слов* end*.*

Доказательство. Доказательство — индукция по построению программы. Обозначим сумму количеств if и while в программе П с помощью $I(\Pi)$, количество end — $E(\Pi)$.

Базис индукции. Если П — оператор присваивания или пустой оператор, то $I(\Pi) = E(\Pi) = 0$.

Шаг индукции.

1) Если $\Pi \circ\!\!- \Pi_1\Pi_2$, Π_1 и Π_2 — программы, то $I(\Pi) = I(\Pi_1)+I(\Pi_2)$, $E(\Pi) = E(\Pi_1) + E(\Pi_2)$. По индукционному предположению, $I(\Pi_1) = E(\Pi_1)$ и $I(\Pi_2) = E(\Pi_2)$. Следовательно, $I(\Pi) = E(\Pi)$.

   Пусть теперь Π_3 — префикс П. Тогда или Π_3 — начальный фрагмент Π_1 и для него утверждение выполняется по индукционному предположению, или $\Pi_3 \circ\!\!- \Pi_1\Pi_4$, где Π_4 — префикс Π_2. По индукционному предположению $I(\Pi_1) = E(\Pi_1)$ и $I(\Pi_4) \geqslant E(\Pi_4)$. Следовательно,

$$I(\Pi_3) = I(\Pi_1) + I(\Pi_4) \geqslant E(\Pi_1) + E(\Pi_4) = E(\Pi_3).$$

2) Пусть

$$\Pi \circ\!\!- \left\{ \begin{array}{l} \textsf{if } e \textsf{ then} \\ \quad \Pi_1 \\ \textsf{end;} \end{array} \right.$$

Π_1 — программа. Тогда

$$I(\Pi) = 1 + I(\Pi_1) = 1 + E(\Pi_1) = E(\Pi).$$

Если Π_3 — префикс Π, то или Π_3 не содержит ничего из Π_1, тогда $I(\Pi_3) = 1 > 0 = E(\Pi_3)$, или

$$\Pi_3 \circ\!\!- \textsf{ if } e \textsf{ then } \Pi_4$$

где Π_4 — префикс Π_1. Тогда

$$I(\Pi_3) = 1 + I(\Pi_4) > E(\Pi_4) = E(\Pi_3).$$

3) Для полного ветвления и цикла доказательство аналогично. □

***Задача 3.1.** Обоснуйте индукционный шаг для полного ветвления и цикла.

Задача 3.2. Верно ли, что в любом суффиксе любой программы количество слов end не меньше, чем суммарное количество if и while?

Лемма 3.2 (О суффиксе программы)**.** *Если* $\Pi \circ\!\!- \Pi_1\Pi_2$, Π *и* Π_1 — *программы,* $\Pi_1 \neq \Pi$, *то* Π_2 — *программа.*

Доказательство. Индукция по построению Π.

Базис индукции. Пусть Π — оператор присваивания или пустой оператор. Любая программа оканчивается точкой с запятой. Внутри Π точка с запятой не встречается. Π_1 — тоже программа, следовательно, она тоже оканчивается точкой с запятой. Но это означает, что $\Pi_1 \circ\!\!- \Pi$. Противоречие. Следовательно, такой случай невозможен.

Шаг индукции.

1) Пусть

$$\Pi \circ\!\!- \begin{cases} \textsf{if } e \textsf{ then} \\ \quad \Pi_3 \\ \textsf{else} \\ \quad \Pi_4 \\ \textsf{end;} \end{cases}$$

Так как Π_1 оканчивается точкой с запятой, то граница между Π_1 и Π_2 проходит или внутри Π_3, или внутри Π_4. Рассмотрим первый случай:

$$\Pi_1 \backsimeq \mathsf{if}\ e\ \mathsf{then}\ \Pi_3'$$
$$\Pi_2 \backsimeq \Pi_3''\ \mathsf{else}\ \Pi_4\ \mathsf{end};$$

Здесь $\Pi_3 \backsimeq \Pi_3'\Pi_3''$. Так как Π_3' — префикс программы Π_3, то в нём сумма количеств if и while не меньше количества end. Но тогда в Π_1 сумма количеств if и while больше количества end, что противоречит предыдущей лемме. Следовательно, такая ситуация невозможна.

Пусть теперь

$$\Pi_1 \backsimeq \mathsf{if}\ e\ \mathsf{then}\ \Pi_3\ \mathsf{else}\ \Pi_4'$$
$$\Pi_2 \backsimeq \Pi_4''\ \mathsf{end};$$

где $\Pi_4 \backsimeq \Pi_4'\Pi_4''$. Так как Π_4' — префикс программы Π_4, то в нём сумма количеств if и while не меньше количества end. Так как Π_3 — программа, то для неё эти величины равны. Но тогда в Π_1 сумма количеств if и while снова больше количества end. Итак, мы доказали, что случай, когда Π — полное ветвление, невозможен.

2) Для неполного ветвления и цикла доказательства аналогичны.

3) Осталось рассмотреть только следование:

$$\Pi \backsimeq \Pi_3\Pi_4$$

Если $\Pi_1 \backsimeq \Pi_3$, то, очевидно, $\Pi_2 \backsimeq \Pi_4$ и Π_2, таким образом, является программой.

Иначе, граница между Π_1 и Π_2 проходит внутри Π_3 или Π_4. Первый случай:

$$\Pi \backsimeq \Pi_1\Pi_5\Pi_4$$

где $\Pi_1\Pi_5 \dot{\circ} \Pi_3$. Π_3 и Π_1 — программы и $\Pi_1 \not\circ \Pi_3$. По индукционному предположению Π_5 — программа. Но тогда $\Pi_2 \dot{\circ} \Pi_5\Pi_4$ — программа.

Второй случай:

$$\Pi \dot{\circ} \Pi_3\Pi_5\Pi_2$$

где $\Pi_5\Pi_2 \dot{\circ} \Pi_4$. $\Pi_1 \dot{\circ} \Pi_3\Pi_5$, Π_3 и Π_1 — программы и $\Pi_1 \not\circ \Pi_3$. По индукционному предположению Π_5 — программа. $\Pi_5\Pi_2 \dot{\circ} \Pi_4$, $\Pi_5 \not\circ \Pi_4$. По индукционному предположению Π_2 — программа. □

*__Задача 3.3.__ Обоснуйте индукционный шаг для неполного ветвления и цикла.

Следствие 3.3 (О двойном представлении программы)**.** *Пусть* Π, Π_1, Π_2, Π_3, Π_4 — *программы,* $\Pi \dot{\circ} \Pi_1\Pi_2$, $\Pi \dot{\circ} \Pi_3\Pi_4$ *и* $\Pi_1 \not\circ \Pi_3$. *Тогда существует программа* Π_5 *такая, что* $\Pi \dot{\circ} \Pi_3\Pi_5\Pi_2$ *или* $\Pi \dot{\circ} \Pi_1\Pi_5\Pi_4$.
*__Задача 3.4.__ Докажите следствие.

§ 3.2. Семантика

До сих пор мы рассматривали только то, каким образом записывать программы, но не говорили о том, что эта запись означает. Теперь рассмотрим этот вопрос.

Семантика программ будет определяться через понятие состояния.

Определение 3.9 (Состояние)**.** *Пусть* V — *некоторое множество переменных. Состояние множества переменных* V — *это частичное отображение* σ *множества переменных в предметную область, для которого* $V \subseteq \operatorname{dom}\sigma$. *Значение переменной* $x \in V$ *на состоянии* σ *множества* V — *это* σx.

Пример 3.7. *Например, если множество переменных*

$$\mathfrak{V} \supseteq \{x, y, z\},$$

то следующие отображения будут состояниями:

$$\sigma_1 = \{(x, 1), (y, 2), (z, 0)\},$$
$$\sigma_2 = \{(x, 100), (y, 50), (z, 75), (u, 34)\}.$$

Для обозначения состояний мы будем использовать буквы σ, τ, ρ, π, возможно, с индексами.

Сначала определим семантику выражений.

Определение 3.10 (Конфигурация выражения)**.** *Конфигурацией выражения e называется состояние множества переменных* $\mathrm{Var}(e)$*. Иначе говоря, всем переменным, которые встречаются в выражении e, состояние σ должно приписывать некоторые значения.*

Задав значения всех переменных, можно определить значение выражения.

Определение 3.11 (Значение выражения)**.** *Пусть e — выражение, σ — его конфигурация. Значением выражения e на σ (обозначается $\sigma(e)$) называется элемент предметной области, который определяется следующим образом.*

1) *Если $e \twoheadrightarrow x$, где x — переменная, то $\sigma(e) = \sigma x$.*

2) *Если $e \twoheadrightarrow c$, где c — константа, то $\sigma(e) = c$.*

3) *Если $e \twoheadrightarrow o(e_1, \ldots, e_n)$, где o — n-местная операция, а $e_1, \ldots, e_n$ — выражения, и $\sigma(e_1) = v_1, \ldots, \sigma(e_n) = v_n$, то $\sigma(e) = o(v_1, \ldots, v_n)$. Таким образом*

$$\sigma(o(e_1, \ldots, e_n)) = o(\sigma(e_1), \ldots, \sigma(e_n)).$$

В частности,

$$\sigma(\mathsf{succ}(e)) = \sigma(e) + 1;$$

$$\sigma(e_1 < e_2) = \begin{cases} 1, & \text{если } \sigma(e_1) < \sigma(e_2), \\ 0, & \text{иначе.} \end{cases}$$

Рассмотрим пример.

Пример 3.8. *Будем рассматривать состояния σ_1 и σ_2 из примера 3.7 на стр. 52.*

$$\sigma_1(0) = 0,$$

$$\sigma_1(x) = 1,$$

$$\sigma_2(\mathsf{succ}(y)) = \sigma_2(y) + 1 = 51,$$

$$\sigma_2(\mathsf{succ}(\mathsf{succ}(x))) = \sigma_2(\mathsf{succ}(x)) + 1 = \sigma_2(x) + 1 + 1 = 102.$$

Задача 3.5. Найдите значения выражений $\mathsf{succ}(z)$ и $\mathsf{succ}(\mathsf{succ}(z))$ на состояниях из примера 3.7 на стр. 52.

Во многих языках программирование ненулевое значение выражения расценивается как истинное, а нулевое — как ложное.

Пример 3.9. *Выражение $(x < y) < \mathsf{succ}(0)$ принимает значение 1, когда значение выражения $x < y$ меньше единицы. Поскольку значением $x < y$ может быть только 0 или 1, то единственно возможный вариант для него — 0. Но нулю значение $x < y$ равняется тогда и только тогда, когда $\sigma x \geqslant \sigma y$. Таким образом, получаем*

$$\sigma((x < y) < \mathsf{succ}(0)) = \begin{cases} 1, & \text{если } \sigma x \geqslant \sigma y, \\ 0, & \text{иначе.} \end{cases}$$

Определение 3.12. *Будем в дальнейшем использовать обозначение вида $e_1 \geqslant e_2$ (или $e_2 \leqslant e_1$) как сокращение для $(e_1 < e_2) < \mathsf{succ}(0)$.*

Пример 3.10. *Рассмотрим выражение $(x < y) < (x \leqslant y)$. Поскольку результатом операций $<$ и $\leqslant$ может быть только 0 или 1, то значение*

выражения на состоянии σ *будет равно* 1 *тогда и только тогда, когда* $\sigma(x < y) = 0$ *и* $\sigma(x \leqslant y) = 1$. *Но первое означает, что* $\sigma x \geqslant \sigma y$, *а второе —* $\sigma x \leqslant \sigma y$, *то есть* $\sigma x = \sigma y$. *Таким образом*

$$\sigma((x < y) < (x \leqslant y)) = \begin{cases} 1, & \text{если } \sigma x = \sigma y, \\ 0, & \text{иначе.} \end{cases}$$

Определение 3.13. *Строку вида* $e_1 == e_2$ *мы будем рассматривать как сокращение для* $(e_1 < e_2) < (e_1 \leqslant e_2)$.

Задача 3.6. Напишите выражения, которые принимали бы значения 0 или 1:

1) $!e_1$, принимает значение 1, если и только если $\sigma(e_1) = 0$;
2) $e_1 \mathrel{!}= e_2$, принимает значение 1, если и только если $\sigma(e_1) \neq \sigma(e_2)$.

Определение 3.14 (Конфигурация)**.** *Для структурированной программы* П *ее к о н ф и г у р а ц и е й называется состояние множества переменных* Var(П).

Задача 3.7. Определите, будут ли состояния из примера 3.7 на стр. 52 конфигурациями программ из примера 3.4 на стр. 45.

Теперь мы готовы к тому, чтобы определить семантику нашего языка программирования. Пусть Σ_Π — множество всевозможных конфигураций программы П.

Для каждой программы П мы определим некоторую одноместную частичную функцию, которая действует из Σ_Π в Σ_Π. Эту функцию мы и будем называть семантикой программы П. Мы будем обозначать эту функцию так же как и программу.

Определение 3.15 (Семантика программы)**.** *С е м а н т и к а п р о г р а м м ы* П *определяется индукцией по построению программы.*

1) *Для пустого оператора:* $;(\sigma) = \sigma$.

2) *Пусть*

$$\Pi \circ x = e;$$

где x — переменная, e — выражение. Пусть σ — конфигурация Π. Тогда $\Pi(\sigma) = \tau$, где τ — конфигурация, которая определяется следующим образом:

$$\tau = \{(y, v) \in \sigma : y \neq x\} \cup \{(x, \sigma(e))\}.$$

То есть, для всех переменных y, отличных от x, $\tau y = \sigma y$, и $\tau x = \sigma(e)$.

3) *Пусть $\Pi \circ \Pi_1\Pi_2$, где Π_1 и Π_2 — программы. Тогда для всякой конфигурации σ определим $\Pi(\sigma) = \Pi_2(\Pi_1(\sigma))$. В этом определении кроется возможная неоднозначность, о которой мы скажем чуть позже.*

4) *Пусть*

$$\Pi \circ \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array} \right.$$

Тогда

   а) *если $\sigma(e) \neq 0$, то $\Pi(\sigma) = \Pi_1(\sigma)$;*

   б) *если $\sigma(e) = 0$, то $\Pi(\sigma) = \sigma$.*

5) *Пусть*

$$\Pi \circ \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end}; \end{array} \right.$$

Тогда

а) *если* $\sigma(e) \neq 0$, *то* $\Pi(\sigma) = \Pi_1(\sigma)$;

б) *если* $\sigma(e) = 0$, *то* $\Pi(\sigma) = \Pi_2(\sigma)$.

6) *Для цикла семантика определяется несколько более сложным образом. Пусть*

$$\Pi \twoheadrightarrow \begin{cases} \textsf{while } e \textsf{ do} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

$\Pi(\sigma)$ *определено, если существует натуральное число* n *и последовательность конфигураций* $(\sigma^i)_{i=0}^n$ *такие, что*

а) $\sigma^0 = \sigma$;

б) $\Pi_1(\sigma^i)$ *определено для* $i = 0, \ldots, n-1$ *и* $\sigma^{i+1} = \Pi_1(\sigma^i)$;

в) $\sigma^i(e) \neq 0$ *тогда и только тогда, когда* $i < n$, *то есть* $\sigma^i(e) \neq 0$, *если* $i < n$, *и* $\sigma^n(e) = 0$.

В этом случае $\Pi(\sigma) = \sigma^n$. *Если же такого числа* n *не существует, то* $\Pi(\sigma)$ *не определено.*

Как видно из определения для цикла результат может оказаться неопределён, даже если для его тела результат всегда определён. Следовательно, если результат программы неопределён, то это в конечном итоге всегда связано с некоторым циклом.

Определение 3.16 (Зацикливание программы)**.** *Если* $\Pi(\sigma)$ *не определено для программы* Π *и конфигурации* σ, *то говорим, что программа* Π *на конфигурации* σ *зацикливается. В противном случае говорим, что программа останавливается.*

Следствие 3.4 (Условие зацикливания)**.** *Пусть*

$$\Pi \Leftarrow \begin{cases} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

Тогда $\Pi(\sigma)$ *не определено тогда и только тогда, когда в последовательности* $(\sigma^i)_i$*, построенной по правилам* а) *и* б)*, для каждого* i *выполняется* $\sigma^i(e) \neq 0$.

Задача 3.8. Докажите следствие.

Рассмотрим несколько примеров.

Пример 3.11. *Пусть* $\sigma = \{(x,1),(y,2),(z,3)\}$.

1) *Рассмотрим программу*

$$x = z;$$

Тогда

$$(x = z;)(\sigma) = \{(x,3),(y,2),(z,3)\}.$$

2) *Для программы*

$$y = \mathsf{succ}(0);$$

получаем

$$(y = \mathsf{succ}(0);)(\sigma) = \{(x,1),(y,1),(z,3)\}.$$

3) *Для следования программ из пп. 1 и 2:*

$$\Pi \Leftarrow x = z;\ y = \mathsf{succ}(0);$$

тогда

$$\Pi(\sigma) = \{(x,3),(y,1),(z,3)\},$$

потому что

$$(x = z;)(\sigma) = \{(x,3),(y,2),(z,3)\}$$

$$(y = \mathsf{succ}(0);)(\{(x,3),(y,2),(z,3)\}) = \{(x,3),(y,1),(z,3)\}.$$

4) *Для неполного ветвления*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{if}\ x < y\ \mathsf{then} \\ \quad x = z; \\ \mathsf{end}; \end{array} \right.$$

Так как $\sigma(x < y) \neq 0$, *то*

$$\Pi(\sigma) = (x = z;)(\sigma) = \{(x, 3), (y, 2), (z, 3)\}.$$

5)

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{if}\ z < y\ \mathsf{then} \\ \quad x = z; \\ \mathsf{end}; \end{array} \right.$$

Тогда $\sigma(z < y) = 0$. *Поэтому* $\Pi(\sigma) = \sigma$.

6) *Для полного ветвления*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{if}\ z < y\ \mathsf{then} \\ \quad x = z; \\ \mathsf{else} \\ \quad y = \mathsf{succ}(0); \\ \mathsf{end}; \end{array} \right.$$

Так как $\sigma(z < y) = 0$, *то*

$$\Pi(\sigma) = (y = \mathsf{succ}(0);)(\sigma) = \{(x, 1), (y, 1), (z, 3)\}.$$

7) *Пусть*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{while}\ x < z\ \mathsf{do} \\ \quad x = \mathsf{succ}(x); \\ \mathsf{end}; \end{array} \right.$$

Рассмотрим $\Pi(\sigma)$. *Построим последовательность:*

$$\begin{aligned} \sigma^0 &= \{(x, 1), (y, 2), (z, 3)\}; \\ \sigma^1 &= \{(x, 2), (y, 2), (z, 3)\}; \\ \sigma^2 &= \{(x, 3), (y, 2), (z, 3)\}. \end{aligned}$$

Очевидно, что последовательность точно соответствует определению семантики для цикла. Далее, $\sigma^0(x < z) \neq 0$, $\sigma^1(x < z) \neq 0$, $\sigma^2(x < z) = 0$. *Следовательно,* $\Pi(\sigma) = \sigma^2$.

8) *Рассмотрим другую программу:*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{if}\ x < z\ \mathsf{then} \\ \quad z = \mathsf{succ}(z); \\ \mathsf{end}; \end{array} \right.$$

Очевидно, что для конфигурации σ последовательность $(\sigma^i)_i$ будет состоять из элементов

$$\sigma^i = \{(x, 1), (y, 2), (z, 3 + i)\}.$$

Следовательно, для каждого i имеет место $\sigma^i(x < z) \neq 0$. Из этого следует по определению, что $\Pi(\sigma)$ не определено.

Задача 3.9. Определите $\Pi(\sigma)$ для программ из примера 3.4 на стр. 45 и конфигураций из примера 3.7 на стр. 52.

Сформулируем ещё несколько утверждений о зацикливании и остановке программ, которые непосредственно следуют из определения семантики структурированных программ.

Следствие 3.5. *Если программа Π не содержит циклов, то $\Pi(\sigma)$ определено для любой конфигурации σ.*

Задача 3.10. Докажите следствие.

Следствие 3.6 (Условие зацикливания)**.** *Пусть*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array} \right.$$

Если для всякой конфигурации σ из $\sigma(e) \neq 0$ следует, что $\Pi_1(\sigma)(e) \neq 0$, то $\Pi(\sigma)$ определено тогда и только тогда, когда $\sigma(e) = 0$.

Задача 3.11. Докажите следствие.

Лемма 3.7 (Условие зацикливания)**.** *Если для цикла*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array} \right.$$

на σ последовательность конфигураций (σ^i) содержит два одинаковых элемента σ^k и σ^l $(k < l)$ и $\sigma^i(e) \neq 0$ для всех $i \leqslant l$, то $\Pi(\sigma)$ не определено.

Доказательство. Пусть $k < l$. Тогда, по определению, получаем, что

$$\sigma^l = \Pi_1(\sigma^{l-1}) = \Pi_1(\Pi_1(\sigma^{l-2})) = \cdots = (\Pi_1)^{l-k}(\sigma^k).$$

Кроме того, для всякого $i \in [k; l)$ выполняется $\sigma^i(e) \neq 0$. Рассмотрим произвольное $j \geqslant k$. Число j может быть представлено в виде

$$j = k + n(l - k) + m,$$

где $m < l - k$. Рассмотрим $(\Pi_1)^{n(l-k)}(\sigma^k)$:

$$\begin{aligned}(\Pi_1)^{n(l-k)}(\sigma^k) &= (\Pi_1)^{(n-1)(l-k)}((\Pi_1)^{l-k}(\sigma^k)) = \\ &= (\Pi_1)^{(n-1)(l-k)}(\sigma^l) = (\Pi_1)^{(n-1)(l-k)}(\sigma^k) = \cdots = \sigma^k.\end{aligned}$$

Тогда

$$\begin{aligned}\sigma^j = (\Pi_1)^{n(l-k)+m}(\sigma^k) = (\Pi_1)^m(\Pi_1^{n(l-k)}(\sigma^k)) = \\ = (\Pi_1)^m(\sigma^k) = \sigma^{k+m}.\end{aligned}$$

Так как $k + m \in [k, l)$, то $\sigma^j(e) \neq 0$. Итак, мы выяснили, что для всякого j имеет место $\sigma^j(e) \neq 0$. Это означает, что $\Pi(\sigma)$ не определено. □

Теперь рассмотрим одну тонкость в определении семантики, о которой мы упоминали, когда определяли семантику следования. Нетрудно заметить, что для пустого оператора, присваивания, ветвлений и цикла семантика определяется однозначным образом. В случае же следования может возникнуть, например, следующая ситуация. Рассмотрим программу:

$$\Pi \circ x = z;\ y = x;\ z = y;$$

Эта программа может быть представлена в виде следования двумя различными способами:

$$\underbrace{x = z;\ y = x;}_{\Pi_1}\ \underbrace{z = y;}_{\Pi_2} \qquad \underbrace{x = z;}_{\Pi_3}\ \underbrace{y = x;\ z = y;}_{\Pi_4}$$

То есть затруднение состоит в том, что одна и та же программа Π может быть представлена в виде следования неоднозначно:

$$\Pi \circ\!\!- \Pi_1\Pi_2 \circ\!\!- \Pi_3\Pi_4$$

причём Π_1 может быть не равно Π_3. Где гарантия, что определяя семантику первым и вторым способом мы получим один и тот же результат? Мы докажем, что противоречия не возникает, пользуясь тем или иным представлением мы получим одно и то же определение семантики.

Теорема 3.8 (Корректность семантики следования)**.** *Если программа Π может быть представлена как следование различными способами: $\Pi \circ\!\!- \Pi_1\Pi_2 \circ\!\!- \Pi_3\Pi_4$, то семантика, которая получается из этих представлений, будет одной и той же.*

Доказательство. Итак, пусть программа Π имеет два таких представления, то есть она может быть разбита на две части двумя различными способами:

$$\Pi \circ\!\!- \Pi_1\Pi_2 \circ\!\!- \Pi_3\Pi_4$$

и $\Pi_1 \not\equiv \Pi_3$. Согласно следствию 3.3 на стр. 52 существует программа Π_5 такая, что $\Pi \circ\!\!- \Pi_3\Pi_5\Pi_2$ или $\Pi \circ\!\!- \Pi_1\Pi_5\Pi_4$.

Рассмотрим первый случай: $\Pi \circ\!\!- \Pi_3\Pi_5\Pi_2$, тогда будет выполнено $\Pi_1 = \Pi_3\Pi_5$ и $\Pi_4 = \Pi_5\Pi_2$. Возьмём произвольную конфигурацию σ. По определению семантики следования, получаем:

$$\Pi(\sigma) = (\Pi_1\Pi_2)(\sigma) = \Pi_2(\Pi_1(\sigma)) = \Pi_2((\Pi_3\Pi_5)(\sigma)) = \Pi_2(\Pi_5(\Pi_3(\sigma))).$$

С другой стороны:

$$\Pi(\sigma) = (\Pi_3\Pi_4)(\sigma) = \Pi_4(\Pi_3(\sigma)) = (\Pi_5\Pi_2)(\Pi_3(\sigma)) = \Pi_2(\Pi_5(\Pi_3(\sigma))).$$

Доказательство для второго случая аналогично.

Как видим, результат не зависит от того, каким именно способом мы разбиваем программу Π на части. Это означает, что определение семантики следования корректно. □

*__Задача 3.12.__ Докажите теорему, когда $\Pi \backsimeq \Pi_1\Pi_5\Pi_4$.

Теперь перейдём к семантике алгоритмов.

Определение 3.17 (Вызов алгоритма)**.** *Вызовом алгоритма A, имеющего n входных переменных $x_1, \ldots, x_n$, мы будем называть набор*

$$\delta = (A, v_1, \ldots, v_n),$$

где $v_1, \ldots, v_n \in \omega$ — элементы предметной области, аргументы вызова.

Начальная конфигурация вызова δ — это такая конфигурация σ программы Π_A, что $\sigma_\delta(x_i) = v_i$ для $i = 1, \ldots, n$. Заметим, что никаких ограничений на значения остальных переменных не налагается, поэтому начальных конфигураций может быть много.

Алгоритм называется детерминированным, если для любого вызова δ и любых его начальных конфигураций σ_δ и τ_δ либо $\Pi_A(\sigma_\delta)$ и $\Pi_A(\tau_\delta)$ одновременно неопределены, либо одновременно определены и при этом выполнено $\Pi_A(\sigma_\delta)(e) = \Pi_A(\tau_\delta)(e)$, где e — выражение-результат, указанное после слова returns *в алгоритме A. В дальнейшем мы будем рассматривать только детерминированные алгоритмы, не упоминая об этом специально.*

Будем говорить, что результат вызова δ (обозначается $\operatorname{Res}\delta$) определен и равен u, если для начальной конфигурации σ_δ выполнено $\Pi_A(\sigma_\delta)(e) = u$, где e — выражение-результат алгоритма

A. Во всех остальных случаях говорим, что результат вызова δ неопределён. Заметим, что поскольку мы договорились рассматривать только детерминированные алгоритмы, то такое определение корректно.

Пример 3.12. *Следующий алгоритм не является детерминированным:*

algorithm A;
arguments x;
returns y;
;
end;

Рассмотрим произвольный вызов (A, x_0). Начальными будут конфигурации вида $\sigma = \{(x, x_0), (y, y_0)\}$, где y_0 — произвольное натуральное число. Тогда очевидно, что $\Pi_A(\sigma) = \sigma y$ будет различным для различных начальных конфигураций вызова δ.

Задача 3.13. Напишите начальные конфигурации для вызовов алгоритмов на рис. 3.1 и 3.2: $(Max, 3, 6)$, $(Add, 2, 3)$. Найдите результат этих вызовов.

Определение 3.18 (Вычисление функции алгоритмом)**.** *Пусть дан алгоритм*

algorithm A;
arguments $x_1, \ldots, x_n$;
returns e;
Π
end;

Будем говорить, что алгоритм A вычисляет n-местную (частичную) функцию Φ_A, если для всякого вызова $\delta = (A, \bar{v})$ алгоритма A

1) $\operatorname{Res}\delta$ *определён тогда и только тогда, когда $\Phi_A(\bar{v})$ определено;*
2) *если $\operatorname{Res}\delta$ определён, то $\operatorname{Res}\delta = \Phi_A(\bar{v})$.*

```
algorithm Add;
arguments x, y;
returns u;
    u = x;                          ⎫
    v = 0;                          ⎪
    while v < y do                  ⎪
        u = succ(u);  ⎫ Π2          ⎬ Π1
        v = succ(v);  ⎭             ⎪
    end;                            ⎭
end;
```

Рис. 3.2: Сложение чисел.

(Частичная) функция, которая вычисляется некоторым алгоритмом, называется (алгоритмически) вычислимой.

Рассмотрим несколько примеров.

Пример 3.13. *Алгоритм Max на рис. 3.1 на стр. 48 вычисляет функцию $f(x, y) = \max\{x, y\}$. Например,*

$$\mathrm{Res}(Max, 1, 3) = 3;$$

$$\mathrm{Res}(Max, 5, 2) = 5.$$

Задача 3.14. Докажите, что результаты вызовов из предыдущего примера действительно равны указанным значениям.

Теперь напишем несколько алгоритмов для вычисления арифметических функций.

Пример 3.14. *Начнём со сложения, покажем, что оно вычисляется алгоритмом на рис. 3.2. По определению результата вызова имеем $\mathrm{Res}(Add, x_0, y_0) = \Pi_1(\sigma)(u)$, где σ — начальная конфигурация.*

Прежде всего отметим, что для любой конфигурации τ имеет место

$$\Pi_2(\tau)(u) = \tau u + 1, \qquad \Pi_2(\tau)(v) = \tau v + 1.$$

Рассмотрим произвольную начальную конфигурацию

$$\sigma = \{(x, x_0), (y, y_0), (u, u_0), (v, v_0)\}.$$

После выполнения первых двух операторов получим новую конфигурацию

$$\sigma_1 = \{(x, x_0), (y, y_0), (u, x_0), (v, 0)\}.$$

Начнем строить последовательность конфигураций $(\sigma^i)_i$ *для цикла:*

$$\sigma^0 = \sigma_1; \quad \sigma^{i+1} = \Pi_2(\sigma^i).$$

Очевидно, что

$$\sigma^i = \{(x, x_0), (y, y_0), (u, x_0 + i), (v, i)\}.$$

Далее, $\sigma^i(v < y) \neq 0$ *тогда и только тогда, когда* $i < y_0$*. Это означает, что последним элементом последовательности будет*

$$\sigma^{y_0} = \{(x, x_0), (y, y_0), (u, x_0 + y_0), (v, y_0)\},$$

так как это — первый элемент последовательности, для которого не выполняется $\sigma^i(v < y) \neq 0$*. Из этого следует, что* $\Pi_1(\sigma) = \sigma^{y_0}$ *и*

$$\Pi_1(\sigma)(u) = x_0 + y_0 = \sigma x + \sigma y.$$

Это и означает, что алгоритм Add вычисляет сложение натуральных чисел:

$$\mathrm{Res}(Add, x_0, y_0) = x_0 + y_0.$$

Пример 3.15. *Теперь рассмотрим алгоритм вычитания (рис. 3.3 на следующей странице). Напомним, что предметная область состоит только из натуральных чисел, поэтому мы напишем программу для вычисления функции* $\lfloor x - y \rfloor$*, где*

$$\lfloor x \rfloor = \frac{x + |x|}{2}.$$

То есть, $\lfloor x \rfloor = x$*, если* $x \geqslant 0$*, и* $\lfloor x \rfloor = 0$*, если* $x \leqslant 0$*. Таким образом, мы собираемся показать, что алгоритм Sub вычисляет такую функцию:*

$$\lfloor x - y \rfloor = \begin{cases} x - y, & \text{если } x \geqslant y, \\ 0 & \text{иначе.} \end{cases}$$

algorithm Sub;
arguments x, y;
returns u;
    $u = y$;
    $v = 0$;
    while $u < x$ do
        $u = \mathsf{succ}(u)$;
        $v = \mathsf{succ}(v)$;
    end;
end;

($u = y$; $v = 0$; while … do … end; — Π_1; $u = \mathsf{succ}(u)$; $v = \mathsf{succ}(v)$; — Π_2)

Рис. 3.3: Вычитание чисел.

Рассмотрим произвольную конфигурацию для вызова (Sub, x_0, y_0)*:*

$$\sigma = \{(x, x_0), (y, y_0), (u, u_0), (v, v_0)\}.$$

Перед началом цикла мы получим конфигурацию

$$\sigma_1 = \{(x, x_0), (y, y_0), (u, y_0), (v, 0)\}.$$

Построим последовательность конфигураций $(\sigma^i)_i$ *для цикла:*

$$\sigma^0 = \sigma_1; \quad \sigma^{i+1} = \Pi_2(\sigma^i).$$

Очевидно, что

$$\sigma^i = \{(x, x_0), (y, y_0), (u, y_0 + i), (v, i)\}.$$

Тогда получаем:

$$\sigma^i(u < x) \neq 0 \iff y_0 + i < x_0.$$

Если $x_0 \leqslant y_0$*, то последним элементом последовательности будет* σ^0*, так как для него уже будет* $\sigma^0(u < x) = 0$*.*

Если же $x_0 \geqslant y_0$*, то последним элементом будет первое* σ^j*, для которого* $\sigma^j(u < x) = 0$*, то есть* $y_0 + j \geqslant x_0$*. Очевидно, что первым из таких* j *будет* $x_0 - y_0$*.*

Следовательно, последний элемент последовательности:

$$\sigma^{x_0-y_0} = \{(x, x_0), (y, y_0), (u, y_0 + x_0 - y_0), (v, x_0 - y_0)\}.$$

Получаем, что

$$\Pi_1(\sigma)(v) = \begin{cases} 0, & \text{если } \sigma x \leqslant \sigma y \\ \sigma x - \sigma y, & \text{если } \sigma x \geqslant \sigma y \end{cases}$$

Это в точности равно $\lfloor \sigma x - \sigma y \rfloor$.

Рассмотрим более сложный пример

Пример 3.16. *Написать структурированную программу, которая находит наибольшее число меньшее* x*, в десятичной записи которого нет троек (рис. 3.4 на противоположной странице).*

Докажем, что приведённый алгоритм действительно решает поставленную задачу. Сначала рассмотрим цикл Π_1.

Утверждение 3.9. *Пусть* σ *— произвольная конфигурация такая, что* $\sigma t = 0$, $(\sigma^i)_i$ *— последовательность конфигураций при работе цикла* Π_1 *на* σ*. Тогда* $\sigma^i u = \left[\sigma u / 10^i\right]$, $\sigma^i t$ *равно количеству троек среди последних* i *цифр числа* σu, $\sigma^i z = \sigma z$ *для любой другой переменной* z.

Доказательство. *Индукция по* i.

Базис индукции. Пусть $i = 0$*. По определению семантики цикла* $\sigma^0 = \sigma$*. Имеем* $\sigma^0 u = \sigma u = \left[\sigma u / 10^0\right]$, $\sigma^i t = 0$ *(по условию),* $\sigma^0 z = \sigma z$ *для всех* z.

Шаг индукции. Пусть для i *утверждение верно. По определению семантики цикла* $\sigma^{i+1} = \Pi_2(\sigma^i)$*. Если* $u_n u_{n-1} \ldots u_1 u_0$ *— десятичная запись числа* σu*, то* $u_n u_{n-1} \ldots u_{i+1} u_i$ *— десятичная запись числа* $\sigma^i u = \left[\sigma u / 10^i\right]$*. Поэтому* $\sigma^i(u \bmod 10) = u_i$ *—* $(i+1)$*-ая цифра числа* σu*. Рассмотрим два случая.*

а) $\sigma^i(u \bmod 10 == 3) = 1$*. В этом случае* $(i+1)$*-ая цифра числа* σu *равна 3. По определению семантики неполного ветвления получаем* $\Pi_3(\sigma^i) = (t = \mathsf{succ}(t);)(\sigma^i)$*. Следовательно,* $\Pi_3(\sigma^i)(t) = \sigma^i t + 1$.

б) $\sigma^i(u \bmod 10 == 3) = 0$*. В этом случае* $(i+1)$*-ая цифра числа* σu *отлична от 3. По определению семантики неполного ветвления получаем* $\Pi_3(\sigma^i) = \sigma^i$*. Следовательно,* $\Pi_3(\sigma^i)(t) = \sigma^i t$.

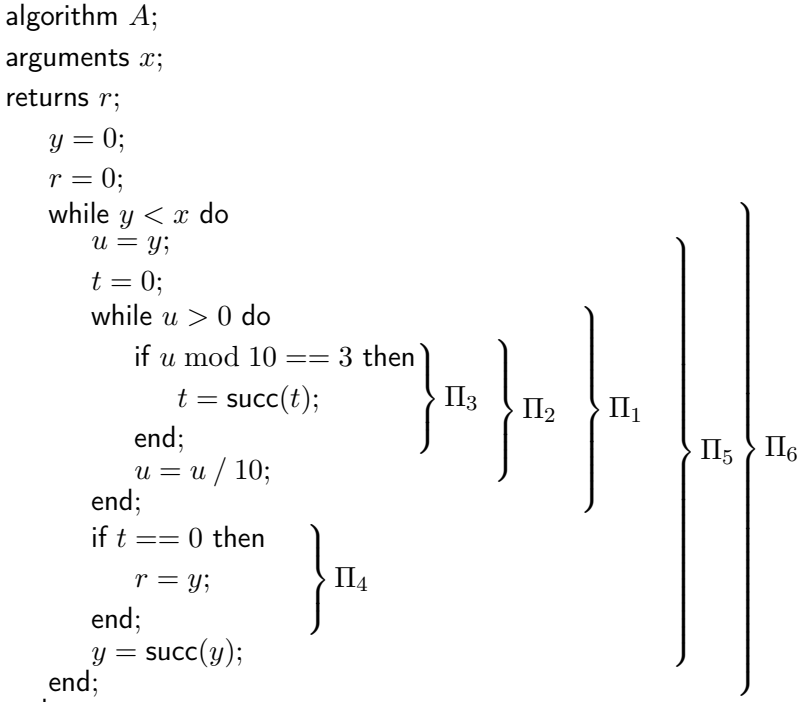


Рис. 3.4: Наибольшее число меньшее x без троек, структурированная программа

Итак, если $(i+1)$-ая цифра числа σu равна 3, то $\Pi_3(\sigma^i)(t) = \sigma^i t + 1$, а если не равна 3, то $\Pi_3(\sigma)(t) = \sigma^i t$. Но по индукционному предположению $\sigma^i t$ есть количество троек среди последних i цифр числа σu. Поэтому $\Pi_3(\sigma^i)(t)$ есть количество троек среди последних $i+1$ цифр числа σu.

Теперь рассмотрим программу Π_2. По определению семантики следования

$$\Pi_2(\sigma^i) = (u = u \,/\, 10;)(\Pi_3(\sigma^i)).$$

По определению семантики присваивания

$$(u = u \,/\, 10;)(\Pi_3(\sigma^i))(t) = \Pi_3(\sigma^i)(t).$$

Поскольку $u \notin \mathrm{LVar}(\Pi_3)$, *то* $\Pi_3(\sigma^i)(u) = \sigma^i u$. *Следовательно,*

$$\sigma^{i+1}u = (u = u \,/\, 10;)(\sigma^i)(u) = \left[\left[\sigma u/10^i\right]/\,10\right] = \left[\sigma u/10^{i+1}\right],$$

$\sigma^{i+1}t = \Pi_3(\sigma^i)(t)$ — *количество троек среди последних* $i+1$ *цифр числа* σu. *Так как* $\mathrm{LVar}(\Pi_2) = \{t, u\}$, *то для любой переменной* $z \notin \{u, t\}$ *получаем* $\sigma^{i+1}z = \sigma^i(z) = \sigma z$ *по индукционному предположению.* □

Утверждение 3.10. *Пусть* σ — *произвольная конфигурация такая, что* $\sigma t = 0$. *Тогда* $\Pi_1(\sigma)$ *определено и* $\Pi_1(\sigma)(t)$ *равно количеству троек в десятичной записи числа* σu.

Доказательство. *Пусть* $(\sigma^i)_i$ — *последовательность конфигураций при работе* Π_1 *на* σ. *По утверждению 3.9 на стр. 68* $\sigma^i u = \left[\sigma u/10^i\right]$. *Если* $\sigma^i(u > 0) = 1$, *то* $\sigma^{i+1}u < \sigma^i u$, *поэтому* $(\sigma^i u)_i$ — *убывающая последовательность натуральных чисел. По теореме об убывающих цепях она конечна. Пусть* $\sigma^n u$ — *её последний элемент. Тогда* $\sigma^n u = 0$, *иначе было бы* $\sigma^n(u > 0) = 1$, *а значит, состояние* σ^n *не было бы последним. Итак, последовательность* $(\sigma^i)_{i=0}^n$ *конечна,* $\Pi_1(\sigma) = \sigma^n$ *и* $\sigma^i(u > 0) = 1$ *тогда и только тогда, когда* $i < n$. *Значит,* n — *наименьшее число такое, что* $\left[\sigma u/10^n\right] = 0$. *Но тогда* n *равно количеству цифр в десятичной записи числа* σu. *По утверждению 3.9 на стр. 68* $\sigma^n t$ *равно количеству троек среди последних* n *цифр числа* σu, *то есть количеству троек в десятичной записи числа* σu. □

Теперь перейдём к внешнему циклу Π_6.

Утверждение 3.11. *Пусть* σ — *произвольная конфигурация такая, что* $\sigma y = 0$, $\sigma r = 0$. *Пусть* $(\sigma^i)_i$ — *последовательность конфигураций при работе* Π_6 *на* σ. *Тогда* $\sigma^i x = \sigma x$, $\sigma^i y = i$, $\sigma^i r$ *равно наибольшему числу на отрезке* $[0, i)$, *в десятичной записи которого нет троек, при* $i > 0$ *и* $\sigma^0 r = 0$.

Доказательство. *Индукция по* i.

Базис индукции. Пусть $i = 0$. *По определению семантики цикла* $\sigma^0 = \sigma$. *Поэтому* $\sigma^0 x = \sigma x$, $\sigma^0 y = \sigma y = 0$, $\sigma^0 r = \sigma r = 0$.

Шаг индукции. Пусть утверждение выполнено для i. *По определению семантики цикла* $\sigma^{i+1} = \Pi_5(\sigma^i)$. *По определению семантики следования*

мы получим

$$\Pi_5(\sigma^i) = (y = \mathsf{succ}(y);)\Big(\Pi_4\big(\Pi_1((u = y;\ t = 0;)(\sigma^i))\big)\Big).$$

Обозначим через τ конфигурацию $(u = y;\ t = 0;)(\sigma^i)$. По определению семантики присваивания и следования получаем $\tau u = \sigma^i y$, $\tau t = 0$. Обозначим через ρ конфигурацию $\Pi_1(\tau)$. По утверждению 3.10 на предыдущей странице ρt равно количеству троек в десятичной записи числа $\tau u = \sigma^i y = i$ (последнее равенство — индукционное предположение). Обозначим $\Pi_4(\rho)$ через π. Рассмотрим два случая.

а) *$\rho(t = 0) = 1$. Это значит, что десятичная запись числа i не содержит троек. Поэтому на отрезке $[0, i+1)$ наибольшее число без троек равно i. По определению семантики неполного ветвления $\pi = (r = y;)(\rho)$, поэтому $\pi r = i$.*

б) *$\rho(t = 0) = 0$. Это значит, что десятичная запись числа i содержит тройки. Поэтому наибольшее число без троек на отрезке $[0, i + 1)$ равно наибольшему числу без троек на отрезке $[0, i)$. По определению семантики неполного ветвления $\pi = \rho$ и $\pi r = \rho r = \sigma^i r$.*

В обоих случаях πr равно наибольшему числу на отрезке $[0, i+1)$, в десятичной записи которого нет троек. Наконец, $\sigma^{i+1} y = (y = \mathsf{succ}(y);)(\pi) = \pi y + 1 = i + 1$, $\sigma^{i+1} x = \sigma^i x = \sigma x$, так как $x \notin \mathrm{LVar}(\Pi_5)$. По определению семантики следования окончательно получаем $\sigma^{i+1} x = \sigma x$, $\sigma^{i+1} y = i + 1$, $\sigma^{i+1} r$ равно наибольшему числу на отрезке $[0, i + 1)$, в десятичной записи которого нет троек. □

Наконец, рассмотрим весь алгоритм.

Утверждение 3.12. *Для любого вызова $\delta = (A, x_0)$ $\mathrm{Res}\,\delta$ равно наибольшему числу меньше x_0, в десятичной записи которого нет троек.*

Доказательство. *Пусть σ_δ — начальная конфигурация вызова δ. Пусть $\sigma = (y = 0;)(\sigma_\delta)$. Тогда $\sigma y = 0$. Пусть $(\sigma^i)_i$ — последовательность конфигураций при работе Π_6 на σ. По утверждению 3.11 на противоположной странице $\sigma^i y = i$ для всех i. Последовательность $(\sigma^i)_i$ конечна, так как в противном случае $(\sigma^i x - \sigma^i y)_i$ была бы бесконечной убывающей последовательностью натуральных чисел, что противоречит теореме об убывающих*

цепях. Пусть σ^n — последний элемент последовательности. По определению семантики цикла n — наименьшее число такое, что $\sigma^n(y < x) = 0$, то есть $n \geqslant x_0$. Значит, $n = x_0$. По определению $\operatorname{Res}\delta = \Pi_A(\sigma_\delta)(r) = \Pi_6(\sigma)(r)$. *По утверждению 3.11 на стр. 70* $\operatorname{Res}\delta$ *равно наибольшему числу на отрезке* $[0, x_0)$ *без троек, то есть* $\operatorname{Res}\delta$ *равно наибольшему числу меньше* x_0*, в десятичной записи которого нет троек.* □

Задача 3.15. Напишите алгоритм, проверяющий чётность заданного числа, и докажите его правильность.

Пример 3.17. *Рассмотрим пример алгоритма, для вычисления НОД методом Евклида (рис. 3.5 на противоположной странице). Номера строк поставлены только для удобства описания, программе они не принадлежат.*

В строках 2–5 мы проверяем специальные случаи, когда один из аргументов равен 0. В этом случае мы полагаем результат равным второму аргументу. Заметим, что если оба аргумента равны 0, то результатом тоже будет 0, поэтому рассматривать этот случай отдельно не нужно.

Строки 7–19 содержат обычный шаг алгоритма Евклида, когда от большего числа отнимается меньшее. Так как вычитание в нашем языке отсутствует, то мы используем строки 8–11 и 14–17 для его выполнения (сравните с телом алгоритма вычитания, рис. 3.3 на стр. 67).

***Задача 3.16.** Докажите правильность алгоритма *Euclid*.

Задача 3.17. Напишите алгоритмы, которые выполняют:

1) умножение двух чисел;
2) нахождение целой части частного двух чисел;
3) нахождение остатка от деления двух чисел;
4) вычисление факториала числа;
5) проверку, является ли число простым,

и докажите их правильность

Естественно, что можно написать много разных алгоритмов, которые будут вычислять одну и ту же (частичную) функцию.

Определение 3.19 (Эквивалентность алгоритмов)**.** *Алгоритмы A и B называются эквивалентными, если они вычисляют*

```
 1: algorithm Euclid;
    arguments x, y;
    returns x;
 2:     if x == 0 then
 3:         x = y;
 4:     else
 5:         if 0 < y then
 6:             while x != y do
 7:                 if x < y then
 8:                     v = 0; w = x;
 9:                     while w < y do
10:                         v = succ(v); w = succ(w);
11:                     end;
12:                     y = v;
13:                 else
14:                     v = 0; w = y;
15:                     while w < x do
16:                         v = succ(v); w = succ(w);
17:                     end;
18:                     x = v;
19:                 end;
20:             end;
21:         end;
22:     end;
23: end;
```

Рис. 3.5: Алгоритм Евклида.

одну и ту же (частичную) функцию:

$$\Phi_A = \Phi_B.$$

Следствие 3.13. *Алгоритмы A и B эквивалентны тогда и только тогда, когда они имеют одинаковое число входных переменных n и для любых $\bar{v} \in \omega^n$ или $\mathrm{Res}(A, \bar{v})$ и $\mathrm{Res}(B, \bar{v})$ одновременно неопределены, или одновременно определены и при этом*

$$\mathrm{Res}(A, \bar{v}) = \mathrm{Res}(B, \bar{v}).$$

***Задача 3.18.** Верно ли следующее утверждение:

> Если $\Pi_A(\sigma) = \Pi_B(\sigma)$ для любой конфигурации σ, то A и B эквивалентны?

Верно ли обратное к нему утверждение? Докажите.

Пример 3.18. *Рассмотрим следующие алгоритмы.*

algorithm A;	algorithm B;	algorithm C;
arguments x, y;	arguments a, b;	arguments u, v;
returns u;	returns b;	returns v;
$u = y$;	;	$v = u$;
end;	end;	end;

Эти три алгоритма вычисляют двухместные функции:

$$\Phi_A(p, q) = \Phi_B(p, q) = q; \qquad \Phi_C(p, q) = p.$$

Таким образом, первые два эквивалентны, а третий не эквивалентен первым двум.

Задача 3.19. Доказать, что каждый алгоритм эквивалентен некоторому алгоритму, который не содержит ветвлений.

Указание. Показать, как ветвления заменить циклами.

§ 3.3. *Свойства структурированных программ

Изучим некоторые свойства структурированных программ.

Сначала продемонстрируем, что значение переменных, которые не встречаются в левой части оператора присваивания в программе П, не изменяется.

Лемма 3.14 (О сохранении значения)**.** *Если* Π *— программа,* x *—переменная,* $x \notin \mathrm{LVar}(\Pi)$, σ, τ *— конфигурации* Π *и* $\Pi(\sigma) = \tau$, *то* $\sigma x = \tau x$.

Доказательство. Индукция по построению программы П.

Базис индукции. Для пустого оператора: $;(\sigma)(x) = \sigma(x)$ по определению семантики.

$\Pi \circ y = e;$ где y — переменная, e — выражение. Очевидно, что $y \neq x$, иначе $x \in \mathrm{LVar}(\Pi)$. По определению семантики для оператора присваивания $\Pi(\sigma) = \tau$ и при этом $\tau y = \sigma(e)$ и $\tau z = \sigma z$ для всех переменных $z \neq y$. Так как $x \neq y$, то $\tau x = \sigma x$.

Шаг индукции.

1) Пусть $\Pi \circ \Pi_1\Pi_2$. По индукции для любой конфигурации σ если $\Pi_1(\sigma) = \rho$, то $\sigma x = \rho x$. По индукции для любой конфигурации ρ если $\Pi_2(\rho) = \tau$, то $\tau x = \rho x$. По определению семантики для следования имеем $\tau = \Pi(\sigma) = \Pi_2(\Pi_1(\sigma)) = \Pi_2(\rho)$. Следовательно, $\tau x = \rho x = \sigma x$.

2) Пусть

$$\Pi \circ \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end;} \end{array} \right.$$

По индукции для любой конфигурации σ

$$\Pi_1(\sigma)(x) = \sigma x, \quad \Pi_2(\sigma)(x) = \sigma x.$$

Если $\sigma(e) \neq 0$, то

$$\Pi(\sigma)(x) = \Pi_1(\sigma)(x) = \sigma x.$$

В противном случае

$$\Pi(\sigma)(x) = \Pi_2(\sigma)(x) = \sigma x.$$

Следовательно, $\Pi(\sigma)(x) = \sigma x$.

3) Для неполного ветвления доказательство аналогично.

4) Пусть

$$\Pi \twoheadleftarrow \begin{cases} \textsf{while } e \textsf{ do} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

Так как $\Pi(\sigma)$ определено, то существуют число n и последовательность конфигураций $(\sigma^i)_{i=0}^n$ такие, что

$$\sigma^0 = \sigma,\ \sigma^{i+1} = \Pi_1(\sigma^i),\ \sigma^i(e) \neq 0 \iff i < n,\ \sigma^n = \tau.$$

По индукционному предположению $\Pi_1(\rho)(x) = \rho x$ для любой конфигурации ρ. Следовательно,

$$\sigma^1 x = \Pi_1(\sigma^0)(x) = \sigma^0 x = \sigma x,$$
$$\sigma^2 x = \Pi_1(\sigma^1)(x) = \sigma^1 x = \sigma x,$$

и т. д. В конце концов мы получим, что $\sigma^n x = \sigma x$, то есть, что $\tau x = \sigma x$. □

Задача 3.20. Обоснуйте шаг для неполного ветвления.

Следующее утверждение показывает, что для работы программы существенно значение только тех переменных, которые в ней используются.

Лемма 3.15 (О неиспользуемых переменных)**.** *Для любой программы* Π *и любой конфигурации* σ

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi(\sigma \restriction \mathrm{Var}(\Pi)).$$

Доказательство. Индукция по построению программы. □

***Задача 3.21.** Докажите лемму.

***Задача 3.22.** Докажите эту же лемму с заменой множества $\mathrm{Var}(\Pi)$ на произвольное множество переменных V такое, что $\mathrm{Var}(\Pi) \subseteq V$.

Следствие 3.16 (О двух конфигурациях)**.** *Если*

$$\sigma \restriction \mathrm{Var}(\Pi) = \tau \restriction \mathrm{Var}(\Pi),$$

то

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi(\tau) \restriction \mathrm{Var}(\Pi).$$

Следствие 3.17 (О результате)**.** *Если* $\Pi(\sigma)$ *определено, то*

$$\Pi(\sigma) = \Pi(\sigma \restriction \mathrm{Var}(\Pi)) \cup \sigma \restriction (\mathrm{dom}\,\sigma \setminus \mathrm{Var}(\Pi)).$$

Задача 3.23. Докажите оба следствия.

Определение 3.20 (Замена переменных)**.** *Если a — это выражение или программа, $x_1, \ldots, x_n$ — попарно различные переменные, $e_1, \ldots, e_n$ — выражения, то с помощью*

$$(a)^{x_1 \ldots x_n}_{e_1 \ldots e_n}$$

мы будем обозначать результат замены в a всех переменных x_i на выражения e_i.

Пример 3.19. *Пусть* $e \mathrel{-\!\!\circ} \mathsf{succ}(x)$. *Тогда* $(e)^x_y \mathrel{-\!\!\circ} \mathsf{succ}(y)$.

Замену можно определить и индукцией по построению выражений и программ.

Определение 3.21 (Замена переменных)**.** *Для выражений:*

1) *если* $e \mathrel{-\!\!\circ} c$, c — *константа, то* $(c)^{x_1 \ldots x_n}_{e_1 \ldots e_n} \mathrel{-\!\!\circ} c$;

2) *если* $e \mathrel{-\!\!\circ} x_i$, *то* $(e)^{x_1 \ldots x_n}_{e_1 \ldots e_n} \mathrel{-\!\!\circ} e_i$;

3) *если* $e \mathrel{-\!\!\circ} z$, z — *переменная,* $z \notin x_i$ *для* $i = 1, \ldots, n$, *то* $(e)^{x_1 \ldots x_n}_{e_1 \ldots e_n} \mathrel{-\!\!\circ} z$;

4) *если* $e \twoheadrightarrow o(d_1, \ldots, d_m)$, *$o$ — операция,* $d_1, \ldots, d_n$ *— выражения, то*

$$(e)^{x_1 \ldots x_n}_{e_1 \ldots e_n} \twoheadrightarrow o\left((d_1)^{x_1 \ldots x_n}_{e_1 \ldots e_n}, \ldots, (d_m)^{x_1 \ldots x_n}_{e_1 \ldots e_n}\right)$$

Задача 3.24. Дайте определение замены для программы индукцией по построению программы.

Лемма 3.18 (О результате замены)**.** *Если $\bar{x}$ — переменные, $\bar{e}$ и d — выражения, то $(d)^{\bar{x}}_{\bar{e}}$ — выражение.*

Доказательство. Индукция по построению выражения d. □

***Задача 3.25.** Докажите лемму.

Замечание 3.2. Очевидно, для того, чтобы результат замены $(\Pi)^{x_1 \ldots x_n}_{e_1 \ldots e_n}$, где Π — программа снова был программой, необходимо, чтобы e_i было переменной для каждого $x_i \in \mathrm{LVar}(\Pi)$.

Задача 3.26. Приведите пример, доказывающий необходимость этого условия.

Покажем, что это условие является одновременно и достаточным.

Лемма 3.19 (О результате замены)**.** *Пусть Π — программа, $\bar{x}$ — переменные, $\bar{e}$ — выражения, причём если $x_i \in \mathrm{LVar}(\Pi)$, то e_i — переменная. Тогда $(\Pi)^{\bar{x}}_{\bar{e}}$ — программа.*

Доказательство. Индукция по построению программы Π. □

***Задача 3.27.** Докажите лемму.

Заметим, что замена осуществляется не последовательно, а одновременно, то есть нельзя сначала заменить x_1 на e_1, затем x_2 на e_2 и так далее.

Пример 3.20. *Пусть $e \twoheadrightarrow x < y$. Тогда $(e)^{xy}_{yz} \twoheadrightarrow y < z$, то есть мы заменили x на y, а y на z. Если бы мы осуществляли замену не одновременно, а последовательно, то получили бы следующее*

$$((e)^x_y)^y_z \twoheadrightarrow (y < y)^y_z \twoheadrightarrow z < z.$$

Как видим, результат получился другой.

Введём понятие замены переменной для состояний.

Определение 3.22 (Замена переменных)**.** *Если σ — состояние, $x_1, \ldots, x_n, y_1, \ldots, y_n$ — переменные, то с помощью $(\sigma)^{x_1 \ldots x_n}_{y_1 \ldots y_n}$ мы обозначаем следующее множество τ:*

$$\tau = (\sigma \setminus \{(x_i, v) : i = 1, \ldots, n, v \in \omega\}) \cup \{(y_1, \sigma x_1), \ldots, (y_n, \sigma x_n)\}.$$

Рассмотрим пример.

Пример 3.21. *Пусть $\sigma = \{(x, 1), (y, 2)\}$. Тогда*

$$(\sigma)^{xy}_{yz} = \{(y, 1), (z, 2)\}.$$

Замечание 3.3. Результат замены переменных в состоянии может не быть состоянием, так как может не быть отображением. Например, для состояния σ из предыдущего примера:

$$(\sigma)^{x}_{y} = \{(y, 1), (y, 2)\}$$

не является отображением.

Сформулируем достаточное условие для того, чтобы результат замены был состоянием.

Лемма 3.20. *Если все переменные y_i не входят в $\operatorname{dom}\sigma$ и все они попарно различны, то $(\sigma)^{x_1 \ldots x_n}_{y_1 \ldots y_n}$ будет состоянием.*

Задача 3.28. Докажите, лемму.

Задача 3.29. Приведите пример, доказывающий, что условие из леммы не является необходимым.

Лемма 3.21. *Пусть σ — конфигурация выражения или структурированной программы a. Если все переменные y_i не входят в $\operatorname{dom}\sigma$ и все они попарно различны, то $(\sigma)^{x_1 \ldots x_n}_{y_1 \ldots y_n}$ будет конфигурацией для $(a)^{x_1 \ldots x_n}_{y_1 \ldots y_n}$.*

Задача 3.30. Докажите, лемму.

Лемма 3.22. *$(\sigma)^{x_1 \ldots x_n}_{y_1 \ldots y_n} = \tau$ тогда и только тогда, когда для любой переменной u выполнено $\sigma u = \tau\left((u)^{x_1 \ldots x_n}_{y_1 \ldots y_n}\right)$.*

Отметим одно из очевидных свойств замены.

Лемма 3.23 (Обратимость замены)**.** *Пусть a — выражение, программа или состояние, $x_1, \ldots, x_n$ — попарно различные переменные. Предположим, что переменные $y_1, \ldots, y_n$ не встречаются в a и попарно различны. Тогда*

$$\left((a)^{x_1 \ldots x_n}_{y_1 \ldots y_n}\right)^{y_1 \ldots y_n}_{x_1 \ldots x_n}$$

совпадает с a.

Доказательство. В самом деле, при замене $(a)^{x_1 \ldots x_n}_{y_1 \ldots y_n}$ вместо каждого x_i всюду появятся соответствующий y_i, а при обратной замене — x_i вместо y_i. Поскольку y_i в a отсутствуют, то в никаких других местах x_i не возникнут. □

***Задача 3.31.** Докажите лемму индукцией по определению выражений и программ.

Следующие две леммы дают основные свойства замены переменных в выражениях и программах.

Лемма 3.24 (Замена переменных в выражениях)**.** *Пусть переменные $\bar{y}$ не входят в выражение e и его конфигурацию σ, а также выполнено* $\mathrm{Var}(e) \subseteq \mathrm{dom}\,\sigma$. *Тогда*

$$(\sigma)^{\bar{x}}_{\bar{y}} \left((e)^{\bar{x}}_{\bar{y}}\right) = \sigma(e)$$

для любых переменных $\bar{x}$.

Доказательство. Индукция по построению выражения e.

Базис индукции.

1) $e \circ c$, где c — константа. Так как $\bar{x}$ — переменные, то $(c)^{\bar{x}}_{\bar{y}} \circ c$. $\rho(c)$ не зависит от ρ. Значит

$$(\sigma)^{\bar{x}}_{\bar{y}} \left((e)^{\bar{x}}_{\bar{y}}\right) = (\sigma)^{\bar{x}}_{\bar{y}}(c) = \sigma(c) = \sigma(e).$$

2) $e \circ z$, z — переменная и $z \neq x_i$, $i = 1, \ldots, n$. Тогда $(e)^{\bar{x}}_{\bar{y}} \circ z$. Кроме того, $\sigma z = (\sigma)^{\bar{x}}_{\bar{y}}(z)$. Следовательно,

$$(\sigma)^{\bar{x}}_{\bar{y}} \left((e)^{\bar{x}}_{\bar{y}}\right) = (\sigma)^{\bar{x}}_{\bar{y}}(z) = \sigma z = \sigma(e).$$

3) $e \circ x_i$. Тогда $(e)^{\bar{x}}_{\bar{y}} \circ y_i$. Но по определению $(\sigma)^{\bar{x}}_{\bar{y}}$ имеем $\sigma x_i = (\sigma)^{\bar{x}}_{\bar{y}}(y_i)$. Следовательно,

$$(\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = (\sigma)^{\bar{x}}_{\bar{y}}(y_i) = \sigma x_i = \sigma(e).$$

Шаг индукции. Пусть $e \circ o(e_1, \ldots, e_m)$, o — операция, $e_1, \ldots, e_m$ — выражения. По индукционному предположению считаем, что

$$(\sigma)^{\bar{x}}_{\bar{y}}\left((e_i)^{\bar{x}}_{\bar{y}}\right) = \sigma(e_i)$$

для всех $i = 1, \ldots, m$. По определению значения выражения

$$\sigma(e) = o(\sigma(e_1), \ldots, \sigma(e_m)).$$

Далее,

$$(\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = o\left((\sigma)^{\bar{x}}_{\bar{y}}\left((e_1)^{\bar{x}}_{\bar{y}}\right), \ldots, (\sigma)^{\bar{x}}_{\bar{y}}\left((e_m)^{\bar{x}}_{\bar{y}}\right)\right) =$$
$$= o(\sigma(e_1), \ldots, \sigma(e_m)) = \sigma(e). \quad \square$$

Лемма 3.25 (Замена переменных в программах)**.** *Пусть переменные $\bar{y}$ не входят в программу* П *и ее конфигурацию σ. Тогда*

$$(\Pi)^{\bar{x}}_{\bar{y}}\left((\sigma)^{\bar{x}}_{\bar{y}}\right) = (\Pi(\sigma))^{\bar{x}}_{\bar{y}}$$

для любых переменных $\bar{x}$.

Доказательство. Индукция по построению программы П.

Базис индукции. Для пустого оператора $;(\sigma) = \sigma$ и утверждение очевидно.

Пусть $\Pi \circ z = e$; где z — переменная, e — выражение. Возможно два случая:

1) $z \neq x_1, \ldots, z \neq x_n$. Тогда $(\Pi)^{\bar{x}}_{\bar{y}} \circ z = (e)^{\bar{x}}_{\bar{y}}$; согласно лемме 3.24 на противоположной странице

$$(\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = \sigma(e).$$

Пусть $\Pi(\sigma) = \tau$. По определению семантики присваивания $\tau u = \sigma u$, если $u \not\equiv z$, и $\tau z = \sigma(e)$. Пусть

$$(\Pi)^{\bar{x}}_{\bar{y}}\left((\sigma)^{\bar{x}}_{\bar{y}}\right) = \rho.$$

По определению семантики присваивания $\rho u = (\sigma)^{\bar{x}}_{\bar{y}} u$, если $u \not\equiv z$, и

$$\rho z = (\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = \sigma(e).$$

Итак, $\rho z = \tau z$. Для любой другой переменной u имеем

$$\tau u = \sigma u = (\sigma)^{\bar{x}}_{\bar{y}}\left((u)^{\bar{x}}_{\bar{y}}\right).$$

Поскольку z отличается от $y_1, \ldots, y_n$ по условию леммы, то $z \not\equiv (u)^{\bar{x}}_{\bar{y}}$. Значит,

$$\tau u = (\sigma)^{\bar{x}}_{\bar{y}}\left((u)^{\bar{x}}_{\bar{y}}\right) = \rho\left((u)^{\bar{x}}_{\bar{y}}\right).$$

Это и означает, что $\rho = (\tau)^{\bar{x}}_{\bar{y}}$.

2) $z \circ\!\!-\!\!\circ x_i$. Тогда $(\Pi)^{\bar{x}}_{\bar{y}} \circ\!\!-\!\!\circ y_i = (e)^{\bar{x}}_{\bar{y}}$. Согласно лемме 3.24 на стр. 80

$$(\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = \sigma(e).$$

Пусть $\Pi(\sigma) = \tau$. По определению семантики присваивания $\tau u = \sigma u$, если $u \not\equiv x_i$, и $\tau x_i = \sigma(e)$. Пусть

$$(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = \rho.$$

По определению семантики присваивания $\rho u = (\sigma)^{\bar{x}}_{\bar{y}} u$, если $u \not\equiv y_i$, и

$$\rho y_i = (\sigma)^{\bar{x}}_{\bar{y}}\left((e)^{\bar{x}}_{\bar{y}}\right) = \sigma(e).$$

Итак, $\rho y_i = \tau x_i$. Для любой другой переменной u имеем

$$\tau u = \sigma u = (\sigma)^{\bar{x}}_{\bar{y}}\left((u)^{\bar{x}}_{\bar{y}}\right) = \rho\left((u)^{\bar{x}}_{\bar{y}}\right).$$

Следовательно, $\rho = (\tau)^{\bar{x}}_{\bar{y}}$.

Ш а г и н д у к ц и и .

1) Пусть $\Pi \circlearrowleft \Pi_1\Pi_2$. Тогда, очевидно,

$$(\Pi)^{\bar{x}}_{\bar{y}} \circlearrowleft (\Pi_1)^{\bar{x}}_{\bar{y}}(\Pi_2)^{\bar{x}}_{\bar{y}}.$$

По определению семантики следования: $\Pi(\sigma) = \Pi_2(\Pi_1(\sigma))$ и

$$(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = (\Pi_2)^{\bar{x}}_{\bar{y}}((\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}})).$$

Пусть $\Pi_1(\sigma) = \tau$. По индукционному предположению

$$(\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = (\tau)^{\bar{x}}_{\bar{y}}.$$

Пусть $\Pi_2(\tau) = \rho$. Тогда

$$(\Pi_2)^{\bar{x}}_{\bar{y}}((\tau)^{\bar{x}}_{\bar{y}}) = (\rho)^{\bar{x}}_{\bar{y}}.$$

Итак,

$$(\Pi(\sigma))^{\bar{x}}_{\bar{y}} = (\Pi_2(\Pi_1(\sigma)))^{\bar{x}}_{\bar{y}} = (\Pi_2(\tau))^{\bar{x}}_{\bar{y}} = (\Pi_2)^{\bar{x}}_{\bar{y}}((\tau)^{\bar{x}}_{\bar{y}}) = \\ = (\Pi_2)^{\bar{x}}_{\bar{y}}((\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}})) = (\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}).$$

2) Пусть

$$\Pi \circlearrowleft \left\{ \begin{array}{l} \textsf{if } e \textsf{ then} \\ \quad \Pi_1 \\ \textsf{else} \\ \quad \Pi_2 \\ \textsf{end;} \end{array} \right.$$

Тогда

$$(\Pi)^{\bar{x}}_{\bar{y}} \circlearrowleft \left\{ \begin{array}{l} \textsf{if } (e)^{\bar{x}}_{\bar{y}} \textsf{ then} \\ \quad (\Pi_1)^{\bar{x}}_{\bar{y}} \\ \textsf{else} \\ \quad (\Pi_2)^{\bar{x}}_{\bar{y}} \\ \textsf{end;} \end{array} \right.$$

По определению семантики для полного ветвления имеем:

$$\Pi(\sigma) = \begin{cases} \Pi_1(\sigma), & \text{если } \sigma(e) \neq 0; \\ \Pi_2(\sigma), & \text{если } \sigma(e) = 0. \end{cases}$$

$$(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = \begin{cases} (\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}), & \text{если } (\sigma)^{\bar{x}}_{\bar{y}}(e)^{\bar{x}}_{\bar{y}} \neq 0; \\ (\Pi_2)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}), & \text{если } (\sigma)^{\bar{x}}_{\bar{y}}(e)^{\bar{x}}_{\bar{y}} = 0. \end{cases}$$

Заменяя в первом равенстве $\bar{x}$ на $\bar{y}$, получим

$$(\Pi(\sigma))^{\bar{x}}_{\bar{y}} = \begin{cases} (\Pi_1(\sigma))^{\bar{x}}_{\bar{y}}, & \text{если } \sigma(e) \neq 0; \\ (\Pi_2(\sigma))^{\bar{x}}_{\bar{y}}, & \text{если } \sigma(e) = 0. \end{cases}$$

По лемме 3.24 на стр. 80,

$$\sigma(e) = (\sigma)^{\bar{x}}_{\bar{y}}(e)^{\bar{x}}_{\bar{y}}.$$

По индукционному предположению

$$(\Pi_1(\sigma))^{\bar{x}}_{\bar{y}} = (\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}), \quad (\Pi_2(\sigma))^{\bar{x}}_{\bar{y}} = (\Pi_2)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}).$$

Поэтому получаем, что $(\Pi(\sigma))^{\bar{x}}_{\bar{y}} = (\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}})$.

3) Для неполного ветвления доказывается аналогично.

4) Пусть

$$\Pi \multimap \begin{cases} \textsf{while } e \textsf{ do} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

Тогда

$$(\Pi)^{\bar{x}}_{\bar{y}} \multimap \begin{cases} \textsf{while } (e)^{\bar{x}}_{\bar{y}} \textsf{ do} \\ \quad (\Pi_1)^{\bar{x}}_{\bar{y}} \\ \textsf{end;} \end{cases}$$

По индукционному предположению для любой конфигурации ρ имеет место:

$$(\Pi_1(\rho))^{\bar{x}}_{\bar{y}} = (\Pi_1)^{\bar{x}}_{\bar{y}}((\rho)^{\bar{x}}_{\bar{y}}).$$

Пусть $(\sigma^i)_i$ — последовательность конфигураций для программы Π на σ, а $(\tau^i)_i$ — для $(\Pi)^{\bar{x}}_{\bar{y}}$ на $(\sigma)^{\bar{x}}_{\bar{y}}$. Докажем индукцией по i, что $(\sigma^i)^{\bar{x}}_{\bar{y}} = \tau^i$.

Базис индукции: $i = 0$. Тогда

$$\sigma^0 = \sigma,\ \tau^0 = (\sigma)^{\bar{x}}_{\bar{y}} = (\sigma^0)^{\bar{x}}_{\bar{y}}.$$

Индукционный шаг: пусть для i доказано. Тогда

$$\sigma^{i+1} = \Pi_1(\sigma^i),\quad \tau^{i+1} = (\Pi_1)^{\bar{x}}_{\bar{y}}(\tau^i).$$

Получаем, что

$$\tau^{i+1} = (\Pi_1)^{\bar{x}}_{\bar{y}}(\tau^i) = (\Pi_1)^{\bar{x}}_{\bar{y}}((\sigma^i)^{\bar{x}}_{\bar{y}}) = (\Pi_1(\sigma^i))^{\bar{x}}_{\bar{y}} = (\sigma^{i+1})^{\bar{x}}_{\bar{y}}.$$

Итак, мы доказали, что $(\sigma^i)^{\bar{x}}_{\bar{y}} = \tau^i$ для всех i. Пусть $\Pi(\sigma)$ определено. Тогда существует n такое, что $\sigma^i(e) \neq 0$ тогда и только тогда, когда $i < n$ и при этом $\Pi(\sigma) = \sigma^n$. Следовательно, по лемме 3.24 на стр. 80

$$(\sigma^i)^{\bar{x}}_{\bar{y}}(e)^{\bar{x}}_{\bar{y}} = \sigma^i(e)$$

и оба выражения не равны нулю при $i < n$. Значит,

$$(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = (\sigma^n)^{\bar{x}}_{\bar{y}} = (\Pi(\sigma))^{\bar{x}}_{\bar{y}}.$$

В обратную сторону. Пусть $(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}})$ определено. По лемме 3.23 на стр. 80

$$((\Pi)^{\bar{x}}_{\bar{y}})^{\bar{y}}_{\bar{x}} \multimap \Pi,\quad ((\sigma)^{\bar{x}}_{\bar{y}})^{\bar{y}}_{\bar{x}} = \sigma.$$

Согласно только что доказанному, это означает, что $\Pi(\sigma)$ определено и при этом

$$((\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}))^{\bar{y}}_{\bar{x}} = \Pi(\sigma).$$

Произведя ещё раз замену $\bar{x}$ на $\bar{y}$, получим

$$(\Pi)^{\bar{x}}_{\bar{y}}((\sigma)^{\bar{x}}_{\bar{y}}) = (\Pi(\sigma))^{\bar{x}}_{\bar{y}}. \qquad \square$$

Задача 3.32. Обоснуйте индукционный шаг для неполного ветвления.

Следствие 3.26. *Пусть дан алгоритм:*

algorithm A;
arguments $\bar{x}$;
returns e;
    Π_A
end;

Попарно различные переменные $\bar{y}$ не встречаются в Π_A, $\bar{x}$ и e. Тогда алгоритм

algorithm B;
arguments $(\bar{x})^{\bar{z}}_{\bar{y}}$;
returns $(e)^{\bar{z}}_{\bar{y}}$;
    $(\Pi_A)^{\bar{z}}_{\bar{y}}$
end;

эквивалентен исходному.

Задача 3.33. Докажите следствие.

§ 3.4. *Простые программы

В этом параграфе мы докажем, что каждая программа эквивалентна программе специального вида.

Определение 3.23 (Простая программа)**.** *Программа* Π *называется* п р о с т о й*, если каждый оператор присваивания имеет один из трёх видов:*

1) $x = y_1$;

2) $x = c$;

3) $x = o(y_1, \ldots, y_n)$,

где $x, y_1, \ldots, y_n$ *— переменные,* c *— константа,* o *— операция, а каждое выражение после* if *или* while *является переменной. Операторы указанного вида мы будем называть* п р о с т ы м и*.*

Алгоритм назовем п р о с т ы м *если его тело является простой программой, а выражение-результат — переменной.*

Пример 3.22. *Программы из примеров 3.14 и 3.15 не являются простыми, например, из-за использования выражений вида* $u < v$ *после* if.

Лемма 3.27 (О разложении выражения)**.** *Пусть* d *— выражение,* y *— переменная,* σ, τ *— конфигурации,* $\tau = (y = d;)(\sigma)$*. Тогда для любого выражения* e

$$\tau(e) = \sigma((e)^y_d).$$

Доказательство. Очевидно, что τ отличается от σ только тем, что $\tau y = \sigma(d)$. Индукция по построению выражения e.
Базис индукции.

1) $e \circ\!\!-\!\!\circ y$. Тогда $\sigma((e)^y_d) = \sigma(d) = \tau y = \tau(e)$.

2) $e \circ\!\!-\!\!\circ z$, z — переменная, $z \neq y$. Тогда $\sigma((e)^y_d) = \sigma z = \tau z = \tau(e)$.

3) $e \circ\!\!-\!\!\circ c$, c — константа: $\sigma((e)^y_d) = \sigma c = \tau c = \tau(e)$.

Ш а г и н д у к ц и и . Пусть для выражений $e_1, \ldots, e_n$ лемма доказана и $e \mathrel{\circ\!\!-} o(e_1, \ldots, e_n)$, где o — операция.

$$\sigma((e)_d^y) = \sigma(o((e_1)_d^y, \ldots, (e_n)_d^y)) =$$
$$= o(\sigma((e_1)_d^y), \ldots, \sigma((e_n)_d^y)) = o(\tau(e_1), \ldots, \tau(e_n)) = \tau(e). \quad \square$$

Напомним ещё раз, что с помощью $\mathfrak{V}$ мы обозначаем множество всевозможных переменных, которые могут использоваться в программах.

Следствие 3.28 (Упрощение присваиваний)**.** *Пусть x, y — переменные, d, e — выражения,*

$$\Pi \mathrel{\circ\!\!-} x = (e)_d^y;$$

$$\Pi_1 \mathrel{\circ\!\!-} y = d; x = e;$$

$$\mathrm{Var}(\Pi) \subseteq V \subseteq \mathfrak{V};$$

$y \notin V$. Тогда для любой конфигурации σ

$$\Pi(\sigma) \restriction V = \Pi_1(\sigma) \restriction V.$$

Доказательство. Очевидно, что достаточно доказать, что $\Pi(\sigma)(x) = \Pi_1(\sigma)(x)$, так как остальные переменные из V не изменяются ни в Π, ни в Π_1. Пусть $\tau = (y = d;)(\sigma)$, тогда $\Pi_1(\sigma) = (x = e;)(\tau)$.

$$\Pi(\sigma)(x) = \sigma((e)_d^y) = \tau(e) = \Pi_1(\sigma)(x). \quad \square$$

Лемма 3.29 (Упрощение полного ветвления)**.** *Пусть e — выражение, Π' и Π'' — программы, y — переменная,*

$$\Pi \mathrel{\circ\!\!-} \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{else} \\ \quad \Pi'' \\ \mathsf{end}; \end{array} \right.$$

$$\mathrm{Var}(\Pi) \subseteq V \subseteq \mathfrak{V};$$

$y \notin V$. Пусть

$$\Pi_1 \mathrel{\overset{\circ}{=}} \begin{cases} y = e; \\ \mathsf{if}\ y\ \mathsf{then} \\ \qquad \Pi' \\ \mathsf{else} \\ \qquad \Pi'' \\ \mathsf{end}; \end{cases}$$

Тогда для любой конфигурации σ выполнено

$$\Pi(\sigma) \upharpoonright V = \Pi_1(\sigma) \upharpoonright V.$$

Доказательство. Пусть $\tau = (y = e;)(\sigma)$. Поскольку y не входит в e, то $\sigma(e) = \tau y$. Так как

$$\sigma \upharpoonright V = \tau \upharpoonright V,$$

то

$$\Pi'(\sigma) \upharpoonright V = \Pi'(\tau) \upharpoonright V,$$

$$\Pi''(\sigma) \upharpoonright V = \Pi''(\tau) \upharpoonright V.$$

Если $\sigma(e) \neq 0$, то $\tau(y) \neq 0$, и мы получаем, что $\Pi(\sigma) = \Pi'(\sigma)$ и $\Pi_1(\tau) = \Pi'(\tau)$. Если $\sigma(e) = 0$, то $\tau(y) = 0$, и, следовательно, $\Pi(\sigma) = \Pi''(\sigma)$ и $\Pi_1(\tau) = \Pi''(\tau)$. И в том, и в другом случае

$$\Pi(\sigma) \upharpoonright V = \Pi_1(\sigma) \upharpoonright V. \qquad \square$$

Лемма 3.30 (Упрощение неполного ветвления)**.** *Пусть e — выражение, Π' — программа, y — переменная,*

$$\Pi \mathrel{\overset{\circ}{=}} \begin{cases} \mathsf{if}\ e\ \mathsf{then} \\ \qquad \Pi' \\ \mathsf{end}; \end{cases}$$

$$\mathrm{Var}(\Pi) \subseteq V \subseteq \mathfrak{V};$$

$y \notin V$. Пусть

$$\Pi_1 \eqsim \left\{ \begin{array}{l} y = e; \\ \mathsf{if}\ y\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{end}; \end{array} \right.$$

Тогда для любой конфигурации σ выполнено

$$\Pi(\sigma) \restriction V = \Pi_1(\sigma) \restriction V.$$

***Задача 3.34.** Докажите лемму.

Лемма 3.31 (Упрощение цикла)**.** *Пусть e — выражение, Π' — программа, y — переменная,*

$$\Pi \eqsim \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi' \\ \mathsf{end}; \end{array} \right.$$

$$\mathrm{Var}(\Pi) \subseteq V \subseteq \mathfrak{V};$$

$y \notin V$. Пусть

$$\Pi_1 \eqsim \left\{ \begin{array}{l} y = e; \\ \mathsf{while}\ y\ \mathsf{do} \\ \quad \Pi' \\ \quad y = e; \\ \mathsf{end}; \end{array} \right.$$

Тогда для любой конфигурации σ выполнено

$$\Pi(\sigma) \restriction V = \Pi_1(\sigma) \restriction V.$$

Доказательство. Пусть $\tau = (y = e;)(\sigma)$. Рассмотрим последовательность конфигураций для Π на σ: $\sigma^0 = \sigma$ и $\sigma^{i+1} = \Pi'(\sigma^i)$.

Рассмотрим последовательность конфигураций для Π_1 на τ: $\tau^0 = \tau$ и $\tau^{i+1} = (y = e;)(\Pi'(\tau^i))$.

Индукцией по i докажем, что $(y = e;)(\sigma^i) = \tau^i$. Из этого, в частности, следует, что

$$\sigma^i \restriction V = \tau^i \restriction V.$$

Базис индукции. $i = 0$. $(y = e;)(\sigma^0) = (y = e;)(\sigma) = \tau = \tau^0$.
Шаг индукции. Пусть для i доказано. Тогда

$$\tau^{i+1} = (y = e;)(\Pi'(\tau^i)).$$

Далее,

$$\tau^{i+1} \restriction V = (y = e;)(\Pi'(\tau^i)) \restriction V = \Pi'(\tau^i) \restriction V = \Pi'(\tau^i \restriction V) = \\ = \Pi'(\sigma^i \restriction V) = \Pi'(\sigma^i) \restriction V = \sigma^{i+1} \restriction V.$$

Кроме того,

$$\tau^{i+1}y = (y = e;)(\Pi'(\tau^i))(y) = \Pi'(\tau^i)(e).$$

Так как e содержит только переменные из V, то

$$\Pi'(\tau^i)(e) = \Pi'(\sigma^i)(e).$$

То есть,

$$\tau^{i+1}y = \Pi'(\sigma^i)(e) = \sigma^{i+1}(e).$$

Это и означает, что $\tau^{i+1} = (y = e;)(\sigma^{i+1})$. Из этого следует, что

$$\tau^i(y) = (y = e;)(\sigma^i)(y) = \sigma^i(e).$$

Пусть $\Pi(\sigma)$ определено. Тогда существует $m \in \omega$ такое, что

$$\sigma^i(e) \neq 0 \iff i < m; \quad \Pi(\sigma) = \sigma^m.$$

Следовательно,

$$\tau^i(y) \neq 0 \iff \sigma^i(e) \neq 0 \iff i < m.$$

Следовательно, $\Pi_1(\tau) = \tau^m$ и

$$\Pi_1(\sigma) \restriction V = \Pi_2(\tau) \restriction V = \tau^m \restriction V = \sigma^m \restriction V = \Pi(\sigma) \restriction V.$$

Обратно, пусть $\Pi_1(\tau)$ определено. Тогда существует m такое, что

$$\tau^i(y) \neq 0 \iff i < m; \quad \Pi_1(\tau) = \tau^m.$$

Получаем, что

$$\sigma^i(e) \neq 0 \iff \tau^i(y) \neq 0 \iff i < m.$$

Это означает, $\Pi(\sigma) = \sigma^m$. Как и в предыдущем рассуждении

$$\Pi_1(\sigma) \restriction V = \Pi(\sigma) \restriction V. \qquad \square$$

Теорема 3.32 (О простой программе)**.** *Для всякой программы* Π *существует простая программа* Π' *такая, что*

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi'(\sigma) \restriction \mathrm{Var}(\Pi)$$

для любой конфигурации σ*.*

Доказательство. Для каждого a — выражения или программы — определим некоторое число — в е с этого объекта — $w(a)$.

Для выражений:

1) $w(x) = 0$, x — переменная;

2) $w(c) = 1$, c — константа;

3) $w(o(e_1, \ldots, e_n)) = w(e_1) + \cdots + w(e_n) + 1$, o — операция, e_i, $i = 1, \ldots, n$ — выражения.

Для программ:

1) $w(;) = 0$;

2) $w(x = e;) = \lfloor w(e) - 1 \rfloor$. То есть, $w(x = e;) = 0$, если $w(e) \leqslant 1$, и $w(x = e;) = w(e) - 1$, если $w(e) \geqslant 2$;

3) $w(\Pi_1\Pi_2) = w(\Pi_1) + w(\Pi_2)$;

4) $w(\mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};) = w(\Pi_1) + w(\Pi_2) + w(e)$;

5) $w(\mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};) = w(\Pi_1) + w(e)$;

6) $w(\mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};) = w(\Pi_1) + 2w(e)$.

Здесь Π_1, Π_2 — программы, e — выражение.

По индукции легко доказать, что

1) $w(e) = 0$ тогда и только тогда, когда e — переменная;

2) $w(e) = 1$ тогда и только тогда, когда e — константа или $e \circ\!\!- o(\bar{x})$, $\bar{x}$ — переменные;

3) $w(\Pi) = 0$ тогда и только тогда, когда Π — простая программа.

Теперь индукцией по сложности программы Π мы докажем, что если $w(\Pi) > 0$, то существует программа Π_1 такая, что $w(\Pi_1) < w(\Pi)$ и

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi).$$

для всех конфигураций σ. Поскольку $w(;) = 0$, то пустой оператор можно не рассматривать.

Базис индукции. $\Pi \circ\!\!- x = e$; так как $w(\Pi) \geqslant 1$, то $w(e) \geqslant 2$. Следовательно, $e \circ\!\!- o(e_1, \ldots, e_n)$ и $w(e_i) \geqslant 1$ для некоторого i. Возьмём новую переменную y, которая не встречается в Π, и построим

$$\Pi_1 \circ\!\!- y = e_i; \underbrace{x = o(e_1, \ldots, e_{i-1}, y, e_{i+1}, \ldots, e_n);}_{\Pi_2}$$

$$w(y = e_i;) = w(e_i) - 1;$$

$$w(\Pi_2) = w(e_1) + \cdots + w(e_{i-1}) + w(e_{i+1}) + \cdots + w(e_n) + 1 = \\ = w(\Pi) - w(e_i).$$

Следовательно,

$$w(\Pi_1) = w(e_i) - 1 + w(\Pi) - w(e_i) = w(\Pi) - 1 < w(\Pi).$$

Из следствия 3.28 на стр. 88 следует, что

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi).$$

Ш а г и н д у к ц и и .

1) Если $\Pi \twoheadleftarrow \Pi_3\Pi_4$, то $w(\Pi_3) > 0$ или $w(\Pi_4) > 0$. Рассмотрим первый случай. По индукционному предположению существует программа Π_{31}:

$$\Pi_3(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_{31}(\sigma) \restriction \mathrm{Var}(\Pi)$$

и $w(\Pi_{31}) < w(\Pi_3)$. Возьмём $\Pi_1 \twoheadleftarrow \Pi_{31}\Pi_4$. Тогда $w(\Pi_1) < w(\Pi)$ и

$$\Pi_1(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_4(\Pi_{31}(\sigma)) \restriction \mathrm{Var}(\Pi) = \\ = \Pi_4(\Pi_{31}(\sigma) \restriction \mathrm{Var}(\Pi)) = \Pi_4(\Pi_3(\sigma) \restriction \mathrm{Var}(\Pi)) = \\ = \Pi_4(\Pi_3(\sigma)) \restriction \mathrm{Var}(\Pi) = \Pi(\sigma) \restriction \mathrm{Var}(\Pi).$$

Аналогично рассматривается и второй случай.

2) Пусть

$$\Pi \twoheadleftarrow \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_3 \\ \mathsf{else} \\ \quad \Pi_4 \\ \mathsf{end}; \end{array} \right.$$

Тогда $w(e) > 0$, $w(\Pi_3) > 0$ или $w(\Pi_4) > 0$. В последних двух случаях доказательство аналогично предыдущему. Если $w(e) > 0$, то возьмём новую переменную y и построим Π_1:

$$\Pi_1 \backsimeq \left\{ \begin{array}{l} y = e; \\ \mathsf{if}\ y\ \mathsf{then} \\ \quad \Pi_3 \\ \mathsf{else} \\ \quad \Pi_4 \\ \mathsf{end}; \end{array} \right.$$

По лемме 3.29 на стр. 88

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi).$$

$$w(\Pi_1) = w(e) - 1 + w(\Pi_3) + w(\Pi_4) = w(\Pi) - 1 < w(\Pi).$$

3) Для неполного ветвления аналогично.

4) Пусть

$$\Pi \backsimeq \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_3 \\ \mathsf{end}; \end{array} \right.$$

Если $w(\Pi_3) > 0$, то утверждение следует из индукционного предположения. Пусть $w(\Pi_3) = 0$. Это означает, что $2w(e) > 0$ и $w(e) > 0$. Возьмём новую переменную y и построим Π_1:

$$\Pi_1 \backsimeq \left\{ \begin{array}{l} y = e; \\ \mathsf{while}\ y\ \mathsf{do} \\ \quad \Pi_3 \\ \quad y = e; \\ \mathsf{end}; \end{array} \right.$$

По лемме 3.31 на стр. 90

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi).$$

По определению веса

$$w(\Pi_1) = 2(w(e) - 1) + 2 \times 0 + w(\Pi_3) = 2w(e) + w(\Pi_3) - 2,$$

$$w(\Pi) = 2w(e) + w(\Pi_3).$$

Следовательно, $w(\Pi_1) < w(\Pi)$.

Итак, мы доказали, что если $w(\Pi) > 0$, то существует программа Π_1 такая, что $w(\Pi_1) < w(\Pi)$ и

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi).$$

для всех конфигураций σ.

Пусть дана исходная программа Π. Построим последовательность программ $(\Pi_i)_i$: $\Pi_0 \twoheadleftarrow \Pi$, Π_{i+1} получается из Π_i с помощью описанной выше процедуры, если $w(\Pi_i) > 0$. Последовательность $(w(\Pi_i))_i$ будет убывающей цепью натуральных чисел, следовательно, она не может быть бесконечной. С другой стороны, для последнего элемента Π_n не может быть $w(\Pi_n) > 0$. Следовательно, $w(\Pi_n) = 0$ и Π_n — простая программа. Так как

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_i(\sigma) \restriction \mathrm{Var}(\Pi)$$

для всех i, то

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_n(\sigma) \restriction \mathrm{Var}(\Pi). \qquad \square$$

***Задача 3.35.** Докажите все пропущенные места в доказательстве теоремы.

Следствие 3.33. *Каждый алгоритм эквивалентен некоторому простому алгоритму.*

Доказательство. Пусть дан произвольный алгоритм

```
algorithm A;
arguments x̄;
returns e;
    Π
end;
```

Пусть y — новая переменная, тогда алгоритм

algorithm A;
arguments $\bar{x}$;
returns y;
    П
    $y = e$;
end;

очевидно эквивалентен исходному. Далее остается только преобразовать тело нового алгоритма к простой программе. □

Задача 3.36. Преобразуйте тело алгоритма *Euclid* из примера 3.17 на стр. 72 в простую программу.

§ 3.5. *Подстановка

Мы рассмотрели язык программирования с очень ограниченным набором конструкций. В реальных ЯП набор операций гораздо шире. В частности, имеется полный набор математических функций. Основная задача этого параграфа — доказать, что подобное обогащение ЯП не приводит в увеличению его возможностей.

Теорема 3.34 (Теорема о подстановке)**.** *Пусть O — алгоритм на языке программирования A, который вычисляет некоторую n-местную функцию o. Пусть язык программирования B получен из языка программирования A добавлением новой операции o, означающей ту же самую функцию. Тогда для каждого алгоритма на B существует эквивалентный ему алгоритм на A.*

Доказательство. Пусть алгоритм, вычисляющий O, имеет вид:

algorithm O;
arguments $x_1, \ldots, x_n$;
returns d;
    П_O
end;

Рассмотрим произвольный алгоритм C на ЯП B:

$$\begin{array}{l}\textsf{algorithm } C;\\ \textsf{arguments } \bar{w};\\ \textsf{returns } r;\\ \quad \Pi^B\\ \textsf{end};\end{array}$$

Согласно теореме о простой программе, можно считать, что программа Π^B является простой, а r — переменная. В частности, это означает, что операция o встречается только в операторах присваивания вида

$$u = o(u_1, \ldots, u_n);$$

Заменим каждую переменную v в алгоритме O на новую переменную $\hat{v}$. Согласно следствию 3.26 на стр. 86, алгоритм будет вычислять ту же самую функцию. Обозначим тело нового алгоритма при помощи $\hat{\Pi}_O$, а выражение-результат — $\hat{d}$.

Теперь построим программу Π'_O следующим образом:

$$\begin{array}{l}\hat{x}_1 = y_1;\\ \vdots\\ \hat{x}_n = y_n;\\ \hat{\Pi}_O\end{array}$$

Здесь $y_1, \ldots, y_n$ — это новые попарно различные переменные. Тогда алгоритм

$$\begin{array}{l}\textsf{algorithm } O';\\ \textsf{arguments } y_1, \ldots, y_n;\\ \textsf{returns } \hat{d};\\ \quad \Pi'_O\\ \textsf{end};\end{array}$$

эквивалентен алгоритму O. В самом деле, рассмотрим результаты вызовов $\delta = (O, \bar{a})$ и $\delta' = (O', \bar{a})$. В результате выполнения первых n

операторов программы Π'_O мы получим конфигурацию τ' такую, что $\tau'\hat{x}_i = \sigma_{\delta'} y_i = \sigma_\delta x_i$. $\Pi_O(\sigma_\delta)$ определено тогда и только тогда когда $\Pi'_O(\tau')$ определено и в этом случае $\Pi_O(\sigma_\delta)(p) = \Pi_O(\tau')(\hat{p})$. Итак, O' действительно эквивалентен O. В частности, это означает, что $\Pi'_O(\sigma)(\hat{d}) = \sigma(o(\bar{y}))$.

Теперь заменим в программе Π^B каждый оператор присваивания вида

$$u = o(v_1, \ldots, v_n);$$

на

$$(\Pi'_O)^{y_1 \ldots y_n}_{v_1 \ldots v_n}\ u = \hat{d};$$

Назовём полученную программу Π^A. Очевидно, что операция o в Π^A не встречается, то есть, Π^A — программа на языке программирования A. Построим алгоритм на ЯП A:

algorithm C;
arguments $\bar{w}$;
returns r;
    Π^A
end;

Чтобы установить эквивалентность нового и старого алгоритмов нужно доказать, что для всякой начальной конфигурации σ выполнено

$$\Pi^B(\sigma)(r) = \Pi^A(\sigma)(r).$$

Индукцией по построению программы Π^B докажем следующее утверждение: для всяких конфигураций σ и τ таких, что

$$\sigma \restriction \mathrm{Var}(\Pi^B) = \tau \restriction \mathrm{Var}(\Pi^B),$$

имеет место

$$\Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B).$$

Базис индукции. Для пустого оператора это очевидно. Рассмотрим произвольный оператор присваивания. Как мы уже сказали выше, возможна одна из четырех ситуаций:

1) $\Pi^B \circ x = y$; x, y — переменные. Но тогда $\Pi^A \circ x = y; \circ \Pi^B$. Поэтому

$$\Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B) = \Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B),$$

так как переменная x изменяется одинаково и в Π^A, и в Π^B.

2) $\Pi^B \circ x = c$; x — переменная, c — константа. Снова $\Pi^A \circ \Pi^B$ и

$$\Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B) = \Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B).$$

3) $\Pi^B \circ x = f(z_1, \ldots, z_n)$; f — операция, $f \neq o$, $x, z_1, \ldots, z_n$ — переменные. Опять $\Pi^A \circ \Pi^B$ и

$$\Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B) = \Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B).$$

4) $\Pi^B \circ x = o(z_1, \ldots, z_n)$; $x, z_1, \ldots, z_n$ — переменные. Тогда

$$\Pi^A \circ (\Pi'_O)^{y_1 \ldots y_n}_{z_1 \ldots z_n}\, x = \hat{d};$$

Согласно лемме о замене переменных

$$(\Pi'_O)^{y_1 \ldots y_n}_{z_1 \ldots z_n}(\sigma)(\hat{d}) = \sigma(o(\bar{z})).$$

Для любых остальных переменных из Π^B программа $(\Pi'_O)^{y_1 \ldots y_n}_{z_1 \ldots z_n}$ не изменяет значение, так как они не входят в неё в левой части оператора присваивания. Это и означает, что

$$\Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B).$$

Шаг индукции. Возможны четыре варианта.

1) $\Pi^B \backsimeq \Pi_1^B \Pi_2^B$. Тогда $\Pi^A \backsimeq \Pi_1^A \Pi_2^A$ и по индукционному предположению

$$\Pi_1^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi_1^A(\tau) \restriction \mathrm{Var}(\Pi^B).$$

Пусть $\Pi_1^B(\sigma) = \sigma_1$ и $\Pi_1^A(\tau) = \tau_1$. По индукционному предположению

$$\Pi_2^B(\sigma_1) \restriction \mathrm{Var}(\Pi^B) = \Pi_2^A(\tau_1) \restriction \mathrm{Var}(\Pi^B).$$

То есть

$$\Pi_2^B(\Pi_1^B(\sigma)) \restriction \mathrm{Var}(\Pi^B) = \Pi_2^A(\Pi_1^A(\tau)) \restriction \mathrm{Var}(\Pi^B).$$

Но по определению семантики следования

$$\Pi_2^B(\Pi_1^B(\sigma)) = \Pi^B(\sigma); \quad \Pi_2^A(\Pi_1^A(\tau)) = \Pi^A(\tau).$$

Следовательно,

$$\Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B).$$

2)

$$\Pi^B \backsimeq \left\{ \begin{array}{l} \mathsf{if}\ x\ \mathsf{then} \\ \quad \Pi_1^B \\ \mathsf{else} \\ \quad \Pi_2^B \\ \mathsf{end}; \end{array} \right.$$

Тогда

$$\Pi^A \backsimeq \left\{ \begin{array}{l} \mathsf{if}\ x\ \mathsf{then} \\ \quad \Pi_1^A \\ \mathsf{else} \\ \quad \Pi_2^A \\ \mathsf{end}; \end{array} \right.$$

и по индукционному предположению

$$\Pi_1^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi_1^A(\tau) \restriction \mathrm{Var}(\Pi^B),$$

$$\Pi_2^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \Pi_2^A(\tau) \restriction \mathrm{Var}(\Pi^B).$$

Заметим, что поскольку x — переменная из $\mathrm{Var}(\Pi^B)$, то $\sigma(x) = \tau(x)$. По определению семантики ветвления имеем:

$$\begin{aligned}
&\Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B) = \\
&\qquad = \left\{ \begin{array}{ll} \Pi_1^B(\sigma) \restriction \mathrm{Var}(\Pi^B), & \text{если } \sigma(x) \neq 0 \\ \Pi_2^B(\sigma) \restriction \mathrm{Var}(\Pi^B), & \text{если } \sigma(x) = 0 \end{array} \right\} = \\
&\qquad = \left\{ \begin{array}{ll} \Pi_1^A(\tau) \restriction \mathrm{Var}(\Pi^B), & \text{если } \tau(x) \neq 0 \\ \Pi_2^A(\tau) \restriction \mathrm{Var}(\Pi^B), & \text{если } \tau(x) = 0 \end{array} \right\} = \\
&\qquad\qquad = \Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B).
\end{aligned}$$

3) Для неполного ветвления — аналогично.

4)
$$\Pi^B \twoheadleftarrow \left\{ \begin{array}{l} \mathsf{while}\ x\ \mathsf{do} \\ \quad \Pi_1^B \\ \mathsf{end}; \end{array} \right.$$

Тогда
$$\Pi^A \twoheadleftarrow \left\{ \begin{array}{l} \mathsf{while}\ x\ \mathsf{do} \\ \quad \Pi_1^A \\ \mathsf{end}; \end{array} \right.$$

и по индукционному предположению для любых конфигураций ρ и π таких, что

$$\rho \restriction \mathrm{Var}(\Pi^B) = \pi \restriction \mathrm{Var}(\Pi^B),$$

имеет место

$$\Pi_1^B(\rho) \restriction \mathrm{Var}(\Pi^B) = \Pi_1^A(\pi) \restriction \mathrm{Var}(\Pi^A).$$

Рассмотрим последовательности конфигураций для Π^B и для Π^A:

$$\sigma^0 = \sigma, \quad \sigma^{i+1} = \Pi_1^B(\sigma^i);$$

$$\tau^0 = \tau, \quad \tau^{i+1} = \Pi_1^A(\tau^i).$$

Покажем, что

$$\sigma^i \restriction \mathrm{Var}(\Pi^B) = \tau^i \restriction \mathrm{Var}(\Pi^B)$$

для всех i. Для $i = 0$ это следует из условия. Пусть для i доказано. Тогда

$$\sigma^{i+1} \restriction \mathrm{Var}(\Pi^B) = \Pi_1^B(\sigma^i) \restriction \mathrm{Var}(\Pi^B) = \\ = \Pi_1^A(\tau^i) \restriction \mathrm{Var}(\Pi^B) = \tau^{i+1} \restriction \mathrm{Var}(\Pi^B).$$

Так как переменная x — из $\mathrm{Var}(\Pi^B)$, то $\sigma^i(x) \neq 0$ тогда и только тогда, когда $\tau^i(x) \neq 0$. Следовательно, последовательность $(\sigma^i)_i$ конечна тогда и только тогда, когда последовательность $(\tau^i)_i$ конечна, и σ^m — последний элемент тогда и только тогда, когда τ^m — последний элемент. То есть, если $\Pi^A(\tau)$ определено, то $\Pi^B(\sigma)$ определено, и

$$\Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B) = \tau^m \restriction \mathrm{Var}(\Pi^B) = \\ = \sigma^m \restriction \mathrm{Var}(\Pi^B) = \Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B).$$

И, если $\Pi^B(\sigma)$ определено, то $\Pi^A(\tau)$ определено, и

$$\Pi^A(\tau) \restriction \mathrm{Var}(\Pi^B) = \Pi^B(\sigma) \restriction \mathrm{Var}(\Pi^B). \qquad \square$$

***Задача 3.37.** Обоснуйте индукционный шаг для неполного ветвления.

Значение доказанной теоремы заключается в следующем. Если мы расширяем язык программирования операциями, которые могут быть вычислены программами на этом языке, то множество функций, которые вычисляются алгоритмами на этом языке, не изменяется.

Например, мы уже видели, что на нашем язык программирования можно написать алгоритмы для сложения и (ограниченного)

вычитания. Следовательно, если мы добавим операции $+$ и $-$ к нашему языку, то его выразительная сила не увеличится. Точно так же, если мы напишем алгоритмы, которые будут вычислять умножение, деление, логарифм и т. д., то мы можем утверждать, что расширение ЯП за счёт этих операций не позволяет написать ничего принципиально нового.

Задача 3.38. Напишите алгоритмы для сложения и вычитания натуральных чисел. Напишите алгоритмы для умножения и деления с использованием операций «$+$» и «$-$». Воспользуйтесь методом, предложенным в теореме, чтобы построить алгоритмы для умножения и деления без использования «$+$» и «$-$».

Задача 3.39. Напишите алгоритмы для возведения чисел в степень и вычисления факториала, используя операцию умножения «$\times$». Воспользуйтесь методом, предложенным в теореме, чтобы построить алгоритмы для этих операций без использования «$\times$». Используйте алгоритмы, полученные при решении предыдущей задачи.

*__Задача 3.40.__ Покажите, что в качестве исходных операций для нашего языка программирования вместо succ и $<$ можно было взять

1) succ и $==$,
2) succ и $!=$,
3) succ и dec, $\mathsf{dec}(x) = \lfloor x - 1 \rfloor$.

*__Задача 3.41.__ Предметная область языка программирования $\mathfrak{L}$ — натуральные числа. В языке программирования имеется константа 1 и две операции — p: $p(n) = 2n$, dec (из задачи 3.40). Доказать, что в $\mathfrak{L}$ можно вычислить любую вычислимую функцию.

Указание. Показать, что функцию succ можно вычислить в таком языке программирования.

*__Задача 3.42.__ Предметная область языка программирования $\mathfrak{L}$ — натуральные числа. В языке программирования имеется константа 0 и операция succ. Доказать, что в $\mathfrak{L}$ нельзя вычислять сложение и вычитание с помощью программ без циклов.

Указание. Показать, если в программе нет циклов, то выходное значение каждой переменной либо ограничено некоторой константой, либо отличает-

ся от входного значения некоторой переменной не более чем на константу, зависящую только от программы.

***Задача 3.43.** Предметная область языка программирования $\mathfrak{L}$ — неотрицательные рациональные числа — $\mathbb{Q}_0$, операции — $+$; $-$; $\times$; $/$, константы — 0; 1. Доказать, что если функция f вычислима в языке программирования $\mathfrak{L}$, то функция f' вычислима в языке программирования с натуральными числами: $f'(n) = [f(n)]$

Указание. Каждую переменную заменить на две.

Глава 4

Программы с метками

§ 4.1. Синтаксис

До сих пор мы рассматривали структурированные программы. Теперь мы опишем иной подход к написанию программ. Исторически он является более ранним, однако, в настоящий момент его считают морально устаревшим, и он используется только в старых языках программирования (таких как Фортран, Бейсик, Фокал) и в языках программирования низкого уровня. Однако, основная конструкция этого подхода (переход к метке) остаётся практически во всех современных языках программирования, что позволяет применять такой принцип, когда это является оправданным.

Определение 4.1 (Метка)**.** *Метка — это имя, служащее для обозначения операторов программы.*

Синтаксис программ с метками следующий:

Определение 4.2 (Программа с метками)**.** *Программа с метками — последовательность операторов одного из двух следующих видов:*

algorithm Min;
arguments x, y;
returns y;
    1 if $x < y$ then 2 else 3
    2 $y = x$; 3
end;

Рис. 4.1: Наименьшее из двух чисел.

1) *оператор присваивания:*

   α $x = e$; β

   где x *— переменная,* e *— выражение,* α, β *— метки;*

2) *оператор условного перехода:*

   α if e then β else γ

   где e *— выражение,* α, β, γ *— метки.*

Здесь метка α *— это метка оператора, а метки* β *и* γ *— метки перехода. В программе не допускаются два оператора с одинаковыми метками оператора. Кроме того, мы для удобства будем предполагать, что имена меток не совпадают ни с именами переменных, ни с константами, ни с другими, используемыми в программе словами (*if, arguments, *. . .).*

Через Start(П), *где* П *— программа с метками, мы будем обозначать метку первого оператора программы* П *— начальную метку. Если метка перехода* α *не является меткой никакого оператора, то это — заключительная (или завершающая) метка.*

Пример 4.1. *Рассмотрим пример (рис. 4.1). Здесь* $\mathrm{Start}(\Pi_{Min}) = 1$ *— начальная метка.* 3 *— заключительная метка.*

§ 4.2. Семантика

Конфигурация программы с метками устроена сложнее, чем для структурированных программ, включая в себя одну из меток программы.

Определение 4.3 (Конфигурация)**.** *Конфигурация программы с метками* Π *— это пара* $(\sigma; \alpha)$*, где* σ *— состояние множества переменных* $\mathrm{Var}(\Pi)$*, а* α *— некоторая метка программы* Π*. С помощью* Σ^E_Π *мы обозначаем множество всевозможных конфигураций программы* Π*.*

Пример 4.2. *Рассмотрим программу из примера 4.1 на предшествующей странице. Следующие объекты будут конфигурациями:*

$$(\{(x,1);(y,4);(z,312)\},1),$$

$$(\{(x,2);(y,5);(z,32)\},3),$$

$$(\{(x,10);(y,8);(z,2)\},4).$$

Определение 4.4 (Семантика программы с метками)**.** *Сначала для всякой программы* Π *определим частичное отображение* $\Sigma^E_\Pi \to \Sigma^E_\Pi$*. Полагаем, что* $\Pi(\sigma,\alpha) = (\tau,\beta)$*, если существует натуральное число* n *и последовательность конфигураций* $(\sigma^i, \alpha_i)_{i=0}^n$ *такие, что*

1) $\sigma^0 = \sigma$; $\alpha_0 = \alpha$;
2) $\sigma^n = \tau$; $\alpha_n = \beta$;
3) *в программе* Π *есть операторы с метками* $\alpha_0, \ldots, \alpha_{n-1}$;
4) *в программе* Π *нет оператора с меткой* α_n *(то есть* α_n *— заключительная метка);*
5) *если для* $i < n$ *в* Π *имеется оператор вида*

$\alpha_i \quad x = e; \quad \gamma$

то $\alpha_{i+1} = \gamma$, *и* $\sigma^{i+1} = (x = e;)(\sigma^i)$;

6) *если для* $i < n$ *в* Π *имеется оператор вида*

$\alpha_i \quad$ if e then β else γ

то $\sigma^{i+1} = \sigma^i$, *а* α_{i+1} *равняется* β, *если* $\sigma^i(e) \neq 0$, *и* γ, *если* $\sigma^i(e) = 0$.

Если такого числа n *не существует, то считаем, что* $\Pi(\sigma, \alpha)$ *неопределено.*

Считаем, что $\Pi(\sigma) = \tau$, *если*

$$\Pi(\sigma, \mathrm{Start}(\Pi)) = (\tau, \beta)$$

для некоторой метки β *(необходимо заключительной). В противном случае* $\Pi(\sigma)$ *неопределено.*

Укажем несколько простейших утверждений, напрямую следующих из определения семантики.

Следствие 4.1. *Для каждой программы с метками* Π *существует программа с метками* Π', *которая содержит не более одной заключительной метки,* $\mathrm{Var}(\Pi') = \mathrm{Var}(\Pi)$ *и* $\Pi(\sigma) = \Pi'(\sigma)$ *для всякого состояния* σ *переменных* $\mathrm{Var}(\Pi)$.

Задача 4.1. Докажите следствие.

Как и в случае структурированных программ, если $\Pi(\sigma)$ не определено, то говорим, что Π на σ зацикливается.

Следствие 4.2 (Условие зацикливания)**.** *Пусть* Π — *программа с метками,* (σ, α) — *ее конфигурация. Тогда* $\Pi(\sigma, \alpha)$ *определено в том и только том случае, когда последовательность конфигураций, построенная с помощью пунктов 1, 5 и 6 определения семантики программ с метками, является конечной.*

Следующее следствие доказывается аналогично доказанному ранее следствию о зацикливании структурированных программ.

Следствие 4.3 (Условие зацикливания)**.** *Если в последовательности конфигураций, построенной с помощью пунктов 1, 5 и 6, есть два одинаковых, то* $\Pi(\sigma, \alpha)$ *неопределено.*

***Задача 4.2.** Докажите оба следствия.

Пример 4.3. *Рассмотрим алгоритм на рис. 4.1 на стр. 107. Возьмем состояние* $\sigma_1 = \{(x,1),(y,2),(z,3)\}$*. Тогда*

$$\Pi(\sigma_1, 3) = (\underbrace{\{(x,1),(y,2),(z,2)\}}_{\sigma_3}; 4)$$

Последовательность: $(\sigma_1, 3), (\sigma_3, 4)$.

$$\Pi(\sigma_1, 2) = (\underbrace{\{(x,1),(y,2),(z,1)\}}_{\sigma_2}; 4)$$

Последовательность: $(\sigma_1, 2), (\sigma_2, 4)$.

$\Pi(\sigma_1, 1) = (\sigma_2, 4)$*, потому что* $\sigma_1(x < y) \neq 0$*. Таким образом,* $\Pi(\sigma_1) = \sigma_2$.

Определение 4.5 (Числа Фибоначчи)**.** *Числа Фибоначчи* F_i, $i \in \omega$ *определяются по индукции:*

$$F_0 = 0,\ F_1 = 1,\ F_{i+1} = F_i + F_{i-1},$$

при $i \geqslant 1$.

Первые числа Фибоначчи:

n	0	1	2	3	4	5	6	7	8	9	10	11	...
F_n	0	1	1	2	3	5	8	13	21	34	55	89	...

Задача 4.3. Найдите все числа Фибоначчи, которые меньше 1000.

Пример 4.4. *На рис. 4.2 на противоположной странице изображён алгоритм для вычисления числа Фибоначчи по его номеру, тело которого является программой с метками.*

```
algorithm Fib;
arguments i;
returns f;
    1   if i < succ(succ(0)) then 2 else 3
    2   f = i;   15
    3   j = succ(0);   4
    4   f = j;   5
    5   p = 0;   6
    6   t = 0;   7
    7   s = f;   8
    8   if t < p then 9 else 11
    9   t = succ(t);   10
    10  s = succ(s);   8
    11  p = f;   12
    12  f = s;   13
    13  j = succ(j);   14
    14  if j < i then 6 else 15
end;
```

Рис. 4.2: Числа Фибоначчи.

Задача 4.4. Постройте последовательность конфигураций при вызове $(Fib, 3)$.

Докажем, что алгоритм Fib и в самом деле вычисляет числа Фибоначчи. Предположим, что в начальной конфигурации $(\sigma, 1)$ выполнено $\sigma i = i_0$.

Если $i_0 < 2$, то последовательность конфигураций содержит всего три элемента: $(\sigma, 1)$, $(\sigma, 2)$, $(\tau, 100)$, где τ совпадает с σ, кроме того, что $\tau(Fib) = \sigma i = i_0$. Но $F_n = n$ для $n < 2$, следовательно, $Fib(\sigma)(Fib) = F_{i_0}$.

Допустим, что $i_0 \geqslant 2$. Первые пять конфигураций таковы: $(\sigma, 1)$, $(\sigma, 3)$, $(\sigma_3, 4)$, $(\sigma_4, 5)$, $(\sigma_5, 6)$, где:

- $\sigma_3 = \sigma$ кроме того, что $\sigma_3 j = 1$;
- $\sigma_4 = \sigma_3$ кроме того, что $\sigma_4 f = 1$;
- $\sigma_5 = \sigma_4$ кроме того, что $\sigma_5 p = 0$.

Утверждение 4.4. *В последовательности конфигураций после любого элемента вида* $(\sigma^k, 6)$ *имеется элемент вида* $(\sigma^l, 14)$. *Если взять наименьшее из таких* l, *то*

$$\sigma^l p = \sigma^k f, \quad \sigma^l f = \sigma^k p + \sigma^k f, \quad \sigma^l j = \sigma^k j + 1.$$

Доказательство. После $(\sigma^k, 6)$ должны идти $(\sigma^{k+1}, 7)$ и $(\sigma^{k+2}, 8)$ такие, что $\sigma^{k+1} t = 0$ и $\sigma^{k+2} s = \sigma^k f$.

Затем должна идти последовательность из 0 или более троек вида

$$(\sigma^{k+3m}, 9), \quad (\sigma^{k+3m+1}, 10), \quad (\sigma^{k+3m+2}, 8)$$

таких, что

$$\sigma^{k+3m+2} t = \sigma^{k+3(m-1)+2} t + 1; \quad \sigma^{k+3m+2} s = \sigma^{k+3(m-1)+2} s + 1.$$

Так как $\sigma^{k+3*0+2} t = 0$ и $\sigma^{k+3*0+2} s = f$, то, очевидно,

$$\sigma^{k+3m+2} t = \sigma^{k+3(m-1)+2} t + 1 = \cdots = \sigma^{k+3(m-m)+2} t + m = m$$
$$\sigma^{k+3m+2} s = \sigma^{k+3(m-1)+2} s + 1 = \cdots = \sigma^{k+3(m-m)+2} s + m = \sigma^k f + m.$$

Очевидно, что количество этих троек должно быть равно $\sigma^k p$, так как для $m = \sigma^k p$

$$\sigma^{k+3m+2} t = \sigma^{k+3m+2} p,$$

то есть значение выражения $t < p$ равно нулю.

Возьмём $l = k + 3\sigma^k p + 6$. Тогда после этой последовательности троек будут идти

$$(\sigma^{l-3}, 11),\ (\sigma^{l-2}, 12),\ (\sigma^{l-1}, 13),\ (\sigma^l, 14).$$

Таким образом, мы действительно выбрали наименьшее из l больших k, для которых метка — 14. Рассмотрим σ^l:

$$\begin{aligned}
&\sigma^l p = \sigma^{l-4} f = \cdots = \sigma^k f \\
&\sigma^l f = \sigma^{l-4} s = \sigma^{k+3\sigma^k p+2} s = \sigma^k f + \sigma^k p \\
&\sigma^l j = \sigma^k j + 1
\end{aligned}$$

□

Следствие 4.5. *Для любой конфигурации вида* $(\sigma^k, 6)$ *такой, что* $\sigma^k f = F_{\sigma^k j}$ *и* $\sigma^k p = F_{\sigma^k j - 1}$, *существует конфигурация* $(\sigma^l, 14)$ *такая, что* $\sigma^l j = \sigma^k j + 1$, $\sigma^l f = F_{\sigma^l j}$ *и* $\sigma^l p = F_{\sigma^l j - 1}$.

Доказательство. Выберем l из утверждения.

$$\begin{aligned}
&\sigma^l f = \sigma^k f + \sigma^k p = F_{\sigma^k j} + F_{\sigma^k j - 1} = F_{\sigma^k j + 1} = F_{\sigma^l j}. \\
&\sigma^l p = \sigma^k f = F_{\sigma^k j} = F_{\sigma^l j - 1}.
\end{aligned}$$

□

Утверждение 4.6. *Алгоритм Fib вычисляет* F_{i_0}.

Доказательство. Покажем следующее: для всякого $2 \leqslant n \leqslant i_0$ существует конфигурация $(\sigma^{l_n}, 14)$ такая, что

$$\sigma^{l_n} j = n,\ \sigma^{l_n} p = F_{n-1},\ \sigma^{l_n} f = F_n.$$

Базис индукции. $n = 2$. Мы уже видели, что

$$\sigma^5 p = 0 = F_{\sigma^5 j - 1},\ \sigma^5 f = 1 = F_{\sigma^5 j}.$$

Согласно следствию, существует номер l_2 такой, что

$$\sigma^{l_2} j = \sigma^5 j + 1 = 2,\ \ \sigma^{l_2} p = F_{\sigma^{l_2} j - 1} = F_1,\ \ \sigma^{l_2} f = F_{\sigma^{l_2} j} = F_2.$$

Шаг индукции. Пусть для n доказано и $n < i_0$. Тогда

$$\sigma^{l_n} j = n < i_0 = \sigma^{l_n} i.$$

Значит, следом за конфигурацией $(\sigma^{l_n}, 14)$ идет $(\sigma^{l_n}, 6)$ и для нее выполнены все условия следствия. Следовательно, существует l_{n+1} такой, что

$$\sigma^{l_{n+1}} j = \sigma^{l_n} j + 1 = n + 1,$$
$$\sigma^{l_{n+1}} p = F_{\sigma^{l_{n+1}} j - 1} = F_n,$$
$$\sigma^{l_{n+1}} f = F_{\sigma^{l_{n+1}} j} = F_{n+1}.$$

То есть утверждение выполняется и для $n + 1$.

Если $n = i_0$, то после $(\sigma^{l_n}, 14)$ идёт $(\sigma^{l_n}, 15)$. Следовательно, результатом вызова будет $\sigma^{l_n}(f) = F_{i_0}$. □

Рассмотрим ещё один пример.

Пример 4.5. *Написать программу с метками, которая проверяет, можно ли из одиннадцатеричной записи натурального числа x отбросить несколько последних цифр, так чтобы оно стало квадратом (рис. 4.3 на следующей странице).*

Поскольку основные принципы мы подробно разобрали в предыдущем примере, то доказательства для этой программы мы дадим более кратко.

Утверждение 4.7. *Пусть σ — произвольное состояние, такое что $\sigma u = 1$, $\sigma y > 0$. Пусть σ^i — состояние, полученное из σ после i «итераций цикла» в строках 6–7. Тогда $\sigma^i u = i + 1$, $\sigma y > i^2$.*

Доказательство. *Индукция по i.*

Базис индукции. Пусть $i = 0$. Тогда $\sigma^0 = \sigma$. По условию $\sigma^0 u = 0 + 1$, $\sigma^0 y > 0$.

Шаг индукции. Пусть для i утверждение верно. По индукционному предположению $\sigma^i u = i + 1$. Так как σ^{i+1} существует, то тест в строке 6 истинен, то есть $\sigma y > (i + 1)^2$. В строке 7 переменная u увеличивается на 1, поэтому $\sigma^{i+1} u = \sigma^i u + 1 = i + 2$. □

Утверждение 4.8. *Пусть σ — произвольное состояние, такое что $\sigma y > 0$. Пусть состояние τ получается из σ после срабатывания строк 5–7. Тогда $\tau(u \times u == y) = 1$ тогда и только тогда, когда σy является полным квадратом.*

```
algorithm A;
arguments x;
returns r;
    1    y = x;   2
    2    y = y / 11;   3
    3    if y == 0 then 4 else 5
    4    r = 0;   100
    5    u = 1;   6
    6    if u × u < y then 7 else 8
    7    u = u + 1;   6
    8    if u × u == y then 9 else 2
    9    r = 1;   100
end;
```

Рис. 4.3: Отбрасывание последних цифр, программа с метками

Доказательство. *После строки 5 переменная u принимает значение 1. Пусть «цикл» в строках 6–7 прервётся после n «итераций», то есть $\tau = \sigma^n$. По утверждению 4.7 на противоположной странице выполняется $\tau u = n + 1$, $\tau y > n^2$ (так как y не меняется). Поскольку «цикл» прервался, то $\tau y \leqslant (\tau u)^2$. Итак, $(\tau u - 1)^2 < \tau y \leqslant (\tau u)^2$. Если $\tau y = (\tau u)^2$, то τy является полным квадратом. Если же $\tau y \neq (\tau u)^2$, то τy лежит между двумя соседними полными квадратами и, следовательно, не является полным квадратом.* □

Утверждение 4.9. *Для любого вызова $\delta = (A, x_0)$*

$$\operatorname{Res}\delta = \begin{cases} 1, & \text{если из одиннадцатеричной записи числа } x_0 \text{ можно} \\ & \text{отбросить несколько последних цифр так,} \\ & \text{чтобы оно стало квадратом;} \\ 0, & \text{иначе.} \end{cases}$$

Доказательство. *Пусть σ — начальная конфигурация вызова δ. Пусть $(\sigma^i)_i$ — последовательность состояний при работе «цикла» в строках 2–9. Индукцией по i докажем, что $\sigma^i y = \left[\sigma x / 11^i\right]$ и ни одно из чисел $\left[\sigma x / 11^j\right]$ не является полным квадратом для $1 \leqslant j \leqslant i$.*

Базис индукции. Пусть $i = 0$. Тогда после строки 1 $\sigma^0 y = \sigma x = \left[\sigma x / 11^0\right]$. Второе условие также выполняется, так как не существует числа j такого, что $1 \leqslant j \leqslant 0$.

Шаг индукции. Пусть для i утверждение верно. Обозначим через τ состояние, получающееся из σ^i после присваивания в строке 2. По индукционному предположению $\sigma^i y = \left[\sigma x / 11^i\right]$, поэтому $\tau y = \left[\left[\sigma x / 11^i\right] / 11\right] = \left[\sigma x / 11^{i+1}\right]$. Так как далее на текущей «итерации» y не меняется, то $\sigma^{i+1} y = \left[\sigma x / 11^{i+1}\right]$. Так как σ^{i+1} существует, то $\tau y > 0$. Обозначим через ρ состояние, которое получится из τ после срабатывания строк 5–7. Тогда $\rho(u \times u == y) = 0$, иначе состояние σ^i было бы последним элементом последовательности. По утверждению 4.8 на стр. 114 τy не является полным квадратом. Используя индукционное предположение, получаем, что числа $\left[\sigma x / 11^j\right]$ не являются полными квадратами для $1 \leqslant j \leqslant i + 1$.

Пусть из числа x_0 можно отбросить несколько последних цифр, так чтобы оно стало квадратом. Пусть k — наименьшее количество цифр, которые необходимо отбросить. Тогда состояние σ^{k-1} существует. На k-й «итерации» как и выше будет получено состояние ρ. Так как число $\rho y = \left[\sigma x / 11^k\right]$ является полным квадратом, то по утверждению 4.8 на стр. 114 $\rho(u \times u == y) = 1$. После этого в строке 9 переменная r получит значение 1.

Пусть из числа x_0 нельзя отбросить несколько цифр, так чтобы оно стало квадратом. Допустим, x_0 содержит k цифр. Тогда в конфигурации σ^{k-1} мы получим $\sigma^{k-1} y = \left[\sigma x / 11^{k-1}\right]$. Последнее число состоит из одной цифры. На следующей «итерации» переменная y получит значение 0 в строке 2. Тест в строке 3 окажется истинным, поэтому переменная r получит значение 0 в строке 4. □

Если сравнить доказательства для программ с метками с доказательствами для структурированных программ, то нельзя не заметить, что первые являются значительно более сложными. Это связано с тем, что программы с метками не имеют формально выра-

женных логических частей. В случае структурированной программы мы доказывали её правильность двигаясь от отдельных операторов и постепенно рассматривая всё более крупные блоки. Каждый такой шаг делался согласно определению семантики для соответствующей конструкции. В случае же программы с метками такой метод не работает, мы должны рассматривать всю программу сразу, целиком. Это затрудняет понимание того, как работает программа и доказательство её правильности. Именно то обстоятельство, что программы с метками не имеют чётко выраженных логических блоков, и привело к тому, что они были вытеснены в значительной мере языками структурного программирования.

Задача 4.5. Напишите алгоритм, который подсчитывает сумму цифр заданного числа в десятичной системе счисления, и докажите его правильность. Используйте операции «+», «−», «×», «/» — целочисленное деление.

§ 4.3. Построение программ с метками

Мы рассмотрели два языка программирования — структурированный и с метками. Мы докажем, что оба этих языка эквивалентны, то есть функции, которые можно вычислить с помощью структурированных программ, можно вычислить и с помощью программ с метками, и наоборот.

Мы ранее доказывали леммы о замене переменных, которые утверждали, что если одни переменные заменить другими, то получится эквивалентная программа. Докажем аналогичное утверждение и для меток.

Замена в программе П метки α на β определяется точно так же как замена одной переменной на другую. Результат такой замены обозначается аналогично:

$$(\text{П})^{\alpha}_{\beta}$$

То есть $(\Pi)^{\alpha}_{\beta}$ получается из Π заменой всюду метки α на метку β.

Замечание 4.1. Результат замены $(\Pi)^{\alpha}_{\beta}$ может не быть программой с метками так как в результате замены в программе могут оказаться две одинаковых метки оператора. Однако, если метка β не является меткой оператора программы Π, то $(\Pi)^{\alpha}_{\beta}$ — программа с метками.

Лемма 4.10 (О замене метки)**.** *Пусть метка β не встречается в программе Π. Тогда для любого состояния σ переменных* $\mathrm{Var}(\Pi)$ *и для любой метки α имеет место* $\Pi(\sigma) = (\Pi)^{\alpha}_{\beta}(\sigma)$.

Доказательство. Докажем следующее утверждение: если $(\Pi)^{\alpha}_{\beta}(\sigma)$ определено, то $\Pi(\sigma)$ определено и $(\Pi)^{\alpha}_{\beta}(\sigma) = \Pi(\sigma)$.

Рассмотрим последовательность конфигураций $(\sigma^j, \lambda_j)_j$ для $(\Pi)^{\alpha}_{\beta}(\sigma)$. Так как $(\Pi)^{\alpha}_{\beta}(\sigma)$ определено, то последовательность $(\sigma^j, \lambda_j)_j$ является конечной. Пусть (σ^m, λ_m) — последняя конфигурация. Это означает, что λ_m — заключительная метка программы $(\Pi)^{\alpha}_{\beta}$. Определим

$$\mu_i \Lsh \begin{cases} \lambda_i, & \text{если } \lambda_i \neq \beta, \\ \alpha, & \text{если } \lambda_i \Lsh \beta. \end{cases}$$

Докажем, что последовательность конфигураций $(\sigma^j, \mu_j)_{j=0}^{m}$ является последовательностью конфигураций при работе Π на σ. Используем индукцию по длине последовательности.

Базис индукции. $i = 0$. Что σ^0 является начальным состоянием — очевидно. С метками возможны две ситуации:

1) $\lambda_0 \neq \beta$ и $\mu_0 = \lambda_0$. Так как $\lambda_0 = \mathrm{Start}((\Pi)^{\alpha}_{\beta})$, то $\mu_0 \Lsh \lambda_0$ должна быть и начальной меткой программы Π, потому что мы заменяем только метку α на β.

2) $\lambda_0 \Lsh \beta$. Тогда $\mu_0 \Lsh \alpha$, потому что метки β в программе Π отсутствуют, они могут появиться в $(\Pi)^{\alpha}_{\beta}$ только в результате замены. Значит, $\mathrm{Start}(\Pi) \Lsh \alpha \Lsh \mu_0$.

Это доказывает базис утверждения.

Ш а г и н д у к ц и и . Пусть для i доказано, что последовательность конфигураций $(\sigma^j, \mu_j)_{j=0}^{i}$ является правильной и $i < m$. Рассмотрим $i+1$. Возможны следующие варианты:

1) В программе $(\Pi)_\beta^\alpha$ есть оператор вида

   $\lambda_i \quad x = e; \quad \lambda_{i+1}$

   Тогда в программе Π есть оператор

   $\mu_i \quad x = e; \quad \mu_{i+1}$

   σ^{i+1} отличается от σ^i только тем, что $\sigma^{i+1}x = \sigma^i(e)$. Это относится и к $(\Pi)_\beta^\alpha$, и к Π. Следующей меткой для Π должна быть μ_{i+1}. Значит, $(\sigma^{i+1}, \mu_{i+1})$ — очередной элемент.

2) В программе $(\Pi)_\beta^\alpha$ есть оператор

   $\lambda_i \quad \mathsf{if}\ e\ \mathsf{then}\ \lambda_{i+1}\ \mathsf{else}\ \delta$

   и $\sigma^i(e) \neq 0$. Тогда в программе Π есть оператор

   $\mu_i \quad \mathsf{if}\ e\ \mathsf{then}\ \mu_{i+1}\ \mathsf{else}\ \delta'$

   Поэтому следующим элементом последовательности $(\sigma^j, \mu_j)_j$ будет $(\sigma^{i+1}, \mu_{i+1})$, где $\sigma^{i+1} = \sigma^i$.

3) В программе $(\Pi)_\beta^\alpha$ есть оператор

   $\lambda_i \quad \mathsf{if}\ e\ \mathsf{then}\ \delta\ \mathsf{else}\ \lambda_{i+1}$

   и $\sigma^i(e) = 0$. Тогда в программе Π есть оператор

   $\mu_i \quad \mathsf{if}\ e\ \mathsf{then}\ \delta'\ \mathsf{else}\ \mu_{i+1}$

   Следовательно, за элементом (σ^i, μ_i) должен идти $(\sigma^{i+1}, \mu_{i+1})$, где $\sigma^{i+1} = \sigma^i$.

Рассмотрим μ_m. Если в Π существует метка оператора μ_m, то в программе $(\Pi)_\beta^\alpha$ должна быть метка оператора λ_m. Но последнее не выполняется, так как λ_m — заключительная метка. Значит, μ_m является заключительной меткой программы Π. Таким образом,

последовательность $(\sigma^j, \mu_j)_j$ заканчивается конфигурацией (σ^m, μ_m). То есть $\Pi(\sigma)$ определено и

$$\Pi(\sigma) = \sigma^m = (\Pi)^{\alpha}_{\beta}(\sigma).$$

Это доказывает наше утверждение.

Доказать утверждение в обратную сторону легко, если заметить, что $\Pi \fallingdotseq ((\Pi)^{\alpha}_{\beta})^{\beta}_{\alpha}$ и при этом метка α не встречается в программе $(\Pi)^{\alpha}_{\beta}$. Согласно только что доказанному, мы получаем: если $((\Pi)^{\alpha}_{\beta})^{\beta}_{\alpha}(\sigma)$ определено, то $(\Pi)^{\alpha}_{\beta}(\sigma)$ тоже будет определено и при этом $((\Pi)^{\alpha}_{\beta})^{\beta}_{\alpha}(\sigma) = (\Pi)^{\alpha}_{\beta}(\sigma)$. А так как $\Pi \fallingdotseq ((\Pi)^{\alpha}_{\beta})^{\beta}_{\alpha}$, то это и означает следование в обратную сторону. □

Определение 4.6 (Эффективное построение)**.** *Мы скажем, что по объекту A из некоторого множества можно э ф ф е к т и в н о построить объект B, если существует алгоритм, который выполняет такое построение для любого объекта A из этого класса.*

Пример 4.6. *Например, замена $b \fallingdotseq (a)^{x}_{y}$ — может быть эффективно построена, потому что существует алгоритм, выполняющий такую замену для любого слова a. В общих словах алгоритм замены следующий: идти по слову a и копировать все символы в слово b, кроме символа x, вместо которого копируется y.*

Замечание 4.2. Не всякое построение эффективно. Например, для всякого алгоритма A существует алгоритм M_A, который имеет минимальную длину среди всех эквивалентных A алгоритмов. Однако построение M_A по A неэффективно. Не существует алгоритма, который выполнял бы это построение для любого алгоритма A.

Теорема 4.11 (О построении программы с метками)**.** *По всякой структурированной программе Π^S, содержащей хотя бы одну переменную x, можно эффективно построить программу с метками Π^L такую, что $\mathrm{Var}(\Pi^S) = \mathrm{Var}(\Pi^L)$ и $\Pi^S(\sigma) = \Pi^L(\sigma)$ для любой конфигурации σ программы Π^S.*

Доказательство. Докажем теорему индукцией по построению Π^S.

Базис индукции. Для пустого оператора $\Pi^S \twoheadleftarrow$; берем

$$\Pi^L \twoheadleftarrow 1 \quad x = x;\ 2.$$

Для оператора присваивания

$$\Pi^S \twoheadleftarrow x = e;$$

где x — переменная, e — выражение, возьмём

$$\Pi^L \twoheadleftarrow 1 \quad x = e;\ 2.$$

Очевидно, что и в том и в другом случае для всякого состояния σ переменных $\mathrm{Var}(\Pi^S)$ имеет место $\Pi^S(\sigma) = \Pi^L(\sigma)$.

Шаг индукции. Возможны четыре варианта.

1) $\Pi^S \twoheadleftarrow \Pi_1^S \Pi_2^S$. По индукции для Π_1^S и для Π_2^S существуют программы Π_1^L и Π_2^L. Согласно следствию 4.1 на стр. 109, считаем, что Π_1^L имеет единственную заключительную метку γ. Выберем для каждой метки α из программы Π_2^L метку $\hat{\alpha}$, которая не встречалась бы ни в Π_1^L ни в Π_2^L. Построим $\tilde{\Pi}_2^L$, заменив в Π_2^L все метки α на $\hat{\alpha}$. Пусть δ — начальная метка $\tilde{\Pi}_2^L$, $\tilde{\Pi}_1^L \twoheadleftarrow (\Pi_1^L)_\delta^\gamma$. Построим $\Pi^L \twoheadleftarrow \tilde{\Pi}_1^L \tilde{\Pi}_2^L$. Докажем, что Π^L — искомая программа.

   Пусть $\Pi^S(\sigma)$ определено. Тогда $\Pi_1^S(\sigma)$ определено, допустим, $\Pi_1^S(\sigma) = \tau$, и $\Pi_2^S(\tau)$ определено, допустим, $\Pi_2^S(\tau) = \rho$. Согласно индукционному предположению $\Pi_1^L(\sigma) = \Pi_1^S(\sigma) = \tau$. Согласно лемме о замене метки $\tilde{\Pi}_1^L(\sigma) = \Pi_1^L(\sigma) = \tau$. Согласно индукционному предположению $\Pi_2^L(\tau) = \Pi_2^S(\tau) = \rho$. Согласно лемме о замене метки $\tilde{\Pi}_2^L(\tau) = \Pi_2^L(\tau) = \rho$. Рассмотрим последовательность конфигураций для $\Pi^L(\sigma)$. Так как $\tilde{\Pi}_1^L(\sigma)$ определено, то последовательность конфигураций $(\sigma^i, \lambda_i)_i$ для $\tilde{\Pi}_1^L(\sigma)$ является конечной. Пусть (σ^m, λ_m) — её последний элемент. Тогда

$\sigma^m = \tau$ и $\lambda_m = \delta$, так как δ — заключительная метка программы $\tilde{\Pi}_1^L$. Заметим, что $(\sigma^m, \lambda_m) = (\tau, \delta)$ является первой конфигурацией для $\tilde{\Pi}_2^L(\tau)$. Так как $\tilde{\Pi}_2^L(\tau)$ определено, то эта последовательность должна быть конечной: $(\sigma^i, \lambda_i)_{i=m}^n$. Очевидно, что $\sigma^n = \rho$. Рассмотрим последовательность $(\sigma^i, \lambda_i)_{i=0}^n$. Эта последовательность является последовательностью конфигураций для $\Pi^L(\sigma)$. Так как заключительная метка программы Π^L совпадает с заключительной меткой программы $\tilde{\Pi}_2^L$, то $\Pi^L(\sigma) = \sigma^n = \rho$. Итак, $\Pi^S(\sigma) = \Pi^L(\sigma)$.

Прежде чем доказывать утверждение в обратную сторону отметим следующее. Единственная метка из $\tilde{\Pi}_2^L$, которая встречается в $\tilde{\Pi}_1^L$, — это δ. Но δ является начальной меткой $\tilde{\Pi}_2^L$, то есть меткой оператора. А заключительная метка не может являться меткой оператора.

Пусть теперь $\Pi^L(\sigma)$ определено. Рассмотрим последовательность конфигураций $(\sigma^i, \lambda_i)_{i=0}^n$. Метка λ_n должна быть заключительной меткой программы Π^L. Но единственной заключительной меткой программы Π^L является заключительная метка программы $\tilde{\Pi}_2^L$. Тогда λ_{n-1} должна быть меткой оператора программы $\tilde{\Pi}_2^L$, потому что сама заключительная метка программы $\tilde{\Pi}_2^L$ в программе $\tilde{\Pi}_1^L$, как мы отметили выше, встречаться не может. Итак, метка λ_0 является меткой оператора программы $\tilde{\Pi}_1^L$, а метка λ_{n-1} — меткой оператора $\tilde{\Pi}_2^L$. Следовательно, в последовательности должна быть конфигурация (σ^m, λ_m) такая, что все λ_i при $i < m$ — метки операторов программы $\tilde{\Pi}_1^L$, а сама λ_m — метка оператора программы $\tilde{\Pi}_2^L$. Но единственной меткой, которая удовлетворяет этим условиям является δ.

Итак, рассмотрим последовательность $(\sigma^i, \lambda_i)_{i=0}^m$. Тогда она является последовательностью конфигураций для $\tilde{\Pi}_1^L(\sigma)$ и

$\lambda_m = \delta$ — заключительная метка $\tilde{\Pi}_1^L$. Значит, $\tilde{\Pi}_1^L(\sigma) = \sigma^m$. По индукционному предположению

$$\Pi_1^S(\sigma) = \Pi_1^L(\sigma) = \tilde{\Pi}_1^L(\sigma) = \sigma^m.$$

Теперь рассмотрим последовательность $(\sigma^i, \lambda_i)_{i=m}^n$. Тогда она является последовательностью конфигураций для $\tilde{\Pi}_2^L(\sigma^m)$. Поэтому $\tilde{\Pi}_2^L(\sigma^m) = \sigma^n$. По индукционному предположению

$$\Pi_2^S(\sigma^m) = \Pi_2^L(\sigma^m) = \tilde{\Pi}_2^L(\sigma^m) = \sigma^n.$$

Получаем, что

$$\Pi^S(\sigma) = \Pi_2^S(\Pi_1^S(\sigma)) = \Pi_2^S(\sigma^m) = \sigma^n.$$

таким образом, $\Pi^S(\sigma)$ определено и $\Pi^S(\sigma) = \Pi^L(\sigma)$.

2)

$$\Pi^S \Leftarrow \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1^S \\ \mathsf{else} \\ \quad \Pi_2^S \\ \mathsf{end}; \end{array} \right.$$

По индукционному предположению для Π_1^S и для Π_2^S существуют программы Π_1^L и Π_2^L. Построим $\tilde{\Pi}_2^L$ так же, как и в предыдущем случае. Пусть γ — заключительная метка программы Π_1^L, а δ — заключительная метка $\tilde{\Pi}_2^L$. Пусть $\tilde{\Pi}_1^L \Leftarrow (\Pi_1^L)_\delta^\gamma$. Пусть α — новая метка, которая не встречается ни в $\tilde{\Pi}_1^L$ ни в $\tilde{\Pi}_2^L$. Возьмём

$$\Pi^L \Leftarrow \left\{ \begin{array}{ll} \alpha & \mathsf{if}\ e\ \mathsf{then}\ \mathrm{Start}(\tilde{\Pi}_1^L)\ \mathsf{else}\ \mathrm{Start}(\tilde{\Pi}_2^L) \\ & \tilde{\Pi}_1^L \\ & \tilde{\Pi}_2^L \end{array} \right.$$

Возможны два случая: либо $\sigma(e) \neq 0$, либо $\sigma(e) = 0$. Рассмотрим первый случай.

Пусть $\Pi^S(\sigma)$ определено. По индукционному предположению, $\tilde{\Pi}_1^L(\sigma)$ определено и $\tilde{\Pi}_1^L(\sigma) = \Pi_1^S(\sigma) = \tau$. Значит, существует последовательность конфигураций $(\sigma^i, \lambda_i)_{i=0}^n$, для которой $\sigma^0 = \sigma$, $\lambda_0 = \mathrm{Start}(\tilde{\Pi}_1^L)$, $\sigma^n = \tau$ и $\lambda_n = \delta$. Рассмотрим последовательность конфигураций, полученную из предыдущей добавлением в начало конфигурации (σ, α): $(\sigma, \alpha), (\sigma^i, \lambda_i)_{i=0}^n$. Легко показать, что это — последовательность конфигураций для $\Pi^L(\sigma)$. Кроме того, δ — заключительная метка Π^L, поэтому $\Pi^L(\sigma) = \sigma^n = \tau = \Pi^S(\sigma)$.

Пусть теперь $\Pi^L(\sigma)$ определено. Следовательно, последовательность конфигураций $(\sigma^i, \lambda_i)_i$ для $\Pi^L(\sigma)$ является конечной. Предположим, она содержит $n+1$ элемент. Очевидно, что $\lambda_1 = \mathrm{Start}(\tilde{\Pi}_1^L)$. Последовательность $(\sigma^i, \lambda_i)_{i=1}^n$ является последовательностью конфигураций для $\tilde{\Pi}_1^L(\sigma)$, так как метки операторов программы $\tilde{\Pi}_2^L$ в $\tilde{\Pi}_1^L$ не встречаются. Тогда последовательность

$$(\sigma^i, \lambda_i)_{i=1}^{n-1}, (\sigma^n, \gamma)$$

является последовательностью конфигураций для $\Pi_1^L(\sigma)$. По индукционному предположению $\Pi_1^S(\sigma)$ определено и $\Pi_1^S(\sigma) = \sigma^n$. То есть

$$\Pi^S(\sigma) = \sigma^n = \Pi^L(\sigma).$$

Второй случай, когда $\sigma(e) = 0$ рассматривается аналогично, только вместо $\Pi_1^S, \Pi_1^L, \tilde{\Pi}_1^L$ рассматриваются $\Pi_2^S, \Pi_2^L, \tilde{\Pi}_2^L$.

3) Для неполного ветвления построение аналогично.

4) Пусть

$$\Pi^S \Leftarrow \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1^S \\ \mathsf{end}; \end{array} \right.$$

и для Π_1^S существует программа с метками Π_1^L. Пусть β — начальная метка Π_1^L, γ — заключительная, а α — новая метка. Возьмём

$$\Pi^L \Leftarrow \left\{ \begin{array}{ll} \gamma & \mathsf{if}\ e\ \mathsf{then}\ \beta\ \mathsf{else}\ \alpha \\ & \tilde{\Pi}_1^L \end{array} \right.$$

Пусть $\Pi^S(\sigma)$ определено. Это означает, что существует последовательность конфигураций $(\sigma_S^i)_{i=0}^n$ такая, что

$$\sigma_S^0 = \sigma, \quad \sigma_S^{i+1} = \Pi_1^S(\sigma_S^i), \quad \sigma_S^i(e) \neq 0 \iff i < n.$$

Рассмотрим последовательность конфигураций $(\sigma_L^j, \lambda_j)_j$ для $\Pi^L(\sigma)$. Пусть j_i — i-ый элемент, для которого $\lambda_{j_i} = \gamma$. В частности, $j_0 = 0$, так как $\lambda_0 = \gamma$. Индукцией по i докажем, что $\sigma_L^{j_i} = \sigma_S^i$.

Б а з и с и н д у к ц и и. $i = 0$. Тогда $j_0 = 0$ и $\sigma_L^{j_0} = \sigma_L^0 = \sigma = \sigma_S^0$.

Ш а г и н д у к ц и и. Пусть доказано, что $\sigma_L^{j_i} = \sigma_S^i$ и $i < n$. Это означает, что $\sigma_S^i(e) \neq 0$ и $\sigma_L^{j_i}(e) \neq 0$. Это означает, что вслед за $(\sigma_L^{j_i}, \gamma)$ должна идти конфигурация $(\sigma_L^{j_i}, \beta)$. Она является начальной конфигурацией для $\Pi_1^L(\sigma_L^{j_i})$. Из этого следует, что

$$\Pi_1^L(\sigma_L^{j_i}) = \Pi_1^S(\sigma_L^{j_i}) = \Pi_1^S(\sigma_S^i) = \sigma_S^{i+1}.$$

Это означает, что последовательность конфигураций для $\Pi_1^L(\sigma_L^{j_i})$ является конечной, и ее последним элементом является (σ_S^{i+1}, γ). Заметим, что в этой последовательности метка γ встречается только один раз в конце.

Следовательно, в последовательности конфигураций для $\Pi^L(\sigma)$ после $(\sigma_L^{j_i}, \beta)$ должна встретиться (σ_S^{i+1}, γ) и это — первое место

после j_i, где меткой является γ. Очевидно, что j_{i+1} — номер этой конфигурации, и $\sigma_L^{j_{i+1}} = \sigma_S^{i+1}$.

Итак, мы доказали, что $\sigma_L^{j_i} = \sigma_S^i$. Значит,

$$\sigma_L^{j_i}(e) \neq 0 \iff \sigma_S^i(e) \neq 0 \iff i < n.$$

То есть, $\sigma_L^{j_n}(e) = 0$. Значит, после конфигурации $(\sigma_L^{j_n}, \gamma)$ будет идти $(\sigma_L^{j_n}, \alpha)$. Так как α является заключительной меткой, то $\Pi^L(\sigma) = \sigma_L^{j_n} = \sigma_S^n = \Pi^S(\sigma)$.

Пусть теперь $\Pi^L(\sigma)$ определено. Тогда последовательность конфигураций $(\sigma_L^i, \lambda_i)_i$ является конечной. Пусть (σ_L^n, λ_n) — последний элемент. Пусть j_i — i-ый элемент, в котором метка — γ. Очевидно, что $j_0 = 0$. Кроме того, $\lambda_{n-1} = \gamma$, потому что метка α встречается в Π^L только в одном месте. Пусть $n - 1 = j_m$ для некоторого m. Кроме того, $\sigma_L^{j_i}(e) \neq 0$ тогда и только тогда, когда $i < m$.

Рассмотрим последовательность конфигураций σ_S^i для $\Pi^S(\sigma)$. Аналогично доказывается, что $\sigma_L^{j_i} = \sigma_S^i$. То есть,

$$\sigma_S^i(e) \neq 0 \iff \sigma_L^{j_i}(e) \neq 0 \iff i < m.$$

Это означает, что

$$\Pi^S(\sigma) = \sigma_S^m = \sigma_L^{j_m} = \sigma_L^{n-1} = \sigma_L^n = \Pi^L(\sigma). \qquad \square$$

***Задача 4.6.** Завершите доказательство корректности построения для полного ветвления.

***Задача 4.7.** Покажите, как построить программу с метками для неполного ветвления, и докажите правильность построения.

Задача 4.8. Докажите, что если исходная структурированная программа была простой, то с методом, предложенным в теореме, будет построена простая программа с метками.

Итак, мы научились по любой структурированной программе строить эквивалентную ей программу с метками.

Задача 4.9. Пользуясь методом, предложенным в теореме, постройте программы с метками по ранее построенным программам

1) для сложения чисел;
2) для вычитания чисел;
3) для умножения чисел;
4) для определения простоты числа.

§ 4.4. Построение структурированных программ

В этом параграфе мы решаем обратную задачу: доказываем, что по программе с метками можно построить эквивалентную структурированную программу.

Теорема 4.12 (Структуризация)**.** *По всякой программе с метками* Π^L *можно эффективно построить структурированную программу* Π^S *такую, что* $\mathrm{Var}(\Pi^L) \subseteq \mathrm{Var}(\Pi^S)$ *и*

$$\Pi^L(\sigma) \restriction \mathrm{Var}(\Pi^L) = \Pi^S(\sigma) \restriction \mathrm{Var}(\Pi^L)$$

для любой конфигурации σ *программы* Π^S*. Программу* Π^S *можно выбрать таким образом, чтобы она содержала только один оператор цикла.*

Доказательство. Итак, пусть дана программа с метками Π^L. Без ограничения общности можно предполагать, что все её метки операторов являются натуральными числами большими 0, а единственная заключительная метка — 0. Это означает, что программа Π^L имеет следующий вид:

$$\Pi^L \twoheadleftarrow \begin{cases} o_1 \\ o_2 \\ \vdots \\ o_m \end{cases}$$

$$\varphi(o_k) \rightleftharpoons \begin{cases} \textsf{if } w \textsf{ then} \\ \quad \textsf{if } y == \widehat{\alpha}_k \textsf{ then} \\ \quad\quad x_k = e_k; \\ \quad\quad y = \widehat{\beta}_k; \\ \quad\quad w = 0; \\ \quad \textsf{end}; \\ \textsf{end}; \end{cases},$$

Рис. 4.4: Построение $\varphi(o_k)$ для присваивания.

Каждый из операторов o_k имеет один из видов:

1) $\alpha_k \quad x_k = e_k; \quad \beta_k$

2) $\alpha_k \quad \textsf{if } e_k \textsf{ then } \beta_k \textsf{ else } \gamma_k$

Пусть w и y — новые переменные, которые не встречаются в программе Π^L. Для каждого оператора o_k одного из этих видов определим структурированную программу $\varphi(o_k)$ как показано на рис. 4.4 для операторов присваивания и на рис. 4.5 на следующей странице для условного перехода. Здесь через $\widehat{n}$ мы обозначаем следующее выражение:

$$\widehat{n} \rightleftharpoons \underbrace{\textsf{succ}(\textsf{succ}(\ldots\textsf{succ}(}_{n \text{ раз}}0\underbrace{)\ldots))}_{n \text{ раз}}$$

Очевидно, что $\sigma(\widehat{n}) = n$ для любого состояния σ. Построим структурированную программу Π^S как показано на рис. 4.6 на стр. 130.

Докажем, что $\Pi^S(\sigma)$ определено тогда и только тогда, когда $\Pi^L(\sigma)$ определено, для любой конфигурации σ программы Π^S и в этом случае

$$\Pi^S(\sigma) \upharpoonright \mathrm{Var}(\Pi^L) = \Pi^L(\sigma) \upharpoonright \mathrm{Var}(\Pi^L).$$

$$\varphi(o_k) \Leftarrow \left\{ \begin{array}{l} \textsf{if } w \textsf{ then} \\ \quad \textsf{if } y == \widehat{\alpha}_k \textsf{ then} \\ \quad\quad \textsf{if } e_k \textsf{ then} \\ \quad\quad\quad y = \widehat{\beta}_k; \\ \quad\quad \textsf{else} \\ \quad\quad\quad y = \widehat{\gamma}_k; \\ \quad\quad \textsf{end;} \\ \quad\quad w = 0; \\ \quad \textsf{end;} \\ \textsf{end;} \end{array} \right. .$$

Рис. 4.5: Построение $\varphi(o_k)$ для условного перехода.

По определению семантики программ с метками мы должны рассматривать последовательность конфигураций $(\sigma_L^i, \lambda_i)_i$, для которой $\sigma_L^0 = \sigma$ и $\lambda_0 = \alpha_1$.

Теперь рассмотрим Π^S. Через σ'_S обозначим конфигурацию Π^S, которая получается из σ применением первого оператора присваивания программы Π^S:

$$\sigma'_S = (y = \widehat{\alpha}_1;)(\sigma).$$

Очевидно, что

$$\sigma'_S y = \alpha_1; \quad \sigma'_S \restriction \mathrm{Var}(\Pi^L) = \sigma \restriction \mathrm{Var}(\Pi^L),$$

поскольку никакие переменные из программы Π^L этот оператор не изменяет. Для оператора цикла мы должны строить последовательность конфигураций $(\sigma_S^i)_i$ такую, что $\sigma_S^0 = \sigma'_S$ и $\sigma_S^{i+1} = \Pi_1^S(\sigma_S^i)$, где Π_1^S — тело цикла.

Индукцией по i докажем следующее утверждение:

1) $\sigma_S^i y = \lambda_i$;

$$\Pi^S \multimap \begin{cases} y = \widehat{\alpha}_1; \\ \mathsf{while}\ y\ \mathsf{do} \\ \quad w = \mathsf{succ}(0); \\ \quad \varphi(o_1) \\ \quad \varphi(o_2) \\ \quad \vdots \\ \quad \varphi(o_m) \\ \mathsf{end}; \end{cases}$$

Рис. 4.6: Построение Π^S.

2) $\sigma_S^i \restriction \mathrm{Var}(\Pi^L) = \sigma_L^i \restriction \mathrm{Var}(\Pi^L)$.

Б а з и с и н д у к ц и и . Для $i = 0$ утверждение, очевидно, выполнено, так как $\sigma_S^0 = \sigma'_S$, а свойства σ'_S мы описали выше.
Ш а г и н д у к ц и и . Пусть для i утверждение доказано. Тогда рассмотрим $\Pi_1^S(\sigma_S^i) = \sigma_S^{i+1}$. По определению семантики следования имеем:

$$\sigma_S^{i+1} = \Pi_1^S(\sigma_S^i) = \varphi(o_m)(\dots\varphi(o_1)((w = \mathsf{succ}(0);)(\sigma_S^i))\dots).$$

Пусть

$$\rho_S^i = (w = \mathsf{succ}(0);)(\sigma_S^i).$$

Тогда ρ_S^i отличается от σ_S^i только тем, что $\rho_S^i w = 1$. Пусть $\lambda_i = \alpha_{j_i}$.

Рассмотрим $\varphi(o_k)(\rho_S^i)$ при $k < j_i$. Очевидно, что

$$\rho_S^i(w) \neq 0$$

потому что $\rho_S^i w = 1$, но, в то же время,

$$\rho_S^i(y == \widehat{\alpha}_k) = 0$$

потому что

$$\rho_S^i y = \sigma_S^i y = \lambda_i = \alpha_{j_i} \neq \alpha_k,$$

а в программе с метками нет двух одинаковых меток операторов. Из определения семантики для неполного ветвления получаем, что $\varphi(o_k)(\rho_S^i) = \rho_S^i$.

Рассмотрим $\varphi(o_k)(\rho_S^i)$ при $k = j_i$. Тогда

$$\rho_S^i(w) \neq 0$$

и

$$\rho_S^i(y == \widehat{\alpha}_k) \neq 0$$

так как

$$\rho_S^i y = \sigma_S^i y = \lambda_i = \alpha_{j_i} = \alpha_k.$$

Пусть $\pi_S^i = \varphi(o_k)(\rho_S^i)$.

1) Если

$$o_k \quad \circ\!\!- \quad \alpha_k \; x_k = e_k; \; \beta_k,$$

то по определению семантики присваивания и следования (для структурированных программ) и по определению семантики программы с метками получим, что

$$\begin{aligned} &\pi_S^i y = \beta_k = \lambda_{i+1}, \\ &\pi_S^i w = 0, \\ &\pi_S^i x_k = \rho_S^i(e_k) = \sigma_S^i(e_k) = \sigma_L^i(e_k) = \sigma_L^{i+1}(x_k). \end{aligned}$$

Для всех остальных переменных z будем иметь

$$\pi_S^i z = \rho_S^i z = \sigma_S^i z = \sigma_L^i z = \sigma_L^{i+1} z.$$

2) Рассмотрим оператор o_k вида

α_k if e_k then β_k else γ_k

Тогда

$$\pi_S^i \restriction \mathrm{Var}(\Pi^L) = \rho_S^i \restriction \mathrm{Var}(\Pi^L) = \\ = \sigma_S^i \restriction \mathrm{Var}(\Pi^L) = \sigma_L^i \restriction \mathrm{Var}(\Pi^L) = \sigma_L^{i+1} \restriction \mathrm{Var}(\Pi^L).$$

Кроме того, $\pi_S^i w = 0$ и

$$\pi_S^i y = \left\{ \begin{array}{ll} \beta_k, & \text{если } \rho_S^i(e_k) \neq 0 \\ \gamma_k, & \text{если } \rho_S^i(e_k) = 0 \end{array} \right\} = \\ = \left\{ \begin{array}{ll} \beta_k, & \text{если } \sigma_L^i(e_k) \neq 0 \\ \gamma_k, & \text{если } \sigma_L^i(e_k) = 0 \end{array} \right\} = \lambda_{i+1}.$$

Теперь рассмотрим состояния $\varphi(o_k)(\pi_S^i)$ при $k > j_i$. Очевидно, что $\pi_S^i(w) = 0$. Значит, $\varphi(o_k)(\pi_S^i) = \pi_S^i$.

Итак, мы доказали, что $\sigma_S^{i+1} = \Pi_1^S(\sigma_S^i) = \pi_S^i$. Но при рассмотрении случая $k = j_i$ мы показали, что π_S^i удовлетворяет условиям утверждения. Следовательно, утверждение доказано для $i + 1$ и, по индукции, для всех i, для которых метка λ_{i-1} не является заключительной.

Пусть $\Pi^L(\sigma)$ определено. Тогда последовательность конфигураций $(\sigma_L^i, \lambda_i)_i$ конечна. Предположим, что (σ_L^n, λ_n) — её последний элемент. Тогда λ_n должна быть заключительной меткой, то есть — нулем. Тогда $\sigma_S^n(y) = 0$ и эта конфигурация будет завершающей для цикла. Согласно утверждению,

$$\sigma_S^n \restriction \mathrm{Var}(\Pi^L) = \sigma_L^n \restriction \mathrm{Var}(\Pi^L).$$

Пусть теперь $\Pi^L(\sigma)$ неопределено. Значит, в последовательности конфигураций $(\sigma_L^i, \lambda_i)_i$ заключительная метка 0 не встречается, то есть $\lambda_i \neq 0$ для всех i. Согласно нашему утверждению, $\sigma_S^i(y) = \lambda_i \neq 0$, что означает, что у цикла завершающего сосояния не будет, так как выражение y никогда не примет значения 0. Следовательно, $\Pi^S(\sigma)$ неопределено. □

Задача 4.10. Структуризируйте программы Min с рис. 4.1 на стр. 107 и Fib с рис. 4.2 на стр. 111 методом, предложенным в теореме.

*__Задача 4.11.__ Модифицируйте доказательство теоремы 4.12 так, что если исходная программа с метками является простой, то построенная структурированная программа тоже будет простой.

*__Задача 4.12.__ Модифицируйте доказательство теоремы 4.12 так, чтобы можно было обойтись только одной дополнительной переменной.

Итак, мы доказали, что структурированные программы и программы с метками вычисляют одни и те же функции. Выведем следствие.

Следствие 4.13 (Теорема о цикле)**.** *Для всякой структурированной программы* Π *существует структурированная программа* Π_1*, которая содержит только один оператор цикла и*

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi)$$

для любой конфигурации σ *программы* Π_1*.*

Доказательство. Для Π существует программа с метками Π_2 такая, что

$$\Pi(\sigma) = \Pi_2(\sigma)$$

для любой конфигурации σ программы Π. По теореме о структуризации существует структурированная программа Π_1, которая содержит один оператор цикла и

$$\Pi_1(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_2(\sigma) \restriction \mathrm{Var}(\Pi)$$

для любой конфигурации σ программы Π_1. Следовательно,

$$\Pi(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi_1(\sigma) \restriction \mathrm{Var}(\Pi)$$

для любой конфигурации σ. □

*__Задача 4.13.__ Сформулируйте теорему о подстановке для программ с метками и, используя теоремы о построении программ с метками и о структуризации, докажите её.

*__Задача 4.14.__ Сформулируйте и докажите теорему о простой программе для программ с метками.

§ 4.5. Блок-схемы

Один из популярных ранее методов представления алгоритмов — блок-схемы. Он и сейчас используется для иллюстрации несложных последовательностей инструкций. Как мы покажем, понятие блок-схемы полностью эквивалентно понятию программы с метками.

Определение 4.7 (Блок-схема)**.** *Блок-схема — размеченный граф* $\mathcal{B} = (G, L_V, L_E)$*, в котором* $\operatorname{rng} L_E \subseteq \{+, -\}$, $\operatorname{rng} L_V$ *состоит из операторов присваивания, выражений и конечных последовательностей имён переменных через запятую. Должны также выполняться следующие условия:*

1) *Имеется в точности одна вершина, в которую не входит ни одно ребро. Из неё исходит в точности одно ребро, а метка является последовательностью имён переменных. Такая вершина называется* вершиной начала*. Переменные из этой последовательности называем* входными переменными *или* параметрами *блок-схемы.*

2) *Из всякой вершины, помеченной оператором присваивания, исходит в точности одно ребро, это —* вершина присваивания*.*

3) *Если из вершины, помеченной выражением, исходит в точности два ребра, то одно из них имеет метку «+», другое — «−». Это —* вершина ветвления*.*

4) *Все остальные вершины помечены выражением и из них не исходит ни одного ребра. Такие вершины называем* вершинами конца*.*

С помощью $\operatorname{Var}(\mathcal{B})$ *будем обозначать множество переменных, встречающихся в метках вершин блок-схемы* $\mathcal{B}$*.*

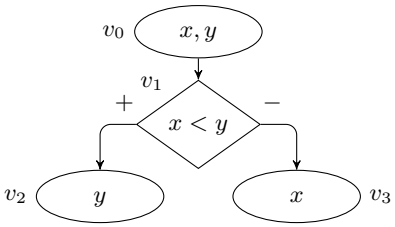


Рис. 4.7: Нахождение максимума.

Традиционно, когда рисуют блок-схемы, вершины присваивания обозначают прямоугольниками, вершины ветвления — ромбами, вершины начала и конца — эллипсами.

Пример 4.7. *Рассмотрим пример блок-схемы (см. рис. 4.7). Здесь вершина v_0 — вершина начала, v_1 — вершина ветвления, v_2 и v_3 — вершины конца.*

Определение 4.8 (Семантика блок-схем)**.** *Конфигурация блок-схемы $\mathcal{B}$ — пара (σ, v), где σ — состояние переменных* $\mathrm{Var}(\mathcal{B})$*, v — одна из вершин. Результат применения блок-схемы к конфигурации (σ, v) (обозначаем его с помощью $\mathcal{B}(\sigma, v)$) — это конфигурация (τ, w), если существует конечная последовательность конфигураций блок-схемы $(\sigma^i, v^i)_{i=0}^n$ такая, что*

1) *вершины $v^0, \ldots, v^n$ образуют путь в графе $\mathcal{B}$, причем v^n — вершина конца;*

2) $(\sigma^0, v^0) = (\sigma, v)$, $(\tau, w) = (\sigma^n, v^n)$;

3) *если вершина v^0 — вершина начала (вершина начала не может иметь другой номер), то $\sigma^1 = \sigma^0$;*

4) *если вершина* v^i, $i < n$ — *вершина присваивания,*

$$L_V(v^i) \circ x = e;$$

то $\sigma^{i+1} = (x = e;)(\sigma^i)$;

5) *если вершина* v^i, $i < n$ — *вершина ветвления,*

$$L_V(v^i) \circ e,$$

то $\sigma^{i+1} = \sigma^i$, *и ребро* (v^i, v^{i+1}) *помечено «+», если* $\sigma^i(e) \neq 0$, *помечено «−», если* $\sigma^i(e) = 0$.

Если σ — *состояние переменных* $\mathrm{Var}(\mathcal{B})$, *то полагаем* $\mathcal{B}(\sigma) = \tau$, *когда* $\mathcal{B}(\sigma, v^0) = (\tau, w)$ *для вершины начала* v^0 *и некоторой вершины конца* w.

Пример 4.8. *Рассмотрим блок-схему, изображенную на рис. 4.7 на предшествующей странице. Возьмем конфигурацию* (σ, v_0), *где* $\sigma = \{(x, 5), (y, 10)\}$. *Очевидно, что требуемый путь:* v_0, v_1, v_3. *Неоднозначность возникает только при рассмотрении* v_1. *Состояние в этой вершине будет* σ, *следовательно,* $\sigma(x < y) = 1$, *поэтому из двух ребер выбираем то, которое помечено «+». Заключительной будет конфигурация* (σ, v_3).

Аналогично определяются вызов блок-схемы, начальная конфигурация вызова, результат вызова и функция, вычисляемая блок-схемой.

Определение 4.9 (Вызов блок-схемы). *Вызов блок-схемы* $\mathcal{B}$ — *это упорядоченная* n-*ка* $(\mathcal{B}, u_1, \ldots, u_n)$, *где* n *равно количеству входных переменных блок-схемы,* $u_1, \ldots, u_n$ — *элементы предметной области, аргументы вызова. Начальная конфигурация вызова* — (σ, v_0), *где* v_0 — *вершина начала, а* $\sigma(x_i) = u_i$, *если* x_i — i-*ая входная переменная блок-схемы. Результат вызова равен* t, *если для любой начальной конфигурации выполнено* $\mathcal{B}(\sigma, v_0) = (\tau, w)$, *вершина конца* w *помечена выражением* e, *и*

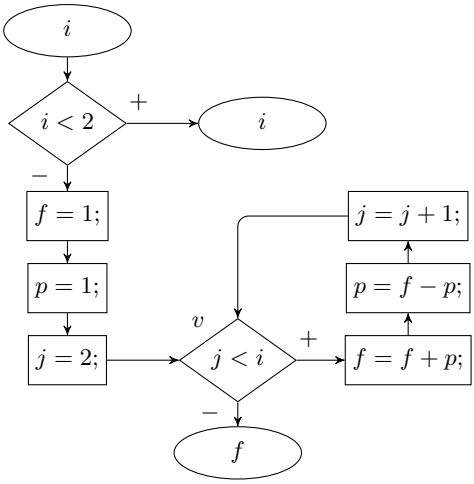


Рис. 4.8: Числа Фибоначчи.

$t = \tau(e)$. *Блок-схема вычисляет частичную функцию* f, *если* $f(\bar{t}) = \mathrm{Res}(\mathcal{B}, \bar{t})$ *для любых* $\bar{t}$.

Задача 4.15. Докажите, что блок-схема на рис. 4.7 на стр. 135 вычисляет функцию максимума.

Пример 4.9. *На рис. 4.8 изображена блок-схема для вычисления числа Фибоначчи по его номеру. Если значение* i *меньше двух, то число Фибоначчи будет совпадать со значением* i *(*$F_0 = 0$, $F_1 = 1$*).*

Иначе, дойдя до вершины v, *мы получим состояние*

$$\sigma_2 = \{(i, i_0), (j, 2), (p, 1), (f, 1)\}.$$

Далее каждый раз при переходе к v *будет состояние*

$$\sigma_k = \{(i, i_0), (j, k), (p, F_{k-1}), (f, F_k)\}$$

пока $k < i_0$, *в чем нетрудно убедиться по индукции:*

$$\sigma_{k+1} f = \sigma_k f + \sigma_k p = F_k + F_{k-1} = F_{k+1},$$

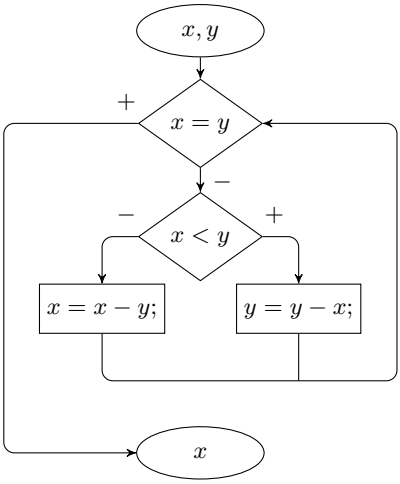


Рис. 4.9: Алгоритм Евклида.

$$\sigma_{k+1}p = \sigma_{k+1}f - \sigma_k p = F_{k+1} - F_{k-1} = F_k.$$

При $k = i_0$ следующей за v окажется вершина конца, и результат вызова будет равен $\sigma_k f = F_{i_0}$.

В качестве дополнительных иллюстраций без пояснений приведём блок-схемы для алгоритма Евклида (рис. 4.9) и для нахождения среднего из трёх чисел (рис. 4.10 на противоположной странице).

Отметим, что всегда можно полагать, что вершина конца в блок-схеме является единственной.

Утверждение 4.14. *Для всякой блок-схемы существует эквивалентная, имеющая ровно одну вершину конца.*

Доказательство. Пусть z — новая переменная. Добавим к блок-схеме новую вершину w с меткой z, она будет новой вершиной конца. Для каждой v — «старой» вершины конца заменим в ней выражение e на оператор присваивания $z = e$; и добавим ребро (v, w). Нетрудно заметить, что если раньше последовательность конфигураций окан-

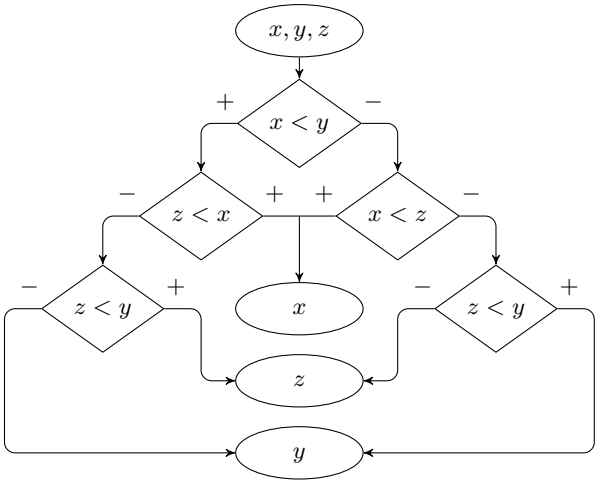


Рис. 4.10: Среднее из трех чисел.

чивалась парой (σ, v) и результатом было $\sigma(e)$, то в новой блок-схеме после (σ, v) будет идти еще один элемент (τ, w), где $\tau = (z = e;)(\sigma)$, что означает $\tau(z) = \sigma(e)$. Таким образом, результат новой блок схемы будет равен результату старой. Аналогично доказывается обратное. □

Таким образом блок-схему для нахождения максимума можно преобразовать к виду на рис. 4.11 на следующей странице.

Между блок-схемами и программами с метками существует тесная связь. Эту связь можно обнаружить, просто сравнив определения семантики программ с метками и блок-схем. В обоих случаях речь идёт о последовательностях вида $(\sigma_i, v_i)_i$, где v_i тем или иным образом идентифицируют элементы программы или блок-схемы, причём правила построения этих последовательностей тоже схожи. Докажем эту эквивалентность.

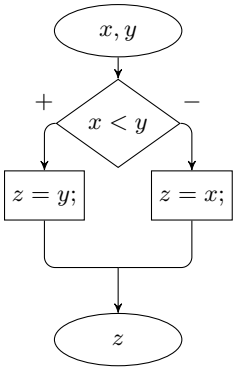


Рис. 4.11: Нахождение максимума.

Теорема 4.15. *По каждому алгоритму, тело которого является программой с метками, может быть эффективно построена блок-схема, вычисляющая ту же самую частичную функцию.*

Доказательство. Пусть A — исходный алгоритм, а Π — его тело, программа с метками. Пусть L — множество меток программы Π. Построим блок-схему $\mathcal{B}$. В качестве множества вершин нашей блок-схемы возьмём множество меток программы Π плюс одна дополнительная вершина v_0, которая будет начальной.

Определим разметку вершин. В качестве метки вершины v_0 берём список входных переменных алгоритма A, v_0 будет вершиной начала. Если λ — заключительная метка в Π, то в блок-схеме $\mathcal{B}$ она будет отмечена выражением-результатом алгоритма A. Если в Π имеется оператор

$$\beta\ x = e;\ \gamma,$$

то вершина β будет помечена оператором присваивания $x = e;$, а если в Π есть оператор

$$\beta\ \mathsf{if}\ e\ \mathsf{then}\ \gamma\ \mathsf{else}\ \delta,$$

то вершину β помечаем выражением e.

Теперь определим ребра и их разметку. Во-первых, добавляем ребро $(v_0, \mathrm{Start}(\Pi))$. Если в программе Π имеется оператор

$$\beta\ x = e;\ \gamma,$$

то добавляем ребро (β, γ). Если в Π имеется оператор

$$\beta\ \mathsf{if}\ e\ \mathsf{then}\ \gamma\ \mathsf{else}\ \delta,$$

то из β будет исходить два ребра: в вершину γ с меткой «+» и в вершину δ с меткой «−».

Поскольку мы в качестве множества вершин блок-схемы взяли множество меток программы Π, то каждую конфигурацию (σ, β) блок-схемы можно рассматривать как конфигурацию программы с метками Π (кроме случая $\beta = v_0$) и наоборот.

Продемонстрируем, что для любой метки β программы Π и любого состояния σ выполнено $\Pi(\sigma, \beta) = \mathcal{B}(\sigma, \beta)$. Для этого покажем, что последовательность конфигураций $(\sigma^i, \lambda_i)_i$ программы Π совпадет с последовательностью конфигураций $(\tau^i, \mu_i)_i$ блок-схемы $\mathcal{B}$. Используем индукцию по длине последовательности.

Базис индукции. Первая конфигурация и в том, и в другом случае будет (σ, β).

Шаг индукции. Пусть для i доказано: $(\sigma^i, \lambda_i) = (\tau^i, \mu_i)$.

Если λ_i — заключительная метка, то μ_i — вершина конца. Следовательно, и (σ^i, λ_i), и (τ^i, μ_i) будут последними элементами последовательностей.

Пусть в программе Π имеется оператор $\lambda_i\ x = e;\ \lambda_{i+1}$. Тогда следующей конфигурацией Π будет $(\sigma^{i+1}, \lambda_{i+1})$, где $\sigma^{i+1} = (x = e;)(\sigma^i)$. С другой стороны, по построению $\mathcal{B}$ в ней существует ребро λ_i в λ_{i+1}, а сама вершина λ_i помечена оператором присваивания $x = e;$. Следовательно, следующая конфигурация блок-схемы также будет $(\sigma^{i+1}, \lambda_{i+1})$.

Пусть в Π есть оператор

$$\lambda_i \text{ if } e \text{ then } \gamma \text{ else } \delta.$$

Тогда следующей конфигурацией Π будет $(\sigma^i, \lambda_{i+1})$, где λ_{i+1} есть γ или δ в зависимости от значения e в σ^i. В блок-схеме $\mathcal{B}$ по построению существуют два ребра, ведущие из вершины λ_i: в γ с меткой «+», в δ с меткой «−». Но тогда следующей конфигурацией блок-схемы будет (σ^i, γ), если $\sigma^i(e) \neq 0$, или (σ^i, δ) в противном случае. Таким образом, в обеих ситуациях получается новая конфигурация, совпадающая с $(\sigma^i, \lambda_{i+1})$. □

Задача 4.16. Докажите, что блок-схема, которая построена в доказательстве теоремы 4.15, будет вычислять ту же самую (частичную) функцию, что и исходный алгоритм.

Задача 4.17. Используя приведенный в доказательстве метод, постройте эквивалентную блок-схему для алгоритма из задачи 4.5 на стр. 117.

Теорема 4.16. *По каждой блок-схеме может быть эффективно построен алгоритм, вычисляющий ту же частичную функцию.*

Доказательство. Это доказательство симметрично предыдущему. Пусть $\mathcal{B}$ — блок-схема, V — множество ее вершин. Пусть v_0 — вершина начала, а v_1 — вершина, в которую ведёт ребро из v_0. Построим программу с метками Π следующим образом. По каждой вершине $v \in V$ (кроме вершин начала и конца) определим $\varphi(v)$ — оператор программы Π. Если v — вершина присваивания, помеченная оператором $x = e$;, из которой ведёт ребро в вершину w, то $\varphi(v)$ будет оператором $v\ x = e;\ w$. Если v — вершина ветвления, помеченная выражением e, из которой ведут ребра в вершины w_1 и w_2 с метками «+» и «−» соответственно, то $\varphi(v)$ будет оператором

$$v \text{ if } e \text{ then } w_1 \text{ else } w_2.$$

Соединим все $\varphi(v)$ в одну программу так, чтобы первым был оператор с меткой v_1.

Как и в предыдущем случае, каждая конфигурация (σ, v) полученной программы с метками Π можно рассматривать как конфигурацию блок-схемы $\mathcal{B}$. Точно такими же рассуждениями, что и в предыдущей теореме доказывается, что для любой конфигурации (σ, v) будет выполнено $\Pi(\sigma, v) = \mathcal{B}(\sigma, v)$.

Для построения законченной записи алгоритма потребуется ещё несколько действий. Возьмём новую заключительную метку α, новую переменную u, а для любой старой вершины конца v, помеченной выражением e_v, добавим к программе оператор

$$v\ u = e_v;\ \alpha.$$

Пусть $\bar{x}$ — список входных переменных из вершины начала. Теперь наш алгоритм может быть записан следующим образом:

```
algorithm A;
arguments x̄;
returns u;
    Π
end;
```

□

Задача 4.18. Докажите, что алгоритм, построенный в конце доказательства будет вычислять ту же (частичную) функцию, что и исходная блок-схема.

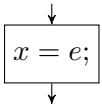
Задача 4.19. Постройте по блок-схеме на рис. 4.8 на стр. 137 эквивалентную программу с метками.

Таким образом, программы с метками и блок-схемы являются разными реализациями одной и той же концепции программирования.

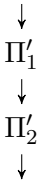
Используя блок-схемы, можно проинтерпретировать доказательство теоремы 4.11 на стр. 120 следующим образом. По каждой структурированной программе Π строится часть блок-схемы Π' с одним входом и одним выходом:

1) для пустого оператора:

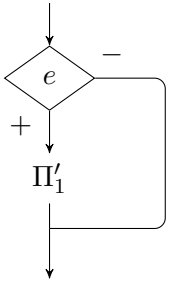
2) для присваивания $x = e$;:



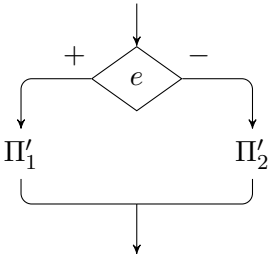
3) для следования $\Pi_1\Pi_2$:



4) для неполного ветвления if e then Π end;:



5) для полного ветвления if e then Π_1 else Π_2 end;:



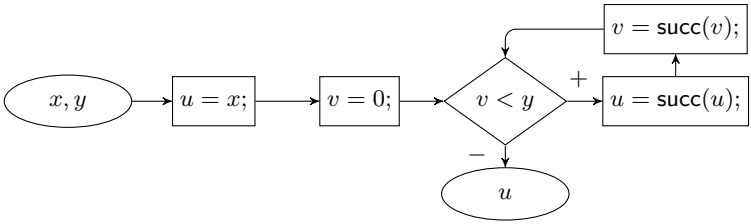
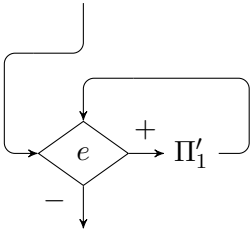
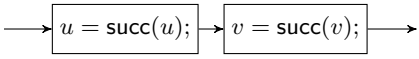


Рис. 4.12: Блок-схема для алгоритма сложения чисел

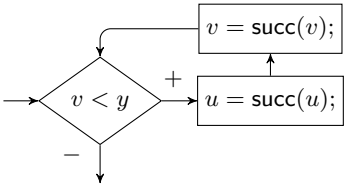
6) для цикла while e do Π end;:



Пример 4.10. *Построим таким образом блок-схему по структурированной программе на рис. 3.2 на стр. 65. Для тела цикла* Π_2 *мы получим:*



Для всего цикла будем иметь



Вся программа представлена в виде блок-схемы на рис. 4.12.

Задача 4.20. Используя приведенный в примере метод, постройте эквивалентную блок-схему для алгоритма Евклида (рис. 3.5 на стр. 73).

Задача 4.21. Докажите, что если тело алгоритма A — структурированная программа П не содержит циклов, то можно построить эквивалентную блок-схему в виде дерева.

Глава 5

Корректность структурированных программ

Мы уже несколько раз доказывали, что написанные нами программы действительно работают корректно, то есть делают именно то, для чего мы их писали. Эти доказательства были достаточно сложными и заключались в рассмотрении различных конфигураций и их преобразованиях.

Имеется другой путь — сформулировать некоторые формальные правила, следуя которым можно из истинных утверждений о программах получать истинные, и, пользуясь этим, доказывать корректность.

§ 5.1. Исчисления

Сначала дадим общее определение.

Определение 5.1 (Исчисление). *Исчислением называется тройка* $\mathfrak{I} = (\mathfrak{S}, \mathfrak{A}, \mathfrak{R})$, *где* $\mathfrak{S}$ — *некоторое множество строк,* $\mathfrak{A} \subseteq \mathfrak{S}$ — *множество аксиом,* $\mathfrak{R}$ — *множество правил вывода.*

Каждое правило вывода r — *это частичное отображение*

$$r : \mathfrak{S}^{<\omega} \to \mathcal{P}(\mathfrak{S}).$$

То есть по каждой конечной последовательности $s_1, \ldots, s_n$, $s_i \in \mathfrak{S}$ *правило вывода даёт множество* $\mathfrak{S}' \subseteq \mathfrak{S}$. *Если* $s \in r(s_1, \ldots, s_n)$, *то говорят, что строка* s *получена из* $s_1, \ldots, s_n$ *по правилу* r.

Таким образом, исчисление содержит исходные строки (аксиомы) и правила, с помощью которых из одних строк можно получать другие.

Определение 5.2 (Вывод в исчислении). *Выводом в исчислении* $\mathfrak{I} = (\mathfrak{S}, \mathfrak{A}, \mathfrak{R})$ *называется конечная последовательность строк* $(s_1, \ldots, s_n)$, *где* $s_i \in \mathfrak{S}$, *и для каждого* $i = 1, \ldots, n$ *выполнено одно из двух:*

- $s_i \in \mathfrak{A}$ *(то есть* s_i — *аксиома) или*
- *есть правило вывода* $r \in \mathfrak{R}$ *такое, что* $s_i \in r(s_1, \ldots, s_{i-1})$ *(говоря другими словами, строка* s_i *может быть получена из предыдущих строк с помощью одного из правил вывода).*

Каждый вывод является выводом своего последнего элемента.

Следовательно, вывод s — это вывод вида $(s_1, \ldots, s_n, s)$. В простейшем случае вывод состоит из одного элемента: (s), в этом случае строка s должна быть аксиомой.

Определение 5.3 (Выводимость). *Выводимость в исчислении $\mathfrak{I}$ строки s (обозначается $\underset{\mathfrak{I}}{\rhd} s$) означает возможность построения в $\mathfrak{I}$ вывода, последним элементом которого является s. Обозначение*

$$A_1, \ldots, A_n \underset{\mathfrak{I}}{\rhd} s$$

означает выводимость s в исчислении $\mathfrak{I}'$, полученном из $\mathfrak{I}$ добавлением аксиом $A_1, \ldots, A_n$.

Теперь сформулируем определения, которые будут использованы для установления корректности.

Определение 5.4 (Условие). *Условие для программы* П *— это некоторое свойство, сформулированное с использованием переменных программы* П.

Пример 5.1. *Пусть x, y и z — переменные. Примеры условий:*

$$x < x^2 + y^2; \quad x < y \ \text{ и } \ y < z - x^3;$$

существует v такое, что $x < v < y$ и v делится на z^2

Мы будем использовать для записи условий язык математической логики и записывать условия в виде формул логики предикатов с использованием обычных знаков арифметических операций и отношений:

$$x < x^2 + y^2; \quad x < y \wedge y < z - x^3; \quad (\exists v)(x < v < y \wedge z^2 \mid v).$$

Определение 5.5 (Истинность условия). *Условие φ истинно на состоянии σ: $\sigma \models \varphi$, если при подстановке в φ вместо переменных x их значений на σ, то есть σx, то получается формула, которая истинна на предметной области. Истинность условия φ на состоянии σ записываем в виде $\sigma \models \varphi$.*

Ложность условия φ на состоянии σ ($\sigma \not\models \varphi$) означает отсутствие истинности.

Замечание 5.1. Мы отличаем истинность на состоянии от истинности на предметной области. Истинность на предметной области означает истинность на любом состоянии (то есть по свободным переменным неявно навешиваются кванторы всеобщности).

Условие может содержать нелогические символы, которые являются общеизвестными (например, сложение или умножение натуральных чисел). Таким образом, речь идёт о предметной области не просто как о множестве, а как об алгебраической системе, то есть о множестве с определёнными на нём отношениями и операциями.

Пример 5.2. *Пусть* $\sigma = \{(x, 20), (y, 25), (z, 4)\}$. *Тогда*

1) $\sigma \models x < y + z^2$, *так как* $20 < 25 + 4^2$;
2) $\sigma \models x < y \wedge y < z + x^2$, *так как* $20 < 25$ *и* $25 < 4 + 20^2$;
3) $\sigma \not\models (\exists u)(x < u < y \wedge z^2 \mid u)$, *потому что никакое число между 20 и 25 не делится на 16.*

Задача 5.1. Определите, истинны ли три условия из предыдущего примера на состояниях из примера 3.7 на стр. 52.

Определение 5.6 (Тройка Хоара). *Тройка Хоара — строка вида* $\{\varphi\}\,\Pi\,\{\psi\}$, *где* Π — *структурированная программа, а* φ *и* ψ — *условия для* Π.

В тройке $\{\varphi\}\,\Pi\,\{\psi\}$ *предусловие — это формула* φ, *а постусловие —* ψ.

Для длинных троек мы будем записывать предусловие, программу и постусловие в столбик, ограничивая его по бокам прямыми линиями:

$$\left|\begin{array}{c}\{\varphi\}\\ \Pi \\ \{\psi\}\end{array}\right|$$

Рассмотрим пример.

Пример 5.3. *Возьмём следующую программу:*

$$\Pi \Leftarrow z = x;\ x = y;\ y = z;$$

Тройки Хоара для данной программы:

$$\left\{x < y + z^2\right\} \Pi \left\{x < y\right\};$$

$$\left\{x < y\right\} \Pi \left\{(\exists u)(x < u < y \wedge z^2 \mid u)\right\};$$

$$\left\{(\exists u)(x < u < y \wedge z^2 | u)\right\} \Pi \left\{x < y + z^2\right\}.$$

Определение 5.7 (Частичная, полная корректность)**.** *Пусть S — некоторое множество конфигураций структурированной программы* Π.

Тройка Хоара $\{\varphi\}\Pi\{\psi\}$ называется частично корректной на множестве S, если для всякой конфигурации $\sigma \in S$ такой, что $\sigma \models \varphi$ и $\Pi(\sigma)$ определено, выполнено $\Pi(\sigma) \models \psi$. Тройка называется частично корректной, если она частично корректна на множестве всех конфигураций программы Π.

Тройка Хоара $\{\varphi\}\Pi\{\psi\}$ называется полностью корректной (или вполне корректной) на множестве конфигураций S, если для любой конфигурации $\sigma \in S$ из $\sigma \models \varphi$ следует, что $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \psi$. Тройка называется полностью корректной, если она полностью корректна на множестве всех конфигураций программы Π.

Интуитивно эти понятия означают следующее. Предусловие — это свойство, которому должны удовлетворять входные данные программы. Постусловие описывает результат работы программы. Частичная корректность: если программа останавливается для допустимых входных данных, то результат верен. Полная корректность: если входные данные допустимы, то программа останавливается и результат верен.

Следствие 5.1. *Если тройка Хоара полностью корректна, то она частично корректна.*

Следствие 5.2. *Каждая тройка полностью корректна на пустом множестве.*

Задача 5.2. Докажите оба следствия.

Пример 5.4. *Рассмотрим следующую программу:*

```
while x < y do
    x = succ(x);
    y = succ(y);
    if y == 5 then
        y = 0;
    end;
end;
```

Семантика этой программы такова:

- *если* $\sigma \not\models x < y$, *то* $\Pi(\sigma) = \sigma$;
- *если* $\sigma \models x < y$ *и* $\sigma y < 5$, *то*

$$\Pi(\sigma) = \{(y, 0); (x, \sigma x - \sigma y + 5)\};$$

- *если* $\sigma \models x < y$ *и* $\sigma y \geqslant 5$, *то* $\Pi(\sigma)$ *не определено.*

Пусть

$$\varphi \circ\!\!- x = y \wedge y < 3, \quad \psi \circ\!\!- y^2 < 5.$$

Если $\sigma \models \varphi$, *то* $\sigma \not\models x < y$, *следовательно,* $\Pi(\sigma) = \sigma$. *Это означает, что тройка* $\{\varphi\}\Pi\{\psi\}$ *полностью корректна, так как если* $\sigma \models \varphi$, *то* $\Pi(\sigma)$ *определено и* $\Pi(\sigma) \models \psi$.

Пусть

$$\varphi \circ\!\!- y - x = 4, \quad \psi \circ\!\!- x + y = 1.$$

Пусть $\sigma \models \varphi$. *Тогда* $\sigma \models x < y$. *Если* $\sigma y < 5$, *то* $\sigma y \leqslant 4$. *Это значит, что* $\sigma y = 4$ *и* $\sigma x = 0$, *следовательно,* $\Pi(\sigma)$ *определено и*

$$\Pi(\sigma) = \{(y, 0); (x, 1)\}.$$

Для остальных σ таких, что $\sigma \not\models \varphi$, $\Pi(\sigma)$ не определено. Итак, тройка $\{\varphi\}\Pi\{\psi\}$ является частично корректной, но не является полностью корректной.

Пусть

$$\varphi \twoheadleftarrow x + y = 5, \qquad \psi \twoheadleftarrow x - y = 3.$$

Рассмотрим следующую конфигурацию: $\sigma = \{(x, 2), (y, 3)\}$. Очевидно, что $\sigma \models \varphi$. Далее, выполнено $\sigma \models x < y$ и $\sigma y < 5$. Следовательно, $\Pi(\sigma)$ определено и

$$\Pi(\sigma) = \{(y, 0); (x, 4)\}.$$

Ясно, что $\Pi(\sigma) \not\models \psi$. Таким образом, тройка $\{\varphi\}\Pi\{\psi\}$ не является частично корректной (и, тем более, полностью корректной).

Задача 5.3. Для каждой из программ из примера 3.4 на стр. 45 напишите по три тройки Хоара, одна из которых должна быть полностью корректной, другая частично корректной, но не полностью корректной, третья не быть частично корректной. Для всех ли программ это можно сделать?

Замечание 5.2. Если $\Pi(\sigma)$ определено для любой конфигурации σ, то для любой тройки Хоара с программой Π полная и частичная корректности эквивалентны.

Следствие 5.3. *Чтобы установить, что алгоритм*

```
algorithm A;
arguments x_1, ..., x_n;
returns e;
    Π
end;
```

вычисляет функцию Φ достаточно доказать, что тройка

$$\{\top\}\Pi\{e = \Phi(\bar{x})\}$$

полностью корректна.

Задача 5.4. Докажите следствие.

§ 5.2. *Исчисление Хоара

Мы определим исчисление Хоара, которое обозначаем с помощью $\mathfrak{IH}$[1]. Строками $\mathfrak{IH}$ будут тройки Хоара. Выводимость тройки Хоара в этом исчислении будет означать полную и/или частичную корректность этой тройки.

У нас будут аксиомы видов:

АКС$_0$ $\{\varphi\};\{\varphi\}$.

АКС $\{(\varphi)^x_e\}x = e;\{\varphi\}$, где x — переменная, e — выражение.

Нужно сделать одну оговорку про замену переменных в формулах. С помощью $(\varphi)^x_e$ как и ранее обозначен результат замены в формуле φ переменной x на выражение e. Однако, в отличие от замены переменных в выражениях и программах, не все вхождения x в формулу φ подлежат замене, а только свободные, то есть не связанные никакими кванторами. Кроме того, есть еще одно ограничение: требуется, чтобы после замены переменные выражения e не оказались в области действия кванторов по этим переменным. Иначе, в результате, например, замены в формуле $(\exists x)(x \neq y)$ переменной y на выражение x мы бы получили формулу $(\exists x)(x \neq x)$, то есть формула из истинной превратилась в ложную.

Правила вывода исчисления $\mathfrak{IH}$:

УПР Усиление предусловия: если выводимо $\{\varphi\}\Pi\{\psi\}$ и формула $\theta \to \varphi$ истинна на предметной области, то выводимо $\{\theta\}\Pi\{\psi\}$.

ОСП Ослабление постусловия: если выводимо $\{\varphi\}\Pi\{\psi\}$ и формула $\psi \to \theta$ истинна на предметной области, то выводимо $\{\varphi\}\Pi\{\theta\}$.

[1]На самом деле мы определим два исчисления — одно для частичной корректности, другое — полной. Эти исчисления будут отличаться только правилом, которое используется для корректности цикла.

СЛД Правило следования: если выводимы тройки $\{\varphi\}\Pi_1\{\theta\}$ и $\{\theta\}\Pi_2\{\psi\}$, то выводимо $\{\varphi\}\Pi_1\Pi_2\{\psi\}$.

НВТ Правило неполного ветвления: если выводима тройка $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ и формула $\varphi \wedge e = 0 \to \psi$ истинна на предметной области, то выводимо

$$\{\varphi\}\textsf{if } e \textsf{ then } \Pi_1 \textsf{ end;}\{\psi\}.$$

ПВТ Правило полного ветвления: если выводимы тройки $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ и $\{\varphi \wedge e = 0\}\Pi_2\{\psi\}$, то выводимо

$$\{\varphi\}\textsf{if } e \textsf{ then } \Pi_1 \textsf{ else } \Pi_2 \textsf{ end;}\{\psi\}.$$

Докажем, что с помощью аксиом можно получить только вполне корректные тройки Хоара, а с помощью правил **УПР**–**ПВТ** из корректных троек получать корректные. Говоря языком математической логики, докажем непротиворечивость исчисления Хоара.

1) **АКС**$_0$ Пусть Π — ; и $\sigma \models \varphi$. Так как $\Pi(\sigma) = \sigma$, то $\Pi(\sigma) \models \varphi$, то есть тройка $\{\varphi\};\{\varphi\}$ полностью корректна.

2) **АКС** Пусть Π — $x = e$; и $\sigma \models (\varphi)^x_e$, то есть φ истинно, если вместо x подставить $\sigma(e)$. Значит, если τ совпадает с σ во всём кроме того, что $\tau x = \sigma(e)$, то $\tau \models \varphi$. Но по определению семантики оператора присваивания, $\Pi(\sigma)$ определено и $\Pi(\sigma) = \tau$. То есть, $\Pi(\sigma) \models \varphi$. Это означает полную корректность тройки $\{(\varphi)^x_e\}\Pi\{\varphi\}$.

3) **УПР** Пусть тройка $\{\varphi\}\Pi\{\psi\}$ корректна и формула $\theta \to \varphi$ истинна на предметной области. Пусть $\sigma \models \theta$. Из истинности $\theta \to \varphi$ следует, что $\sigma \models \varphi$.

а) Частичная корректность. Пусть $\Pi(\sigma)$ определено. Тогда $\Pi(\sigma) \models \psi$ из-за частичной корректности $\{\varphi\}\Pi\{\psi\}$. Следовательно, тройка $\{\theta\}\Pi\{\psi\}$ частично корректна.

б) Полная корректность. Так как $\sigma \models \varphi$ и тройка $\{\varphi\}\Pi\{\psi\}$ вполне корректна, то $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \psi$. То есть тройка $\{\theta\}\Pi\{\psi\}$ вполне корректна.

4) **ОСП** Пусть тройка $\{\varphi\}\Pi\{\psi\}$ корректна и формула $\psi \to \theta$ истинна на предметной области. Пусть $\sigma \models \varphi$.

а) Частичная корректность. Пусть $\Pi(\sigma)$ определено. Тогда $\Pi(\sigma) \models \psi$ из-за частичной корректности $\{\varphi\}\Pi\{\psi\}$. Из истинности $\psi \to \theta$ получаем $\Pi(\sigma) \models \theta$. Следовательно, тройка $\{\varphi\}\Pi\{\theta\}$ частично корректна.

б) Полная корректность. Так как $\sigma \models \varphi$ и тройка $\{\varphi\}\Pi\{\psi\}$ вполне корректна, то $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \psi$. Из истинности $\psi \to \theta$ получаем $\Pi(\sigma) \models \theta$. То есть тройка $\{\varphi\}\Pi\{\theta\}$ вполне корректна.

5) **СЛД** Пусть тройки $\{\varphi\}\Pi_1\{\theta\}$ и $\{\theta\}\Pi_2\{\psi\}$ корректны. Пусть $\sigma \models \varphi$.

а) Частичная корректность. Пусть $(\Pi_1\Pi_2)(\sigma) = \Pi_2(\Pi_1(\sigma))$ определено. Тогда $\Pi_1(\sigma)$ определено и $\Pi_2(\Pi_1(\sigma))$ определено. Из частичной корректности $\{\varphi\}\Pi_1\{\theta\}$ следует, что $\Pi_1(\sigma) \models \theta$. Из частичной корректности $\{\theta\}\Pi_2\{\psi\}$ следует $\Pi_2(\Pi_1(\sigma)) \models \psi$. То есть $(\Pi_1\Pi_2)(\sigma) \models \psi$ и тройка $\{\varphi\}\Pi_1\Pi_2\{\psi\}$ частично корректна.

б) Полная корректность. Из полной корректности $\{\varphi\}\Pi_1\{\theta\}$ следует, что $\Pi_1(\sigma)$ определено и $\Pi_1(\sigma) \models \theta$. Из полной корректности $\{\theta\}\Pi_2\{\psi\}$ следует, что $\Pi_2(\Pi_1(\sigma))$ определено и $\Pi_2(\Pi_1(\sigma)) \models \psi$. Но $(\Pi_1\Pi_2)(\sigma) = \Pi_2(\Pi_1(\sigma))$. То есть

$(\Pi_1\Pi_2)(\sigma)$ определено и $(\Pi_1\Pi_2)(\sigma) \models \psi$ Это значит, что тройка $\{\varphi\}\Pi_1\Pi_2\{\psi\}$ вполне корректна.

6) **НВТ** Пусть

$$\Pi \Leftarrow \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};$$

Пусть тройка $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ корректна и формула $\varphi \wedge e = 0 \to \psi$ истинна на предметной области. Пусть $\sigma \models \varphi$.

а) Частичная корректность. Пусть $\Pi(\sigma)$ определено.
Если $\sigma \models e \neq 0$, то $\Pi(\sigma) = \Pi_1(\sigma)$ и $\sigma \models \varphi \wedge e \neq 0$. Из-за частичной корректности $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ имеем $\Pi_1(\sigma) \models \psi$ и $\Pi(\sigma) \models \psi$.
Если $\sigma \models e = 0$, то $\Pi(\sigma) = \sigma$. Кроме того, $\sigma \models \varphi \wedge e = 0$. Следовательно, $\sigma \models \psi$, то есть $\Pi(\sigma) \models \psi$.
Итак, в любом случае $\Pi(\sigma) \models \psi$, то есть тройка $\{\varphi\}\Pi\{\psi\}$ частично корректна.

б) Полная корректность.
Если $\sigma \models e \neq 0$, то $\Pi(\sigma) = \Pi_1(\sigma)$ и $\sigma \models \varphi \wedge e \neq 0$. Так как $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ полностью корректна, то $\Pi_1(\sigma)$ определено и $\Pi_1(\sigma) \models \psi$. Следовательно, $\Pi(\sigma) \models \psi$.
Если $\sigma \models e = 0$, то $\Pi(\sigma) = \sigma$. Кроме того, $\sigma \models \varphi \wedge e = 0$. Следовательно, $\sigma \models \psi$, то есть $\Pi(\sigma) \models \psi$.
Итак, в любом случае $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \psi$, то есть тройка $\{\varphi\}\Pi\{\psi\}$ вполне корректна.

7) **ПВТ** Аналогично.

***Задача 5.5.** Докажите, что с помощью правила **ПВТ** из частично (полностью) корректных троек можно получить только частично (полностью) корректные тройки.

****Замечание 5.3.** Обычно, когда вводят какие-либо исчисления, требуют, чтобы множество выводимых строк было (алгоритмически) перечислимо, то есть существовал бы алгоритм, результатом которого могли быть все

выводимые строки и только они. В нашем случае это не так, поскольку в общем случае установить истинность формулы на алгебраической системе алгоритмически невозможно.

Можно ограничиться какой-либо перечислимой частью теории алгебраической системы. Например, в нашем случае, когда алгебраическая система — $(\omega, +, \times)$ — можно ограничиться арифметикой Пеано, и требовать, чтобы формулы из правил **УПР**, **ОСП** и **НВТ** были не истинны на $(\omega, +, \times)$, а были теоремами арифметики Пеано. Это приведёт к сокращению множества выводимых троек.

Вводя исчисление Хоара, мы преследовали цель сделать его перечислимым, а удобным для доказательства корректности программ.

Замечание 5.4 (Корректность программ)**.** Если тройка

$$\{\top\}\Pi\{\psi\}$$

является частично или полностью корректной, то это означает, что после остановки программы постусловие ψ будет выполнено всегда, так как предусловие выполняется всегда.

Задача 5.6. Что означают частичная и полная корректность троек вида

$$\{\varphi\}\,\Pi\,\{\bot\}?$$

С помощью аксиомы **АКС** и правил **УПР**–**ПВТ** можно доказывать корректность программ без циклов.

Замечание 5.5. Так как для любой программы Π без циклов $\Pi(\sigma)$ определено для любой конфигурации σ, то для них полная и частичная корректности эквивалентны.

Прежде чем рассматривать пример, отметим, что для любого состояния σ выполнено

$$\sigma \models x < y \iff \sigma \models (x < y) \neq 0.$$

Это следует из определения операции $<$ (стр. 40). То же самое выполняется и для других введенных сравнений ($==$, $!=$ и т. д.). Поэтому мы в дальнейшем будем при написании условий вместо $(x < y) \neq 0$ писать более короткое $x < y$.

Пример 5.5. *Рассмотрим следующую программу:*

```
if x < y then
    z = x;
else
    z = y;
end;
```

Докажем, что

$$\underset{\mathfrak{IH}}{\triangleright} \{\top\}\Pi\{z = \min\{x, y\}\}.$$

[1] **АКС**. $\{x = \min\{x, y\}\}z = x;\{z = \min\{x, y\}\}$.

[2] **УПР** : [1]. *Так как формула*

$$\top \wedge x < y \to x = \min\{x, y\}$$

истинна[2], *то*

$$\{\top \wedge x < y\}z = x;\{z = \min\{x, y\}\}.$$

[3] **АКС**. $\{y = \min\{x, y\}\}z = y;\{z = \min\{x, y\}\}$.

[4] **УПР** : [3]. *Так как формула*

$$\top \wedge \neg x < y \to y = \min\{x, y\}$$

истинна, то

$$\{\top \wedge \neg x < y\}z = y;\{z = \min\{x, y\}\}.$$

[5] **ПВТ** : [2], [4]. $\{\top\}\Pi\{z = \min\{x, y\}\}$.

Запись вида **АКС** : *означает, что далее следует аксиома, а запись вида* **ПВТ** [2], [4] : *означает, что текущая тройка Хоара выводится из троек Хоара, полученных на шагах 2 и 4 с помощью правила* **ПВТ**.

[2] Здесь и дальше, говоря об истинности формулы, мы подразумеваем её истинность на нашей предметной области — множестве натуральных чисел со стандартными операциями.

Пример 5.6. *Для следующей программы докажем полную корректность тройки*

$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y}\} \\ \Pi \\ \{x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}\} \end{array}\right|$$

Иными словами, при выполнении программы множество $\{x, y\}$ не изменяется, но x становится не больше y.

$$\Pi \rightleftharpoons \left\{\begin{array}{l} \mathsf{if}\ y < x\ \mathsf{then} \\ \quad t = y; \\ \quad y = x; \\ \quad x = t; \\ \mathsf{end}; \end{array}\right.$$

[1] **АКС**. $\{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} t = y; \{x = \hat{x} \wedge t = \hat{y} \wedge t < x\}$.

[2] **АКС**. $\{x = \hat{x} \wedge t = \hat{y} \wedge t < x\} y = x; \{y = \hat{x} \wedge t = \hat{y} \wedge t < y\}$.

[3] **АКС**. $\{y = \hat{x} \wedge t = \hat{y} \wedge t < y\} x = t; \{y = \hat{x} \wedge x = \hat{y} \wedge x < y\}$.

[4] **СЛД** : [1], [2].

$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} \\ t = y; y = x; \\ \{y = \hat{x} \wedge t = \hat{y} \wedge t < y\} \end{array}\right|.$$

[5] **СЛД** : [4], [3].

$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} \\ t = y; y = x; x = t; \\ \{y = \hat{x} \wedge x = \hat{y} \wedge x < y\} \end{array}\right|.$$

[6] **ОСП** : [5]. *Формула*

$$y = \hat{x} \wedge x = \hat{y} \wedge x < y \to x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}$$

истинна. Следовательно,

$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} \\ t = y; y = x; x = t; \\ \{x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}\} \end{array}\right|.$$

[7] **НВТ** : [6]. *Формула*

$$x = \hat{x} \wedge y = \hat{y} \wedge \neg y < x \to x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}$$

истинна. Следовательно,

$$\{x = \hat{x} \wedge y = \hat{y}\}\Pi\{x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}\}.$$

Рассмотрим более сложный пример.

Пример 5.7. *Рассмотрим программу, которая находит наибольшее из трёх чисел* x, y, z.

$$\left.\begin{array}{l} u = x; \\ \left.\begin{array}{l} \mathsf{if}\ u < y\ \mathsf{then} \\ \quad u = y; \\ \mathsf{end}; \end{array}\right\}\Pi_1 \\ \left.\begin{array}{l} \mathsf{if}\ u < z\ \mathsf{then} \\ \quad u = z; \\ \mathsf{end}; \end{array}\right\}\Pi_2 \end{array}\right\}\Pi$$

Докажем полную корректность тройки $\{\top\}\Pi\{u = \max\{x, y, z\}\}$.

[1] **АКС**. $\{x = x\}u = x; \{u = x\}$

[2] **УПР** : [1]. *Формула* $\top \to x = x$ *истинна, поэтому*

$$\{\top\}u = x; \{u = x\}.$$

[3] **АКС**. $\{y = \max\{x, y\}\}u = y; \{u = \max\{x, y\}\}$

[4] **УПР** : [3]. *Формула*

$$(u = x \wedge u < y) \to y = \max\{x, y\}$$

истинна, поэтому

$$\{u = x \wedge u < y\}u = y; \{u = \max\{x, y\}\}.$$

[5] **НВТ** : [4]. *Формула*

$$(u = x \wedge \neg u < y) \to u = \max\{x, y\}$$

истинна, поэтому

$$\{u = x\}\Pi_1\{u = \max\{x, y\}\}.$$

[6] **СЛД** : [2], [5]. $\{\top\}u = x; \Pi_1\{u = \max\{x, y\}\}$

[7] **АКС**. $\{z = \max\{x, y, z\}\}u = z; \{u = \max\{x, y, z\}\}$

[8] **УПР** : [7]. *Формула*

$$(u = \max\{x, y\} \wedge u < z) \to z = \max\{x, y, z\}$$

истинна, поэтому

$$\{u = \max\{x, y\} \wedge u < z\}u = z; \{u = \max\{x, y, z\}\}.$$

[9] **НВТ** : [8]. *Формула*

$$(u = \max\{x, y\} \wedge \neg u < z) \to u = \max\{x, y, z\}$$

истинна, поэтому

$$\{u = \max\{x, y\}\}\Pi_2\{u = \max\{x, y, z\}\}.$$

[10] **СЛД** : [6], [9]. $\{\top\}\Pi\{u = \max\{x, y, z\}\}$

Общая схема вывода такова. Сначала мы выводим, что после оператора $u = x$; *переменная* u *станет равна* x *(тройки 1–2). Затем, что после выполнения* Π_1 *переменная* u *станет равной максимальному из чисел* x *и* y *(тройки 3–6). Наконец, что после* Π_2 *переменная* u *станет равной наибольшему из трёх чисел (тройки 7–10).*

Рассмотрим тройки 1–2. Тройка 1 — это аксиома, потому что $(u = x)^u_x \leftrightarrows x = x$. *Возможно, что предусловие этой тройка выгдядит несколько неестественно, но такая тройка является единственно возможной, так как должно выполняться постусловие* $u = x$. *Заменяя в нём переменную из левой части присваивания (то есть* u*) на выражение из правой (то есть* x*), мы получаем предусловие* $x = x$. *Поскольку такое предусловие нам не нужно, мы заменяем его на эквивалентное предусловие* $\top$ *с помощью правила* **УПР**. *Написать сразу тройку* $\{\top\}u = x; \{u = x\}$ *нельзя, потому что она не является аксиомой.*

Тройки 3–6. Мы уже знаем, что $u = x$. *Далее мы выводим, что если* $u < y$ *истинно, то программа присвоит* u *максимальное из двух чисел. Если же* $u < y$ *ложно, то максимальное число уже хранится в* u. *Тройка*

3 говорит, что если y был равен $\max\{x, y\}$ и мы присвоили его в u, то теперь $u = \max\{x, y\}$. Постусловие $u = \max\{x, y\}$ — это та формула, которую мы хотим вывести. Тогда предусловие $y = \max\{x, y\}$ получается из постусловия однозначно с помощью замены. Теперь мы замечаем, что формула $y = \max\{x, y\}$ не похожа ни на одну конструкцию, встречающуюся в программе. С помощью правила **УПР** *мы заменяем $y = \max\{x, y\}$ на формулу $u = x \wedge u < y$. Здесь $u = x$ — это условие после первого присваивания, а $u < y$ — то, что проверяется в ветвлении. В результате получается тройка 4. Она говорит, что если $u = x$ и $u < y$, то после присваивания внутри программы Π_1 в u окажется нужное значение. Далее мы видим, что формула $(u = x \wedge \neg u < y) \to u = \max\{x, y\}$ истинна. Эта формула говорит, что если $u = x$ и условие ветвления не выполнено, то в u уже хранится нужное значение. Оказалось, что независимо от истинности $u < y$ после выполнения Π_1 в u окажется $\max\{x, y\}$. Применяя правило* **НВТ**, *получаем тройку 5. Соединяя тройки 2 и 5 по правилу* **СЛД**, *получаем тройку 6.*

Тройки 7–10 аналогичны тройкам 3–6.

Пример 5.8. *Следующая программа переставляет значения переменных x и y, не используя других переменных:*

$$\left.\begin{array}{l} x = x + y; \\ y = x - y; \\ x = x - y; \end{array}\right\} \Pi$$

Докажем полную корректность тройки $\{x = a \wedge y = b\}\Pi\{x = b \wedge y = a\}$.

[1] **АКС**. $\{x + y = a + b \wedge y = b\}x = x + y;\{x = a + b \wedge y = b\}$

[2] **УПР** : [1]. *Формула*

$$(x = a \wedge y = b) \to (x + y = a + b \wedge y = b)$$

истинна, поэтому

$$\{x = a \wedge y = b\}x = x + y;\{x = a + b \wedge y = b\}.$$

[3] **АКС**. $\{x = a + b \wedge x - y = a\}y = x - y;\{x = a + b \wedge y = a\}$

[4] **УПР** : [3]. *Формула*

$$(x = a + b \wedge y = b) \to (x = a + b \wedge x - y = a)$$

истинна, поэтому

$$\{x = a + b \wedge y = b\}y = x - y; \{x = a + b \wedge y = a\}.$$

[5] **СЛД** : [2], [4]. $\{x = a \wedge y = b\}x = x + y; y = x - y; \{x = a + b \wedge y = a\}$

[6] **АКС**. $\{x - y = b \wedge y = a\}x = x - y; \{x = b \wedge y = a\}$

[7] **УПР** : [6]. *Формула*

$$(x = a + b \wedge y = a) \to (x - y = b \wedge y = a)$$

истинна, поэтому

$$\{x = a + b \wedge y = a\}x = x - y; \{x = b \wedge y = a\}.$$

[8] **СЛД** : [5], [7]. $\{x = a \wedge y = b\}\Pi\{x = b \wedge y = a\}$

*__Задача 5.7.__ Докажите, что программа на рис. 5.1 на противоположной странице находит среднее из трёх чисел.

§ 5.3. Корректность циклов

Итак, мы научились доказывать корректность программ, не содержащих циклов. Прежде чем переходить к циклам, дадим несколько определений.

Определение 5.8 (Инвариант программы)**.** *Условие φ называется (частичным) инвариантом программы* П, *если тройка $\{\varphi\}\Pi\{\varphi\}$ является полностью (соответственно, частично) корректной.*

```
algorithm Mid;
arguments x, y, z;
returns m;
    if x < y then
        if x < z then
            if y < z then
                m = y;
            else
                m = z;
            end;
        else
            m = x;
        end;
    else
        if x < z then
            m = x;
        else
            if y < z then
                m = z;
            else
                m = y;
            end;
        end;
    end;
end;
```

Рис. 5.1: Среднее из трёх чисел.

Определение 5.9 (Инвариант цикла)**.** *Условие φ называется (частичным) инвариантом цикла*

```
while e do
    Π
end;
```

если тройка $\{\varphi \wedge e \neq 0\}\Pi\{\varphi\}$ полностью (частично) корректна,

Очевидно, что каждый инвариант программы (цикла) является и частичным инвариантом программы (цикла). Обратное, естественно, неверно.

Пример 5.9. *Возьмём $\Pi \Leftarrow t = x; x = y; y = t;$ и $\varphi \Leftarrow \{x, y\} = \{u, v\}$. В этом случае φ является инвариантом программы Π. В самом деле, пусть $\sigma x = u, \sigma y = v$. Тогда $\sigma \models \varphi$, $\Pi(\sigma) = \tau$, где $\tau x = v, \tau y = u$, то есть $\tau \models \varphi$. Аналогично для случая, когда $\sigma x = v, \sigma y = u$.*

Пример 5.10. *Покажем, что условие $u - v = w$ является инвариантом следующего цикла:*

```
while u < x do
    u = succ(u);   } Π
    v = succ(v);   }
end;
```

Пусть $\sigma \models u - v = w$, то есть $\sigma u - \sigma v = \sigma w$. Очевидно, что

$$\Pi(\sigma)(u) = \sigma u + 1; \quad \Pi(\sigma)(v) = \sigma v + 1$$

и $\Pi(\sigma)(w) = \sigma w$. Следовательно,

$$\Pi(\sigma)(u) - \Pi(\sigma)(v) = \sigma u + 1 - (\sigma v + 1) = \sigma u - \sigma v = \sigma w = \Pi(\sigma)(w).$$

Итак, $\Pi(\sigma) \models u - v = w$.

Замечание 5.6. Если φ — инвариант программы Π, то φ — инвариант цикла

```
while e do Π end;
```

для любого e.

Обратное неверно. Рассмотрим пример.

Пример 5.11. *Тот же цикл, но*

$$\varphi \twoheadrightarrow u - v = w \wedge u \leqslant x.$$

Пусть $\sigma \models \varphi$. *Мы уже знаем из предыдущего примера, что* $\Pi(\sigma) \models u - v = w$. *Так как* $\sigma \models u \leqslant x \wedge u < x$, *то* $\sigma \models u + 1 \leqslant x$. *Значит,*

$$\Pi(\sigma)(u) = \sigma u + 1 \leqslant \sigma x = \Pi(\sigma)(x).$$

То есть $\Pi(\sigma) \models u \leqslant x$. *Следовательно,* φ — *инвариант цикла.*

То, что φ *не является инвариантом тела цикла, легко убедиться, рассмотрев конфигурацию* σ, *для которой* $\sigma u = \sigma x = \sigma v + \sigma w$. *Тогда*

$$\Pi(\sigma)(u) = \sigma u + 1 > \sigma x = \Pi(\sigma)(x),$$

то есть $\Pi(\sigma) \not\models \varphi$, *хотя* $\sigma \models \varphi$.

Понятия частичного инварианта достаточно для доказательства частичной корректности оператора цикла.

Лемма 5.4 (Частичная корректность цикла)**.** *Пусть дан цикл*

$$\Pi \twoheadrightarrow \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end;} \end{array} \right.$$

φ — *его частичный инвариант,* $\Pi(\sigma)$ *определено для конфигурации* σ. *Тогда* $\Pi(\sigma) \models \varphi \wedge e = 0$.

Доказательство. Рассмотрим последовательность конфигураций цикла $(\sigma^i)_{i=0}^n$: $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma)$, $\Pi(\sigma) = \sigma^n$. Индукцией по i докажем, что $\sigma^{i+1} \models \varphi$ для $i = 0, \ldots, n$.

Базис индукции. Если $i = 0$, то $\sigma^0 = \sigma \models \varphi$.

Шаг индукции. Пусть для $i < n$ доказано, что $\sigma^i \models \varphi$. Так как $i < n$, то $\sigma^i(e) \neq 0$, то есть $\sigma^i \models e \neq 0$. Следовательно, $\sigma^i \models \varphi \wedge e \neq 0$. Так как тройка $\{\varphi \wedge e \neq 0\}\, \Pi_1\, \{\varphi\}$ частично корректна и $\Pi_1(\sigma^i)$ определено, то $\sigma^{i+1} = \Pi_1(\sigma^i) \models \varphi$.

Так как $\Pi(\sigma) = \sigma^n$, то $\sigma^n(e) = 0$, следовательно,

$$\Pi(\sigma) = \sigma^n \models \varphi \wedge e = 0. \qquad \square$$

Для доказательства полной корректности цикла инварианта недостаточно.

Определение 5.10 (Ограничитель цикла)**.** *Пусть дан цикл*

```
while e do
    Π
end;
```

и условие φ. Ограничителем цикла при условии φ называется выражение L_φ такое, что

1) *$\sigma(L_\varphi) \in \omega$ для всякой конфигурации σ;*
2) *$\Pi(\sigma)(L_\varphi) < \sigma(L_\varphi)$ для всякой конфигурации σ такой, что $\sigma \models \varphi \wedge e \neq 0$.*

Если условие φ истинно на предметной области, то такой ограничитель будет ограничителем и при любом другом условии. Мы будем называть его просто ограничителем.

Пример 5.12. *Рассмотрим тот же цикл, что в примере 5.10 на стр. 166. Пусть $\varphi \Leftrightarrow \top$ и $L = \lfloor x - u \rfloor$. Очевидно, что $\sigma(L) \in \omega$ для любого σ. Пусть $\sigma \models \varphi \wedge e \neq 0$, то есть $\sigma \models u < x$. Тогда*

$$\Pi(\sigma)(u) = \sigma u + 1, \quad \Pi(\sigma)(x) = \sigma x.$$

Следовательно,

$$\Pi(\sigma)(L) = \lfloor \Pi(\sigma)(x) - \Pi(\sigma)(u) \rfloor = \lfloor \sigma x - (\sigma u + 1) \rfloor.$$

Так как $\sigma \models u < x$, то

$$\sigma x - \sigma u > 0, \quad \sigma x - (\sigma u + 1) \geqslant 0.$$

Таким образом,

$$\Pi(\sigma)(L) = \sigma x - \sigma u - 1 < \sigma x - \sigma u = \sigma(L).$$

Итак, $L = \lfloor x - u \rfloor$ — ограничитель цикла при условии φ, следовательно, L — универсальный ограничитель.

Лемма 5.5 (Полная корректность цикла)**.** *Пусть дан цикл*

$$\Pi \Leftarrow \left\{ \begin{array}{l} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end;} \end{array} \right.$$

φ — инвариант цикла, L_φ — ограничитель. Тогда $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \varphi \wedge e = 0$ для всякой конфигурации σ, для которой $\sigma \models \varphi$.

Доказательство. Пусть $\sigma \models \varphi$. Рассмотрим последовательность конфигураций цикла $(\sigma^i)_i$: $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma)$. Индукцией по i докажем, что если $\sigma^i \models e \neq 0$, то $\Pi_1(\sigma^i)$ определено и $\sigma^{i+1} \models \varphi$.

Базис индукции. Если $i = 0$, то $\sigma^0 \models \varphi$.

Шаг индукции. Пусть для i доказано и $\sigma^i \models e \neq 0$. Так как тройка $\{\varphi \wedge e \neq 0\}\, \Pi_1\, \{\varphi\}$ полностью корректна, то $\Pi_1(\sigma^i)$ определено и $\sigma^{i+1} \models \varphi$.

Рассмотрим последовательность натуральных чисел $(\sigma^i(L_\varphi))_i$. Если предположить, что $\sigma^i \models e \neq 0$ для всех $i \in \omega$, то, так как L_φ — ограничитель, и $\sigma^i \models \varphi \wedge e \neq 0$, $(\sigma^i(L_\varphi))_i$ — убывающая цепь натуральных чисел. Следовательно, она не может быть бесконечной, противоречие. Это означает, что должен существовать элемент σ^n такой, что $\sigma^n \models e = 0$ и $\sigma^i(e) \neq 0$ для $i < n$. По определению семантики цикла $\Pi(\sigma)$ определено и $\Pi(\sigma) = \sigma^n$. Получаем, что $\Pi(\sigma) \models \varphi \wedge e = 0$. □

Лемма 5.6 (Об ограничителе цикла)**.** *Если переменная z не входит в условия φ, ψ, выражения e и L и программу Π_1 и тройка*

$$\{\varphi \wedge e \neq 0 \wedge L = z\}\, \Pi_1\, \{\psi \wedge L < z\}$$

полностью корректна и $\sigma(L) \in \omega$ для любой конфигурации σ, то L — ограничитель цикла

$$\mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};$$

при условии φ.

*__Задача 5.8.__ Докажите лемму.

Задача 5.9. Приведите пример, доказывающий, что условие невхождения z в программу Π_1 существенно.

Теперь мы можем определить правила вывода для циклов. Пусть дан цикл

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

ЦКЧ Для частичной корректности. Если φ — частичный инвариант цикла Π, то $\{\varphi\}\Pi\{\varphi \wedge e = 0\}$. То есть, из частичной корректности $\{\varphi \wedge e \neq 0\}\Pi_1\{\varphi\}$ следует частичная корректность $\{\varphi\}\Pi\{\varphi \wedge e = 0\}$.

ЦКП Для полной корректности. Если φ — инвариант цикла Π и L_φ — ограничитель Π, то $\{\varphi\}\Pi\{\varphi \wedge e = 0\}$. То есть, из полной корректности тройки

$$\{\varphi \wedge e \neq 0 \wedge L = z\}\Pi_1\{\varphi \wedge L < z\},$$

где z не входит в φ, e, L и Π_1, следует полная корректность тройки $\{\varphi\}\Pi\{\varphi \wedge e = 0\}$. Предполагается, что выражение L должно принимать только натуральные значения.

С помощью этих правил из корректных троек получаются корректные, что следует из лемм 5.4, 5.5 и 5.6.

Рассмотрим пример.

Пример 5.13. *Мы докажем полную корректность тройки*

$$\{\top\}\Pi\{u = \lfloor x - y \rfloor\}$$

для следующей программы:

$$
\left.\begin{array}{l} u = 0;\ v = y; \\ \textsf{while } v < x \textsf{ do} \\ \quad \left.\begin{array}{l} u = \mathsf{succ}(u); \\ v = \mathsf{succ}(v); \end{array}\right\} \Pi_3 \\ \textsf{end;} \end{array}\right\} \Pi_2 \Bigg\} \Pi
$$

[1] **АКС**. $\{y = y \wedge 0 = 0\}u = 0; \{y = y \wedge u = 0\}$.

[2] **АКС**. $\{y = y \wedge u = 0\}v = y; \{v = y \wedge u = 0\}$.

[3] **ОСП** : [2]. *Формула*

$$
y = v \wedge u = 0 \rightarrow \\
\rightarrow \underbrace{(v = y \wedge x \leqslant y \wedge u = \lfloor x - y \rfloor \bigvee v \leqslant x \wedge v \geqslant y \wedge u = \lfloor v - y \rfloor)}_{\psi}
$$

истинна, поэтому $\{y = y \wedge u = 0\}v = y; \{\psi\}$.

[4] **СЛД** : [1], [3]. $\{0 = 0 \wedge y = y\}u = 0; v = y; \{\psi\}$.

[5] **УПР** : [4]. *Очевидно, что формула*

$$\top \rightarrow y = y \wedge 0 = 0$$

истинна. Поэтому

$$\{\top\}u = 0; v = y; \{\psi\}.$$

Прежде, чем продолжать, преобразуем вторую часть ψ*:*

$v \leqslant x \wedge v \geqslant y \wedge u = \lfloor v - y \rfloor \equiv v \leqslant x \wedge v + 1 > y \wedge u + 1 = \lfloor (v + 1) - y \rfloor$.

Рассмотрим $\psi \wedge v < x$*:*

$$
\begin{array}{l}
\psi \wedge v < x \equiv (v = y \wedge x \leqslant y \wedge u = \lfloor x - y \rfloor \bigvee \\
\qquad \bigvee v \leqslant x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor) \wedge v < x \equiv \\
\qquad \equiv v = y \wedge x \leqslant y \wedge u = \lfloor x - y \rfloor \wedge v < x \bigvee \\
\qquad \bigvee v \leqslant x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor \wedge v < x \equiv \\
\qquad \equiv \left\{ \begin{array}{c} v = y \wedge x \leqslant y \wedge v < x \equiv \bot \\ v \leqslant x \wedge v < x \equiv v < x \equiv v + 1 \leqslant x \end{array} \right\} \equiv \\
\qquad \qquad \equiv v + 1 \leqslant x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor
\end{array}
$$

[6] **АКС**.
$$\left|\begin{array}{c}\{v+1\leqslant x\wedge v+1>y\wedge u+1=\lfloor v+1-y\rfloor\}\\ u=\mathsf{succ}(u);\\ \{v+1\leqslant x\wedge v+1>y\wedge u=\lfloor v+1-y\rfloor\}\end{array}\right|.$$

[7] **АКС**.
$$\left|\begin{array}{c}\{v+1\leqslant x\wedge v+1>y\wedge u=\lfloor v+1-y\rfloor\}\\ v=\mathsf{succ}(v);\\ \{v\leqslant x\wedge v>y\wedge u=\lfloor v-y\rfloor\}\end{array}\right|.$$

[8] **ОСП** : [7]. *Формула*
$$v\leqslant x\wedge v>y\wedge u=\lfloor v-y\rfloor\to\psi$$
истинна. Значит
$$\{v+1\leqslant x\wedge v+1>y\wedge u=\lfloor v+1-y\rfloor\}v=\mathsf{succ}(v);\{\psi\}.$$

[9] **СЛД** : [6], [8]. $\{v+1\leqslant x\wedge v+1>y\wedge u+1=\lfloor v+1-y\rfloor\}\Pi_3\{\psi\}$.

[10] **УПР** : [9]. $\{\psi\wedge v<x\}\Pi_3\{\psi\}$.

[11] **ЦКП** : [10]. ψ — *инвариант.* $L_\psi=x-v$ — *ограничитель. Значит* $\{\psi\}\Pi_2\{\psi\wedge\neg v<x\}$.

Преобразуем последнюю формулу:

$$\begin{array}{c}\psi\wedge\neg v<x\equiv\psi\wedge v\geqslant x\equiv\\ \equiv v=y\wedge x\leqslant y\wedge u=\lfloor x-y\rfloor\wedge v\geqslant x\bigvee\\ \bigvee v\leqslant x\wedge v\geqslant y\wedge u=\lfloor v-y\rfloor\wedge v\geqslant x\equiv\\ \equiv\left\{\begin{array}{c}v\leqslant x\wedge v\geqslant x\equiv x=v\\ v=y\wedge x\leqslant y\wedge v\geqslant x\equiv v=y\wedge x\leqslant v\end{array}\right\}\equiv\\ \equiv v=y\wedge x\leqslant v\wedge u=\lfloor x-y\rfloor\bigvee v\geqslant y\wedge v=x\wedge u=\lfloor v-y\rfloor\equiv\\ \equiv\{\ v=x\wedge u=\lfloor v-y\rfloor\equiv v=x\wedge u=\lfloor x-y\rfloor\ \}\equiv\\ \equiv u=\lfloor x-y\rfloor\wedge(v=y\wedge x\leqslant v\bigvee v\geqslant y\wedge v=x)\end{array}$$

Это значит, что формула $\psi\wedge\neg v<x\to u=\lfloor x-y\rfloor$ *истинна.*

[12] **ОСП** : [11]. $\{\psi\}\Pi_2\{u = \lfloor x - y \rfloor\}$.

[13] **СЛД** : [5], [12]. $\{\top\}\Pi\{u = \lfloor x - y \rfloor\}$.

Мы формально доказали, что программа Π *действительно выполняет вычитание чисел.*

Пример 5.14. *Рассмотрим программу для вычисления суммы* x *и* y*:*

$$\left.\begin{array}{l} u = x;\ v = 0; \\ \mathsf{while}\ v < y\ \mathsf{do} \\ \quad \left.\begin{array}{l} v = v + 1; \\ u = u + 1; \end{array}\right\}\Pi_2 \\ \mathsf{end;} \end{array}\right\}\Pi_1 \Bigg\}\Pi$$

Докажем полную корректность тройки $\{\top\}\Pi\{u = x + y\}$.

[1] **АКС**. $\{x = x \wedge 0 = 0\}u = x;\{u = x \wedge 0 = 0\}$

[2] **АКС**. $\{u = x \wedge 0 = 0\}v = 0;\{u = x \wedge v = 0\}$

[3] **СЛД** : [1], [2]. $\{x = x \wedge 0 = 0\}u = x; v = 0;\{u = x \wedge v = 0\}$

[4] **УПР** : [3]. *Формула* $\top \to (x = x \wedge 0 = 0)$ *истинна, поэтому*

$$\{\top\}u = x; v = 0;\{u = x \wedge v = 0\}.$$

Пусть $\varphi \rightleftharpoons v \leqslant y \wedge u = x + v$.

[5] **ОСП** : [4]. *Формула* $(u = x \wedge v = 0) \to \varphi$ *истинна, поэтому*

$$\{\top\}u = x; v = 0;\{\varphi\}.$$

Рассмотрим формулу $\varphi \wedge v < y$*:*

$$\varphi \wedge v < y \equiv v < y \wedge u = x + v \equiv v + 1 \leqslant y \wedge u + 1 = x + v + 1.$$

[6] **АКС**.

$$\left|\begin{array}{c} \{v + 1 \leqslant y \wedge u + 1 = x + v + 1 \wedge y - (v + 1) + 1 = z\} \\ v = v + 1; \\ \{v \leqslant y \wedge u + 1 = x + v \wedge y - v + 1 = z\} \end{array}\right|$$

[7] **АКС**.

$$\left|\begin{array}{c}\{v \leqslant y \wedge u+1 = x+v \wedge y-v+1 = z\}\\ u = u+1;\\ \{v \leqslant y \wedge u = x+v \wedge y-v+1 = z\}\end{array}\right|$$

[8] **СЛД** : [6], [7].

$$\left|\begin{array}{c}\{v+1 \leqslant y \wedge u+1 = x+v+1 \wedge y-(v+1)+1 = z\}\\ v = v+1;\ u = u+1;\\ \{v \leqslant y \wedge u = x+v \wedge y-v+1 = z\}\end{array}\right|$$

[9] **УПР** : [8]. *Формула*

$$(\varphi \wedge v < y \wedge y - v = z) \rightarrow \\ \rightarrow (v+1 \leqslant y \wedge u+1 = x+v+1 \wedge y-(v+1)+1 = z)$$

истинна, поэтому

$$\left|\begin{array}{c}\{\varphi \wedge v < y \wedge y - v = z\}\\ v = v+1;\ u = u+1;\\ \{v \leqslant y \wedge u = x+v \wedge y-v+1 = z\}\end{array}\right|.$$

[10] **ОСП** : [9]. *Формула*

$$(v \leqslant y \wedge u = x+v \wedge y-v+1 = z) \rightarrow (\varphi \wedge y - v < z)$$

истинна, поэтому

$$\{\varphi \wedge v < y \wedge y - v = z\} v = v+1;\ u = u+1; \{\varphi \wedge y - v < z\}.$$

[11] **ЦКП** : [10]. $\{\varphi\}\Pi_1\{\varphi \wedge \neg v < y\}$

[12] **ОСП** : [11]. *Формула* $(\varphi \wedge \neg v < y) \rightarrow u = x + y$ *истинна, поэтому* $\{\varphi\}\Pi_1\{u = x + y\}$.

[13] **СЛД** : [5], [12]. $\{\top\}\Pi\{u = x + y\}$

Прокомментируем построенный вывод.

Сначала нужно определить инвариант цикла. Инвариант должен «объяснять», как работает цикл. В нашем случае это формула

$$\varphi \mathrel{\overset{\circ}{-}} v \leqslant y \wedge u = x + v.$$

Она утверждает, что при выполнении тела цикла всегда будет выполнено $u = x + v$ и всегда $v \leqslant y$. Это верно, потому что сначала $u = x$, $v = 0$, а на каждой новой итерации u и v увеличиваются на единицу. Кроме того, если программа вошла в цикл, то $v < y$, поэтому после увеличения v на 1 выполняется условие $v \leqslant y$. Когда цикл прервётся, $v < y$ будет ложно, то есть будет выполняться $v \geqslant y$. Поскольку инвариант тоже истинен, то будет истинна формула

$$\varphi \wedge v \geqslant y \mathrel{\overset{\circ}{-}} v \leqslant y \wedge u = x + v \wedge v \geqslant y.$$

Из неё следует, что $u = x + y$, что нам и нужно.

Тройки 1–5: инициализация. Тройка 1 описывает присваивание $u = x$, а тройка 2 — присваивание $v = 0$. Тройка 3 описывает первые два присваивания. В первой тройке нельзя опустить условие $0 = 0$. Если мы его опустим, то получится тройка $\{x = x\}u = x; \{u = x\}$. Но к этой тройке и тройке 2 неприменимо правило **СЛД**. *На шаге 4 мы заменяем тождественно истинное предусловие $x = x \wedge 0 = 0$ на формулу $\top$. На шаге 5 мы начинаем готовиться к выводу корректности цикла. Мы замечаем, что после первых двух присваиваний выполняется формула φ, то есть наш инвариант.*

Тройки 6–11: цикл. Наша цель — показать, что если перед началом очередной итерации выполняется формула $\varphi \wedge v < y$, то после этой итерации выполняется формула φ. Сначала мы преобразуем формулу $\varphi \wedge v < y$. Поскольку мы знаем, что аксиомам нужны правые части присваиваний, то формулу преобразуем так, чтобы u и v встречались в ней только в выражениях $u + 1$ и $v + 1$. Затем обычным образом описываем работу двух присваиваний в теле цикла (тройки 6–8). Чтобы вывести полную корректность тройки для цикла, нужно указать ограничитель. Для нашего цикла в качестве ограничителя можно взять выражение $y - v$. Оно неотрицательно, так как всегда $v \leqslant y$, и оно убывает, так как y не меняется, а v растёт. Чтобы проверить, что выражение L — ограничитель, нужно запомнить его значение до очередной итерации в некоторой новой переменной z, а

затем убедиться, что в конце итерации $L < z$. Это проделано на шагах 6 (запомнили старое значение) и 10 (с помощью ослабления постусловия вывели, что новое значение меньше z). Также на шаге 10 мы замечаем, что из имеющейся формулы следует φ. Тройка 9 была нужна для того, чтобы заменить «некрасивую» формулу, оставшуюся после использования аксиом, на формулу того вида, который нужен правилу **ЦКП**. *В результате на шаге 10 получилась тройка, которая говорит, что φ — инвариант, а $y - v$ — ограничитель. Применяя на шаге 11 правило* **ЦКП**, *получаем тройку $\{\varphi\}\Pi_1\{\varphi \wedge \neg v < y\}$. Она говорит, что если перед запуском цикла выполнялся инвариант φ, то после остановки цикла будет по-прежнему выполняться φ (так как инвариант сохраняется на каждой итерации) и будет ложно $v < y$ (так как цикл прервался). Остановку цикла гарантирует ограничитель: он не может убывать бесконечно.*

Тройки 12–13: завершение вывода. На шаге 11 мы замечаем, что если истинна формула $\varphi \wedge \neg v < y$, то $v = y$, а потому $u = x + y$. Далее применяем правило **СЛД**.

Приведём теперь менее формальное доказательство корректности программы из последнего примера.

Утверждение 5.7. *Формула $\varphi \circ\!\!- v \leqslant y \wedge u = x + v$ является инвариантом цикла Π_1.*

Доказательство. Сначала отметим эквивалентность

$$\varphi \wedge v < y \equiv v < y \wedge u = x + v.$$

Допустим, выполнено $\sigma \models v < y \wedge u = x + v$ и $\tau = \Pi_2(\sigma)$. По определению семантики присваивания $\tau u = \sigma u + 1$, $\tau v = \sigma v + 1$, $\tau x = \sigma x$, $\tau y = \sigma y$. Поэтому имеем

$$\tau u = \sigma u + 1 = \sigma x + \sigma v + 1 = \tau x + \tau v,$$

то есть $\tau \models u = x + v$. Поскольку $\sigma v < \sigma y$, то $\sigma v + 1 \leqslant \sigma y$, $\tau v \leqslant \tau y$, то есть $\tau \models v \leqslant y$. Итак, мы доказали, что если $\sigma \models \varphi \wedge v < y$, то $\Pi_2(\sigma) \models \varphi$. Это и означает, что φ — инвариант цикла Π_1. □

Утверждение 5.8. *Выражение $L_\varphi \circ\!\!- y - v$ является ограничителем цикла Π_1 при предусловии φ.*

Доказательство. Пусть $\sigma \models \varphi \wedge v < y$. Пусть $\Pi_2(\sigma) = \tau$. Тогда $\tau v = \sigma v + 1$, $\tau y = \sigma y$. Поэтому

$$\tau(L_\varphi) = \tau y - \tau v = \sigma y - \sigma v - 1 = \sigma(L_\varphi) - 1 < \sigma(L_\varphi),$$

так как $\sigma(L_\varphi) > 0$. Поскольку всегда выполнено $v \leqslant y$, то выражение $y - v$ может принимать только натуральные значения. Следовательно, $y - v$ — это ограничитель. □

Утверждение 5.9. *Для любой конфигурации σ имеет место* $\Pi(\sigma) \models u = x + y$.

Доказательство. После первых двух присваиваний истинна формула $u = x \wedge v = 0$. Из истинности этой формулы следует истинность инварианта φ. Поскольку существует ограничитель, то $\Pi(\sigma)$ определено. Тогда $\Pi(\sigma) \models \varphi \wedge \neg v < y$. Преобразуем последнюю формулу:

$$\varphi \wedge \neg v < y \equiv v \leqslant y \wedge u = x + v \wedge v \geqslant y \equiv$$
$$\equiv u = x + v \wedge y = v \equiv u = x + y \wedge y = v.$$

Из последней формулы тривиально следует $u = x + y$, поэтому получаем $\Pi(\sigma) \models u = x + y$. □

В дальнейшем мы неоднократно будем пользоваться следующей леммой.

Лемма 5.10 (О замене переменной)**.** *Если*

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\Pi\{\psi\},$$

то

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^x_y\}(\Pi)^x_y\{(\psi)^x_y\},$$

где y — переменная, которая не встречается в выводе тройки $\{\varphi\}\Pi\{\psi\}$ (и, следовательно, в программе Π).

Доказательство. Пусть $\mathcal{D}$ — вывод тройки $\{\varphi\}\Pi\{\psi\}$. Заменим всюду в $\mathcal{D}$ переменную x на y. Нужно теперь доказать, что полученная последовательность снова будет выводом. Индукция по длине вывода.

Базис индукции. Аксиома: $\{(\varphi)^z_e\}z = e;\{\varphi\}$. Если $z \neq x$, то

$$((\varphi)^z_e)^x_y \multimap ((\varphi)^x_y)^z_{(e)^x_y}$$

$$(z = e;)^x_y \multimap z = (e)^x_y;$$

Получаем

$$\{((\varphi)^x_y)^z_{(e)^x_y}\}z = (e)^x_y;\{(\varphi)^x_y\}$$

что является аксиомой.

Если $z \multimap x$, то

$$((\varphi)^x_e)^x_y \multimap (\varphi)^x_{(e)^x_y} \multimap ((\varphi)^x_y)^y_{(e)^x_y}$$

$$(x = e;)^x_y \multimap y = (e)^x_y;$$

Получаем

$$\{((\varphi)^x_y)^y_{(e)^x_y}\}y = (e)^x_y;\{(\varphi)^x_y\}$$

что тоже является аксиомой.

Для пустого оператора утверждение очевидно: вместо аксиомы $\{\varphi\};\{\varphi\}$ мы получим аксиому $\{(\varphi)^x_y\};\{(\psi)^x_y\}$.

Шаг индукции. Индукционный шаг состоит в доказательстве того, что после замены переменных все применения правил останутся верными. Докажем два из них.

1) **НВТ** Пусть тройка

$$\{\varphi\}\mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};\{\psi\}$$

получена из тройки $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ и истинности $\varphi \wedge e = 0 \to \psi$. По индукционному предположению

$$\underset{\mathfrak{Ih}}{\rhd} \{(\varphi)^x_y \wedge (e)^x_y \neq 0\}(\Pi_1)^x_y\{(\psi)^x_y\}.$$

Формула $(\varphi)^x_y \wedge \neg(e)^x_y = 0 \to (\psi)^x_y$ так же будет истинна. По правилу **НВТ** получаем

$$\{(\varphi)^x_y\}\mathsf{if}\ (e)^x_y\ \mathsf{then}\ (\Pi_1)^x_y\ \mathsf{end};\{(\psi)^x_y\}.$$

2) **ЦКП** Пусть из полной корректности тройки

$$\{\varphi \wedge e \neq 0 \wedge L = z\}\Pi_1\{\varphi \wedge L < z\}$$

получена полная корректность

$$\{\varphi\}\mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};\{\varphi \wedge e = 0\}.$$

По индукционному предположению получаем выводимость

$$\{(\varphi)^x_y \wedge (e)^x_y \neq 0 \wedge (L)^x_y = z\}(\Pi_1)^x_y\{(\varphi)^x_y \wedge (L)^x_y < z\}.$$

Так как y отличается в том числе и от z, то с помощью правила **ЦКП** можно вывести

$$\{(\varphi)^x_y\}\mathsf{while}\ (e)^x_y\ \mathsf{do}\ (\Pi_1)^x_y\ \mathsf{end};\{(\varphi)^x_y \wedge (e)^x_y = 0\}. \qquad \square$$

****Задача 5.10.** Докажите, что из истинности формулы $\varphi \wedge e = 0 \to \psi$ следует истинность $(\varphi)^x_y \wedge (e)^x_y = 0 \to (\psi)^x_y$.

***Задача 5.11.** Докажите индукционный шаг для остальных правил.

Рассмотрим еще один вариант замены.

Лемма 5.11 (О замене переменной)**.** *Если*

$$\underset{\mathfrak{Ih}}{\rhd} \{\varphi\}\Pi\{\psi\},$$

то

$$\underset{\text{ЦКП}}{\triangleright} \{(\varphi)^x_d\}(\Pi)^x_d\{(\psi)^x_d\},$$

где d *— выражение, которое не содержит переменных из* $\mathrm{LVar}(\Pi)$*, а* x *не принадлежит* $\mathrm{LVar}(\Pi)$ *и отличается от переменных, используемых в правилах* **ЦКП**.

Доказательство. Прежде всего отметим, что при такой замене из программ снова получатся программы (лемма 3.19 на стр. 78). В остальном доказательство аналогично доказательству предыдущей леммы. □

В практике программирования процесс установления корректности обычно делится на две части.

Определение 5.11 (Спецификация программы)**.** *Спецификация программы* Π *— это определение предусловия* φ *и постусловия* ψ *для каждой логически законченной части программы* Π.

Спецификация может быть выполнена до написания текста программы. Если мы перед тем как писать программу определяем, из каких частей она должна состоять, то мы можем сформулировать, что эти части должны делать, следовательно, мы в состоянии определить предусловие и постусловие для каждой из них.

Определение 5.12 (Верификация программы)**.** *Верификация — доказательство корректности (частичной или полной) каждой тройки* $\{\varphi\}\Pi\{\psi\}$*, построенной в ходе спецификации.*

Итак, спецификация — условия, которым должна удовлетворять программа. Верификация — доказательство того, что она им действительно удовлетворяет.

Обычно спецификация и верификация выполняются в виде комментариев к программе. Конечно, подробно записывать доказательство корректности обычно не требуется, достаточно написать только

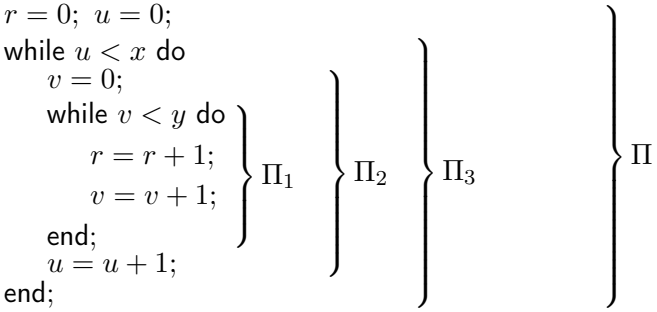


Рис. 5.2: Вычисление произведения

наиболее сложные вещи: инварианты циклов, ограничители циклов и ограничители вызовов (для рекурсивных подпрограмм), а так же предусловия и постусловия для подпрограмм.

§ 5.4. *Примеры формальных выводов

Рассмотрим еще несколько примеров.

Пример 5.15. *Докажем полную корректность тройки* $\{\top\}\Pi\{r = xy\}$ *для программы на рис. 5.2. Для краткости введём две следующие формулы* φ *и* ψ*:*

$$\varphi \twoheadleftarrow u \leqslant x \wedge r = uy,$$

$$\psi \twoheadleftarrow u < x \wedge v \leqslant y \wedge r = uy + v.$$

[1] **АКС**. $\{0 = 0 \wedge 0 = 0\}r = 0; \{r = 0 \wedge 0 = 0\}$

[2] **АКС**. $\{r = 0 \wedge 0 = 0\}u = 0; \{r = 0 \wedge u = 0\}$

[3] **СЛД** : [1], [2]. $\{0 = 0 \wedge 0 = 0\}r = 0;\ u = 0; \{r = 0 \wedge u = 0\}$

[4] **УПР** : [3]. *Формула* $\top \to (0 = 0 \wedge 0 = 0)$ *истинна, поэтому*

$$\{\top\}r = 0;\ u = 0; \{r = 0 \wedge u = 0\}.$$

[5] **ОСП** : [4]. *Формула* $(r = 0 \wedge u = 0) \to \varphi$ *истинна, поэтому*

$$\{\top\} r = 0;\ u = 0; \{\varphi\}.$$

Рассмотрим формулу $\psi \wedge v < y$*:*

$$\psi \wedge v < y \equiv u < x \wedge v < y \wedge r = uy + v \equiv$$
$$\equiv u < x \wedge v + 1 \leqslant y \wedge r + 1 = uy + v + 1.$$

[6] **АКС**.

$$\left| \begin{array}{c} \{u < x \wedge v + 1 \leqslant y \wedge r + 1 = uy + v + 1\} \\ r = r + 1; \\ \{u < x \wedge v + 1 \leqslant y \wedge r = uy + v + 1\} \end{array} \right|$$

[7] **АКС**.

$$\left| \begin{array}{c} \{u < x \wedge v + 1 \leqslant y \wedge r = uy + v + 1\} \\ v = v + 1; \\ \{u < x \wedge v \leqslant y \wedge r = uy + v\} \end{array} \right|$$

[8] **СЛД** : [6], [7].

$$\left| \begin{array}{c} \{u < x \wedge v + 1 \leqslant y \wedge r + 1 = uy + v + 1\} \\ r = r + 1;\ v = v + 1; \\ \{u < x \wedge v \leqslant y \wedge r = uy + v\} \end{array} \right|$$

[9] **УПР** : [8]. *Формула*

$$(\psi \wedge v < y) \to (u < x \wedge v + 1 \leqslant y \wedge r + 1 = uy + v + 1)$$

истинна, поэтому

$$\{\psi \wedge v < y\} r = r + 1;\ v = v + 1; \{\psi\}.$$

[10] **ЦКП** : [9]. ψ *— инвариант цикла* Π_1, $L_\psi \twoheadleftarrow y - v$ *— ограничитель. Значит,*

$$\{\psi\} \Pi_1 \{\psi \wedge \neg v < y\}.$$

Преобразуем формулу $\psi \wedge \neg v < y$*:*

$$\psi \wedge \neg v < y \equiv u < x \wedge v = y \wedge r = uy + v \equiv u < x \wedge v = y \wedge r = (u+1)y.$$

[11] **ОСП** : [10]. *Формула*

$$(\psi \wedge \neg v < y) \to (u + 1 \leqslant x \wedge r = (u + 1)y)$$

истинна, поэтому

$$\{\psi\}\Pi_1\{u + 1 \leqslant x \wedge r = (u + 1)y\}.$$

[12] **АКС**. $\{u + 1 \leqslant x \wedge r = (u + 1)y\}u = u + 1; \{u \leqslant x \wedge r = uy\}$

[13] **СЛД** : [11], [12]. $\{\psi\}\Pi_1 u = u + 1; \{u \leqslant x \wedge r = uy\}$

Рассмотрим формулу $\varphi \wedge u < x$*:*

$$\varphi \wedge u < x \equiv u < x \wedge r = uy.$$

[14] **АКС**. $\{u < x \wedge r = uy \wedge 0 = 0\}v = 0; \{u < x \wedge r = uy \wedge v = 0\}$

[15] **ОСП** : [14]. *Формула* $(u < x \wedge r = uy \wedge 0 = 0) \to \psi$ *истинна, поэтому*

$$\{u < x \wedge r = uy \wedge 0 = 0\}v = 0; \{\psi\}.$$

[16] **СЛД** : [15], [13]. $\{u < x \wedge r = uy \wedge 0 = 0\}\Pi_2\{u \leqslant x \wedge r = uy\}$

[17] **УПР** : [16]. *Формула*

$$(\varphi \wedge u < x) \to (u < x \wedge r = uy \wedge 0 = 0)$$

истинна, поэтому

$$\{\varphi \wedge u < x\}\Pi_2\{\varphi\}.$$

[18] **ЦКП** : [17]. φ — *инвариант цикла* Π_3, $L_\varphi \circ\!\!- x - u$ — *ограничитель. Значит,*

$$\{\varphi\}\Pi_3\{\varphi \wedge \neg u < x\}.$$

[19] **ОСП** : [18]. *Формула* $(\varphi \wedge \neg u < x) \to r = xy$ *истинна, поэтому*

$$\{\varphi\}\Pi_3\{r = xy\}.$$

[20] **СЛД** : [5], [19]. $\{\top\}\Pi\{r = xy\}$

$$
\left.\begin{array}{l}
u = 0; \\
\left.\begin{array}{l}
\mathsf{while}\ u < x\ \mathsf{do} \\
\quad u = \mathsf{succ}(\mathsf{succ}(u)); \\
\mathsf{end};
\end{array}\right\} \Pi_1 \\
\left.\begin{array}{l}
\mathsf{if}\ u == x\ \mathsf{then} \\
\quad E = \mathsf{succ}(0); \\
\mathsf{else} \\
\quad E = 0; \\
\mathsf{end};
\end{array}\right\} \Pi_2
\end{array}\right\} \Pi
$$

Рис. 5.3: Определение четности

Пример 5.16. *Рассмотрим задачу определения чётности натурального числа (рис. 5.3). Мы докажем, что тройка* $\{\top\}\Pi\{E = \mathrm{ev}(x)\}$ *является полностью корректноой. С помощью* ev *мы обозначили функцию четности:*

$$
\mathrm{ev}(x) = \begin{cases} 1, & \text{если } x \text{ — чётное число,} \\ 0 & \text{иначе.} \end{cases}
$$

[1] **АКС**. $\{0 = 0\}u = 0;\{u = 0\}$

[2] **АКС**.

$$
\left|\begin{array}{c} \{\mathrm{ev}(\mathsf{succ}(\mathsf{succ}(u))) = 1 \wedge \mathsf{succ}(\mathsf{succ}(u)) \leqslant x + 1\} \\ u = \mathsf{succ}(\mathsf{succ}(u)); \\ \{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1\} \end{array}\right|
$$

[3] **УПР** : [2]. *Формула*

$$
\begin{aligned}
&(\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1 \wedge u < x) \to \\
&\qquad \to (\mathrm{ev}(\mathsf{succ}(\mathsf{succ}(u))) = 1 \wedge \mathsf{succ}(\mathsf{succ}(u)) \leqslant x + 1)
\end{aligned}
$$

истинна, поэтому

$$
\left|\begin{array}{c} \{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1 \wedge u < x\} \\ u = \mathsf{succ}(\mathsf{succ}(u)); \\ \{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1\} \end{array}\right|
$$

[4] **ЦКП** : [3]. $\varphi \Leftrightarrow \mathrm{ev}(u) = 1 \wedge u \leqslant x + 1$ — *инвариант,* $L_\varphi \Leftrightarrow x - u$ — *ограничитель. Значит,*

$$\{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1\}\Pi_1\{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1 \wedge \neg u < x\}.$$

[5] **УПР** : [4]. *Формула* $u = 0 \to (\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1)$ *истинна, поэтому*

$$\{u = 0\}\Pi_1\{\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1 \wedge \neg u < x\}.$$

Обозначим через ψ *следующую формулу:*

$$\psi \Leftrightarrow (\mathrm{ev}(x) = 1 \wedge u = x) \vee (\mathrm{ev}(x) = 0 \wedge \neg u = x).$$

[6] **ОСП** : [5]. *Формула*

$$(\mathrm{ev}(u) = 1 \wedge u \leqslant x + 1 \wedge \neg u < x) \to \psi$$

истинна, поэтому

$$\{u = 0\}\Pi_1\{\psi\}.$$

[7] **АКС**. $\{\mathsf{succ}(0) = \mathrm{ev}(x)\}E = \mathsf{succ}(0); \{E = \mathrm{ev}(x)\}$

[8] **АКС**. $\{0 = \mathrm{ev}(x)\}E = 0; \{E = \mathrm{ev}(x)\}$

[9] **УПР** : [7]. *Формула* $(\psi \wedge u = x) \to \mathsf{succ}(0) = \mathrm{ev}(x)$ *истинна, поэтому*

$$\{\psi \wedge u = x\}E = \mathsf{succ}(0); \{E = \mathrm{ev}(x)\}.$$

[10] **УПР** : [8]. *Формула* $(\psi \wedge \neg u = x) \to 0 = \mathrm{ev}(x)$ *истинна, поэтому*

$$\{\psi \wedge \neg u = x\}E = 0; \{E = \mathrm{ev}(x)\}.$$

[11] **ПВТ** : [9], [10]. $\{\psi\}\Pi_2\{E = \mathrm{ev}(x)\}$

[12] **СЛД** : [6], [11]. $\{u = 0\}\Pi_1\Pi_2\{E = \mathrm{ev}(x)\}$

[13] **СЛД** : [1], [13]. $\{0 = 0\}\Pi\{E = \mathrm{ev}(x)\}$

[14] **УПР** : [13]. *Формула* $\top \to 0 = 0$ *истинна, поэтому*

$$\{\top\}\Pi\{E = \mathrm{ev}(x)\}.$$

Пример 5.17. *Следующая программа предназначена для поиска наименьшего простого делителя числа:*

$$\left.\begin{array}{l} u = 2; \\ \textsf{while } x \bmod u > 0 \textsf{ do} \\ \quad u = \textsf{succ}(u); \\ \textsf{end}; \end{array}\right\}\Pi_1\Bigg\}\Pi$$

Докажем полную корректность тройки $\{x > 1\}\Pi\{u = \mathrm{md}(x)\}$, *где через* $\mathrm{md}(x)$ *обозначен наименьший простой делитель числа* x.

[1] **АКС**. $\{x > 1 \wedge 2 = 2\}u = 2; \{x > 1 \wedge u = 2\}$

[2] **АКС**.

$$\left|\begin{array}{c} \{x > 1 \wedge (\forall y)((y > 1 \wedge y < \textsf{succ}(u)) \to x \bmod y > 0)\} \\ u = \textsf{succ}(u); \\ \{x > 1 \wedge (\forall y)((y > 1 \wedge y < u) \to x \bmod y > 0)\} \end{array}\right|$$

[3] **УПР** : [2]. *Формула*

$$\begin{array}{r} (x > 1 \wedge (\forall y)((y > 1 \wedge y < u) \to x \bmod y > 0) \wedge x \bmod u > 0) \to \\ \to (x > 1 \wedge (\forall y)((y > 1 \wedge y < \textsf{succ}(u)) \to x \bmod y > 0)) \end{array}$$

истинна, поэтому

$$\left|\begin{array}{c} \{(x > 1 \wedge (\forall y)((y > 1 \wedge y < u) \to x \bmod y > 0) \wedge x \bmod u > 0\} \\ u = \textsf{succ}(u); \\ \{x > 1 \wedge (\forall y)((y > 1 \wedge y < u) \to x \bmod y > 0)\} \end{array}\right|.$$

[4] **ЦКП** : [3]. *Формула*

$$\varphi \mathrel{-\!\!\circ} x > 1 \wedge (\forall y)((y > 1 \wedge y < u) \to x \bmod y > 0)$$

является инвариантом, выражение $L_\varphi \mathrel{-\!\!\circ} \mathrm{md}(x) - u$ *— ограничителем. Значит, получаем* $\{\varphi\}\Pi_1\{\varphi \wedge \neg x \bmod u > 0\}$.

[5] **ОСП** : [4]. *Формула* $(\varphi \wedge \neg x \bmod u > 0) \to u = \mathrm{md}(x)$ *истинна, поэтому*

$$\{\varphi\}\Pi_1\{u = \mathrm{md}(x)\}.$$

[6] **ОСП** : [1]. *Формула* $(x > 1 \wedge u = 2) \to \varphi$ *истинна, поэтому*

$$\{x > 1 \wedge 2 = 2\}u = 2; \{\varphi\}.$$

[7] **СЛД** : [6], [5]. $\{x > 1 \wedge 2 = 2\}\Pi\{u = \mathrm{md}(x)\}$

[8] **УПР** : [7]. *Формула* $x > 1 \to (x > 1 \wedge 2 = 2)$ *истинна, поэтому*

$$\{x > 1\}\Pi\{u = \mathrm{md}(x)\}.$$

Рассмотрим некоторые способы упростить формальные выводы корректности.

Определение 5.13 (Допустимость)**.** *Аксиома или правило вывода называются* д о п у с т и м ы м и *в исчислении* $\mathfrak{I}$*, если их добавление к исчислению* $\mathfrak{I}$ *не увеличивает множество выводимых строк.*

То есть аксиома или правило допустимы, если всё, что можно вывести с их использованием, можно вывести и без них. Введём некоторые новые аксиомы и правила, которые могут упростить выводы, и докажем их допустимость.

1) **ПРИСВ** Аксиома:

$$\{\varphi\}x = e; \{(\exists u)((\varphi)^x_u \wedge x = (e)^x_u)\}$$

при условии, что переменная u отличается от x и не входит ни в φ, ни в e.

Доказательство. Аксиома **АКС**:

$$\{(\exists u)(((\varphi)^x_u)^x_e \wedge e = ((e)^x_u)^x_e)\}x = e; \{(\exists u)((\varphi)^x_u \wedge x = (e)^x_u)\}$$

Так как u не входит ни в φ, ни в e, то $(\exists u)(((\varphi)^x_u)^x_e \wedge e = (e)^x_u)$ следует из φ. По правилу **УПР** получаем:

$$\{\varphi\}x = e; \{(\exists u)((\varphi)^x_u \wedge x = (e)^x_u)\}. \qquad \square$$

****Задача 5.12.** Докажите истинность формулы:

$$\varphi \to (\exists u)(((\varphi)^x_u)^x_e \wedge e = ((e)^x_u)^x_e).$$

2) **ПРИСВ*** Аксиома:

$$\{\varphi\}x = e; \{\varphi \wedge x = e\}$$

при условии, что x не входит ни в φ, ни в e.

Доказательство. Предыдущая аксиома:

$$\{\varphi\}x = e; \{(\exists u)((\varphi)^x_u \wedge x = (e)^x_u)\}.$$

Так как x не входит в e, то $(e)^x_u \doteq e$. Так как x не входит в φ, то $(\varphi)^x_u \doteq \varphi$. Так как u не входит в формулу под квантором, то этот квантор можно удалить. По правилу **ОСП** получаем

$$\{\varphi\}x = e; \{\varphi \wedge x = e\}. \qquad \square$$

С помощью предыдущих аксиом и правила **СЛД** легко получить два правила:

3) **ПРИСВ**, **ПРИСВ*** Если выводимо $\{\varphi\}\Pi\{\psi\}$, то выводимы тройки

$$\{\varphi\}\Pi\ x = e; \{(\exists u)((\psi)^x_u \wedge x = (e)^x_u)\};$$

$$\{\varphi\}\Pi\ x = e; \{\psi \wedge x = e\}.$$

Для первого правила необходимо, чтобы u не входило ни в φ, ни в e, для второго — чтобы x не входило ни в φ, ни в e.

4) **ДОБ** Правило вывода: если выводимо $\{\varphi\}\Pi\{\psi\}$ и $\mathrm{LVar}(\Pi) \cap \mathrm{Var}(\theta) = \varnothing$, то выводимо

$$\{\varphi \wedge \theta\}\Pi\{\psi \wedge \theta\}.$$

Доказательство. Пусть $\mathcal{D}$ — вывод тройки $\{\varphi\}\Pi\{\psi\}$.

Прежде всего, в выводе могут быть использования правила **ЦКП**, в которых переменная z входит в θ. Покажем, как убрать все такие переходы, индукцией по их количеству. Если их нет, то делать ничего не надо. Пусть для выводов, содержащих меньше n переходов, доказано, и $\mathcal{D}$ содержит их n штук. Пусть t — тройка, которая выводится по правилу **ЦКП**. Следовательно, $\mathcal{D}$ имеет вид $\mathcal{D}_1 t \mathcal{D}_2$, причём в $\mathcal{D}_1$ и в $\mathcal{D}_2$ переходов по правилу **ЦКП** меньше чем в $\mathcal{D}$. Заменим в $\mathcal{D}_1$ все вхождения переменной z на новую переменную $\hat{z}$. Получим новый вывод $\hat{\mathcal{D}}_1$. Рассмотрим последовательность $\hat{\mathcal{D}}_1 t \mathcal{D}_1 \mathcal{D}_2$. Тогда эта последовательность является выводом той же тройки, что и исходный вывод, и он содержит меньше переходов по правилу **ЦКП** с использованием z.

Теперь можно использовать индукцию по выводу $\{\varphi\}\Pi\{\psi\}$.

Базис индукции. Если применяется аксиома **АКС**, то

$$\{(\varphi)_e^x\}x = e;\{\varphi\}.$$

Так как $x \in \mathrm{LVar}(\Pi)$, то $x \notin \mathrm{Var}(\theta)$. Следовательно, $(\theta)_e^x \doteqdot \theta$, поэтому:

$$\{(\varphi)_e^x \wedge \theta\}x = e;\{\varphi \wedge \theta\}$$

тоже является аксиомой.

Шаг индукции. Рассмотрим, например, индукционный шаг для **ПВТ**. В прежнем выводе тройка

$$\{\varphi\}\mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\psi\}$$

получалась из троек $\{\varphi \wedge e \neq 0\}\,\Pi_1\,\{\psi\}$ и $\{\varphi \wedge e = 0\}\,\Pi_2\,\{\psi\}$. По индукционному предположению тройки

$$\{\varphi \wedge \theta \wedge e \neq 0\}\,\Pi_1\,\{\psi \wedge \theta\} \text{ и } \{\varphi \wedge \theta \wedge e = 0\}\,\Pi_2\,\{\psi \wedge \theta\}$$

выводимы. Следовательно, выводима и тройка

$$\{\varphi \wedge \theta\}\mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\psi \wedge \theta\}$$

по тому же правилу **ПВТ**. □

***Задача 5.13.** Докажите, что последовательность $(\hat{\mathcal{D}}_1 t \mathcal{D}_1 \mathcal{D}_2)$, которая построена в доказательстве, является выводом той же тройки Хоара, что и вывод $(\mathcal{D}_1 t \mathcal{D}_2)$.

***Задача 5.14.** Завершите доказательство допустимости правила **ДОБ**, обосновав индукционный шаг для остальных правил исчисления Хоара.

****Задача 5.15.** Докажите, что если в исчислении $\mathfrak{IH}$ аксиомы **АКС** заменить аксиомами **ПРИСВ**, то множество выводимых троек Хоара не изменится.

***Задача 5.16.** Доказать, что следующие аксиомы допустимы в исчислении $\mathfrak{IH}$.

1) Для любой программы Π и любой формулы φ выводима тройка $\{\bot\}\,\Pi\,\{\varphi\}$.
2) Для любой программы без циклов Π и любой формулы φ выводима тройка $\{\varphi\}\,\Pi\,\{\top\}$.

***Задача 5.17.** Доказать, что следующие правила вывода допустимы в исчислении $\mathfrak{IH}$.

1) Если выводима тройка

$$\{\varphi \wedge x < e\}\,\Pi_1\,\{\varphi \wedge x = e\}\,,$$

то выводима тройка

$$\{\varphi\}\,\mathsf{while}\ x < e\ \mathsf{do\ if}\ x < e\ \mathsf{then}\ \Pi_1\ \mathsf{end};\ \mathsf{end};\{\varphi \wedge x \geqslant e\}\,.$$

Здесь Π_1 — программа, x — переменная, e — выражение.

2) Если выводимы тройки

$$\{\varphi \wedge e \neq 0\}\,\Pi_1\,\{\varphi \wedge x = y\}\ \text{ и }\ \{\varphi \wedge e = 0\}\,\Pi_2\,\{\varphi \wedge x > y\}\,,$$

```
algorithm Div;
arguments x, y;
returns v;
    u = 0; v = 0;
    while u + y ⩽ x do
        u = u + y;
        v = v + 1;
    end;
end;
```

Π_1

Рис. 5.4: Деление чисел

то выводима тройка

$$\{\varphi\}\,\mathsf{while}\ x < y\ \mathsf{do}\ \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\ \mathsf{end};\{\varphi \wedge x \geqslant y\}\,.$$

Здесь Π_1 и Π_2 — программы, x и y — переменные, e — выражение, φ — формула.

3) Если выводима тройка

$$\{\varphi \wedge e \neq 0 \wedge L = z\}\,\Pi_1\,\{\varphi \wedge e' = 0 \wedge L < z\}\,,$$

то выводима тройка

$$\{\varphi\}\,\mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{if}\ e'\ \mathsf{!=}\ 0\ \mathsf{then}\ \Pi_2\ \mathsf{end};\ \mathsf{end};\{\varphi \wedge e = 0\}\,.$$

Здесь Π_1 и Π_2 — программы, φ — формула, e, e' и L — выражения, z — новая переменная. Можно считать известным, что $\underset{\mathfrak{IH}}{\rhd}\ \{\bot\}\,\Pi\,\{\psi\}$ для любой программы Π и любой формулы ψ.

Пример 5.18. *Докажем корректность программы деления, приведённой на рис. 5.4.*

[1] **ПРИСВ***. $\{\top\}u = 0;\{u = 0\}$[3].

[3] На самом деле здесь использована аксиома **ПРИСВ*** и применено правило **ОСП**, чтобы убрать из постусловия $\top$: формула $\varphi \wedge \top \to \varphi$, очевидно, истинна.

[2] **ПРИСВ*** : [1]. $\{\top\}u = 0; v = 0; \{u = 0 \wedge v = 0\}$.

[3] **ОСП** : [2]. *Формула*

$$u = 0 \wedge v = 0 \to vy = u \wedge u \leqslant x,$$

истинна, следовательно,

$$\{\top\}\, u = 0; v = 0; \{vy = u \wedge u \leqslant x\}\,.$$

[4] **ДОБ** : [3]. $\{y > 0\}u = 0; v = 0; \{vy = u \wedge u \leqslant x \wedge y > 0\}$[4].

[5] **ПРИСВ**.

$$\left|\begin{array}{c} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x}\} \\ u = u + y; \\ \{(\exists w)(vy = w \wedge w \leqslant x \wedge w + y \leqslant x \wedge u = w + y \wedge x - w = \hat{x})\} \end{array}\right|$$

[6] **ОСП** : [5]. *Формула*

$$(\exists w)(vy = w \wedge w \leqslant x \wedge w + y \leqslant x \wedge u = w + y \wedge x - w = \hat{x}) \to \\ \to vy = u - y \wedge u \leqslant x \wedge x - u = \hat{x} - y$$

истинна, поэтому

$$\left|\begin{array}{c} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x}\} \\ u = u + y; \\ \{vy = u - y \wedge u \leqslant x \wedge x - u = \hat{x} - y\} \end{array}\right|$$

[7] **ПРИСВ** : [6].

$$\left|\begin{array}{c} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x}\} \\ u = u + y;\ v = v + 1; \\ \{(\exists z)(zy = u - y \wedge u \leqslant x \wedge x - u = \hat{x} - y \wedge v = z + 1)\} \end{array}\right|$$

[4] Как и в предыдущем замечании здесь применено правило **ДОБ**, а затем — **УПР**, чтобы убрать из предусловия $\top$: формула $\varphi \to \varphi \wedge \top$ истинна. В дальнейшем, мы будем убирать $\top$ такими способами, не упоминая об этом специально.

[8] **ОСП** : [7]. *Формула*

$$(\exists z)(zy = u - y \wedge u \leqslant x \wedge x - u = \hat{x} - y \wedge v = z + 1) \to \\ \to vy = u \wedge u \leqslant x \wedge x - u = \hat{x} - y$$

истинна, из чего следует

$$\left|\begin{matrix} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x}\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leqslant x \wedge x - u = \hat{x} - y\} \end{matrix}\right|$$

[9] **ДОБ** : [8].

$$\left|\begin{matrix} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x} \wedge y > 0\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leqslant x \wedge x - u = \hat{x} - y \wedge y > 0\} \end{matrix}\right|$$

[10] **ОСП** : [9]. *Формула*

$$vy = u \wedge u \leqslant x \wedge x - u = \hat{x} - y \wedge y > 0 \to \\ \to vy = u \wedge u \leqslant x \wedge x - u < \hat{x} \wedge y > 0$$

истинна, получаем

$$\left|\begin{matrix} \{vy = u \wedge u \leqslant x \wedge u + y \leqslant x \wedge x - u = \hat{x} \wedge y > 0\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leqslant x \wedge x - u < \hat{x} \wedge y > 0\} \end{matrix}\right|$$

[11] **ЦКП** : [10].

$$\left|\begin{matrix} \{vy = u \wedge u \leqslant x \wedge y > 0\} \\ \Pi_1 \\ \{vy = u \wedge u \leqslant x \wedge y > 0 \wedge u + y > x\} \end{matrix}\right|$$

[12] **ОСП** : [11]. *Формула*

$$vy = u \wedge u \leqslant x \wedge y > 0 \wedge u + y > x \to v = \left[x/y\right]$$

algorithm Sum;
arguments x;
returns r;
    $r = 0;\ y = x;$
    while $0 < y$ do
        $r = r + y \bmod 10;$ } Π_1
        $y = y \,/\, 10;$ } Π_1
    end; } Π_2
end;

Рис. 5.5: Сумма цифр числа.

истинна, поэтому

$$\{vy = u \wedge u \leqslant x \wedge y > 0\}\,\Pi_1\,\{v = [x/y]\}$$

С помощью квадратных скобок мы обозначаем целую часть частного, например,

$$[10/3] = 3; \quad [6/4] = 1.$$

[13] **СЛД** : [4], [12]. $\{y > 0\}\,u = 0; v = 0; \Pi_1\,\{v = [x/y]\}$

Пример 5.19. *Рассмотрим алгоритм нахождения суммы цифр числа, приведённый на рис. 5.5. Пусть $l(i)$ — последняя цифра числа i, а $s(i)$ — сумма его цифр.*

[1] **ПРИСВ***. $\{\top\}r = 0; \{r = 0\}$

[2] **ПРИСВ*** : [1]. $\{\top\}r = 0; y = x; \{r = 0 \wedge y = x\}$.

[3] **ОСП** : [2]. $\{\top\}r = 0; y = x; \{s(x) = s(y) + r\}$.

[4] **АКС**.

$$\left|\begin{array}{c}\{s(x) + l(y) = s(y) + r + y \bmod 10\}\\ r = r + y \bmod 10;\\ \{s(x) + l(y) = s(y) + r\}\end{array}\right|$$

[5] **УПР** : [4].

$$\left|\begin{array}{c}\{s(x) = s(y) + r \wedge 0 < y\}\\ r = r + y \bmod 10;\\ \{s(x) + l(y) = s(y) + r\}\end{array}\right|$$

[6] **ОСП** : [5].

$$\{s(x) = s(y) + r \wedge 0 < y\} r = r + y \bmod 10; \{s(x) = s(y/10) + r\}$$

[7] **АКС**. $\{s(x) = s(y/10) + r\} y = y \,/\, 10; \{s(x) = s(y) + r\}$

[8] **СЛД** : [6], [7].

$$\left|\begin{array}{c}\{s(x) = s(y) + r \wedge 0 < y\}\\ r = r + y \bmod 10; y = y \,/\, 10;\\ \{s(x) = s(y) + r\}\end{array}\right|$$

[9] **ЦКП** : [8]. *Ограничитель — y.*

$$\left|\begin{array}{c}\{s(x) = s(y) + r\}\\ \Pi_2\\ \{s(x) = s(y) + r \wedge \neg 0 < y\}\end{array}\right|$$

[10] **ОСП** : [9]. $\{s(x) = s(y) + r\}\Pi_2\{s(x) = r\}$

[11] **СЛД** : [3], [10]. $\{\top\}\Pi_{Sum}\{s(x) = r\}$

Пример 5.20. *Следующая программа определяет, является ли число x простым.*

$$\left.\begin{array}{l} y = 2;\ f = 1;\\ \left.\begin{array}{l}\text{while } y < x \text{ do}\\ \quad \left.\begin{array}{l}\text{if } x \bmod y == 0 \text{ then}\\ \qquad f = 0;\ y = x;\\ \text{end;}\end{array}\right\}\Pi_1\\ \quad y = \text{succ}(y);\\ \text{end;}\end{array}\right\}\Pi_2\end{array}\right\}\Pi$$

Докажем полную корректность тройки $\{x > 1\}\Pi\{f = P(x)\}$, где

$$P(x) = \begin{cases} 1, & \text{если } x \text{ — простое число,}\\ 0, & \text{в противном случае.}\end{cases}$$

[1] **ПРИСВ***. $\{\top\}y = 2; \{y = 2\}$

[2] **ПРИСВ*** : [1]. $\{\top\}y = 2;\ f = 1; \{y = 2 \wedge f = 1\}$

[3] **ДОБ** : [2]. $\{x > 1\}y = 2;\ f = 1; \{y = 2 \wedge f = 1 \wedge x > 1\}$

Обозначим через φ следующую формулу:

$$\varphi \leftrightarrows \Big((f = 1 \wedge y \leqslant x \wedge (\forall z)((z > 1 \wedge z < y) \to x \bmod z > 0)) \bigvee \bigvee (f = 0 \wedge y = x + 1 \wedge P(x) = 0)\Big) \wedge x > 1 \wedge y \geqslant 2.$$

[4] **ОСП** : [3]. *Формула* $(y = 2 \wedge f = 1 \wedge x > 1) \to \varphi$ *истинна, поэтому*

$$\{x > 1\}y = 2;\ f = 1; \{\varphi\}.$$

[5] **ПРИСВ***. $\{\top\}f = 0;\ \{f = 0\}$

[6] **ПРИСВ*** : [5]. $\{\top\}f = 0;\ y = x; \{f = 0 \wedge y = x\}$

[7] **ДОБ** : [6].

$$\left|\begin{array}{c} \{x > 1 \wedge P(x) = 0\} \\ f = 0;\ y = x; \\ \{f = 0 \wedge y = x \wedge x > 1 \wedge P(x) = 0\} \end{array}\right|$$

Рассмотрим формулу $\varphi \wedge y < x$:

$$\varphi \wedge y < x \equiv f = 1 \wedge \mathsf{succ}(y) \leqslant x \wedge$$
$$\wedge (\forall z)((z > 1 \wedge z < \mathsf{succ}(y) - 1) \to x \bmod z > 0) \wedge x > 1 \wedge y \geqslant 2.$$

[8] **УПР** : [7]. *Формула*

$$(\varphi \wedge y < x \wedge x \bmod y = 0) \to (x > 1 \wedge P(x) = 0)$$

истинна, поэтому получим

$$\left|\begin{array}{c} \{\varphi \wedge y < x \wedge x \bmod y = 0\} \\ f = 0;\ y = x; \\ \{f = 0 \wedge y = x \wedge x > 1 \wedge P(x) = 0\} \end{array}\right|$$

Обозначим через ψ следующую формулу:

$$\psi \leftrightarrows \Big((f = 1 \wedge \mathsf{succ}(y) \leqslant x \wedge \\ \wedge (\forall z)((z > 1 \wedge z < \mathsf{succ}(y)) \to x \bmod z > 0)) \bigvee \\ \bigvee (f = 0 \wedge \mathsf{succ}(y) = x + 1 \wedge P(x) = 0)\Big) \wedge x > 1 \wedge \mathsf{succ}(y) \geqslant 2.$$

Легко видеть, что $(\varphi)^y_{\mathsf{succ}(y)} \leftrightarrows \psi$.

[9] **ОСП** : [8]. *Формула*

$$(f = 0 \wedge y = x \wedge x > 1 \wedge P(x) = 0) \to \psi$$

истинна, поэтому

$$\{\varphi \wedge y < x \wedge x \bmod y = 0\} f = 0;\ y = x; \{\psi\}.$$

[10] **НВТ** : [9]. *Формула* $(\varphi \wedge y < x \wedge \neg x \bmod y = 0) \to \psi$ *истинна, поэтому*

$$\{\varphi \wedge y < x\} \Pi_1 \{\psi\}.$$

[11] **АКС**. $\{\psi\} y = \mathsf{succ}(y); \{\varphi\}$

[12] **СЛД** : [10], [11]. $\{\varphi \wedge y < x\} \Pi_1 y = \mathsf{succ}(y); \{\varphi\}$

[13] **ЦКП** : [12]. φ *— инвариант цикла* Π_2, $L_\varphi \leftrightarrows x + 1 - y$ *— ограничитель. Значит,*

$$\{\varphi\} \Pi_2 \{\varphi \wedge \neg y < x\}.$$

[14] **ОСП** : [13]. *Формула* $(\varphi \wedge \neg y < x) \to f = P(x)$ *истинна, поэтому*

$$\{\varphi\} \Pi_2 \{f = P(x)\}.$$

[15] **СЛД** : [4], [14]. $\{x > 1\} \Pi \{f = P(x)\}$

Пример 5.21. *Докажем корректность программы на рис. 5.6 на следующей странице, которая находит наименьшее число после* x*, в пятеричной записи которого все цифры одинаковы.*

Введём следующие обозначения.

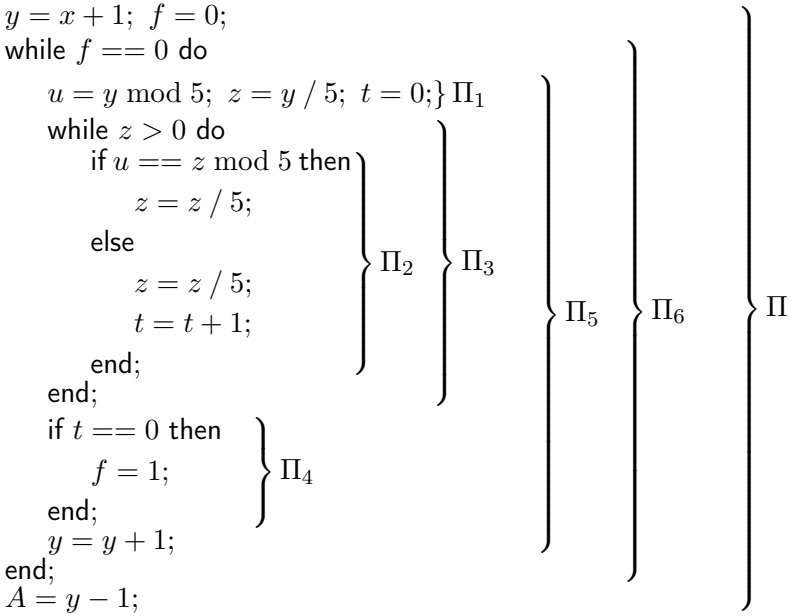


Рис. 5.6: Следующее за x число, состоящее из одинаковых цифр

- $R(x) = \begin{cases} 1, & \textit{если все пятеричные цифры числа } x \textit{ одинаковы} \\ 0, & \textit{в противном случае;} \end{cases}$
- $d(x, i)$ — *i-ая цифра в пятеричной записи числа x;*
- $l(z)$ — *количество цифр в пятеричной записи числа z;*
- $K(y, i)$ — *количество цифр от y_1 до y_{i-1} отличных от y_0 (здесь $y_n y_{n-1} \dots y_1 y_0$ — пятеричная запись числа y);*
- $K(y)$ — *количество пятеричных цифр числа y отличных от последней цифры;*
- $M(x)$ — *наименьшее число после x, в пятеричной записи которого все цифры одинаковы (такое число всегда существует).*

Докажем полную корректность тройки $\{\top\}\Pi\{A = M(x)\}$. *Введём две формулы* φ *и* ψ:

$$\varphi \twoheadleftarrow (\forall a)((a > x \wedge a < y - 1) \to R(a) = 0) \bigwedge$$
$$\bigwedge \big((f = 0 \wedge (y = x + 1 \vee R(y - 1) = 0)) \vee (f = 1 \wedge R(y - 1) = 1)\big);$$

$$\psi \twoheadleftarrow u = d(y, 0) \wedge (\exists i)\left(i \leqslant l(y) \wedge z = \left[y/5^i\right] \wedge t = K(y, i)\right).$$

[1] **ПРИСВ***. $\{\top\}y = x + 1; \{y = x + 1\}$

[2] **ПРИСВ*** : [1]. $\{\top\}y = x + 1;\ f = 0; \{y = x + 1 \wedge f = 0\}$

[3] **ОСП** : [2]. *Формула* $(y = x + 1 \wedge f = 0) \to \varphi$ *истинна, поэтому*

$$\{\top\}y = x + 1;\ f = 0; \{\varphi\}.$$

[4] **ПРИСВ***. $\{\top\}u = y \bmod 5; \{u = y \bmod 5\}$

[5] **ПРИСВ*** : [4]. $\{\top\}u = y \bmod 5;\ z = y\ /\ 5; \{u = y \bmod 5 \wedge z = y/5\}$

[6] **ПРИСВ*** : [5]. $\{\top\}\Pi_1\{u = y \bmod 5 \wedge z = y/5 \wedge t = 0\}$

[7] **ОСП** : [6]. *Формула* $(u = y \bmod 5 \wedge z = y/5 \wedge t = 0) \to \psi$ *истинна, поэтому*

$$\{\top\}\Pi_1\{\psi\}.$$

[8] **ПРИСВ**.

$$\left|\begin{array}{c} \{\psi \wedge z > 0 \wedge u = z \bmod 5\} \\ z = z\ /\ 5; \\ \{(\exists z')((\psi)^z_{z'} \wedge z' > 0 \wedge u = z' \bmod 5 \wedge z = z'/5)\} \end{array}\right|$$

[9] **ОСП** : [8]. *Формула*

$$(\exists z')((\psi)^z_{z'} \wedge z' > 0 \wedge u = z' \bmod 5 \wedge z = z'/5) \to \psi$$

истинна, поэтому

$$\{\psi \wedge z > 0 \wedge u = z \bmod 5\}z = z\ /\ 5; \{\psi\}.$$

[10] **ПРИСВ**.

$$\left|\begin{array}{c} \{\psi \wedge z > 0 \wedge \neg u = z \bmod 5\} \\ z = z \,/\, 5; \\ \{(\exists z')((\psi)^{z}_{z'} \wedge z' > 0 \wedge \neg u = z' \bmod 5 \wedge z = z'/5)\} \end{array}\right|$$

[11] **ПРИСВ** : [10].

$$\left|\begin{array}{c} \{\psi \wedge z > 0 \wedge \neg u = z \bmod 5\} \\ z = z \,/\, 5;\ t = t + 1; \\ \left\{\begin{array}{l} (\exists t')\big((\exists z')((\psi)^{z\ t}_{z'\ t'} \wedge z' > 0 \wedge \\ \qquad \wedge \neg u = z' \bmod 5 \wedge z = z'/5) \wedge t = t' + 1\big) \end{array}\right\} \end{array}\right|$$

[12] **ОСП** : [11]. *Формула*

$$(\exists t')\Big((\exists z')((\psi)^{z\ t}_{z'\ t'} \wedge z' > 0 \wedge \neg u = z' \bmod 5 \wedge \\ \wedge z = z'/5) \wedge t = t' + 1\Big) \to \psi$$

истинна, поэтому

$$\{\psi \wedge z > 0 \wedge \neg u = z \bmod 5\} z = z \,/\, 5;\ t = t + 1; \{\psi\}.$$

[13] **ПВТ** : [9], [12]. $\{\psi \wedge z > 0\}\Pi_2\{\psi\}$

[14] **ЦКП** : [13]. ψ — *инвариант*, z — *ограничитель. Значит,*

$$\{\psi\}\Pi_3\{\psi \wedge \neg z > 0\}.$$

[15] **ОСП** : [13]. *Формула* $(\psi \wedge \neg z > 0) \to t = K(y)$ *истинна, поэтому*

$$\{\psi\}\Pi_3\{t = K(y)\}.$$

[16] **СЛД** : [7], [15]. $\{\top\}\Pi_1\Pi_3\{t = K(y)\}$

[17] **ДОБ** : [16]. $\{\varphi \wedge f = 0\}\Pi_1\Pi_3\{t = K(y) \wedge \varphi \wedge f = 0\}$

[18] **ПРИСВ**.

$$\left|\begin{array}{c} \{\varphi \wedge f = 0 \wedge t = K(y) \wedge t = 0\} \\ f = 1; \\ \{(\exists f')((\varphi)^{f}_{f'} \wedge f' = 0 \wedge t = K(y) \wedge t = 0 \wedge f = 1)\} \end{array}\right|$$

Заменим в формуле φ y *на* $y+1$.

$$(\varphi)^y_{y+1} \leftrightarrows (\forall a)((a > x \wedge a < (y+1)-1) \to R(a) = 0) \bigwedge$$
$$\bigwedge \Big((f = 0 \wedge (y+1 = x+1 \vee R((y+1)-1) = 0)) \vee$$
$$\vee (f = 1 \wedge R((y+1)-1) = 1) \Big)$$

[19] **ОСП** : [18]. *Формула*

$$(\exists f')((\varphi)^f_{f'} \wedge f' = 0 \wedge t = K(y) \wedge t = 0 \wedge f = 1) \to (\varphi)^y_{y+1}$$

истинна, поэтому

$$\{\varphi \wedge f = 0 \wedge t = K(y) \wedge t = 0\} f = 1; \{(\varphi)^y_{y+1}\}.$$

[20] **НВТ** : [19]. *Формула* $(\varphi \wedge f = 0 \wedge t = K(y) \wedge \neg t = 0) \to (\varphi)^y_{y+1}$ *истинна, поэтому*

$$\{\varphi \wedge f = 0 \wedge t = K(y)\} \Pi_4 \{(\varphi)^y_{y+1}\}.$$

[21] **СЛД** : [17], [20]. $\{\varphi \wedge f = 0\} \Pi_1 \Pi_3 \Pi_4 \{(\varphi)^y_{y+1}\}$

[22] **АКС**. $\{(\varphi)^y_{y+1}\} y = y+1; \{\varphi\}$

[23] **СЛД** : [21], [22]. $\{\varphi \wedge f = 0\} \Pi_5 \{\varphi\}$

[24] **ЦКП** : [23]. φ — *инвариант,* $\lfloor M(x) - y \rfloor$ — *ограничитель. Значит,*

$$\{\varphi\} \Pi_6 \{\varphi \wedge \neg f = 0\}.$$

[25] **ОСП** : [24]. *Формула* $(\varphi \wedge \neg f = 0) \to y - 1 = M(x)$ *истинна, поэтому*

$$\{\varphi\} \Pi_6 \{y - 1 = M(x)\}.$$

[26] **АКС**. $\{y - 1 = M(x)\} A = y - 1; \{A = M(x)\}$

[27] **СЛД** : [25], [26]. $\{\varphi\} \Pi_6\ A = y - 1; \{A = M(x)\}$

[28] **СЛД** : [3], [27]. $\{\top\} \Pi \{A = M(x)\}$

Задача 5.18. Докажите корректность ранее написанной программы нахождения остатка (используя сложение и вычитание).

*__Задача 5.19.__ Напишите программы и докажите их корректность:

1) нахождение чисел Фибоначчи;
2) вычисление факториала;
3) нахождение двоичного логарифма.

Из операций разрешается использовать только succ.

Задача 5.20. Написать программы для решения следующих задач и доказать их правильность с помощью исчисления Хоара. Разрешается использовать операции сравнения, succ, $+$, $-$, $\times$, $/$, $\bmod$, $P(x)$ (проверка на простоту: $P(x) = 1$, если число x является простым, $P(x) = 0$ в противном случае).

1) Найти количество чисел на отрезке $[x, y)$, последняя десятичная цифра которых нечётна.
2) Найти сумму простых чисел на отрезке $[x, y)$.
3) Найти количество чётных цифр в семеричной записи числа x.
4) Найти количество чётных делителей числа x.
5) Найти произведение ненулевых пятеричных цифр числа x.

§ 5.5. *Исчисление предусловий

Легко понять, что для одной и той же программы можно составить много корректных троек Хоара. Однако не все из них представляют практический интерес. Например, если предусловие является тождественно ложным, то такая тройка будет полностью корректной, однако это ничего не говорит о свойствах программы. Поэтому интерес представляют, прежде всего, такие тройки, у которых предусловие описывает как можно больше ситуаций, то есть является как можно более слабым.

Определение 5.14 (Слабейшее предусловие). *Пусть* Π *— программа,* ψ *— условие. Условие* φ *называется* *слабейшим предусловием для* $\Pi\{\psi\}$, *если тройка Хоара* $\{\varphi\}\Pi\{\psi\}$ *частично корректна и для любой конфигурации* σ, *если* $\Pi(\sigma)$ *определено и* $\Pi(\sigma) \models \psi$, *то* $\sigma \models \varphi$.

Интуитивно, слабейшее предусловие — необходимое и достаточное условие для того, чтобы после выполнения программы выполнялось постусловие.

Определение 5.15 (Полное слабейшее предусловие). *Слабейшее предусловие* φ *для* $\Pi\{\psi\}$ *называется* *полным, если тройка* $\{\varphi\}\Pi\{\psi\}$ *является полностью корректной.*

Следствие 5.12. *Всякое полное слабейшее предусловие является слабейшим предусловием.*

Следствие 5.13. *Если тройка* $\{\top\}\Pi\{\psi\}$ *частично (полностью) корректна, то* $\top$ *— (полное) слабейшее предусловие для* $\Pi\{\psi\}$.

Задача 5.21. Докажите оба следствия.

Следствие 5.14 (Об эквивалентность предусловий). *Если* φ *и* ψ *— полные слабейшие предусловия для* $\Pi\{\theta\}$, *то* $\varphi \equiv \psi$.

Доказательство. Пусть $\sigma \models \varphi$, тогда $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \theta$. Так как ψ — полное слабейшее предусловие для $\Pi\{\theta\}$, то $\sigma \models \psi$. Итак, из φ следует ψ. Аналогично в обратную сторону. □

Пример 5.22. *Пусть* Π *— программа такая, что* $\Pi(\sigma)$ *не определено для любого* σ. *Тогда для любого постусловия* ψ *любое предусловие* φ *является слабейшим для* $\Pi\{\psi\}$, *но не является полным слабейшим.*

Предположим, что мы имеем программу Π и постусловие ψ. Это означает, что в наличии есть текст программы и требования того, что она должна делать. Требуется определить слабейшее предусловие для $\Pi\{\psi\}$, то есть найти необходимые и достаточные условия, при которых программа Π действительно делает то, что нужно.

Для решения этой задачи введём следующее исчисление предусловий — $\mathfrak{IP}$. Строками $\mathfrak{IP}$ будут те же тройки Хоара, но выводимость тройки $\{\varphi\}\Pi\{\psi\}$ будет означать, что φ — полное слабейшее предусловие для $\Pi\{\psi\}$.

Мы сразу будем давать правила и доказывать непротиворечивость исчисления предусловий. Мы приведём только правила для нахождения полных слабейших предусловий. Поскольку все предусловия, о которых мы будем говорить в дальнейшем будут полными, то слово «полное» мы часто будем опускать.

1) **СПР$_0$** Обе аксиомы будут такими же, как и в исчислении Хоара. Для аксиомы $\{\varphi\};\{\varphi\}$ утверждение очевидно, так как $(;)(\sigma) = \sigma$.

2) **СПР** $\{(\psi)^x_e\}x = e;\{\psi\}$, то есть $(\psi)^x_e$ — слабейшее предусловие для $x = e;\{\psi\}$.

   Доказательство. Очевидно, $(x = e;)(\sigma)$ всегда определено. Пусть $(x = e;)(\sigma) \models \psi$. $(x = e;)(\sigma) = \tau$, если $\tau = \sigma$ кроме того, что $\tau x = \sigma(e)$. Следовательно, ψ истинно при подстановке τx вместо x или, что то же самое, $\sigma(e)$. Но тогда $\sigma \models (\psi)^x_e$. □

3) **СЭП** Если $\{\varphi\}\Pi_1\{\psi\}$ и формула $\theta \leftrightarrow \varphi$ истинна, то $\{\theta\}\Pi_1\{\psi\}$.

Задача 5.22. Докажите, что с помощью правила **СЭП** можно получать только тройки с слабейшими предусловиями, если исходная тройка такова.

4) **ССЛ** Если выводимы тройки $\{\varphi\}\Pi_1\{\theta\}$ и $\{\theta\}\Pi_2\{\psi\}$, то выводимо $\{\varphi\}\Pi_1\Pi_2\{\psi\}$.

   Доказательство. Пусть $(\Pi_1\Pi_2)(\sigma) \models \psi$, по определению семантики следования, $\Pi_2(\Pi_1(\sigma)) \models \psi$. Значит, $\Pi_1(\sigma) \models \theta$, так как θ — слабейшее предусловие для $\Pi_2\{\psi\}$. Следовательно, $\sigma \models \varphi$, так как φ — слабейшее предусловие для $\Pi_1\{\theta\}$. То есть,

если $(\Pi_1\Pi_2)(\sigma) \models \psi$, то $\sigma \models \varphi$, что означает, φ — слабейшее предусловие для $\Pi_1\Pi_2\{\psi\}$.

Корректность тройки $\{\varphi\}\Pi_1\Pi_2\{\psi\}$ сразу получается из корректности $\{\varphi\}\Pi_1\{\theta\}$ и $\{\theta\}\Pi_2\{\psi\}$ по правилу **СЛД**. □

5) **СНВ** Пусть

$$\Pi \twoheadleftarrow \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};$$

Пусть φ — слабейшее предусловие для $\Pi_1\{\psi\}$. Тогда

$$\theta \twoheadleftarrow \varphi \wedge e \neq 0 \vee \psi \wedge e = 0$$

слабейшее предусловие для $\Pi\{\psi\}$.
Доказательство. Предположим, что $\Pi(\sigma) \models \psi$.

а) Если $\sigma \models e \neq 0$, то $\Pi(\sigma) = \Pi_1(\sigma)$. $\Pi_1(\sigma) \models \psi$, значит $\sigma \models \varphi$. Итак, $\sigma \models e \neq 0$ и $\sigma \models \varphi$, следовательно, $\sigma \models \varphi \wedge e \neq 0$ и $\sigma \models \theta$.

б) Если $\sigma \models e = 0$, то $\Pi(\sigma) = \sigma$. То есть $\sigma \models \psi \wedge e = 0$ и $\sigma \models \theta$.

В любом случае, как мы видим, $\sigma \models \theta$.

Теперь докажем корректность тройки $\{\theta\}\Pi\{\psi\}$.

$$\theta \wedge e \neq 0 \twoheadleftarrow (\varphi \wedge e \neq 0 \vee \psi \wedge e = 0) \wedge e \neq 0 \equiv \varphi \wedge e \neq 0$$

$$\theta \wedge e = 0 \twoheadleftarrow (\varphi \wedge e \neq 0 \vee \psi \wedge e = 0) \wedge e = 0 \equiv \psi \wedge e = 0$$

Корректность $\{\varphi \wedge e \neq 0\}\Pi_1\{\psi\}$ получается из $\{\varphi\}\Pi_1\{\psi\}$ усилением предусловия. Формула $\theta \wedge e = 0 \to \psi$ истинна. Следовательно, корректность тройки $\{\theta\}\Pi\{\psi\}$ получается по правилу **НВТ**. □

6) **СПВ** Пусть

$$\Pi \Leftarrow \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};$$

Пусть φ_1 — слабейшее предусловие для $\Pi_1\{\psi\}$, и φ_2 — слабейшее предусловие для $\Pi_2\{\psi\}$. Тогда

$$\varphi \Leftarrow \varphi_1 \wedge e \neq 0 \vee \varphi_2 \wedge e = 0$$

слабейшее предусловие для $\Pi\{\psi\}$.

В частности, если $\varphi_1 \equiv \varphi_2$, то

$$\varphi \equiv \varphi_1 \wedge e \neq 0 \vee \varphi_1 \wedge e = 0 \equiv \varphi_1 \wedge (e \neq 0 \vee e = 0) \equiv \varphi_1 \equiv \varphi_2.$$

Задача 5.23. Докажите, что φ — действительно слабейшее предусловие для $\Pi\{\psi\}$.

Перед формулировкой правила нахождения слабейшего предусловия для оператора цикла отметим следующее.

Следствие 5.15. *Если программа* Π *не содержит циклов и в исчислении* $\mathfrak{IP}$ *выводимо* $\{\varphi\}\Pi\{\psi\}$*, то в исчислении* $\mathfrak{IH}$ *также выводимо* $\{\varphi\}\Pi\{\psi\}$.

Доказательство. Всякая аксиома из $\mathfrak{IP}$ является также аксиомой из $\mathfrak{IH}$. Если же тройка получена по одному из правил **СЭП**, **ССЛ**, **СНВ**, **СПВ**, то её можно вывести и в $\mathfrak{IH}$, что было показано выше при доказательстве непротиворечивости этих правил. □

7) **СЦП** Пусть

$$\Pi \Leftarrow \mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};$$

Пусть $\varphi_0 \Leftarrow \psi \wedge e = 0$, φ'_{i+1} — полное слабейшее предусловие для $\Pi_1\{\varphi_i\}$ и $\varphi_{i+1} \Leftarrow \varphi'_{i+1} \wedge e \neq 0$. Пусть

$$\varphi \equiv \bigvee\{\varphi_i : i \in \omega\}.$$

Тогда φ — полное слабейшее предусловие для $\Pi\{\psi\}$.

Доказательство. Пусть $\Pi(\sigma) = \tau$ определено и $\tau \models \psi$. Значит существует число n и последовательность конфигураций $(\sigma^i)_{i=0}^n$ такие, что

$$\sigma^0 = \sigma, \quad \sigma^n = \tau, \quad \sigma^{i+1} = \Pi_1(\sigma^i), \quad \sigma^i \models e \neq 0 \iff i < n.$$

Если $n = 0$, то $\sigma = \tau$, $\sigma \models \psi \wedge e = 0$ и, следовательно, $\sigma \models \varphi$.

Пусть $n > 0$. Индукцией по i докажем, что $\sigma^{n-i} \models \varphi_i$.

Базис индукции. Для $i = 0$ имеем $\sigma^n = \tau$, $\tau \models \psi \wedge e = 0$, $\tau \models \varphi_0$.

Шаг индукции. Пусть для i выполнено $\sigma^{n-i} \models \varphi_i$. Так как $\Pi_1(\sigma^{n-(i+1)}) = \sigma^{n-i}$, и φ'_{i+1} — слабейшее предусловие для $\Pi_1\{\varphi_i\}$, то $\sigma^{n-(i+1)} \models \varphi'_{i+1}$. Так как $n-(i+1) < n$, то $\sigma^{n-(i+1)} \models e \neq 0$. Итак, $\sigma^{n-(i+1)} \models \varphi_{i+1}$.

Следовательно, $\sigma^0 \models \varphi_n$ и $\sigma \models \varphi$.

Теперь докажем полную корректность тройки $\{\varphi\}\Pi\{\psi\}$. Для этого докажем полную корректность тройки

$$\{\varphi \wedge e \neq 0 \wedge L = z\}\Pi_1\{\varphi \wedge L < z\}$$

для некоторого L и переменной z, которая нигде больше не встречается. Возьмём $L \mathrel{\dot{-}} \min\{i \in \omega : \varphi_i\}$. Заметим, что если $\sigma \models \varphi$, то $\sigma(L)$ определено.

Предположим, что $\sigma \models \varphi \wedge e \neq 0 \wedge L = z$. Пусть $i = \sigma(L)$. Следовательно, $\sigma \models \varphi_i$ и $\sigma \not\models \varphi_j$ для $j < i$. Если $i = 0$, то $\sigma \models \psi \wedge e = 0$, что противоречит тому, что $\sigma \models e \neq 0$. Следовательно, $i > 0$. Тогда $\varphi_i \wedge e \neq 0 \equiv \varphi'_i \wedge e \neq 0$. Так как φ'_i — слабейшее предусловие для $\Pi_1\{\varphi_{i-1}\}$, то $\Pi_1(\sigma) \models \varphi_{i-1}$. Следовательно, $\Pi_1(\sigma) \models \varphi$, и $\Pi_1(\sigma)(L) < i$. Так как $z \notin \mathrm{LVar}(\Pi_1)$, то по лемме о сохранении значения

$$\Pi_1(\sigma)(L) < i = \sigma z = \Pi_1(\sigma)(z).$$

Следовательно, $\Pi_1(\sigma) \models \varphi \wedge L < z$.

По правилу **ЦКП** получаем, что тройка $\{\varphi\}\Pi\{\varphi \wedge e = 0\}$ является полностью корректной. Формула $\varphi \wedge e = 0 \to \psi$ истинна, следовательно, тройка $\{\varphi\}\Pi\{\psi\}$ полностью корректна.

Это и означает, что φ — полное слабейшее предусловие для $\Pi\{\psi\}$. □

Рассмотрим несколько частных случаев правил для циклов.

Следствие 5.16. *Пусть* $\varphi_i \equiv \Phi(i)$ *для некоторой формулы* Φ *для всех* $i \geqslant 1$. *Тогда*

$$\varphi \equiv \psi \wedge e = 0 \vee (\exists i)(i \geqslant 1 \wedge \Phi(i)).$$

Следствие 5.17. *Пусть* $\Phi(i) \equiv i = e \wedge \Phi'(i)$, *где выражение* e *не содержит* i. *Тогда*

$$\varphi \equiv \psi \wedge e = 0 \vee \Phi'(e) \wedge e \geqslant 1.$$

Задача 5.24. Докажите оба следствия.

Возникает естественный вопрос: а существует ли вообще такая формула φ, о которой говорится в правиле **СЦП**? Ведь она должна быть эквивалентна бесконечной дизъюнкции. В случае произвольной предметной области, о которой ничего заранее неизвестно, существование такой формулы утверждать нельзя. Однако, в нашем случае, когда предметной областью выступает множество натуральных чисел со сложением и умножением, ответ на этот вопрос положителен.

Теорема 5.18 (О существовании слабейших предусловий)**.** *Для всякой программы* Π *и постусловия* ψ *для* $\Pi\{\psi\}$ *существует полное слабейшее предусловие.*

Доказательство теоремы существенно использует некоторые сведения из теории алгоритмов и теории доказательств. По этим причинам доказательство теоремы вынесено в отдельный параграф, который неподготовленные читатели могут пропустить.

```
algorithm Fib;
arguments n;
returns f;
    f = 1;  p = 0;  i = 1;
    while i < n do                 ⎫
        i = succ(i);   ⎫           ⎪
        t = f + p;     ⎬ Π2        ⎬ Π1
        p = f;         ⎪           ⎪
        f = t;         ⎭           ⎪
    end;                           ⎭
end;
```

Рис. 5.7: Числа Фибоначчи.

Формула, которая строится в доказательстве этой теоремы, является очень громоздкой и непригодной для практических целей. Обычно получить необходимую формулу можно, построив несколько φ_i и сделав обобщение (которое, конечно, нужно доказать). Этот метод мы демонстрируем в следующих примерах.

Пример 5.23. *Рассмотрим алгоритм на рис. 5.7.*

Очевидно, что постусловием должна быть формула $\psi \rightleftharpoons f = F_n$.

Теперь найдём слабейшее предусловие для

$$\Pi_2\{\underbrace{f = F_n \wedge i \geqslant n}_{\varphi_0}\}.$$

$\mathbf{f = t;}$ $t = F_n \wedge i \geqslant n.$

$\mathbf{p = f;}$ $t = F_n \wedge i \geqslant n.$

$\mathbf{t = f + p;}$ $f + p = F_n \wedge i \geqslant n.$

$\mathbf{i = succ(i);}$ $f + p = F_n \wedge i + 1 \geqslant n.$

Получили, что

$$\varphi_1' \multimap f + p = F_n \wedge i + 1 \geqslant n,$$
$$\varphi_1 \multimap f + p = F_n \wedge i + 1 \geqslant n \wedge i < n \equiv$$
$$\equiv f + p = F_n \wedge i + 1 = n.$$

Слабейшее предусловие для $\Pi_2\{\varphi_1\}$ *будет*

$$\varphi_2' \multimap 2f + p = F_n \wedge i + 2 = n,$$
$$\varphi_2 \multimap 2f + p = F_n \wedge i + 2 = n \wedge i < n \equiv$$
$$\equiv 2f + p = F_n \wedge i + 2 = n.$$

Аналогично получаем φ_3*:*

$$\varphi_3 \multimap 3f + 2p = F_n \wedge i + 3 = n.$$

Сравнивая φ_1*,* φ_2 *и* φ_3 *можно высказать предположение, что*

$$\varphi_j \multimap F_{j+1}f + F_j p = F_n \wedge i + j = n.$$

Проверим его: найдём слабейшее предусловие для $\Pi_1\{\varphi_j\}$*.*

f = t; $F_{j+1}t + F_j p = F_n \wedge i + j = n.$

p = f; $F_{j+1}t + F_j f = F_n \wedge i + j = n.$

t = f + p; $F_{j+1}(f + p) + F_j f = F_n \wedge i + j = n.$ *Так как*

$$F_{j+1} + F_j = F_{j+1+1},$$

то последняя формула эквивалентна

$$F_{j+1+1}f + F_{j+1}p = F_n \wedge i + j = n.$$

i = succ(i); $F_{j+1+1}f + F_{j+1}p = F_n \wedge i + j + 1 = n.$

Наше предположение подтвердилось:

$$\varphi_j \equiv F_{j+1}f + F_j p = F_n \wedge j = n - i.$$

Таким образом, по правилу **СЦП** *(следствие 5.17 на стр. 208) получаем, что полным слабейшим предусловием для* $\Pi_1\{\psi\}$ *будет*

$$f = F_n \wedge i \geqslant n \bigvee F_{n-i+1}f + F_{n-i}p = F_n \wedge n - i \geqslant 1.$$

Продолжим.

```
algorithm Primes;
arguments x, y;
returns v;
    u = x;
    v = 0;
    while u < y do
        if P(u) then
            v = v + 1;
        end;
        u = u + 1;
    end;
end;
```

Рис. 5.8: Подсчитывание простых чисел.

i = 1; $f = F_n \wedge 1 \geqslant n \bigvee F_n f + F_{n-1} p = F_n \wedge n - 1 \geqslant 1.$

p = 0; $f = F_n \wedge 1 \geqslant n \bigvee F_n f = F_n \wedge n \geqslant 2.$

f = 1; $1 = F_n \wedge 1 \geqslant n \bigvee F_n = F_n \wedge n \geqslant 2.$

Заметим, что

$$F_n = F_n \equiv \top,$$

$$1 = F_n \equiv n = 1 \vee n = 2,$$

$$1 = F_n \wedge 1 \geqslant n \equiv n = 1.$$

Поэтому, полным слабейшим предусловием для $\Pi\{f = F_n\}$ *будет*

$$n = 1 \vee n \geqslant 2 \equiv n \geqslant 1.$$

Итак, программа работает правильно, если $n \geqslant 1$.

Пример 5.24. *Рассмотрим программу, подсчитывающую количество простых чисел на промежутке* $[x, y)$, *на рис. 5.8. Мы считаем, что значение функции* $P(x)$ *равно* 1 *или* 0 *в в зависимости от того, является ли число*

x простым или нет. Обозначим с помощью $P(a,b)$ — количество простых чисел на промежутке $[a,b)$. Постусловие должно выглядеть так:

$$v = P(x,y).$$

Для цикла:

$$\varphi_0 \rightleftharpoons v = P(x,y) \wedge u \geqslant y$$

$$\Phi(i) \rightleftharpoons v = P(x,u) \wedge i = y - u$$

В самом деле:

u = u + 1; $v = P(x, u+1) \wedge i + 1 = y - u$

v = v + 1; $v + 1 = P(x, u+1) \wedge i + 1 = y - u$

if ... end;

$$\begin{aligned}(v = P(x, u+1) \wedge \neg P(u) \bigvee \\ \bigvee v + 1 = P(x, u+1) \wedge P(u)) \wedge i + 1 = y - u \equiv \\ \equiv v = P(x,u) \wedge i + 1 = y - u \rightleftharpoons \Phi(i+1)\end{aligned}$$

while ... end; $v = P(x,y) \wedge u \geqslant y \vee v = P(x,u) \wedge y - u \geqslant 1$

v = 0; $0 = P(x,y) \wedge u \geqslant y \vee 0 = P(x,u) \wedge y - u \geqslant 1$

u = x; $0 = P(x,y) \wedge x \geqslant y \vee 0 = P(x,x) \wedge y - x \geqslant 1 \equiv \top$

Следовательно, слабейшим предусловием будет $\top$, и программа корректно работает всегда.

Иногда найти полное слабейшее предусловие можно, не используя исчисление $\mathfrak{IP}$.

Пример 5.25. *Найдём полное слабейшее предусловие для следующей программы Π и формулы $\psi \rightleftharpoons u = 1$:*

```
v = 2;
while u > 1 do                 ⎫
    if u mod v == 0 then  ⎫    ⎪
        u = u / v;        ⎬ Π2 ⎬ Π1
        v = v + 1;        ⎪    ⎪
    end;                  ⎭    ⎪
end;                           ⎭
```

Утверждение 5.19. *Пусть σ — произвольное состояние, такое что $\sigma v = 2$, n — наибольшее натуральное число, такое что для некоторого $k \in \omega$ имеет место $\sigma x = n! \times k$. Пусть $(\sigma^i)_i$ — последовательность состояний при работе цикла Π_1 на σ. Тогда $\sigma^i u = {}^{n!}/_{(i+1)!} \times k$, $\sigma^i v = i + 2$ для $0 \leqslant i \leqslant n - 1$.*

Доказательство. *Индукция по i.*

Базис индукции. Пусть $i = 0$, тогда $\sigma^0 = \sigma$. Имеем $\sigma^0 u = n! \times k = {}^{n!}/_{(0+1)!} \times k$, $\sigma^0 v = 2 = 2 + 0$.

Шаг индукции. Пусть для i утверждение верно и пусть $i + 1 \leqslant n - 1$. По определению семантики цикла $\sigma^{i+1} = \Pi_2(\sigma^i)$. По индукционному предположению $\sigma^i u = {}^{n!}/_{(i+1)!} \times k$, $\sigma^i v = i + 2$. Поскольку $n!$ является произведением всех чисел от 1 до n, а $i+2 \leqslant n$, то ${}^{n!}/_{(i+1)!} \times k$ делится на $i+2$. Следовательно, $\sigma^i(u \bmod v == 0) = 1$ и $\sigma^{i+1} = (u = u / v;\ v = v+1;)(\sigma^i)$. Получаем $\sigma^{i+1} u = \sigma^i u / \sigma^i v = {}^{n!}/_{(i+2)!} \times k$, $\sigma^{i+1} v = \sigma^i v + 1 = i + 3$. □

Утверждение 5.20. *Полным слабейшим предусловием для $\Pi\{\psi\}$ является формула $\varphi \Leftrightarrow (\exists x) u = x!$.*

Доказательство. *Сначала докажем, что тройка Хоара $\{\varphi\}\,\Pi\,\{\psi\}$ полностью корректна. Пусть σ — произвольное состояние, такое что $\sigma \models \varphi$. Пусть $\sigma' = (v = 2;)(\sigma)$. Тогда $\sigma' v = 2$, $\sigma' y = \sigma y$ для $y \neq v$. Так как предусловие не содержит v, то оно истинно на σ'. Следовательно, $\sigma' u = n!$ для некоторого $n \in \omega$. Пусть $(\sigma_i)_i$ — последовательность состояний при работе Π_1 на σ'. По утверждению 5.19 $\sigma^{n-1} u = 1$. Поэтому $\sigma^{n-1}(u > 1) = 0$, $\Pi(\sigma) = \sigma^{n-1}$ и $\Pi(\sigma) \models u = 1$.*

Теперь докажем, что предусловие φ является слабейшим. Пусть σ — произвольное состояние, такое что $\sigma \not\models \varphi$. Тогда $\sigma u = n! \times k$, где k и n такие же, как и в утверждении 5.19. Поэтому $k > 1$ и k не делится на $n + 1$, так как в противном случае n не было бы наибольшим. По утверждению 5.19 $\sigma^{n-1} u = k$. Значит, $\sigma^{n-1}(u \bmod v == 0) = 0$ и $\sigma^n = \sigma^{n-1}$. Следовательно, $\Pi(\sigma)$ не определено. □

Задача 5.25. Напишите программы и найдите для них слабейшие предусловия:

1) Вычисление факториала (используя умножение).
2) Определение простоты числа (используя умножение и целочисленное деление).

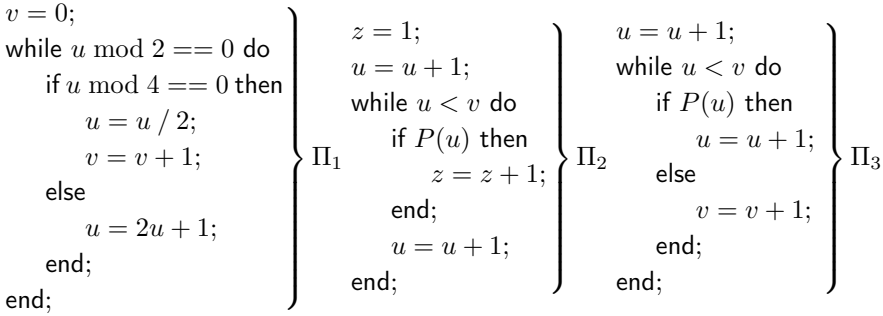


Рис. 5.9: Программы к задаче 5.27

3) Нахождение количества делителей числа (используя умножение и целочисленное деление).

*__Задача 5.26.__ Напишите эти же программы и найдите слабейшие предусловия для них, используя только сложение и вычитание.

__Задача 5.27.__ Найдите полные слабейшие предусловия для следующих программ и формул (рис. 5.9), определение функции P дано в примере 5.20 на стр. 195:

1) $\Pi_1\{v > 0\}$;
2) $\Pi_2\{z = 1\}$;
3) $\Pi_3\{\top\}$.

Ранее мы доказывали непротиворечивость исчислений $\mathfrak{IH}$ и $\mathfrak{IP}$. Непротиворечивость означала, что каждый выводимый объект (тройка Хоара), обладает некоторыми свойствами (частичная, полная корректность, слабейшее предусловие), для выявления которых и построено исчисление. Обратное свойство называется полнота исчисления. Исчисление полно, если каждый объект, обладающий необходимыми свойствами, выводим в исчислении. Полнота

исчисления Хоара означает, что каждая частично или полностью корректная тройка выводима в $\mathfrak{IH}$. Полнота исчисления предусловий означает, что если φ — полное слабейшее предусловие для $\Pi\{\psi\}$, то тройка $\{\varphi\}\Pi\{\psi\}$ выводима в $\mathfrak{IP}$. Докажем, что и то и другое исчисление являются полными, если говорить о полной корректности троек.

Теорема 5.21 (Полнота исчисления предусловий)**.** *Если φ — полное слабейшее предусловие для $\Pi\{\psi\}$, то*

$$\underset{\mathfrak{IP}}{\rhd} \{\varphi\}\Pi\{\psi\}.$$

Доказательство. Для каждой пары $\Pi\{\psi\}$ исчисление $\mathfrak{IP}$ даёт некоторое полное слабейшее предусловие θ. По следствию об эквивалентности предусловий имеем $\theta \equiv \varphi$. По правилу **СЭП** получаем

$$\underset{\mathfrak{IP}}{\rhd} \{\varphi\}\Pi\{\psi\}. \qquad \square$$

Теорема 5.22 (Полнота исчисления Хоара)**.** *Для каждой полностью корректной тройки $\{\varphi\}\Pi\{\psi\}$ выполнено*

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\Pi\{\psi\}.$$

Доказательство. Докажем полноту исчисления Хоара индукцией по количеству операторов цикла в программе.

Базис индукции. Если циклов в программе нет, то по $\Pi\{\psi\}$ исчисление предусловий позволяет построить слабейшее предусловие φ', причём тройка $\{\varphi'\}\,\Pi\,\{\psi\}$ будет выводима и в исчислении $\mathfrak{IH}$, согласно следствию 5.15 на стр. 206. Докажем истинность формулы $\varphi \to \varphi'$, после чего выводимость тройки $\{\varphi\}\,\Pi\,\{\psi\}$ будет следовать по правилу **УПР**.

Рассмотрим любую конфигурацию $\sigma \models \varphi$. Так как тройка $\{\varphi\}\,\Pi\,\{\psi\}$ вполне корректна, то $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \psi$. Так как φ' — слабейшее предусловие для $\Pi\{\psi\}$, то $\sigma \models \varphi'$.

Ш а г и н д у к ц и и . Пусть для программ, которые содержат меньше чем n циклов, теорема доказана. Достаточно доказать теорему только для оператора цикла. Рассмотрим любую вполне корректную тройку

$$\{\varphi\}\underbrace{\mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};}_{\Pi}\{\psi\},$$

где Π_1 содержит меньше n циклов. Исчисление предусловий позволяет построить формулу φ' — слабейшее предусловие для $\Pi\{\psi\}$. Тройка Хоара

$$\{\varphi' \wedge e \neq 0 \wedge L = z\}\Pi_1\{\varphi' \wedge L < z\}$$

(см. доказательство непротиворечивости **СЦП**) является вполне корректной. По индукционному предположению, она выводима в $\mathfrak{IH}$. По правилу **ЦКП** получаем выводимость тройки

$$\{\varphi'\}\,\Pi\,\{\varphi' \wedge e = 0\}\,.$$

Формула $\varphi' \wedge e = 0 \to \psi$ является истинной, следовательно, тройка $\{\varphi'\}\,\Pi\,\{\psi\}$ выводима с помощью **ОСП**. Точно также как и для базиса доказывается, что формула $\varphi \to \varphi'$ истинна, следовательно,

$$\underset{\mathfrak{IH}}{\rhd}\ \{\varphi\}\,\Pi\,\{\psi\}\,. \qquad \square$$

§ 5.6. **Существование слабейших предусловий

Прежде чем доказывать теорему о существовании полного слабейшего предусловия, установим несколько лемм.

Лемма 5.23 (Форма предусловий)**.** *Если программа* Π *не содержит циклов, то полное слабейшее предусловие* φ *для* $\Pi\{\psi \wedge \theta\}$ *может быть эффективно представлено конечной дизъюнкцией формул вида*

$$(\psi)^{\bar{x}}_{\bar{e}} \wedge \theta',$$

где θ и θ' — бескванторные формулы.

Доказательство. Индукция по сложности программы. □

Задача 5.28. Докажите лемму.

Следующие две теоремы доказываются в теории алгоритмов.

Теорема 5.24 (О представлении функций)**.** *Для каждой вычислимой функции f существует формула арифметики Ψ_f, которая представляет f. То есть $\Psi_f(\bar{x}, y)$ истинна тогда и только тогда, когда $f(\bar{x}) = y$.*

Поскольку все программы работают с числами, то необходим некоторый способ представления логических формул и состояний натуральными числами. Пусть $[a]$ — номер a.

Теорема 5.25 (О представлении формул)**.** *Для каждого $n \in \omega$ существует формула арифметики Ψ_n, которая представляет истинность любой арифметической формулы с не более чем n кванторами на любом состоянии. То есть формула $\Psi_n([\chi], [\sigma])$ истинна тогда и только тогда, когда формула χ содержит не более чем n кванторов и $\sigma \models \chi$.*

Лемма 5.26 (О представлении предусловий)**.** *Для всякой формулы ψ существует арифметическая формула Ψ_ψ, представляющая истинность всех формул вида*

$$\varphi \leftrightarrows \bigvee_{j<m} (\psi)^{\bar{x}}_{\bar{e}_j} \wedge \theta_j,$$

на любом состоянии σ, где $m \in \omega$, $\bar{e}_j$ — наборы выражений θ_j — бескванторные формулы.

Доказательство. Пусть функции g и h таковы, что для любой формулы φ вышеуказанного вида

$$g([\varphi], j) = [(\psi)^{\bar{x}}_{\bar{e}_j}]$$

$$h([\varphi], j) = [\theta_j].$$

Если $j \geqslant m$, то

$$g([\varphi], j) = [\bot]$$
$$h([\varphi], j) = [\bot].$$

То есть функции g и h разлагают формулу φ на составные части. Обе эти функции вычислимы, по теореме обе эти функции представляются арифметическими формулами Ψ_g и Ψ_h соответственно. Очевидно, что формулы $(\psi)^{\bar{x}}_{\bar{e}_j}$ содержит столько же кванторов, сколько и ψ. Пусть это количество равно n. Тогда истинность всех формул $(\psi)^{\bar{x}}_{\bar{e}_j}$ и θ_j представляется арифметической формулой Ψ_n. Пусть

$$\Psi_\psi(y, x) \twoheadleftarrow (\exists j)(\exists u_\psi)(\exists u_\theta) \\ (\Psi_g(y, j, u_\psi) \wedge \Psi_h(y, j, u_\theta) \wedge \Psi_n(u_\theta, x) \wedge \Psi_n(u_\theta, x)).$$

Предположим, что $\sigma \models \varphi$. Тогда существует $j^0 < m$ такое, что $\sigma \models (\psi)^{\bar{x}}_{\bar{e}_{j^0}}$ и $\sigma \models \theta_{j^0}$. Пусть

$$u^0_\psi = [(\psi)^{\bar{x}}_{\bar{e}_{j^0}}]; \quad u^0_\theta = [\theta_{j^0}].$$

Тогда истинны формулы

$$\Psi_g([\varphi], j^0, u^0_\psi),\ \Psi_h([\varphi], j^0, u^0\theta),\ \Psi_n(u^0_\psi, [\sigma]),\ \Psi_n(u^0_\theta, [\sigma]).$$

Следовательно, истинна и формула $\Psi_\psi([\varphi], [\sigma])$.

Пусть истинна $\Psi_\psi([\varphi], [\sigma])$. Тогда для некоторых j^0, u^0_ψ и u^0_θ истинны формулы

$$\Psi_g([\varphi], j^0, u^0_\psi),\ \Psi_h([\varphi], j^0, u^0\theta),\ \Psi_n(u^0_\psi, [\sigma]),\ \Psi_n(u^0_\theta, [\sigma]).$$

Если $j^0 \geqslant m$, то по определению функции g имеем $g([\varphi], j) = [0 = 1]$. Так как Ψ_g представляет g и $\Psi_g([\varphi], j, u^0_\psi)$ истинна, то $u^0_\psi = [0 = 1]$. Так как Ψ_n представляет истинность всех формул с не более чем n кванторами, то она представляет истинность и $0 = 1$. Но формула

$0 = 1$ ложна на всех состояниях, следовательно, $\Psi_n([0 = 1], [\sigma])$ должна быть ложна для всякого σ, противоречие.

Следовательно, $j^0 < m$. Это означает, что

$$u^0_\psi = [(\psi)^{\bar{x}}_{\bar{e}_{j^0}}]; \quad u^0_\theta = [\theta_{j^0}].$$

Из истинности $\Psi_n(u^0_\psi, [\sigma])$ и $\Psi_n(u^0_\theta, [\sigma])$ следует, что $\sigma \models (\psi)^{\bar{x}}_{\bar{e}_{j^0}}$ и $\sigma \models \theta_{j^0}$. Следовательно, $\sigma \models (\psi)^{\bar{x}}_{\bar{e}_{j^0}} \wedge \theta_{j^0}$ и $\sigma \models \varphi$. □

Лемма 5.27 (О представлении предусловий)**.** *Пусть множество формул $\{\varphi_i : i \in \omega\}$ алгоритмически перечислимо по i и каждое φ_i является дизъюнкцией формул вида $(\psi)^{\bar{x}}_{\bar{e}} \wedge \theta$, где θ — бескванторная формула. Тогда существует формула $\Phi(i, x)$ такая, что для любых $i \in \omega$ и состояний σ формула $\Phi(i, [\sigma])$ истинна тогда и только тогда, когда $\sigma \models \varphi_i$.*

Доказательство. Так как множество $\{\varphi_i : i \in \omega\}$ перечислимо, то существует вычислимая функция f такая, что $f(i) = [\varphi_i]$. По теореме существует формула Ψ_f такая, что $\Psi_f(i, [\varphi_i])$ истинна тогда и только тогда, когда $f(i) = [\varphi_i]$. По лемме 5.26 на стр. 217, существует арифметическая формула Ψ_ψ, представляющая истинность всех формул φ_i. Пусть

$$\Phi \leftrightarrows (\exists u_\varphi)(\Psi_f(i, u_\varphi) \wedge \Psi_\psi(u_\varphi, x)).$$

Пусть $\sigma \models \varphi_i$. Возьмём $u^0_\varphi = [\varphi_i]$. Тогда истинны формулы $\Psi_f(i, u^0_\varphi)$ и $\Psi_\psi(u^0_\varphi, [\sigma])$. Следовательно, истинна и $\Phi(i, [\sigma])$.

Пусть истинна $\Phi(i, [\sigma])$. Это означает, что для некоторого u^0_φ истинны и $\Psi_f(i, u^0_\varphi)$, и $\Psi_\psi(u^0_\varphi, [\sigma])$. Из первого следует, что $u^0_\varphi = [\varphi_i]$. Из второго — $\sigma \models \varphi_i$. □

Лемма 5.28 (О существовании предусловий)**.** *Пусть*

$$\Pi \leftrightarrows \mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};$$

и Π_1 не содержит циклов. Тогда для всякого постусловия ψ существует слабейшее полное предусловие φ для $\Pi\{\psi\}$.

Доказательство. По лемме 5.23 на стр. 216 полное слабейшее предусловие для $\Pi\{\psi\}$ может быть выражено бесконечной дизъюнкцией

$$\varphi \equiv \bigvee\{\varphi_i : i \in \omega\},$$

которая удовлетворяет условиям леммы 5.27 на предыдущей странице. Это означает, что существует формула Φ такая, что $\Phi(i, [\sigma])$ истинна тогда и только тогда, когда $\sigma \models \varphi_i$.

Будем рассматривать конфигурации программы Π. Пусть переменные из $\mathrm{Var}(\Pi)$ образуют набор $\bar{x}$ из n переменных. Пусть функция f по $\bar{v} \in \omega^n$ даёт номер конфигурации σ такой, что $x_i = v_i$. Тогда эта функция представляется формулой Ψ_f. Возьмём формулу

$$\Theta(i, \bar{x}) \leftrightarrows (\exists u_\sigma)(\Phi(i, u_\sigma) \wedge \Psi_f(\bar{x}, u_\sigma)).$$

Тогда

$$\bigvee\{\varphi_i : i \in \omega\} \equiv (\exists i)\Theta.$$

Следовательно, слабейшее предусловие для $\Pi\{\psi\}$ выражается формулой $(\exists i)\Theta$. □

Задача 5.29. Докажите, что

$$\bigvee\{\varphi_i : i \in \omega\} \equiv (\exists i)\Theta.$$

Теперь мы можем доказать теорему о существовании полного слабейшего предусловия.

Доказательство. Пусть даны программа Π и постусловие ψ. По теореме о цикле существует программа Π_2 такая, что

$$\Pi_2(\sigma) \restriction \mathrm{Var}(\Pi) = \Pi(\sigma) \restriction \mathrm{Var}(\Pi).$$

Кроме того, можно считать, что Π_2 имеет вид

$y = \alpha;$
$w = 0;$
while e do
  Π_1
end;
} Π_3

где y и w — единственные переменные не входящие в Π, α — константа, Π_1 не содержит циклов. По предыдущей лемме для $\Pi_3\{\psi\}$ существует полное слабейшее предусловие φ'. Для $\Pi_2\{\psi\}$ полным слабейшим предусловием будет

$$\varphi \Leftarrow (\varphi')^{y\,w}_{\alpha 0}.$$

Очевидно, что оно же будет полным слабейшим предусловием и для $\Pi\{\psi\}$. □

Ещё раз подчеркнём, что в доказательстве существенно используются свойства арифметики натуральных чисел. Если речь идёт о другой предметной области, то утверждение о существовании слабейших предусловий может быть неверным.

§ 5.7. *Невозможность автоматической верификации

Несмотря на то, что введенные нами правила достаточно формальны, тем не менее процесс верификации не может быть полностью автоматизирован.

Следующая теорема является вариантом неразрешимости проблемы остановки из теории алгоритмов.

Теорема 5.29. *Не существует эффективного способа определить частичную корректность произвольной тройки Хоара.*

Доказательство. Будем доказывать утверждение от противного: допустим, что существует алгоритм C, который принимает на вход тройку Хоара (в некотором закодированном виде) в виде переменных x, y, z (коды предусловия, программы и постусловия соответственно), а результатом его является 1 или 0 в зависимости

от того, является тройка частично корректной или нет:

```
algorithm C;
arguments x, y, z;
returns r;
    Π
end;
```

С помощью $[\alpha]$ будем обозначать код α (условия или программы). Без ограничения общности будем считать, что тело алгоритма является структурированной программой Π, а выражение-результат — это переменная r. Модифицируем программу, приписав к ней в конце цикл, а в начале несколько присваиваний:

```
algorithm C';
arguments y;
returns r;
    x = [y =] & y;        ⎫
    z = [⊥];              ⎪
    Π                     ⎬ Π'
    while r == 0 do       ⎪
        ;                 ⎪
    end;                  ⎭
end;
```

С помощью амперсанда мы обозначили результат конкатенации, то есть после выполнения первого присваивания переменная x содержит код условия $y = [\Pi'']$, если y содержит код программы Π''. Также новая программа будет зацикливаться, если результат старой был 0. Таким образом, новый алгоритм C' делает следующее: на вход в переменной y подается код программы Π'', затем проверяется корректность тройки

$$\{y = [\Pi'']\}\,\Pi''\,\{\bot\}\,,$$

и если тройка не оказывается частично корректной, то программа Π' зацикливается, а в противном случае результатом будет 1.

Подадим на вход алгоритму C' код программы Π': то есть рассмотрим вызов $(C', [\Pi'])$. Предположим, что результат этого вызова определен. Это означает, что после выполнения Π переменная r содержит ненулевое значение, то есть тройка $\{y = [\Pi']\}\,\Pi'\,\{\bot\}$ является частично корректной. Но формула $\bot$ всегда является ложной, поэтому эта тройка может быть частично корректной, только если программа Π' зацикливается, что противоречит предположению об определенности вызова $(C', [\Pi'])$.

Полученное противоречие показывает, что результат вызова $(C', [\Pi'])$ не может быть определен. Из определения сразу вытекает, что тройка $\{y = [\Pi']\}\,\Pi'\,\{\bot\}$ является частично корректной. Но тогда после выполнения Π значение r будет равно 1 и Π' тоже остановится, то есть результат вызова $(C', [\Pi'])$ будет определен. Снова получено противоречие.

Итак, мы выяснили, что любое предположение о результате вызова $(C', [\Pi'])$ приводит к противоречию. Но это означает, что наше исходное допущение неверно и алгоритма C не может существовать. □

Заметим, что никаких троек Хоара, кроме троек вида $\{\varphi\}\,\Pi\,\{\bot\}$ в доказательстве этой теоремы не применяется, поэтому ее можно усилить:

Следствие 5.30. *Не существует эффективного способа определить частичную корректность произвольной тройки Хоара вида* $\{\varphi\}\,\Pi\,\{\bot\}$.

Аналогичным методом можно доказать, что эффективное определение полной корректности троек Хоара невозможно.

Теорема 5.31. *Не существует эффективного способа определить полную корректность произвольной тройки Хоара.*

Доказательство. Доказательство будет аналогично предыдущему. Предположим, что существует алгоритм C, который принимает на вход тройку Хоара в виде переменных x, y, z, а результатом его является 1 или 0 в зависимости от того, является ли тройка полностью корректной:

```
algorithm C;
arguments x, y, z;
returns r;
    Π
end;
```

Построим модифицированный алгоритм C':

```
algorithm C′;
arguments y;
returns r;
    x = [y =] & y;   ⎫
    z = [⊤];         ⎪
    Π                ⎬ Π′
    while r do       ⎪
        ;            ⎪
    end;             ⎭
end;
```

Новая программа будет зацикливаться, если результат старой был 1. Таким образом, новый алгоритм C' делает следующее: на вход в переменной y подается код программы Π'', затем проверяется корректность тройки

$$\{y = [\Pi'']\}\, \Pi''\, \{\top\},$$

и если тройка оказывается полностью корректной, то программа Π' зацикливается, а в противном случае результатом будет 0.

Подадим на вход алгоритму C' код программы Π': $y = [\Pi']$. Тогда программа Π' остановится в том и только том случае, когда тройка

$$\{y = [\Pi']\}\, \Pi'\, \{\top\}$$

окажется не вполне корректной.

Предположим, что результат вызова $(C', [\Pi'])$ определен, то есть тройка $\{y = [\Pi']\}\,\Pi'\,\{\top\}$ не вполне корректна. Поскольку постусловие всегда выполнено, то отсутствие полной корректности может означать только то, что программа зацикливается для некоторого значения переменной y, для которого выполнено предусловие $y = [\Pi']$. Но такое значение только одно: $[\Pi']$, то есть вызов $(C', [\Pi'])$ неопределен. Получили противоречие.

Рассмотрим противоположный случай, когда результат вызова $(C', [\Pi'])$ неопределен. Из определения сразу вытекает, что тройка $\{y = [\Pi']\}\,\Pi'\,\{\top\}$ не может быть вполне корректной. Но тогда после выполнения Π значение r будет равно 0 и Π' тоже остановится, то есть результат вызова $(C', [\Pi'])$ будет определен. Снова противоречие.

Поскольку оба возможных варианта привели к противоречию, то исходное предположение о существовании алгоритма C неверно. $\square$

Глава 6

Подпрограммы, функциональное программирование

§ 6.1. Подпрограммы

Мы уже доказывали, что тело одной программы можно встроить в тело другой. Но это не всегда удобно, например, если один алгоритм используется внутри другого не один раз, а несколько, то это означает, что мы должны записать тело первого алгоритма внутри другого столько раз, сколько он используется. Например, в алгоритме Евклида (рис. 3.5 на стр. 73) мы были вынуждены два раза написать фрагмент, выполняющий вычитание чисел. Это нерационально, так как от повторения написанного информации больше не становится. Поэтому в программировании появилось понятие п о д п р о г р а м м ы — части алгоритма, которая написана один раз, но может использоваться многократно. Обогатим наши

языки программирования подпрограммами. Синтаксис подпрограмм будет похож на способ записи алгоритмов.

Определение 6.1 (Подпрограмма)**.** *Синтаксис подпрограмм:*

function *Имя подпрограммы*;
arguments *Список входных переменных*;
returns *Выражение-результат*;
Тело подпрограммы
end;

Переменные, которые стоят после слова arguments *— это входные переменные подпрограммы (или параметры подпрограммы). Тело подпрограммы (его мы будем обозначать* Π_A*, если* A *— имя подпрограммы) такое же как и тело алгоритмов.*

Во многих языках программирования подпрограммы называют функциями. Мы не будем так делать, что не путать их с функциями в математическом смысле. Также подпрограммы иногда называют процедурами.

Пример 6.1. *Рассмотрим две подпрограммы:*

function A;
arguments x;
returns succ(succ(x));;
end;

function Max;
arguments x, y;
returns m;
1 if $x < y$ then 2 else 3
2 $m = y$; 4
3 $m = x$; 4
end;

В частном случае подпрограмма может не иметь параметров, то есть список переменных после arguments может быть пустым.

Вызов подпрограммы f, как и вызов алгоритма, — это набор вида

$$\delta = (f, \bar{v}),$$

где $\bar{v}$ — набор элементов предметной области (аргументов) длины равной количеству входных переменных f. Точно так же определяется и р е з у л ь т а т в ы з о в а $\mathrm{Res}\,\delta$. Как и алгоритм, каждая детерминированная подпрограмма A вычисляет некоторую частичную функцию Φ_A, определяемую так же, как для алгоритмов: $\Phi_A(\bar{v}) = \mathrm{Res}(A, \bar{v})$ для всех $\bar{v}$.

Чтобы использовать уже определённые подпрограммы, расширим понятие выражения.

Определение 6.2 (Выражение)**.** *Кроме уже известных нам выражений мы будем считать в ы р а ж е н и е м также следующую строку:*

$$\mathsf{apply}(f, e_1, \ldots, e_n),$$

где f — имя подпрограммы с n параметрами, а e_1, …, e_n — выражения. В частности, если f — подпрограмма без параметров, то выражением будет строка $\mathsf{apply}(f)$.

Пример 6.2. *Примеры выражений, в которых используются имена подпрограмм из примера 6.1 на предыдущей странице:*

$$\mathsf{apply}(A, 0);$$

$$\mathsf{apply}(Max, x, \mathsf{succ}(y));$$

$$\mathsf{apply}(Max, \mathsf{apply}(A, x), \mathsf{succ}(0)).$$

Определим теперь, что такое значение выражения e на конфигурации σ. Очевидно, что к уже существующему определению нужно добавить определение того, что такое $\sigma(\mathsf{apply}(f, e_1, \ldots, e_n))$, где f — имя подпрограммы.

Определение 6.3 (Значение выражения)**.** *Будем считать, что*

$$\sigma(\mathsf{apply}(f, e_1, \ldots, e_n)) = v,$$

если

1) $\sigma(e_1), \ldots, \sigma(e_n)$ *определены;*

2) $v = \mathrm{Res}(f, \sigma(e_1), \ldots, \sigma(e_n))$.

Таким образом,

$$\sigma(\mathsf{apply}(f, e_1, \ldots, e_n)) = \mathrm{Res}(f, \sigma(e_1), \ldots, \sigma(e_n)).$$

Замечание 6.1. Рассмотрим вызов некоторой подпрограммы f: $\delta = (f, \sigma(e_1), \ldots, \sigma(e_n))$. Чтобы определить значение $\sigma(\mathsf{apply}(f, \bar{e}))$ нужно определить $\Pi_f(\sigma_\delta)$. Однако, для определения $\Pi_f(\sigma_\delta)$ может снова потребоваться определить $\tau(\mathsf{apply}(f, \bar{d}))$ и так далее. Если подобная цепочка оказывается бесконечной, то мы считаем, что $\sigma(\mathsf{apply}(f, \bar{e}))$ не определено.

Пример 6.3. *Рассмотрим выражения из предыдущего примера на конфигурации* $\sigma = \{(x, 1), (y, 2)\}$.

1) $\mathsf{apply}(A, 0)$. *Очевидно, что* A *вычисляет функцию* $\Phi_A(x) = x + 2$. *Следовательно,* $\sigma(\mathsf{apply}(A, 0)) = 2$.

2) $Max(x, \mathsf{succ}(y))$. *Подпрограмма* Max *вычисляет функцию-максимум из двух аргументов. Поэтому*

$$\sigma(\mathsf{apply}(Max, x, \mathsf{succ}(y))) = \max\{1, 3\} = 3.$$

3) $\sigma(\mathsf{apply}(Max, \mathsf{apply}(A, x), \mathsf{succ}(0))) = \max\{3, 1\} = 3$.

Пример 6.4. *Рассмотрим подпрограмму, которая вычисляет частичную функцию, не являющуюся всюду определённой (рис. 6.1 на следующей странице). Легко видеть, что* $\Pi_B(\tau)$ *определено тогда и только тогда, когда* $x \geqslant y$ *и в этом случае* $\Pi_B(\tau)(B) = \tau x$. *Рассмотрим выражение* $\mathsf{apply}(B, x, \mathsf{succ}(y))$ *на той же конфигурации* σ. $\mathrm{Res}(B, 1, 3)$ *не определён, следовательно,* $\sigma(\mathsf{apply}(B, x, \mathsf{succ}(y)))$ *тоже не определено.*

Пример 6.5. *Проиллюстрируем замечание 6.1.*

```
function C;
arguments x;
returns apply(C, succ(x));;
end;
```

```
function B;
arguments x, y;
returns x;
    z = y;
    while x < z do
        z = succ(z);
    end;
end;
```

Рис. 6.1:

```
function D;
arguments x;
returns y;
    if x > 0 then
        y = apply(D, x − 1) + 2;
    else
        y = 0;
    end;
end;
```

Рис. 6.2:

Чтобы определить значение $\sigma(\mathsf{apply}(C, e))$ *нужно определить* $\operatorname{Res}\delta$, *где* $\delta = (C, \sigma(e))$. *Чтобы определить* $\operatorname{Res}\delta$, *нужно определить* $\Pi_C(\sigma_\delta)$, *а для этого снова нужно определять* $\sigma_\delta(\mathsf{apply}(C, \mathsf{succ}(x)))$ *и так далее. Следовательно, согласно замечанию, мы считаем, что* $\sigma(\mathsf{apply}(C, e))$ *не определено ни для каких* e *и* σ.

Пример 6.6. *Приведём ещё один пример (рис. 6.2). Здесь мы имеем схожую ситуацию — для определения* $\sigma(\mathsf{apply}(D, e))$ *нужно определить* $\sigma_\delta(\mathsf{apply}(D, x - 1))$, *где* $\sigma_\delta = \{(x, \sigma(e)), (y, a)\}$. *Однако, в этом случае по-*

следовательность не оказывается бесконечной. Индукцией по σx докажем, что $\sigma(\mathsf{apply}(D, x)) = 2\sigma x$.
Базис индукции. Если $\sigma x = 0$, то $\sigma_\delta x = 0$, следовательно,

$$\Pi_D(\sigma_\delta) = \{(x, \sigma_\delta x), (y, 0)\}.$$

Поэтому $\sigma(\mathsf{apply}(D, x)) = 0$.
Шаг индукции. Пусть для $\sigma x < x_0$ утверждение уже доказано и $\sigma x = x_0$. Тогда

$$\Pi_D(\sigma_\delta) = \{(x, \sigma_\delta x), (y, \sigma_\delta(\mathsf{apply}(D, x-1) + 2))\}.$$

Так как $\sigma_\delta(x-1) = x_0 - 1 < x_0$, то по индукционному предположению получаем

$$\sigma_\delta(\mathsf{apply}(D, x-1)) = 2\sigma_\delta(x-1) = 2x_0 - 2.$$

Поэтому

$$\sigma(\mathsf{apply}(D, x)) = 2x_0 - 2 + 2 = 2x_0 = 2\sigma x.$$

Отметим один частный случай apply, который будет неоднократно использоваться в дальнейшем.

Лемма 6.1. *Предположим, что $\bar{x}$ — это входные переменные подпрограммы f, а r_f — выражение-результат. Тогда для любой конфигурации σ выполнено*

$$\sigma(\mathsf{apply}(f, \bar{x})) = \Pi_f(\sigma)(r_f).$$

Доказательство. По определению $\sigma(\mathsf{apply}(f, \bar{x})) = \operatorname{Res}\delta$, где вызов δ равен $(f, \sigma x_1, \ldots, \sigma x_n)$. Конфигурация τ будет начальной для δ тогда и только тогда, когда $\tau x_i = \sigma x_i$. В частности, само σ тоже является начальной конфигурацией для δ. По определению значения вызова получаем

$$\operatorname{Res}\delta = \Pi_f(\sigma)(r_f),$$

что и требовалось. □

Добавим к нашим языкам программирования ещё одну операцию.

```
algorithm cond;
arguments x, y, z;
returns u;
    if x then
        u = y;
    else
        u = z;
    end;
end;
```

Рис. 6.3: Алгоритм *cond*.

Определение 6.4 (Условная операция)**.** *Значение трёхместной операции* cond *определяется следующим образом:*

$$\sigma(\mathsf{cond}(e, e_1, e_2)) = \begin{cases} \sigma(e_1), & \text{если } \sigma(e) \neq 0, \\ \sigma(e_2), & \text{иначе.} \end{cases}$$

Таким образом, программа $x = \mathsf{cond}(e, e_1, e_2)$; эквивалентна такому ветвлению:

```
if e then
    x = e1;
else
    x = e2;
end;
```

Заметим, что для вычисления функции cond легко написать алгоритм (рис. 6.3), поэтому в силу теоремы о подстановке добавление этой операции к языку программирования не расширяет его возможностей.

```
function Add;
arguments x, y;
returns cond(x == 0,
    y,
    succ(apply(Add, x − 1, y)));;
end;
```

Рис. 6.4: Сложение чисел.

Используя такой синтаксис можно упростить функцию D:

```
function D;
arguments x;
returns cond(x > 0,
    apply(D, x − 1) + 2,
    0);;
end;
```

*__Задача 6.1.__ Докажите, что подпрограмма на рис. 6.4 вычисляет сумму чисел.

Примеры показывают, что у нас появился принципиально новый случай в определении семантики. До сих пор значение любого выражения мы считали определённым для любой конфигурации. Для новой семантики это уже не так. Поэтому мы должны усовершенствовать определение семантики и для всех остальных программных конструкций. Для этого достаточно добавить к определению случаи, когда значение какого-либо выражения не определено.

Определение 6.5 (Семантика программы)**.**

1) $\Pi \twoheadrightarrow x = e;\ \Pi(\sigma)$ *определено тогда и только тогда, когда* $\sigma(e)$ *определено.*

2) $\Pi \twoheadleftarrow \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};$ *если* $\sigma(e)$ *не определено, то* $\Pi(\sigma)$ *тоже не определено.*

3) $\Pi \twoheadleftarrow \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};$ *если* $\sigma(e)$ *не определено, то* $\Pi(\sigma)$ *тоже не определено.*

4) $\Pi \twoheadleftarrow \mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};$ $\Pi(\sigma)$ *определено, если существует последовательность конфигураций* $\sigma^0, \ldots, \sigma^n$, $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma^i)$, $\sigma^i(e)$ *определено для* $i = 0, \ldots, n$ *и* $\sigma^i(e) \neq 0$ *тогда и только тогда, когда* $i < n$.

Как видим, для определённости $\Pi(\sigma)$ *теперь необходима определённость всех выражений, которые встречаются в* Π *и значения которых нужно определять.*

Аналогично для программ с метками. Пусть Π *— программа,* σ *— состояние переменных* $\mathrm{Var}(\Pi)$. *Пусть* $(\sigma^i, \alpha_i)_i$ *— последовательность конфигураций* Π. *Если для определения очередной конфигурации* $(\sigma^{i+1}, \alpha_{i+1})$ *требуется определение значения выражения* e *в конфигурации* σ^i, *и оно не определено, то* $\Pi(\sigma)$ *мы тоже считаем неопределённым.*

В остальном определение семантики не изменяется.

Пример 6.7. *Рассмотрим подпрограмму из примера 6.4 на стр. 229. Если*

$$\Pi \twoheadleftarrow y = \mathsf{apply}(B, y, \mathsf{succ}(y));$$

то $\Pi(\sigma)$ *не определено.*

Рассмотрим логически законченный пример алгоритма с подпрограммами.

Пример 6.8. *Запишем алгоритм Евклида с использованием подпрограмм, см. рис. 6.5 на противоположной странице.*

Как видим, по сравнению с алгоритмом на рис. 3.5 на стр. 73 текст программы стал намного легче для восприятия и короче за счёт того, что фрагменты, которые выполняли вычитание теперь вынесены в отдельную подпрограмму, то есть вместо двух раз он написан единожды.

```
function S;
arguments x, y;
returns u;
    u = 0; v = y;
    while v < x do
        u = succ(u);
        v = succ(v);
    end;
end;
```

```
algorithm Euclid;
arguments x, y;
returns u;
    u = x; v = y;
    if u == 0 then
        u = v;
    else
        if v then
            while u != v do
                if u < v then
                    v = apply(S, v, u);
                else
                    u = apply(S, u, v);
                end;
            end;
        end;
    end;
end;
```

Рис. 6.5: Алгоритм Евклида.

Рассмотрим еще несколько примеров, показывающих как можно применять подпрограммы.

Пример 6.9. *Напишем программу для вычисления факториала числа (рис. 6.6 на следующей странице).*

Утверждение 6.2. *Для любого $n \geqslant 0$ выполняется* $\mathrm{Res}(F, n) = n!$.

Доказательство. *Обозначим через e выражение*

$$\mathsf{cond}(x == 0, 1, \mathsf{apply}(F, x - 1) \times x).$$

```
function F;
arguments x;
returns cond(x == 0,
    1,
    apply(F, x − 1) × x); ;
end;
```

Рис. 6.6: Факториал числа

Пусть $\delta = (F, n)$ *— произвольный вызов, а* $\sigma_\delta = \{(x, n)\}$ *— начальная конфигурация. Проведём индукцию по* n.
Базис индукции. Пусть $n = 0$. *Тогда* $\sigma_\delta(x == 0) = 1$ *и* $\mathrm{Res}(F, n) = \Pi_F(\sigma)(e) = 1 = n!$.
Шаг индукции. Пусть для всех вызовов (F, k), *где* $k < n$ *утверждение верно. Рассмотрим вызов* $\delta = (F, n)$, *где* $n > 0$. *Тогда* $\sigma_\delta(x == 0) = 0$, *значит,*

$$\mathrm{Res}(F, n) = \sigma_\delta(\mathsf{apply}(F, x - 1) \times x) = \sigma_\delta(\mathsf{apply}(F, x - 1)) \times n.$$

По определению значение выражения $\sigma_\delta(\mathsf{apply}(F, x-1))$ *равно* $\mathrm{Res}(F, n-1)$. *По индукционному предположению* $\mathrm{Res}(F, n-1) = (n-1)!$. *Следовательно,* $\mathrm{Res}(F, n) = (n-1)! \times n = n!$. □

Пример 6.10. *Напишем программу для нахождения суммы чётных цифр числа* x *в десятичной записи (рис. 6.7 на следующей странице).*

Через $\mathrm{Sed}(x)$ *обозначим сумму чётных цифр числа* x *и отметим два очевидных свойства этой функции.*

Утверждение 6.3. *1) Если последняя цифра числа* a *является чётной, то* $\mathrm{Sed}(a) = \mathrm{Sed}\left(\left[a/10\right]\right) + a \bmod 10$.

2) Если последняя цифра числа a *нечётна, то* $\mathrm{Sed}(a) = \mathrm{Sed}\left(\left[a/10\right]\right)$.

Заметим, что утверждение 6.3 описывает только свойства функции Sed *и никак не зависит от программы.*

Утверждение 6.4. *Для любого натурального числа* a *имеет место* $\mathrm{Res}(S, a) = \mathrm{Sed}(a)$.

```
function S;
arguments x;
returns cond(x == 0,
    0,
    cond((x mod 10) mod 2 == 0,
        apply(S, x / 10) + x mod 10,
        apply(S, x / 10)));;
end;
```

Рис. 6.7: Сумма чётных цифр числа

Доказательство. *Пусть* $\delta = (S, a)$ *— произвольный вызов, а* $\sigma_\delta = \{(x, a)\}$ *— начальная конфигурация. Проведём индукцию по* a.
Базис индукции. Если $a = 0$, *то* $\sigma_\delta(x == 0) = 1$ *и* $\mathrm{Res}(S, a) = 0 = \mathrm{Sed}(a)$.
Шаг индукции. Обозначим через e *выражение*

$$\mathsf{cond}((x \bmod 10) \bmod 2 == 0, \mathsf{apply}(S, x \,/\, 10) + x \bmod 10, \mathsf{apply}(S, x \,/\, 10)).$$

Пусть $a > 0$. *Тогда* $\mathrm{Res}(S, a) = \sigma_\delta(e)$. *Возможны два случая.*

1) *Если последняя цифра числа* a *чётна, то*

$$\sigma_\delta((x \bmod 10) \bmod 2 == 0) = 1,$$

поэтому

$$\mathrm{Res}(S, a) = \sigma_\delta(\mathsf{apply}(S, x \,/\, 10) + x \bmod 10) = \\ = \sigma_\delta(\mathsf{apply}(S, x \,/\, 10)) + a \bmod 10.$$

По определению получаем, что значение выражения $\mathsf{apply}(S, x \,/\, 10)$ *равняется* $\mathrm{Res}\left(S, \left[a/10\right]\right)$. *Так как* $\left[a/10\right] < a$, *то по индукционному предположению выполнено* $\mathrm{Res}\left(S, \left[a/10\right]\right) = \mathrm{Sed}\left(\left[a/10\right]\right)$. *Следовательно,*

$$\mathrm{Res}(S, a) = \mathrm{Sed}\left(\left[a/10\right]\right) + a \bmod 10 = \mathrm{Sed}(a)$$

по утверждению 6.3 на стр. 236.

2) *Случай, когда последняя цифра числа a является нечетной, рассматривается аналогично.* □

Задача 6.2. Напишите алгоритмы вычисления факториала и проверки числа на простоту, используя подпрограммы сложения, умножения и т. д.

Определение 6.6 (Основная программа)**.** *Поскольку подпрограммы — неотъемлемая часть алгоритма, то алгоритмом мы впредь будем называть всю совокупность подпрограмм и основной программы — программы, написанной после* algorithm. *Таким образом, если подпрограмм нет, то понятия алгоритма и основной программы совпадают.*

§ 6.2. Графы зависимости, списки и деревья вызовов

Рассмотрим произвольную совокупность подпрограмм A.

Определение 6.7 (Граф зависимости)**.** *Графом зависимости для множества подпрограмм A называется граф, множество вершин которого — A, а множество рёбер:*

$$\{(f, g) : \text{в } f \text{ встречаются выражения вида } \mathsf{apply}(g, \cdots)\}.$$

Пример 6.11. *Рассмотрим следующие подпрограммы:*

function A;	function B;	function C;
arguments x;	arguments x;	arguments x;
returns apply$(B, \mathsf{apply}(C, x))$;;	returns apply(C, x);;	returns apply(B, x);;
end;	end;	end;

Граф зависимости для данного множества подпрограмм изображён на рис. 6.8 на противоположной странице.

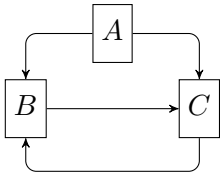


Рис. 6.8: Граф зависимости для примера 6.11

Задача 6.3. Постройте графы зависимости для подпрограмм из алгоритмов из задачи 6.2 на предыдущей странице.

Определение 6.8 (Рекурсия)**.** *Рекурсия — цикл в графе зависимостей. Рекурсивная подпрограмма — подпрограмма, которая входит в один из циклов. Прямая рекурсия — цикл, состоящий из одной вершины, косвенная рекурсия — цикл, состоящий более чем из одной вершины.*

Пример 6.12. *В предыдущем примере имеется рекурсия. B и C — рекурсивные подпрограммы. Рекурсия является косвенной.*

Определение 6.9 (Список вложенных вызовов)**.** *Список вложенных вызовов для выражения e и конфигурации σ (обозначаем с помощью* $\mathrm{Fc}(\sigma, e)$*) определяется по индукции:*

1) $e \multimap x$ *или* $e \multimap c$*, x — переменная, c — константа. Тогда* $\mathrm{Fc}(\sigma, e)$ *— пуст:* $\mathrm{Fc}(\sigma, e) = ()$.

2) $e \multimap o(e_1, \ldots, e_n)$*, где o — операция. Тогда*

$$\mathrm{Fc}(\sigma, e) = \mathrm{Fc}(\sigma, e_1), \ldots, \mathrm{Fc}(\sigma, e_n).$$

Если $\sigma(e_i)$ неопределено, то часть, следующая за $\mathrm{Fc}(\sigma, e_i)$ отбрасывается:

$$\mathrm{Fc}(\sigma, e) = \mathrm{Fc}(\sigma, e_1), \ldots, \mathrm{Fc}(\sigma, e_i).$$

3) $e \Leftarrow \mathsf{apply}(f, e_1, \ldots, e_n)$, *где f — подпрограмма. Тогда*

$$\mathrm{Fc}(\sigma, e) = \mathrm{Fc}(\sigma, e_1), \ldots, \mathrm{Fc}(\sigma, e_n), (f, \sigma(e_1), \ldots, \sigma(e_n)).$$

Если $\sigma(e_i)$ не определено, то часть, следующая за $\mathrm{Fc}(\sigma, e_i)$ считается пустой:

$$\mathrm{Fc}(\sigma, e) = \mathrm{Fc}(\sigma, e_1), \ldots, \mathrm{Fc}(\sigma, e_i).$$

Рассмотрим пример.

Пример 6.13. *Пусть f и g — подпрограммы с одним и двумя входными переменными соответственно. Пусть*

$$e \Leftarrow \mathsf{apply}(g, \mathsf{apply}(f, x), \mathsf{apply}(g, x, y))$$

Рассмотрим следующую конфигурацию: $\sigma = \{(x, 1), (y, 2)\}$. Предположим, что $\mathrm{Res}(f, 1) = 3$ и $\mathrm{Res}(g, 1, 2) = 5$. Тогда

$$\begin{aligned}
\mathrm{Fc}(\sigma, x) &= () \\
\mathrm{Fc}(\sigma, \mathsf{apply}(f, x)) &= ((f, 1)) \\
\mathrm{Fc}(\sigma, y) &= () \\
\mathrm{Fc}(\sigma, \mathsf{apply}(g, x, y)) &= ((g, 1, 2)) \\
\mathrm{Fc}(\sigma, e) &= ((f, 1), (g, 1, 2), (g, 3, 5))
\end{aligned}$$

Определение 6.10 (Список вложенных вызовов)**.** *Для программы Π список вложенных вызовов $\mathrm{Fc}(\sigma, \Pi)$ строится индукцией по построению программы Π:*

1) *Для пустого оператора список вложенных вызовов пуст:* $\mathrm{Fc}(\sigma, ;) = ()$.

2) $\Pi \Rightarrow x = e$; *в этом случае* $\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma, e)$.

3) $\Pi \Rightarrow \Pi_1\Pi_2$. *Тогда*

$$\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma, \Pi_1), \mathrm{Fc}(\Pi_1(\sigma), \Pi_2).$$

Если $\Pi_1(\sigma)$ *неопределено, то*

$$\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma, \Pi_1).$$

4) *Неполное ветвление:*

$$\Pi \Rightarrow \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array} \right.$$

в этом случае

$$\mathrm{Fc}(\sigma, \Pi) = \begin{cases} \mathrm{Fc}(\sigma, e), \mathrm{Fc}(\sigma, \Pi_1), & \text{если } \sigma(e) \neq 0; \\ \mathrm{Fc}(\sigma, e), & \text{иначе.} \end{cases}$$

Как и раньше, если $\sigma(e)$ *не определено, то* $\mathrm{Fc}(\sigma, \Pi_1)$ *из определения исчезает:*

$$\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma, e).$$

5) *Полное ветвление:*

$$\Pi \Rightarrow \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end}; \end{array} \right.$$

в этом случае

$$\mathrm{Fc}(\sigma, \Pi) = \begin{cases} \mathrm{Fc}(\sigma, e), \mathrm{Fc}(\sigma, \Pi_1), \textit{если } \sigma(e) \neq 0; \\ \mathrm{Fc}(\sigma, e), \mathrm{Fc}(\sigma, \Pi_2), \textit{иначе.} \end{cases}$$

Если $\sigma(e)$ не определено, то

$$\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma, e).$$

6) *Цикл:*

$$\Pi \Rightarrow \begin{cases} \textsf{while } e \textsf{ do} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

Пусть $(\sigma^i)_i$ — последовательность конфигураций для цикла: $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma^i)$. Тогда

$$\mathrm{Fc}(\sigma, \Pi) = \mathrm{Fc}(\sigma^0, e), \mathrm{Fc}(\sigma^0, \Pi_1), \mathrm{Fc}(\sigma^1, e), \mathrm{Fc}(\sigma^1, \Pi_1), \ldots$$

Заметим, что в этом случае последовательность вызовов $\mathrm{Fc}(\sigma, \Pi)$ может быть бесконечной, если программа зацикливается. Если последовательность $(\sigma^i)_i$ конечна, то количество частей вида $\mathrm{Fc}(\sigma, \cdots)$ тоже конечно. Оно может быть конечным в трех случаях:

- *$\sigma^n(e) = 0$ для некоторого n. В этом случае последней частью $\mathrm{Fc}(\sigma, \Pi)$ будет $\mathrm{Fc}(\sigma^n, e)$.*
- *$\Pi_1(\sigma^n)$ не определено для некоторого n. Тогда последней частью $\mathrm{Fc}(\sigma, \Pi)$ будет $\mathrm{Fc}(\sigma^n, \Pi_1)$.*
- *$\sigma^n(e)$ не определено для некоторого n. Тогда последней частью $\mathrm{Fc}(\sigma, \Pi)$ будет $\mathrm{Fc}(\sigma^n, e)$.*

Определение 6.11 (Список вложенных вызовов)**.** *Если*

$$\delta = (f, v_1, \ldots, v_n)$$

является вызовом, то списком вложенных вызовов *для* δ *будет*

$$\mathrm{Fc}(\delta) = \mathrm{Fc}(\sigma_\delta, \Pi_f), \mathrm{Fc}(\tau, r_f),$$

если $\Pi_f(\sigma_\delta)$ *определено, или*

$$\mathrm{Fc}(\delta) = \mathrm{Fc}(\sigma_\delta, \Pi_f)$$

в противном случае. Здесь мы считаем, что $\tau = \Pi_f(\sigma_\delta)$*, а* r_f *— выражение-результат подпрограммы* f*.*

Определение 6.12 (Множество вложенных вызовов)**.** *Если* $\mathrm{Fc}(\ldots)$ *— список вложенных вызовов, то с помощью* $\mathrm{Fs}(\ldots)$ *мы будем обозначать множество, которое образуют элементы этого списка.*

***Задача 6.4.** Дайте определение Fs по индукции (аналогично Fc).

Задача 6.5. Постройте $\mathrm{Fc}(\mathit{Euclid}, 6, 4)$ и $\mathrm{Fs}(\mathit{Euclid}, 6, 4)$, см. рис. 6.5 на стр. 235.

Задача 6.6. Постройте примеры списков и множеств вложенных вызовов, используя алгоритмы из задачи 6.2 на стр. 238.

Лемма 6.5 (О вложенных вызовах)**.** *Если* e*,* d *— выражения,* x *— переменная,* $x \in \mathrm{Var}(e)$*,* σ *— конфигурация,* $\tau = (x = d;)(\sigma)$ *и* $\sigma((e)^x_d)$ *определено, то*

$$\mathrm{Fs}(\sigma, (e)^x_d) = \mathrm{Fs}(\sigma, d) \cup \mathrm{Fs}(\tau, e).$$

Доказательство. Индукция по сложности выражения e. Очевидно, что τ отличается от σ только тем, что $\tau x = \sigma(d)$.

Б а з и с и н д у к ц и и . Пусть $e \multimap x$ (во всех остальных случаях не будет выполнено условие $x \in \mathrm{Var}(e)$). Тогда $\mathrm{Fs}(\tau, e) = \varnothing$ и $(e)_d^x \multimap d$, следовательно,

$$\mathrm{Fs}(\sigma, (e)_d^x) = \mathrm{Fs}(\sigma, d) \cup \varnothing.$$

Ш а г и н д у к ц и и . Допустим, что $e \multimap o(e_1, \ldots, e_n)$, где o — операция. Тогда

$$\mathrm{Fs}(\tau, e) = \mathrm{Fs}(\tau, e_1) \cup \cdots \cup \mathrm{Fs}(\tau, e_n),$$
$$\mathrm{Fs}(\sigma, (e)_d^x) = \mathrm{Fs}(\sigma, (e_1)_d^x) \cup \cdots \cup \mathrm{Fs}(\sigma, (e_n)_d^x).$$

По индукционному предположению

$$\mathrm{Fs}(\sigma, (e_i)_d^x) = \mathrm{Fs}(\tau, e_i) \cup \mathrm{Fs}(\sigma, d).$$

Следовательно,

$$\mathrm{Fs}(\sigma, (e)_d^x) = (\mathrm{Fs}(\tau, e_1) \cup \mathrm{Fs}(\sigma, d)) \cup \cdots \cup (\mathrm{Fs}(\tau, e_n) \cup \mathrm{Fs}(\sigma, d)) =$$
$$= (\mathrm{Fs}(\tau, e_1) \cup \cdots \cup \mathrm{Fs}(\tau, e_n)) \cup \mathrm{Fs}(\sigma, d) = \mathrm{Fs}(\tau, e) \cup \mathrm{Fs}(\sigma, d).$$

Случай, когда $e \multimap \mathsf{apply}(f, e_1, \ldots, e_n)$, где f — подпрограмма, рассматривается аналогично. □

Задача 6.7. Обоснуйте индукционный шаг для подпрограмм.

Определение 6.13 (Дерево вложенных вызовов)**.** *Если*

$$\delta = (f, v_1, \ldots, v_n)$$

вызов, то д е р е в о в л о ж е н н ы х в ы з о в о в $\mathrm{Ft}(\delta)$ *— это дерево с корнем* δ*, сыновьями каждой вершины* γ *которого являются вызовы* $\mathrm{Fc}(\gamma)$*, идущие в том же порядке слева направо.*

Следствие 6.6. *В дереве* $\mathrm{Ft}(\delta)$ *каждое поддерево с корнем* γ *является деревом* $\mathrm{Ft}(\gamma)$.

```
function h;
arguments x;
returns r;
    r = x;
    while 1 < r do
        r = g(2, r);
    end;
end;
```

```
function f;
arguments x;
returns r;
    if x < 2 then
        r = x + 1;
    else
        r = apply(g, apply(f, x − 1), x);
    end;
end;
```

```
function g;
arguments x, y;
returns r;
    if x < y then
        r = apply(g, y − x, apply(f, x));
    else
        r = apply(f, x − y);
    end;
end;
```

Рис. 6.9: Подпрограммы из примера 6.14.

*__Задача 6.8.__ Докажите следствие.

Рассмотрим пример.

__Пример 6.14.__ *Рассмотрим подпрограммы на рис. 6.9.*

Рассмотрим вызов $\delta = (h, 4)$. *Дерево вложенных вызовов* $\mathrm{Ft}(\delta)$ *изображено на рисунке 6.10 на следующей странице.*

__Задача 6.9.__ Постройте $\mathrm{Ft}(\mathit{Euclid}, 5, 3)$, см. рис. 6.5 на стр. 235.

__Задача 6.10.__ Постройте примеры деревьев вложенных вызовов, используя алгоритмы из задачи 6.2 на стр. 238.

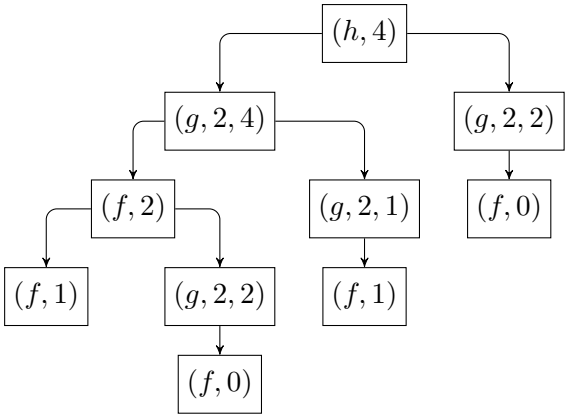


Рис. 6.10: Дерево вызовов программы из примера 6.14.

Определение 6.14 (Лес вложенных вызовов)**.** *Рассмотрим произвольный объект a — выражение или программу. Пусть σ — его конфигурация. Построим $\mathrm{Fs}(\sigma, a)$ и для каждого элемента $\delta \in \mathrm{Fs}(\sigma, a)$ — дерево вызовов $\mathrm{Ft}\,\delta$. Множество этих деревьев мы будем называть* лесом вызовов *$\mathrm{Ff}(\sigma, a)$.*

Пример 6.15. *Рассмотрим подпрограммы из примера 6.14 на предшествующей странице. Лес вызовов $\mathrm{Ff}(\sigma_\delta, \Pi_h)$, будет состоять из двух деревьев — поддеревьев $\mathrm{Ft}\,\delta$, см. рис. 6.10, корнями которых будут $(g, 2, 4)$ и $(g, 2, 2)$ (вторая строка).*

Определение 6.15 (Рекурсивный вызов)**.** *Пусть $\mathrm{Ft}\,\delta$ — дерево вложенных вызовов для $\delta = (f, \bar{v})$. Вершина ε, отличная от корня и имеющая вид $\varepsilon = (f, \bar{v}_1)$, называется* рекурсивным вызовом *для δ.*

Пример 6.16. *В предыдущем примере* $(f, 1)$ *(первый в четвёртой строке) — рекурсивный вызов для* $(f, 2)$*, а* $(g, 2, 2)$ *(в четвёртой строке) — рекурсивный вызов для* $(g, 2, 4)$*.*

Замечание 6.2. Если для вызова $\delta = (f, \bar{v})$ есть рекурсивные вызовы, то f — рекурсивная подпрограмма.

Обратное неверно.

Пример 6.17. *Рассмотрим две подпрограммы:*

```
function f;                    function g;
arguments x, y;                arguments x, y;
returns cond(x < y,            returns cond(x < y,
    2,                             apply(f, x, y),
    apply(g, x, y));;              2);;
end;                           end;
```

Очевидно, что обе подпрограммы рекурсивны (косвенная рекурсия). Однако, рекурсивные вызовы ни для какой из них невозможны.

Задача 6.11. Докажите, что в предыдущем примере рекурсивные вызовы невозможны.

Докажем следующее свойство дерева вызовов:

Лемма 6.7 (О бесконечных деревьях вызовов)**.** *Если дерево* $\mathrm{Ft}(\delta)$ *бесконечно, то результат* $\mathrm{Res}\,\delta$ *не определён.*

Доказательство. Предположим, что $\delta = (f, \bar{v})$. Прежде всего нужно доказать, для того чтобы $\Pi_f(\sigma_\delta)$ было определено необходимо чтобы

1) список $\mathrm{Fc}(\sigma_\delta, \Pi_f)$ был конечным;

2) для всех вызовов γ из списка $\mathrm{Fc}(\sigma, \Pi)$ было определено $\mathrm{Res}\,\gamma$.

Предположим, что дерево $\mathrm{Ft}(\delta)$ бесконечно, но $\mathrm{Res}\,\delta$ определено (следовательно, определено $\Pi_f(\sigma_\delta)$). Индукцией по глубине вершин дерева $\mathrm{Ft}(\delta)$ легко доказывается, что условия (1) и (2) выполняются для всех вершин дерева. Значит, каждая вершина имеет конечно много сыновей. По теореме о конечных деревьях 2.2 на стр. 33 в

дереве $\mathrm{Ft}(\delta)$ должна существовать бесконечная ветвь. Так как подпрограмм конечно много, то вызовы некоторой подпрограммы будут на этой ветви повторяться бесконечно часто. Согласно замечанию 6.1 на стр. 229 мы должны считать, $\mathrm{Res}\,\delta$ неопределён, что противоречит предположению. □

*__Задача 6.12.__ Восполните все пропущенные места в доказательстве леммы.

Обратное, очевидно, неверно:

Пример 6.18. *Если программа не содержит подпрограмм, то дерево вызовов состоит из одного корня, но вовсе необязательно, что программа останавливается:*

```
function A;
arguments x;
returns x;
    while 0 < 1 do
        ;
    end;
end;
```

Очевидно, что результат $\mathrm{Res}(A, x)$ *не определён для всех* $x \in \omega$.

Однако, для программ без циклов обратное верно. Прежде всего докажем следующую лемму.

Лемма 6.8 (Об определённости вложенных вызовов)**.** *Если для программы без циклов каждый элемент* $\mathrm{Fc}(\delta)$ *определён, то* $\mathrm{Res}(\delta)$ *определён.*

Доказательство. Сначала индукцией по построению выражения e доказывается, что если в $\mathrm{Fc}(\sigma, e)$ каждый элемент определён, то значение e определено. Затем то же самое доказывается для программ. □

*__Задача 6.13.__ Докажите лемму полностью.

Теорема 6.9 (О конечных деревьях вызовов)**.** *Если программа не содержит циклов, то дерево* $\mathrm{Ft}\,\delta$ *конечно для вызова* δ *тогда и только тогда, когда* $\mathrm{Res}\,\delta$ *определён.*

```
function S;
arguments x, y;
returns apply(S_1, x, y, 0);;
end;
```

```
function S_1;
arguments x, y, z;
returns cond(y < x,
    apply(S_1, x, succ(y), succ(z)),
    z);;
end;
```

Рис. 6.11: Вычитание чисел.

Доказательство. Индукция по высоте дерева $\mathrm{Ft}(\delta)$.
Базис индукции. Если δ — лист дерева, то список вложенных вызовов $\mathrm{Fc}\,\delta$ пуст, и все его элементы определены. По лемме $\mathrm{Res}\,\delta$ определено.
Шаг индукции. Пусть для всех сыновей вершины δ утверждение доказано. Так как каждый вложенный вызов определён, то $\mathrm{Res}\,\delta$ тоже определено.

В обратную сторону — это лемма 6.7 на стр. 247. □

Лемма 6.10 (Условие зацикливания)**.** *Если в дереве* $\mathrm{Ft}\,\delta$ *есть вершина* $\varepsilon = \delta$*, то* $\mathrm{Res}\,\delta$ *не определён.*

Задача 6.14. Докажите лемму.

§ 6.3. *Функциональное программирование

В этом параграфе мы докажем, что вызовы подпрограмм могут заменить все другие программные конструкции. Но прежде чем делать это в общем случае рассмотрим пример.

Пример 6.19. *Изменим подпрограмму, вычисляющую разность, и введём вспомогательную подпрограмму, см. рис. 6.11.*

Как видим, ни одна из подпрограмм не содержит ни ветвлений, ни циклов, ни присваиваний. Докажем, что подпрограмма S действительно вычисляет разность чисел.

Утверждение 6.11. *Для любых $a, b, c \in \omega$ выполнено*

$$\mathrm{Res}(S_1, a, b, c) = \lfloor a - b \rfloor + c.$$

Доказательство. Докажем это индукцией по $\lfloor a-b \rfloor$. Поскольку тела подпрограмм пусты, то начальная и конечная конфигурации будут совпадать, следовательно, для любого вызова δ

$$\mathrm{Res}\,\delta = \sigma(r),$$

где σ — начальная конфигурация вызова:

$$\sigma = \{(x, a), (y, b), (z, c)\}.$$

Базис индукции. $\lfloor a - b \rfloor = 0$. То есть $b \geqslant a$. Тогда $\sigma(y < x) = 0$ и

$$\mathrm{Res}(S_1, a, b, c) = \sigma(z) = c = \lfloor a - b \rfloor + c,$$

Шаг индукции. Пусть для $\lfloor a - b \rfloor < n$ утверждение доказано, $n > 0$ и $\lfloor a - b \rfloor = n$. Тогда $a > b$, следовательно,

$$\mathrm{Res}(S_1, a, b, c) = \sigma(\mathsf{apply}(S_1, x, \mathsf{succ}(y), \mathsf{succ}(z))).$$

По определению последнее равно результату вызова $\mathrm{Res}(S_1, a, b + 1, c + 1)$. Но

$$\lfloor a - (b + 1) \rfloor = \lfloor a - b - 1 \rfloor < \lfloor a - b \rfloor = n.$$

По индукционному предположению:

$$\mathrm{Res}(S_1, a, b + 1, c + 1) = \lfloor a - (b + 1) \rfloor + (c + 1) = \lfloor a - b \rfloor + c.$$

Следовательно,

$$\mathrm{Res}(S_1, a, b, c) = \mathrm{Res}(S_1, a, b + 1, c + 1) = \lfloor a - b \rfloor + c. \qquad \square$$

Следствие 6.12. $\mathrm{Res}(S_1, a, b, 0) = \lfloor a - b \rfloor$.

Следствие 6.13. $\mathrm{Res}(S, a, b) = \mathrm{Res}(S_1, a, b, 0) = \lfloor a - b \rfloor$.

Итак, мы доказали, что рассмотренный пример корректен.

Теперь рассмотрим, как избавиться от присваиваний и циклов в общем случае.

Определение 6.16 (Функциональная программа)**.** *Программа называется функциональной, если она не содержит никаких операторов, кроме пустых.*

Функциональная программа так называется, потому что единственное разрешённое действие — возврат результата, который может вычисляться с использованием подпрограмм (функций). Разумеется, чтобы обойтись без обычных программных конструкций множество допустимых операций должно быть достаточно большим. Мы покажем, что наличие введённой ранее операции cond является достаточным условием. Существует теория функционального программирования (λ-исчисление), где доказывается, что можно обойтись и без такой операции, если изменить обычные понятия истинности и ложности.

Итак, перейдём к доказательству основных теорем.

Теорема 6.14 (О функциональных программах)**.** *Всякий алгоритм эквивалентен некоторому функциональному алгоритму.*

Доказательство. Прежде всего, несколько замечаний. Будем считать, что тело каждой подпрограммы является структурированной программой. Мы знаем, что этого можно добиться (теорема о структуризации).

Далее, основную программу с телом Π мы выделим в отдельную подпрограмму, например, A'. То есть основную программу

algorithm A;
arguments $x_1, \ldots, x_n$;
returns e;
    Π
end;

заменяем на

function A';
arguments $x_1, \ldots, x_n$;
returns e;
    Π
end;

algorithm A;
arguments $x_1, \ldots, x_n$;
returns apply$(A', x_1, \ldots, x_n)$;;
end;

Ясно, что полученный алгоритм эквивалентен исходному.

Мы будем предполагать, что все переменные любой подпрограммы являются входными. Этого тоже нетрудно добиться, достаточно включить их в список после arguments, а в выражениях с использованием apply подставить для них, например, нули.

Заметим, что при соблюдении этих условий выполняется такое утверждение. Если $\bar{x}$ — входные переменные подпрограммы f, то для любой конфигурации σ

$$\sigma(\mathsf{apply}(f, \bar{x})) = \Pi_f(\sigma)(r_f), \tag{6.1}$$

если r_f — выражение результат для f.

С каждой программой Π свяжем некоторое натуральное число $w(\Pi)$ — вес. Это будет мера «испорченности программы» по сравнению с функциональной. Определим его индуктивно для тела подпрограммы f:

1) Для пустого оператора вес равен нулю: $w(;) = 0$.

2) $\Pi \multimap x = e;\ \ w(\Pi) = 1.$

3) $\Pi \multimap \Pi_1\Pi_2,\ w(\Pi) = w(\Pi_1) + w(\Pi_2) + 1.$

4) $\Pi \multimap \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{end};\ \ w(\Pi) = w(\Pi_1) + 1.$

5) $\Pi \multimap \mathsf{if}\ e\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\ \ w(\Pi) = w(\Pi_1) + w(\Pi_2) + 1.$

6) $\Pi \multimap \mathsf{while}\ e\ \mathsf{do}\ \Pi_1\ \mathsf{end};\ \ w(\Pi) = w(\Pi_1) + 1.$

Если алгоритм A состоит из подпрограмм $f_1, \ldots, f_n$ и основной программы Π_A, то $w(A) = w(\Pi_A) + w(\Pi_{f_1}) + \cdots + w(\Pi_{f_n})$.

Нетрудно заметить, что если вес алгоритма равен 0, то он является функциональным, потому что не может содержать никаких операторов, кроме пустых.

Теперь покажем, что если вес алгоритма больше 0, то он может быть уменьшен.

Пусть тело некоторой подпрограммы f не удовлетворяет определению функциональной программы. Возможны пять вариантов:

1) $\Pi_f \multimap x = e;$

2) $\Pi_f \multimap \Pi'\Pi''$

3) $\Pi_f \multimap \mathsf{if}\ e\ \mathsf{then}\ \Pi'\ \mathsf{end};$

4) $\Pi_f \multimap \mathsf{if}\ e\ \mathsf{then}\ \Pi'\ \mathsf{else}\ \Pi''\ \mathsf{end};$

5) $\Pi_f \multimap \mathsf{while}\ e\ \mathsf{do}\ \Pi'\ \mathsf{end};$

Рассмотрим, как разобраться с каждым из этих случаев. Будем полагать, что r_f — выражение результат подпрограммы f. В каждом из перечисленных случаем мы построим новую подпрограмму $\hat{f}$, эквивалентную исходной, но с меньшим весом.

1) Заменим в r_f переменную x на e:

$$\begin{array}{l}\mathsf{function}\ \hat{f};\\ \mathsf{arguments}\ x_1,\ldots,x_n;\\ \mathsf{returns}\ (r_f)^x_e;;\\ \mathsf{end};\end{array}$$

Согласно лемме 3.27 на стр. 87 имеем

$$(x = e;)(\sigma)(r_f) = \sigma((r_f)^x_e),$$

поэтому $\mathrm{Res}(f, \bar{v}) = \mathrm{Res}(\hat{f}, \bar{v})$.

Рассмотрим, как в ходе такого преобразования изменяется вес:

$$w(\Pi_f) = 1, \quad w(\Pi_{\hat{f}}) = 0 < 1.$$

2) Создадим новую подпрограмму f_1:

$$\begin{array}{ll}\mathsf{function}\ \hat{f};\\ \mathsf{arguments}\ x_1,\ldots,x_n;\\ \mathsf{returns}\ \mathsf{apply}(f_1, x_1, \ldots, x_n);\\ \quad \Pi'\\ \mathsf{end};\end{array} \qquad \begin{array}{ll}\mathsf{function}\ f_1;\\ \mathsf{arguments}\ x_1,\ldots,x_n;\\ \mathsf{returns}\ r_f;\\ \quad \Pi''\\ \mathsf{end};\end{array}$$

Рассмотрим вызов $\delta = (\hat{f}, \bar{v})$. По определению

$$\mathrm{Res}\,\delta = \Pi'(\sigma_\delta)(\mathsf{apply}(f_1, x_1, \ldots, x_n)).$$

Согласно (6.1) имеем

$$\Pi'(\sigma_\delta)(\mathsf{apply}(f_1, x_1, \ldots, x_n)) = \Pi''(\Pi'(\sigma_\delta))(r_f).$$

Имеем $\Pi''(\Pi'(\sigma_\delta)) = (\Pi'\Pi'')(\sigma_\delta)$ по определению семантики следования, значит

$$\mathrm{Res}\,\delta = (\Pi'\Pi'')(\sigma_\delta)(r_f) = \Pi_f(\sigma_\delta)(r_f) = \mathrm{Res}(f, \bar{v}).$$

Рассмотрим, как в ходе такого преобразования изменяется вес:

$$w(\Pi_f) = w(\Pi') + w(\Pi'') + 1,$$

$$w(\Pi_{\hat{f}}) + w(\Pi_{f_1}) = w(\Pi') + w(\Pi'') < w(\Pi_f).$$

3) Аналогично следующему пункту.

4) Построим две новые подпрограммы f_1 и f_2:

function $\hat{f}$; arguments $\bar{x}$; returns cond(e, apply($f_1, \bar{x}$), apply($f_2, \bar{x}$));; end;	function f_1; arguments $\bar{x}$; returns r_f; Π' end;	function f_2; arguments $\bar{x}$; returns r_f; Π'' end;

Рассмотрим вызов $\delta = (\hat{f}, \bar{v})$. Предположим, что $\sigma_\delta(e) \neq 0$. Тогда

$$\operatorname{Res}\delta = \sigma_\delta(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), \mathsf{apply}(f_2, \bar{x}))) = \sigma_\delta(\mathsf{apply}(f_1, \bar{x})).$$

Пользуясь (6.1), получаем

$$\sigma_\delta(\mathsf{apply}(f_1, \bar{x})) = \Pi'(\sigma_\delta)(r_f) = \operatorname{Res}(f, \bar{v}).$$

Последнее равенство получается по определению семантики полного ветвления: $\Pi_f(\sigma_\delta) = \Pi'(\sigma_\delta)$.

Случай $\sigma_\delta(e) = 0$ рассматривается точно так же.

Вес был равен

$$w = w(\Pi') + w(\Pi'') + 1,$$

стал равен:

$$w(\Pi') + w(\Pi'') < w.$$

5) Создадим новую подпрограмму f_1:

function $\hat{f}$;
arguments $\bar{x}$;
returns cond(e,
apply($f_1, \bar{x}$),
r_f);;
end;
	function f_1;
	arguments $\bar{x}$;
	returns apply($\hat{f}, \bar{x}$);
	Π'
	end;

Рассмотрим вызовы $\delta = (f, \bar{v})$ и $\hat{\delta} = (\hat{f}, \bar{v})$. Пусть $\operatorname{Res}\delta$ определено. Значит, последовательность конфигураций цикла $(\sigma^i)_{i=0}^n$: $\sigma^0 = \sigma_\delta$, $\sigma^{i+1} = \Pi'(\sigma^i)$ является конечной, $\Pi'(\sigma^i)$ определено для всех $i < n$ и $\sigma^i(e) \neq 0$ тогда и только тогда, когда $i < n$. Следовательно, $\operatorname{Res}\delta = \sigma^n(r_f)$.

Индукцией по $n - i$ докажем, что $\sigma^i(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f))$ определено и равно $\sigma^n(r_f)$.

Базис индукции. Если $n = i$, то $\sigma^i(e) = 0$ и, следовательно,

$$\sigma^i(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)) = \sigma^i(r_f) = \sigma^n(r_f).$$

Шаг индукции. Пусть для $i+1$ утверждение доказано и $i < n$. Тогда $\sigma^i(e) \neq 0$, и поэтому

$$\sigma^i(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)) = \sigma^i(\mathsf{apply}(f_1, \bar{x})).$$

Согласно (6.1), получаем

$$\sigma^i(\mathsf{apply}(f_1, \bar{x})) = \Pi'(\sigma^i)(\mathsf{apply}(\hat{f}, \bar{x})) = \sigma^{i+1}(\mathsf{apply}(\hat{f}, \bar{x})).$$

Еще раз используем (6.1):

$$\sigma^{i+1}(\mathsf{apply}(\hat{f}, \bar{x})) = (;)(\sigma^{i+1})(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)) = \\ = \sigma^{i+1}(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)).$$

По индукционному предположению последнее определено и равно $\sigma^n(r_f)$, что и требовалось.

При $i = 0$ получим:

$$\operatorname{Res}\delta = \sigma^n(r_f) = \sigma^0(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)) = \\ = \sigma_{\hat{\delta}}(\mathsf{cond}(e, \mathsf{apply}(f_1, \bar{x}), r_f)) = \operatorname{Res}\hat{\delta},$$

то есть результаты обоих вызовов определены и равны.

Пусть теперь $\operatorname{Res}\hat{\delta}$ определено. Тогда дерево вложенных вызовов конечно. Используем индукцию по количеству рекурсивных вызовов в этом дереве, чтобы доказать, что и $\operatorname{Res}\delta$ определено. Доказывать будем такое утверждение: если σ — начальная конфигурация вызова $\hat{\delta}$, то $\Pi_f(\sigma)$ определено.

Б а з и с и н д у к ц и и . В случае, когда рекурсивных вызовов нет, $\sigma(e) = 0$. Если допустить обратное, $\sigma(e) \neq 0$, то

$$\operatorname{Res}\hat{\delta} = \sigma(\mathsf{apply}(f_1, \bar{x})) = \Pi'(\sigma)(\mathsf{apply}(\hat{f}, \bar{x})).$$

Поскольку мы предполагаем, что $\operatorname{Res}\hat{\delta}$ определено, то $\Pi'(\sigma)$ должно быть определено, и в дереве будет рекурсивный вызов.

Итак, $\sigma(e) = 0$, следовательно, тело цикла не выполняется и $\operatorname{Res}\delta = \sigma(r_f)$. Последнее определено, из-за определенности $\operatorname{Res}\hat{\delta}$.

Ш а г и н д у к ц и и . Предположим, что для всех деревьев с меньшим количеством рекурсивных вызовов утверждение доказано. Если $\sigma(e) = 0$, то рассуждения такие же как и прежде. Пусть $\sigma(e) \neq 0$. Тогда

$$\operatorname{Res}\hat{\delta} = \sigma(\mathsf{apply}(f_1, \bar{x})) = \Pi'(\sigma)(\mathsf{apply}(\hat{f}, \bar{x})).$$

Заметим, что как и раньше $\sigma^1 = \Pi'(\sigma)$ определено. Пусть $\bar{v}'$ — значения переменных $\bar{x}$ на конфигурации σ^1, рассмотрим вызовы $\delta' = (f, \bar{v}')$ и $\hat{\delta}' = (\hat{f}, \bar{v}')$. Поскольку в дереве вложенных

вызовов вершина $\hat{\delta}'$ располагается под вершиной $\hat{\delta}$, то количество рекурсивных вызовов для $\hat{\delta}'$ меньше чем для $\hat{\delta}$. По индукционному предположению получаем, что $\operatorname{Res}\delta'$ определено, то есть существует конечная последовательность конфигураций цикла $(\sigma^i)_{i=1}^n$, $\sigma^{i+1} = \Pi'(\sigma^i)$,

$$\sigma^n(e) \neq 0 \iff i < n$$

и $\operatorname{Res}\delta' = \sigma^n(r_f)$. Но тогда последовательность $(\sigma^i)_{i=0}^n$, $\sigma^0 = \sigma$, будет последовательностью конфигураций цикла на σ, следовательно, $\operatorname{Res}\delta = \sigma^n(r_f)$ и тоже определено. Из предыдущей части доказательства получаем, что $\operatorname{Res}\delta = \operatorname{Res}\delta'$.

Вес был:

$$w(\Pi_f) = w(\Pi') + 1,$$

стал:

$$w(\Pi_{\hat{f}}) + w(\Pi_{f_1}) = 0 + w(\Pi') < w(\Pi_f).$$

Будем производить преобразования 1)–5), пока они приводят к уменьшению $w(A)$.

Как только $w(A) = 0$, мы получили функциональную программу. □

*__Задача 6.15.__ Покажите, как убрать неполное ветвление и докажите правильность предложенного построения.

Как мы видим, вызов подпрограммы заменяет следование, операция cond — ветвление, а рекурсивный вызов — оператор цикла.

__Задача 6.16.__ Пользуясь методом из теоремы, по алгоритмам из задачи 6.2 на стр. 238 постройте эквивалентные им функциональные.

*__Задача 6.17.__ Доказать, что если тело структурированной программы не содержит циклов, то при переводе её в функциональную программу можно обойтись без рекурсии и дополнительных функций.

Указание. Внести все операторы присваивания внутрь операторов if, а затем избавиться от присваиваний.

***Задача 6.18.** Предложить способ создания по программе с метками эквивалентной функциональной программы без промежуточного построения структурированной программы.
Указание. Каждый оператор программы преобразовать в отдельную функцию.

§ 6.4. **Удаление подпрограмм

Теперь докажем обратную теорему.

Теорема 6.15 (Об удалении подпрограмм)**.** *По каждому нерекурсивному алгоритму можно построить эквивалентный алгоритм без подпрограмм.*

Доказательство. Рассмотрим граф зависимостей нашего алгоритма. Поскольку он конечен и не содержит циклов, то в нем найдется вершина f, из которой не исходит ни одного ребра. Это должна быть подпрограмма f, в которой не встречаются вызовы никаких других подпрограмм. Пользуясь теоремой о подстановке можно удалить вызовы подпрограммы f из всех остальных подпрограмм и основной программы. Ввиду того, что внутри f другие подпрограммы не вызываются, то после подстановки рекурсии не будет. Поскольку теперь подпрограмма f ниоткуда не вызывается, то ее можно исключить. В новом графе зависимостей циклов по-прежнему не будет, поэтому, пользуясь этой процедурой, можно исключить по очереди все подпрограммы. □

Такой метод не годится, если присутствует рекурсия, так подставляя вместо выражений вида $\mathsf{apply}(f, \bar{e})$ тело подпрограммы f мы снова будем получать выражения такого вида и никогда не сможем исключить их полностью.

Прежде чем показывать, как удаляется рекурсия, докажем следующую лемму.

Лемма 6.16 (О кодировании пар чисел)**.** *Существует вычислимая двухместная функция ν такая, что*

1) *для всяких x, y, x', y':*

$$\nu(x,y) = \nu(x',y') \iff x = x' \text{ и } y = y';$$

2) *существуют вычислимые функции ν_1 и ν_2 такие, что для всяких x, y:*

$$\nu(\nu_1(x), \nu_2(x)) = x; \quad \nu_1(\nu(x,y)) = x; \quad \nu_2(\nu(x,y)) = y.$$

Доказательство. В самом деле, рассмотрим функцию

$$\nu(x,y) = (2x+1)2^y.$$

Чтобы доказать справедливость обоих пунктов, заметим, что каждое положительное натуральное число допускает единственное разложение на простые множители: $z = 2^{i_2}3^{i_3}\dots p_k^{i_{p_k}}$. Следовательно, для каждого натурального $z > 0$ существуют единственные числа a и b такие, что b — нечётное, a — степень двойки и $z = ab$: $a = 2^{i_2}$, $b = 3^{i_3}\dots p_k^{i_{p_k}}$.

Пусть $\nu(x,y) = \nu(x',y') = z$. Тогда для z существуют a и b. Но тогда $a = 2^y = 2^{y'}$, $b = (2x+1) = (2x'+1)$. То есть $x = x'$ и $y = y'$. Аналогично доказывается второй пункт. □

***Задача 6.19.** Завершите доказательство леммы.

Задача 6.20. Напишите подпрограммы для вычисления функций ν, ν_1, ν_2 из леммы без использования рекурсии.

Очевидными свойствами функции ν являются следующие:

Следствие 6.17. $\nu(x,y) > x$, $\nu(x,y) > y$, $\nu(x,y) > 0$.

Следующий наш шаг — доказать, что всякую косвенную рекурсию можно заменить прямой.

```
function f;
arguments x, x_1, ..., x_n;
returns r;
    if x == 1̂ then
        Π'_{f_1}
        r = r_1;
    end;
    ⋮
    if x == m̂ then
        Π'_{f_m}
        r = r_m;
    end;
end;
```

Рис. 6.12: Удаление косвенной рекурсии.

Лемма 6.18 (Об удалении косвенной рекурсии)**.** *Каждый алгоритм эквивалентен некоторому алгоритму, в котором присутствует только одна подпрограмма.*

Доказательство. Прежде всего, будем считать, что все подпрограммы имеют одинаковое количество параметров. Если это не так, всегда можно добавить фиктивные параметры, которые нигде не используются.

Пусть алгоритм содержит m подпрограмм $f_1, \ldots, f_m$, каждая из которых имеет n параметров. Можно считать, что все подпрограммы $f_1, \ldots, f_m$ имеют входные переменные $x_1, \ldots, x_n$. Пусть r_i — выражение-результат для f_i. Рассмотрим подпрограмму f, приведённую на рис. 6.12.

Здесь x и r — новые переменные. Напомним, что для натурального числа a с помощью $\widehat{a}$ мы обозначаем выражение

$$\widehat{n} \Leftarrow \underbrace{\mathsf{succ}(\mathsf{succ}(\ldots\mathsf{succ}(}_{a\text{ раз}}\,0\,\underbrace{)\ldots))}_{a\text{ раз}}.$$

Каждая программа Π'_{f_i} получена из Π_{f_i} заменой каждого выражения вида $\mathsf{apply}(f_i, \bar{e})$ на $\mathsf{apply}(f, i, \bar{e})$. Для каждой конфигурации σ будем иметь

$$\sigma(\mathsf{apply}(f_i, \bar{e})) = \sigma(\mathsf{apply}(f, i, \bar{e})).$$

Проделав точно такую же замену $\mathsf{apply}(f_i, \bar{e})$ на $\mathsf{apply}(f, i, \bar{e})$ в основной программе мы получим алгоритм, эквивалентный исходному. Так как теперь в алгоритме присутствует только одна подпрограмма, то рекурсия может быть только прямой. □

***Задача 6.21.** Докажите, что для любой конфигурации σ

$$\sigma(\mathsf{apply}(f_i, \bar{e})) = \sigma(\mathsf{apply}(f, i, \bar{e}))$$

для любого $i = 1, \ldots, m$, и любых выражений $\bar{e}$.

Теорема 6.19 (Об удалении рекурсии)**.** *По каждому алгоритму можно построить эквивалентный алгоритм без рекурсии.*

Доказательство. Согласно предыдущим утверждениям, мы можем считать, что у нас есть одна подпрограмма f с n параметрами. Прежде всего, несколько модифицируем саму f.

Во-первых, перестроим тело f в простую программу. Это означает, в частности, что каждый вызов подпрограммы f является отдельным оператором присваивания вида

$$u = \mathsf{apply}(f, u_1, \ldots, u_n);$$

Во-вторых, преобразуем тело подпрограммы f в программу с метками. Будем считать, что метками являются натуральные числа, причем единственная заключительная метка — это 0. Заметим, что

все проделанные преобразования привели к эквивалентной подпрограмме.

Итак, подпрограмма f сейчас имеет вид

function f;
arguments $\bar{x}$;
returns r;
    o_1
    o_2
    $\vdots$
    o_N
end;

Каждый из операторов o_k имеет один из видов:

1) $o_k \twoheadrightarrow \alpha_k\ x_k = e_k;\ \beta_k$,

2) $o_k \twoheadrightarrow \alpha_k$ if x_k then β_k else γ_k,

3) $o_k \twoheadrightarrow \alpha_k\ u = \mathsf{apply}(f, u_1, \ldots, u_n);\ \beta_k$,

причём e_k не содержат apply.

Приступим к построению по подпрограмме f подпрограммы f_1 без рекурсивных вызовов. Будем делать это так же, как мы структуризировали программу с метками (см. доказательство теоремы 4.12 на стр. 127). Добавим новые переменные w, y и s. По каждому o_k определим соответствующую программу $\varphi(o_k)$. Для операторов первых двух видов определение $\varphi(o_k)$ не изменится. Для операторов третьего вида строим $\varphi(o_k)$ как показано на рис. 6.13 на следующей странице. Здесь мы предполагаем, что $z_1, \ldots, z_m$ — все переменные программы Π_f (исключая y, w, s, но включая r), которые не являются входными, а $x_1, \ldots, x_n$ — входные.

Построим программу φ_1 как показано на рис. 6.14 на следующей странице. Заметим, что u тоже является одной из переменных

$$
\varphi(o_k) \Rightarrow \left\{
\begin{array}{l}
\textsf{if } w \textsf{ then} \\
\quad \textsf{if } y == \widehat{\alpha}_k \textsf{ then} \\
\qquad s = \nu(s, z_1); \ldots s = \nu(s, z_m); \\
\qquad s = \nu(s, x_1); \ldots s = \nu(s, x_n); \\
\qquad s = \nu(s, \widehat{\beta}_k); \\
\qquad x_1 = u_1; \ldots x_n = u_n; \\
\qquad y = \widehat{\alpha}_1; \\
\qquad w = 0; \\
\quad \textsf{end}; \\
\textsf{end};
\end{array}
\right.
$$

Рис. 6.13: Построение $\varphi(o_k)$.

$$
\varphi_1 \Rightarrow \left\{
\begin{array}{l}
\textsf{if } w \textsf{ then} \\
\quad \textsf{if } s \textsf{ then} \\
\qquad u = r; \\
\qquad y = \nu_2(s); s = \nu_1(s); \\
\qquad \left.
\begin{array}{l}
x_n = \nu_2(s); s = \nu_1(s); \\
\vdots \\
x_1 = \nu_2(s); s = \nu_1(s); \\
z_m = \nu_2(s); s = \nu_1(s); \\
\vdots \\
z_1 = \nu_2(s); s = \nu_1(s);
\end{array}
\right\} \varphi_1' \\
\qquad w = 0; \\
\quad \textsf{end}; \\
\textsf{end};
\end{array}
\right.
$$

Рис. 6.14: Построение φ_1.

```
function f_1;
arguments x̄;
returns r;
    y = α̂_1; w = 0; s = 0;
    while y do
        w = succ(0);
        φ(o_1)
        ⋮
        φ(o_N)
        φ_1
    end;
end;
```

Рис. 6.15: Построение f_1.

$x_1, \ldots, x_n, z_1, \ldots, z_m$, но для нее в φ_1' мы не включаем присваивание $u = \nu_2(s);$.

Теперь мы готовы построить нерекурсивную подпрограмму f_1 эквивалентную исходной (см. рис. 6.15). Докажем эквивалентность f и f_1.

Пусть $\Pi_f(\sigma)$ определено. Следовательно, последовательность конфигураций $(\sigma^i, \lambda_i)_i$ конечна. Рассмотрим последовательность конфигураций цикла Π_{f_1}: $(\sigma_1^j)_j$, и последовательность конфигураций Π_f: $(\sigma^i, \lambda_i)_i$. Пусть $j_0 = 0$. Индукцией по высоте леса вложенных вызовов $\mathrm{Ff}(\sigma, \Pi_f)$ мы докажем следующее утверждение:

Утверждение 6.20. *Для всякого i существует j_i такое, что*

1) $\sigma_1^{j_i} s = \sigma_1^{j_0} s$,

2) $\sigma_1^l s \geqslant \sigma_1^{j_0} s$ *для* $l \in [j_0, j_i]$,

3) $\sigma_1^{j_i} y = \lambda_i$,

4) $\sigma_1^{j_i} w = 0$,

5) $\sigma_1^{j_i} R = \sigma^i f$,

6) $\sigma_1^{j_i} z = \sigma^i z$ *для остальных переменных* z.

Базис индукции. Если вложенные вызовы отсутствуют, то доказательство повторяет доказательство теоремы о структуризации. Шаг индукции. Пусть для лесов высоты не более h утверждение доказано, и предположим, что лес $\mathrm{Ff}(\sigma, \Pi_f)$ имеет высоту $h+1$. Используем индукцию по длине последовательности $(\sigma^i, \lambda_i)_i$. Для операторов первых двух видов доказательство повторяет доказательство теоремы о структуризации.

Рассмотрим i-ый элемент последовательности: (σ^i, λ_i). Пусть $\lambda_i = \alpha_k$, а сам оператор имеет вид

$$\alpha_k \quad u = f(\bar{u}); \quad \beta_k.$$

Пусть δ — вызов, служащий, для вычисления $\sigma^i(f(\bar{u}))$.

В последовательности $(\sigma_1^j)_i$ следующей за $\sigma_1^{j_i}$ будет конфигурация $\sigma_1^{j_i+1}$ такая, что $\sigma_1^{j_i+1} y = \alpha_1$, $\sigma_1^{j_i+1} w = 0$,

$$\sigma_1^{j_i+1} s = \nu(\nu^{j_i}(\nu^{j_i}(s, \bar{z}), \bar{x}), \beta_k),$$

и $\sigma_1^{j_i+1} z = \sigma_\delta z$ для остальных переменных z.

Здесь мы обозначаем

$$\nu^l(s, \bar{w}) = \nu(\nu(\dots \nu(\sigma_1^l s, \sigma_1^l w_1) \dots), \sigma_1^l w_N).$$

Заметим, что $\sigma_1^{j_i+1} s > \sigma_1^{j_i} s \geqslant 0$.

Высота леса $\mathrm{Ff}(\sigma_\delta, \Pi_f)$ не больше h. Так как вызов δ определён, то последовательность конфигураций для $\Pi_f(\sigma_\delta)$ конечна, и заканчивается она конфигурацией вида (σ', γ). По индукционному предположению существует такое j', что $\sigma_1^{j'} s = \sigma_1^{j_i+1} s$, $\sigma_1^{j'} y = \gamma$, $\sigma_1^{j'} w = 0$, $\sigma_1^{j'} R = \sigma' f$, и $\sigma_1^{j'} z = \sigma' z$ для остальных z. При следующем

выполнении тела цикла все выражения $y == \widehat{\alpha_k}$ в операторах $\varphi(o_k)$ будут нулевыми. Следовательно, w и s в φ_1 будут ненулевыми. Пусть $j_{i+1} = j' + 1$. Тогда $\sigma_1^{j_{i+1}} z = \sigma_1^{j_i} z$ для $z \neq u$,

$$\sigma_1^{j_{i+1}} u = \sigma_1^{j'} R = \sigma' f,$$

$\sigma_1^{j_{i+1}} s = \sigma_1^{j_i} s$, $\sigma_1^{j_{i+1}} y = \beta_k = \lambda_{i+1}$, $\sigma_1^{j_{i+1}} w = 0$. Таким образом, $\sigma_1^{j_{i+1}}$ соответствует $(\sigma^{i+1}, \lambda_{i+1})$. Утверждение доказано.

Последовательность $(\sigma^i, \lambda_i)_i$ оканчивается конфигурацией (σ^n, γ). Согласно утверждению существует j_n такое, что $\sigma_1^{j_n} s = \sigma_1^0 s = 0$, $\sigma_1^{j_n} y = \gamma$, $\sigma_1^{j_n} w = 0$, $\sigma_1^{j_n} R = \sigma^n f$. При следующем выполнении тела цикла, мы получим $\sigma_1^{j_n+1} w = 1$, а значение остальных переменных не изменится. Следовательно,

$$\Pi_{f_1}(\sigma)(f_1) = \sigma_1^{j_n} R = \sigma^n f = \Pi_f(f).$$

Теперь докажем обратное, пусть $\Pi_{f_1}(\sigma_1)$ определено.

Утверждение 6.21. *Если $k < l$,*

$$\sigma_1^k s = \sigma_1^l s, \quad \sigma_1^j s \geqslant \sigma_1^k s \ \text{ для } j \in [k, l],$$

то в последовательности конфигураций для $\Pi_f(\sigma^0, \lambda_0)$ где $\lambda_0 = \sigma_1^k y$, $\sigma^0 f = \sigma_1^k R$ и $\sigma^0 z = \sigma_1^k z$ для остальных переменных z, имеется элемент $(\sigma^{l'}, \lambda_{l'})$ такой, что

1) $\sigma_1^l y = \lambda_{l'}$,

2) $\sigma_1^l R = \sigma^{l'} f$,

3) $\sigma_1^l z = l' z$ *для остальных переменных* z.

Индукция по количеству $j \in [k, l]$ таких, что $\sigma_1^j s > \sigma_1^k s$.

Б а з и с и н д у к ц и и . Если таких j нет, то доказательство повторяет доказательство теоремы о структуризации.

Ш а г и н д у к ц и и . Пусть для меньших количеств доказано. Рассмотрим самое первое j, для которого $\sigma_1^{j-1}s < \sigma_1^j s$. Рассмотрим наибольшее из таких p, что $\sigma_1^i s \geqslant \sigma_1^j s$ для всех $i \in [j,p]$.

Последовательности конфигураций $(\sigma^k,\ldots,\sigma^{j-1})$, $(\sigma^j,\ldots,\sigma^p)$ и $(\sigma^{p+1},\ldots,\sigma^l)$ удовлетворяют условиям индукционного предположения. Применим его к этим трём случаям.

В последовательности конфигураций для $\Pi_f(\sigma^0,\lambda_0)$ где $\lambda_0 = \sigma_1^k y$, $\sigma^0 f = \sigma_1^k R$ и $\sigma^0 z = \sigma_1^k z$ для остальных переменных z, имеется элемент $(\sigma^{j'},\lambda_{j'})$ такой, что

1) $\sigma_1^{j-1}y = \lambda_{j'}$,

2) $\sigma_1^{j-1}R = \sigma^{j'}f$,

3) $\sigma_1^{j-1}z = j'z$ для остальных переменных z.

Так как $\sigma_1^{j-1}s < \sigma_1^j s$, то $\sigma_1^{j-1}y = \lambda_{j'}$ и в программе Π_f имеется оператор вида

$$\lambda_{j'}\ u = f(\bar{u});\ \lambda_{j'+1}.$$

Для вычисления $\sigma^{j'}(f(\bar{u}))$ используется вызов

$$\delta = (f, \sigma^{j'}u_1,\ldots,\sigma^{j'}u_n).$$

Заметим, что $\sigma_\delta f = \sigma_1^j R$, $\sigma_\delta z = \sigma_1^j z$ для остальных переменных z программы Π_f.

При переходе от p к $p+1$ значение переменной s уменьшилось, что могло произойти, только при выполнении фрагмента φ_1. Это означает, что результаты всех сравнений $y == \widehat{\alpha}$ будут равны нулю на σ^p. Это означает, что $\sigma^p y = \gamma$ — заключительная метка.

Таким образом, для $\Pi_f(\sigma^0,\lambda_0)$, где $\lambda_0 = \sigma_1^j y = \alpha_1$, $\sigma^0 f = \sigma_1^j R = 0$ и $\sigma^0 z = \sigma_1^j z$ для остальных переменных z, в последовательности конфигураций имеется элемент $(\sigma^{p'},\lambda_{p'})$ такой, что

1) $\gamma = \lambda_{p'}$,

2) $\sigma_1^p R = \sigma^{p'} f$.

Из этого следует, что

$$\operatorname{Res}\delta = \sigma_1^p R.$$

Заметим, что σ_1^{p+1} совпадает с σ_1^{j-1} кроме того, что

$$\sigma_1^{p+1} u = \sigma_p^1 R = \operatorname{Res}\delta = \sigma^{j'}(f(\bar{u})).$$

Это означает, что σ_1^{p+1} соответствует $(\sigma^{j'+1}, \lambda_{j'+1})$ в последовательности конфигураций.

В последовательности конфигураций для $\Pi_f(\sigma^{j'+1}, \lambda_{j'+1})$ где $\lambda_{j'+1} = \sigma_1^{p+1} y$, $\sigma^{j'+1} f = \sigma_1^{p+1} R$ и $\sigma^{j'+1} z = \sigma_1^{p+1} z$ для остальных переменных z, имеется элемент $(\sigma^{l'}, \lambda_{l'})$ такой, что

1) $\sigma_1^l y = \lambda_{l'}$,

2) $\sigma_1^l R = \sigma^{l'} f$,

3) $\sigma_1^l z = l' z$ для остальных переменных z.

Так как $\Pi_{f_1}(\sigma_1)$ определено, то последовательность $(\sigma_1^i)_i$ конечна. То есть для некоторого N выполняется

$$\sigma^n \not\models w = 0.$$

Из этого следует, что $\sigma^{n-1} y = \gamma$ — заключительная метка и $\sigma^{n-1} s = 0$. Согласно утверждению, в последовательности конфигураций для $\Pi_f(\sigma^0, \lambda_0)$, где $\lambda_0 = \alpha_1$, $\sigma^0 f = 0$ и $\sigma^0 z = \sigma_\delta z$ для остальных переменных z, имеется элемент (σ^l, λ_l) такой, что $\lambda_l = \gamma$. То есть $\Pi_f(\sigma)$ определено. □

Задача 6.22. Пользуясь предложенным методом удалите рекурсию из программы из примера 6.19 на стр. 249.

Приведённый способ, конечно, не является оптимальным для практического применения. Обычно разумным решением для задачи удаления рекурсии будет заново написать программу без ее применения.

```
function f;
arguments x;
returns cond(x mod 2 == 0,
    apply(f, x / 2),
    cond(x == 1,
        2,
        apply(f, x + 2)));;
end;
```

Рис. 6.16: Функциональная программа к примеру 6.20

Пример 6.20. *Определить, какую функцию вычисляет подпрограмма на рис. 6.16. Доказать по определению семантики. Написать структурированную программу, вычисляющую ту же самую функцию.*

Подпрограмма f вычисляет следующую функцию g.

$$g(n) = \begin{cases} 2, & \text{если } n \text{ является степенью двойки;} \\ \text{не определено} & \text{в противном случае.} \end{cases}$$

Любое натуральное число n единственным образом представляется в виде $n = 2^a(2b+1)$, где a и b — натуральные числа. Индукцией по a докажем, что $\mathrm{Res}(f, 2^a(2b+1)) = \mathrm{Res}(f, 2b+1)$.

Базис индукции. Если $a = 0$, то получаем просто тождественное равенство $\mathrm{Res}(f, 2b+1) = \mathrm{Res}(f, 2b+1)$.

Шаг индукции. Пусть $a > 0$, σ — начальная конфигурация. Тогда число $2^a(2b+1)$ чётно, поэтому $\sigma(x \bmod 2 == 0) = 1$. По определению имеем

$$\mathrm{Res}(f, 2^a(2b+1)) = \sigma(\mathsf{apply}(f, x \,/\, 2)) = \mathrm{Res}(f, 2^{a-1}(2b+1)).$$

Так как $a - 1 < a$, то по индукционному предположению $\mathrm{Res}(f, 2^{a-1}(2b+1)) = \mathrm{Res}(f, 2b+1)$. *Значит,* $\mathrm{Res}(f, 2^a(2b+1)) = \mathrm{Res}(f, 2b+1)$.

Итак, остаётся определить, что делает программа с нечётными числами вида $2b+1$. Если $b = 0$, то результатом будет 2. Если $b > 0$, то по

```
function f;
arguments x;
returns y;
    while x > 0 do
        if x mod 2 == 0 then
            x = x / 2;
        else
            if x == 1 then
                y = 2; x = 0;
            end;
        end;
    end;
end;
```

Рис. 6.17: Структурированная программа к примеру 6.20

определению семантики получаем

$$\mathrm{Res}(f, 2b+1) = \sigma(\mathsf{apply}(f, x+2)) = \mathrm{Res}(f, 2b+3).$$

Число $2b+3$ *также нечётно, поэтому*

$$\mathrm{Res}(f, 2b+3) = \mathrm{Res}(f, 2b+5) = \ldots$$

Все числа в получившейся последовательности нечётны, значит, последовательность бесконечна. По определению семантики $\mathrm{Res}(f, 2b+1)$ *не определено.*

Случай $\mathrm{Res}(f, 0)$ *аналогичен случаю* $\mathrm{Res}(f, 2b+1)$. *Так как 0 — чётное число и* $0/2 = 0$, *то получится бесконечная последовательность вызовов* $(f, 0)$.

Эквивалентная структурированная программа показана на рис. 6.17.

Задача 6.23. Определить, какие функции вычисляют подпрограммы на рис. 6.18 на следующей странице. Доказать это по определению семантики.

```
function f;
arguments x, y;
returns cond(x < y,
    apply(f, x, y − 3) + 5,
    x + y);;
end;
```

```
function f;
arguments x, y;
returns cond(x ⩾ y,
    cond(x == y,
        0,
        apply(f, 0, 1)),
    apply(f, x, dec(y)));;
end;
```

Рис. 6.18: Функциональные программы к задаче 6.23

Построить структурированные программы, вычисляющие те же функции. Функция dec определена в задаче 3.40 на стр. 104.

*__Задача 6.24.__ Доказать, что если функциональная программа не содержит рекурсии, то она эквивалентна программе без циклов, следований и подпрограмм.

Указание. Использовать индукцию по количеству функций.

§ 6.5. *Корректность подпрограмм

Очевидно, что для доказательства корректности алгоритмов с подпрограммами правил вывода, которые мы изучали раньше, уже недостаточно. В самом деле, если выражение e содержит apply, то мы не можем использовать аксиому

$$\{(\varphi)^x_e\}x = e;\{\varphi\},$$

потому что мы не определяли, что означает apply внутри условия. Истинность условия на состоянии должна определяться самим условием и состоянием, но не программой, для которой это условие написано.

Первое, что мы сформулируем, это правила истинности формул с использованием apply.

ЗАМ Замена. Пусть тройки $\{\varphi\}\, x = e; \{\theta\}$ и $\{\theta\}\, y = d; \{\psi\}$ частично (полностью) корректны, переменные x и y различны и формула ψ не содержит x. Тогда тройка $\{\varphi\}\, y = (d)^x_e; \{\psi\}$ также частично (полностью) корректна.

Докажем непротиворечивость.

Доказательство. По правилу **СЛД** получаем корректность $\{\varphi\}\, x = e; y = d; \{\psi\}$. Далее достаточно отметить, что при применении программ $x = e; y = d;$ и $y = (d)^x_e;$ к одной и той же конфигурации σ или обе они зациклятся, или мы получим конфигурации τ' и τ, отличающиеся только значением переменной x. Но так как x в ψ не входит, то формула ψ будет истинной или ложной на τ и τ' одновременно. □

Заметим, что требование различности переменных x и y несущественно. В случае их совпадения можно даже не требовать, чтобы ψ их не содержало.

ЗАМ$_1$ Замена. Пусть тройки $\{\varphi\}\, y = e; \{\theta\}$ и $\{\theta\}\, y = d; \{\psi\}$ частично (полностью) корректны. Тогда тройка $\{\varphi\}\, y = (d)^y_e; \{\psi\}$ также частично (полностью) корректна.

Доказательство. Выберем новую переменную u.

[1] $\{\varphi\}\, y = e; \{\theta\}$

[2] $\{\theta\}\, y = d; \{\psi\}$

[3] **АКС**. $\{\psi\}\, u = y; \left\{(\psi)^y_u\right\}$

[4] **ЗАМ** : [2], [3] $\{\theta\}\, u = d; \left\{(\psi)^y_u\right\}$

[5] **ЗАМ** : [1], [4] $\{\varphi\}\, u = (d)^y_e; \left\{(\psi)^y_u\right\}$

[6] **АКС**. $\left\{(\psi)^y_u\right\}\, y = u; \{\psi\}$

[7] **ЗАМ** : [5], [6] $\{\varphi\}\, y = (d)^y_e; \{\psi\}$

Здесь нужно только отметить, что поскольку u — новая переменная, то $((d)^y_u)^u_e \circ\!\!- (d)^y_e$ и $((\psi)^y_u)^u_y \circ\!\!- \psi$. □

Второе правило, которое мы сформулируем позволит переходить от тел подпрограмм к их вызовам.

РЕК Правило рекурсии. Пусть Π_f — тело подпрограммы f, r_f — выражение-результат, $\bar{x}$ — параметры f, ψ не содержит переменных из $\mathrm{LVar}(\Pi_f)$ (кроме, может быть, y), выражение L при выполнении φ принимает только натуральные значения и зависит только от $\bar{x}$, а переменная u отличается от y и $\bar{x}$. Тогда если

$$\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\} \underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge L = u\}\, \Pi_f y = r_f; \{\psi\}\,,$$

то тройка $\{\varphi\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$ полностью корректна.

Доказательство. Доказывать непротиворечивость этого правила будем индукцией по общему количеству применений правила **РЕК** при выводе тройки $\{\varphi\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$. Если это количество равно нулю, то правило **РЕК** вообще не применяется.

Предположим, что общее количество равно N, а для меньших количеств утверждение доказано. Поскольку вывод

$$\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\} \underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge L = u\}\, \Pi_f y = r_f; \{\psi\}\,,$$

содержит меньше N применений правила **РЕК**, то по индукционному предположению из полной корректности

$$\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$$

следует полная корректность

$$\{\varphi \wedge L = u\}\, \Pi_f y = r_f; \{\psi\}\,.$$

Заметим, что

$$\varphi \equiv \varphi \wedge \bigvee_{i=0}^{\infty} L = i \equiv \bigvee_{i=0}^{\infty} (\varphi \wedge L = i).$$

Поэтому для доказательства полной корректности тройки $\{\varphi\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$ достаточно доказать полную корректность всех троек вида $\{\varphi \wedge L = i\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$ для $i \in \omega$.

Базис индукции. Пусть $i = 0$. Поскольку L принимает только натуральные значения, то $\varphi \wedge L < 0 \equiv \bot$. Тройка $\{\bot\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$ полностью корректна, поэтому вполне корректной будет и тройка $\{\varphi \wedge L = 0\}\, \Pi_f y = r_f; \{\psi\}$.

Возьмем любую конфигурацию σ, для которой $\sigma \models \varphi \wedge L = 0$. Тогда $\rho = (\Pi_f y = r_f;)(\sigma)$ определено и $\rho \models \psi$. Следовательно, $\tau = (y = \mathsf{apply}(f, \bar{x});)(\sigma)$ тоже определено и может отличаться от ρ только значениями переменных из $\mathrm{LVar}(\Pi_f)$. Но такие переменные не входят в ψ, поэтому из $\rho \models \psi$ следует $\tau \models \psi$, что и требуется.

Шаг индукции. Пусть для всех значений меньших i утверждение доказано, $i > 0$. Тогда полностью корректны все тройки вида

$$\{\varphi \wedge L = j\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$$

для $j < i$, поэтому полностью корректна и тройка

$$\{\varphi \wedge L < i\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}\,.$$

Но тогда вполне корректна и тройка

$$\{\varphi \wedge L = i\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}\,.$$

Дальнейшие рассуждения как для базиса индукции. □

Отметим, что выводимость

$$\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\} \underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge L = u\}\, \Pi_f y = r_f; \{\psi\}$$

не подразумевает, что тройка $\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$ непременно должна появляться в этом выводе. То есть, если удастся вывести $\{\varphi \wedge L = u\}\, \Pi_f y = r_f; \{\psi\}$ без использования $\{\varphi \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi\}$, то это уже является достаточным для применения правила **РЕК**.

Формулировка правила **РЕК** делает невозможным доказательство допустимости правила **ДОБ**. В самом деле, переменная v может не встречаться в программе $y = \mathsf{apply}(f, \bar{x});$, но может встречаться в теле подпрограммы f, поэтому может случиться так, что нельзя будет вывести

$$\{\varphi \wedge \theta \wedge L < u\}\, y = \mathsf{apply}(f, \bar{x}); \{\psi \wedge \theta\} \underset{\mathfrak{IH}}{\rhd} \underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge \theta \wedge L = u\}\, \Pi_f y = r_f; \{\psi \wedge \theta\},$$

если v входит в θ.

С другой стороны, само по себе правило **ДОБ** не перестанет быть корректным, поэтому сформулируем его как самостоятельное правило.

ДОБ Если переменные формулы θ не входят в $\mathrm{LVar}(\Pi)$, то из частичной (полной) корректности $\{\varphi\}\Pi\{\psi\}$ следует частичная (полная) корректность $\{\varphi \wedge \theta\}\Pi\{\psi \wedge \theta\}$.

Доказательство. Пусть тройка $\{\varphi\}\Pi\{\psi\}$ частично корректна и $\sigma \models \varphi \wedge \theta$ и $\Pi(\sigma)$ определено: $\tau = \Pi(\sigma)$. Тогда из $\sigma \models \varphi \wedge \theta$, следует $\sigma \models \varphi$ и $\tau \models \psi$. С другой стороны переменные из θ не встречаются в $\mathrm{LVar}(\Pi)$, поэтому в конфигурации τ переменные из θ имеют то же значение, что и в σ, поэтому $\tau \models \theta$. Следовательно, $\tau \models \psi \wedge \theta$.

Если тройка $\{\varphi\}\Pi\{\psi\}$ является полностью корректной и $\sigma \models \varphi \wedge \theta$, то $\sigma \models \varphi$, поэтому $\Pi(\sigma)$ определено. Далее как в предыдущем случае. □

Используя правило **ДОБ** можно доказать допустимость еще нескольких правил.

ДОБ$_1$ Если выражение d не содержит apply, переменная y отличается от x и переменных выражения d, то из частичной (полной) корректности $\{\varphi\}\, y = e; \{\psi\}$ следует частичная (полная) корректность $\left\{(\varphi)^x_d\right\} y = (e)^x_d; \left\{(\psi)^x_d\right\}$.

Доказательство. Выберем новую переменную u

[1] $\{\varphi\}\, y = e; \{\psi\}$

[2] **ДОБ** : [1]. $\{\varphi \wedge x = u\}\, y = e; \{\psi \wedge x = u\}$.

[3] **ОСП** : [2]. Истинно $\psi \wedge x = u \to (\psi)^x_u$, поэтому

$$\{\varphi \wedge x = u\}\, y = e; \left\{(\psi)^x_u\right\}.$$

[4] **АКС**. $\left\{(\varphi)^x_d \wedge d = u\right\} x = d; \{\varphi \wedge x = u\}$

[5] **ЗАМ** : [4], [3]. $\left\{(\varphi)^x_d \wedge d = u\right\} x = d; y = e; \left\{(\psi)^x_u\right\}$.

[6] **ДОБ** : [5]. $\left\{(\varphi)^x_d \wedge d = u \wedge d = u\right\} y = (e)^x_d; \left\{(\psi)^x_u \wedge d = u\right\}$.

[7] **ОСП** : [6]. Истинно $(\psi)^x_u \wedge d = u \to (\psi)^x_d$, поэтому

$$\left\{(\varphi)^x_d \wedge d = u \wedge d = u\right\} y = (e)^x_d; \left\{(\psi)^x_d\right\}.$$

[8] **АКС**. $\left\{(\varphi)^x_d \wedge d = d \wedge d = d\right\} u = d; \left\{(\varphi)^x_d \wedge d = u \wedge d = u\right\}$.

```
function mcc_1;
arguments x, y, z;
returns r;
    if z mod x then
        r = apply(mcc_1, x, y, z + 1);
    else
        if z mod y then
            r = apply(mcc_1, x, y, z + 1);
        else
            r = z;
        end;
    end;
end;
```

Π_1

```
algorithm mcc;
arguments x, y;
returns apply(mcc_1, x, y, 1);;
end;
```

Рис. 6.19: Наименьшее общее кратное.

[9] **УПР** : [6]. Истинно $(\varphi)_d^x \to (\varphi)_d^x \wedge d = d \wedge d = d$, поэтому

$$\left\{(\varphi)_d^x\right\} u = d; \left\{(\varphi)_d^x \wedge d = u \wedge d = u\right\}.$$

[10] **ЗАМ** : [9], [7].

$$\left\{(\varphi)_d^x\right\} y = (e)_d^x; \left\{(\psi)_d^x\right\}. \qquad \square$$

Пример 6.21. *Рассмотрим пример — подпрограмму для вычисления наименьшего общего кратного, см. рис. 6.19. В качестве L возьмем $xy - z$, отметим, что* $\mathrm{HOK}(x, y) \leqslant xy$.

[1] *Предположим, что*

$$\left|\begin{array}{c} \{0 < z \leqslant \mathrm{HOK}(x, y) \wedge xy - z < u\} \\ m = \mathsf{apply}(mcc_1, x, y, z); \\ \{m = \mathrm{HOK}(x, y)\} \end{array}\right|$$

[2] **ДОБ**$_1$: [1].

$$\left|\begin{array}{c}\{0 < z+1 \leqslant \mathrm{HOK}(x,y) \wedge xy-(z+1) < u\}\\ r = \mathsf{apply}(mcc_1, x, y, z+1);\\ \{r = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[3] **АКС**. $\{z = \mathrm{HOK}(x,y)\}\, r = z; \{r = \mathrm{HOK}(x,y)\}$

[4] **УПР** : [3].

$$0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y = 0 \wedge xy - z = u \to \\ \to z = \mathrm{HOK}(x,y)$$

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y = 0 \wedge xy - z = u\}\\ r = z;\\ \{r = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[5] **УПР** : [2].

$$0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y \neq 0 \wedge xy - z = u \to \\ \to 0 < z+1 \leqslant \mathrm{HOK}(x,y) \wedge xy-(z+1) < u$$

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y \neq 0 \wedge xy - z = u\}\\ r = mcc_1(x,y,z+1);\\ \{r = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[6] **ПВТ** : [4], [5].

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge xy - z = u\}\\ \Pi_1\\ \{mcc_1 = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[7] **УПР** : [2].

$$0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x \neq 0 \wedge xy - z = u \to \\ \to 0 < z+1 \leqslant \mathrm{HOK}(x,y) \wedge xy-(z+1) < u$$

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge z \bmod x \neq 0 \wedge xy - z = u\}\\ r = mcc_1(x,y,z+1);\\ \{r = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[8] **ПВТ** : [7], [6].

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge xy - z = u\}\\ \Pi_{mcc_1}\\ \{r = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[9] **АКС**.

$$\left|\begin{array}{c}\{r = \mathrm{HOK}(x,y)\}\\ m = r;\\ \{m = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[10] **СЛД** : [8], [9].

$$\left|\begin{array}{c}\{0 < z \leqslant \mathrm{HOK}(x,y) \wedge xy - z = u\}\\ \Pi_{mcc_1} m = r;\\ \{m = \mathrm{HOK}(x,y)\}\end{array}\right|$$

[11] **РЕК** : [1] – [10].

$$\{0 < z \leqslant \mathrm{HOK}(x,y)\}\, m = \mathsf{apply}(mcc_1, x, y, z); \{m = \mathrm{HOK}(x,y)\}$$

[12] **ДОБ**$_1$: [11].

$$\{0 < 1 \leqslant \mathrm{HOK}(x,y)\}\, m = \mathsf{apply}(mcc_1, x, y, 1); \{m = \mathrm{HOK}(x,y)\}$$

[13] **УПР** : [12]. $x \geqslant 1 \wedge y \geqslant 1 \to 0 < 1 \leqslant \mathrm{HOK}(x,y)$

$$\{x \geqslant 1 \wedge y \geqslant 1\}\, m = \mathsf{apply}(mcc_1, x, y, 1); \{m = \mathrm{HOK}(x,y)\}$$

[14] **АКС**$_0$. $\{x \geqslant 1 \wedge y \geqslant 1\}\,; \{x \geqslant 1 \wedge y \geqslant 1\}$

[15] **СЛД** : [14], [13].

$$\{x \geqslant 1 \wedge y \geqslant 1\}\,; m = \mathsf{apply}(mcc_1, x, y, 1); \{m = \mathrm{HOK}(x,y)\}$$

[16] **УПР** : [15]. *Из истинности формулы*

$$x \geqslant 1 \wedge y \geqslant 1 \wedge L = u \to x \geqslant 1 \wedge y \geqslant 1$$

получаем

$$\{x \geqslant 1 \wedge y \geqslant 1 \wedge L = u\}\,; m = \mathsf{apply}(mcc_1, x, y, 1); \{m = \mathrm{HOK}(x,y)\}$$

```
function F;
arguments x;
returns r;
    if x then
        r = apply(F, x − 1) × x;
    else
        r = 1;
    end;
end;
```

Рис. 6.20: Факториал.

[17] **РЕК** : [1] – [16].

$$\{x \geqslant 1 \wedge y \geqslant 1\}\, m = \mathsf{apply}(mcc, x, y); \{m = \text{НОК}(x, y)\}$$

Пример 6.22. *Рассмотрим подпрограмму, вычисляющую факториал (рис. 6.20). В качестве L берем x.*

[1] *Предположим, что*

$$\{x < u\}\, y = \mathsf{apply}(F, x); \{y = x!\}$$

[2] **ДОБ**$_1$: [1]. $\{x - 1 < u\}\, y = \mathsf{apply}(F, x - 1); \{y = (x - 1)!\}$

[3] **АКС**. $\{yx = x!\}\, r = y \times x; \{r = x!\}$

[4] **УПР** : [3]. *Из истинности* $y = (x - 1)! \to yx = x!$ *выводим*

$$\{y = (x - 1)!\}\, r = y \times x; \{r = x!\}$$

[5] **ЗАМ** : [2], [4].

$$\{x - 1 < u\}\, r = \mathsf{apply}(F, x - 1) \times x; \{r = x!\}$$

[6] **УПР**. *Истинно* $x = u \wedge x \neq 0 \to x - 1 < u$, *поэтому*

$$\{x = u \wedge x \neq 0\}\, r = \mathsf{apply}(F, x - 1) \times x; \{r = x!\}$$

```
function mcd;
arguments x, y;
returns r;
    if x == y then
        r = x;
    else
        if x < y then                          }
            r = apply(mcd, x, y − x);          }
        else                                   } Π1
            r = apply(mcd, x − y, y);          }
        end;                                   }
    end;
end;
```

Рис. 6.21: Алгоритм Евклида.

[7] **АКС**. $\{1 = x!\}\, r = 1; \{r = x!\}$

[8] **УПР** : [7]. *Очевидно, истинно* $x = u \wedge x = 0 \to 1 = x!$, *поэтому*

$$\{x = u \wedge x = 0\}\, r = 1; \{r = x!\}$$

[9] **ПВТ** : [6], [8]. $\{x = u\}\, \Pi_F\, \{r = x!\}$

[10] **АКС**.

$$\{r = x!\}\, y = r; \{y = x!\}$$

[11] **СЛД** : [9], [10].

$$\{x = u\}\, \Pi_f y = r; \{y = x!\}$$

[12] **РЕК** : [1] – [11].

$$\{\top\}\, y = \mathsf{apply}(F, x); \{y = x!\}$$

Пример 6.23. *Рассмотрим алгоритм Евклида на рис. 6.21.*
В качестве L *берем* $x + y$.

[1] *Предположим, что*

$$\{x > 0 \wedge y > 0 \wedge x + y < u\} z = \mathsf{apply}(mcd, x, y); \{z = \text{НОД}(x, y)\}.$$

[2] **АКС**

$$\{z = \text{НОД}(x, y)\}\, r = z; \{r = \text{НОД}(x, y)\}\,.$$

[3] **ЗАМ** : [1], [2].

$$\{x > 0 \wedge y > 0 \wedge x + y < u\} r = \mathsf{apply}(mcd, x, y); \{r = \text{НОД}(x, y)\}.$$

[4] **ЗАМ** : [3].

$$\left|\begin{array}{c}\{x - y > 0 \wedge y > 0 \wedge (x - y) + y < u\}\\ r = \mathsf{apply}(mcd, x - y, y);\\ \{r = \text{НОД}(x - y, y)\}\end{array}\right|$$

[5] **ЗАМ** : [3].

$$\left|\begin{array}{c}\{x > 0 \wedge y - x > 0 \wedge x + (y - x) < u\}\\ r = \mathsf{apply}(mcd, x, y - x);\\ \{r = \text{НОД}(x, y - x)\}\end{array}\right|.$$

[6] **ОСП** : [4]. *Истинно* $r = \text{НОД}(x - y, y) \to r = \text{НОД}(x, y)$

$$\left|\begin{array}{c}\{x - y > 0 \wedge y > 0 \wedge (x - y) + y < u\}\\ r = \mathsf{apply}(mcd, x - y, y);\\ \{r = \text{НОД}(x, y)\}\end{array}\right|$$

[7] **УПР** : [6]. *Истинно*

$$x > 0 \wedge y > 0 \wedge x + y = u \wedge x \neq y \wedge x \geqslant y \to \\ \to x - y > 0 \wedge y > 0 \wedge (x - y) + y < u$$

$$\left|\begin{array}{c}\{x > 0 \wedge y > 0 \wedge x + y = u \wedge x \neq y \wedge x \geqslant y\}\\ r = \mathsf{apply}(mcd, x - y, y);\\ \{r = \text{НОД}(x, y)\}\end{array}\right|$$

[8] **ОСП** : [5]. *Истинно*

$$r = \text{НОД}(x, y - x) \to r = \text{НОД}(x, y)$$

$$\left|\begin{array}{c}\{x > 0 \wedge y - x > 0 \wedge x + (y - x) < u\} \\ r = \mathsf{apply}(mcd, x, y - x); \\ \{r = \text{НОД}(x, y)\}\end{array}\right|$$

[9] **УПР** : [8]. *Истинно*

$$x > 0 \wedge y > 0 \wedge x + y = u \wedge x \neq y \wedge x < y \to$$
$$\to x > 0 \wedge y - x > 0 \wedge x + (y - x) < u$$

$$\left|\begin{array}{c}\{x > 0 \wedge y > 0 \wedge x + y = u \wedge x \neq y \wedge x < y\} \\ r = \mathsf{apply}(mcd, x, y - x); \\ \{r = \text{НОД}(x, y)\}\end{array}\right|$$

[10] **ПВТ** : [7], [9].

$$\left|\begin{array}{c}\{x > 0 \wedge y > 0 \wedge x + y = u \wedge x \neq y\} \\ \Pi_1 \\ \{r = \text{НОД}(x, y)\}\end{array}\right|$$

[11] **АКС**. $\{x = \text{НОД}(x, y)\}\, r = x; \{r = \text{НОД}(x, y)\}$

[12] **УПР** : [11]. *Истинно*

$$x > 0 \wedge y > 0 \wedge x + y = u \wedge x = y \to x = \text{НОД}(x, y)$$

$$\{x > 0 \wedge y > 0 \wedge x + y = u \wedge x = y\}\, r = x; \{r = \text{НОД}(x, y)\}$$

[13] **ПВТ** : [12], [10].

$$\{x > 0 \wedge y > 0 \wedge x + y = u\}\, \Pi_{mcd}\, \{r = \text{НОД}(x, y)\}$$

[14] **АКС**.

$$\{r = \text{НОД}(x, y)\}\, z = r; \{z = \text{НОД}(x, y)\}$$

[15] **СЛД** : [13], [14].

$$\{x > 0 \wedge y > 0 \wedge x + y = u\}\, \Pi_{mcd} z = r; \{z = \text{НОД}(x, y)\}$$

[16] **РЕК** : [1] – [15].

$$\{x > 0 \wedge y > 0\}\, z = \mathsf{apply}(mcd, x, y); \{z = \text{НОД}(x, y)\}$$

Глава 7

Вычислительная сложность

§ 7.1. Хранение чисел

До сих пор мы говорили о том, что та или иная задача разрешима, но не говорили, например, сколько времени или сколько памяти требуется для её решения.

Чтобы точно описать, что такое время вычисления или требуемая память, введём следующее понятие. Для хранения натуральных чисел нужна некоторая память. Вопрос в том, сколько памяти требуется для хранения некоторого натурального числа. Пусть функция $\mathrm{S} : \omega \to \omega$ описывает о б ъ ё м п а м я т и для хранения каждого из натуральных чисел. Определим, какими свойствами должна обладать эта функция.

1) Для хранения каждого натурального числа требуется некоторая память. Следовательно $\mathrm{S}(x) \geqslant 1$ для каждого $x \in \omega$.

2) Естественно предположить, что в конечном объеме памяти можно хранить конечное число различных натуральных чисел. Таким образом, для всякого $\alpha \in \omega$ множество $\{x \in \omega : \mathrm{S}(x) \leqslant \alpha\}$ является конечным.

3) Мы хотим, чтобы функция S была монотонной, то есть чем больше число, тем больше места требуется для его хранения. Это значит, что если $x > y$, то $\mathrm{S}(x) \geqslant \mathrm{S}(y)$.

4) Мы видели, что существуют двухместные функции, которые однозначно нумеруют пары натуральных чисел, например, $\nu(x, y) = (2x + 1)2^y$. Эта функция является монотонной по обоим аргументам. Легко придумать и другие примеры. Таким образом, можно считать, что число $\nu(x, y)$ содержит информацию и об x, и об y одновременно. Потребуем, чтобы для всякой такой функции f существовала константа $c \in \omega$, для которой было выполнено

$$\mathrm{S}(f(x, y)) \geqslant \mathrm{S}(x) + \mathrm{S}(y) - c$$

для любых x и y.

Рассмотрим пример. Пусть $\mathrm{S}(x) = [\log_k x] + 1$, $[x]$ — целая часть x. Эта функция описывает длину числа в его k-ичном представлении. Очевидно, что S положительна, монотонна, и множество $\{x \in \omega : [\log_2 x] + 1 \leqslant \alpha\}$ конечно для каждого $\alpha \in \omega$. Чтобы доказать последнее свойство, заметим, что для любой монотонной разнозначной нумерующей функции f выполнено $f(x, y) \geqslant xy+x+y$. Это следует из того, что для всяких x' и y' таких, что $x' \leqslant x$ и $y' \leqslant y$, должно быть $f(x', y') \leqslant f(x, y)$. Всего пар (x', y'), удовлетворяющих этому условию, не меньше чем $(x + 1)(y + 1)$. Следовательно, $f(x, y)$ должно быть не меньше чем каждое из первых $(x + 1)(y + 1)$ натуральных чисел. Это означает, что

$$f(x, y) \geqslant (x + 1)(y + 1) - 1 = xy + x + y.$$

Итак,

$$\begin{aligned}\mathrm{S}(f(x,y)) &= [\log_k f(x,y)] + 1 \geqslant [\log_k(xy + x + y)] + 1 \geqslant \\ &\geqslant [\log_k xy] + 1 \geqslant [\log_k x] + [\log_k y] + 1 = \\ &= ([\log_k x] + 1) + ([\log_k y] + 1) - 1 = \mathrm{S}(x) + \mathrm{S}(y) - 1.\end{aligned}$$

Мы можем взять $c = 1$.

Теперь мы покажем, что такой объём является наименьшим возможным при соблюдении указанных выше четырёх условий.

Лемма 7.1 (О минимальном объёме)**.** *Для всякой функции* S, *удовлетворяющей условиям 1–4, существует константа* $k \in \omega$ *такая, что* $\mathrm{S}(x) \geqslant [\log_k x] + 1$ *для всех* $x \in \omega$.

Доказательство. Пусть

$$f(x,y) = \frac{(x+y)(x+y+1)}{2} + x$$

Эта функция однозначно нумерует пары чисел, следовательно, существует константа c для которой:

$$\mathrm{S}(f(x,y)) \geqslant \mathrm{S}(x) + \mathrm{S}(y) - c$$

для любых $x, y \in \omega$. Пусть $x = y$. Тогда

$$f(x,x) = \frac{2x(2x+1)}{2} + x = 2x^2 + 2x.$$

Итак,

$$\mathrm{S}(2x^2 + 2x) \geqslant 2\mathrm{S}(x) - c.$$

Выберем k настолько большим, чтобы

$$\mathrm{S}(k) \geqslant c + 6, \quad k^3 \geqslant 3k^2 \quad \text{и} \quad k^2 \geqslant 2k.$$

Следовательно, $\mathrm{S}(k) \geqslant \log_k k + c + 5$.

Если $k \leqslant x < 3k^2$, то

$$\mathrm{S}(x) \geqslant \log_k k + c + 5 \geqslant \log_k k + \log_k k^2 + c + 3 = \\ = \log_k k^3 + c + 3 \geqslant \log_k x + c + 3.$$

Пусть для $x \leqslant 3k^{2n}$ доказано, что

$$\mathrm{S}(x) \geqslant \log_k x + c + 3.$$

Отметим, что

$$\mathrm{S}(3k^{2n}) \geqslant 2\mathrm{S}(k^n) - c \geqslant 2(\log_k k^n + c + 3) - c = 2n + c + 6$$

Если $3k^{2n} \leqslant x < 3k^{2n+2}$, то

$$\mathrm{S}(x) \geqslant 2n + c + 6 = \log_k k^{2n+3} + c + 3 \geqslant \log_k x + c + 3.$$

Итак, для всех $x \geqslant k$ имеет место неравенство

$$\mathrm{S}(x) \geqslant \log_k x + c + 3 \geqslant [\log_k x] + 1.$$

Для $x < k$ имеем:

$$\mathrm{S}(x) \geqslant 1 = [\log_k x] + 1. \qquad \square$$

***Задача 7.1.** Докажите, что функция

$$f(x, y) = \frac{(x + y)(x + y + 1)}{2} + x$$

монотонно нумерует пары чисел.

***Задача 7.2.** Приведите пример записи чисел, для которой количество цифр, требуемых для записи, изменяется немонотонно.

§ 7.2. *Вычисления

Теперь приступим к описанию того, что такое время и память вычисления. Прежде всего, определим, что такое вычисление структурированной программы на конфигурации σ. Будем называть вычислением последовательность конфигураций. Вычисление на конфигурации σ будем обозначать $\mathrm{Cnt}(a, \sigma)$, где a — выражение или

программа. Вычисление определено, только если определено $\sigma(a)$ для выражений или $a(\sigma)$ для программ.

Определение 7.1 (Вычисление выражения)**.** *Определим, что такое вычисление выражения e, по индукции. Далее везде мы будем предполагать, что для любого выражения d переменная u_d является уникальной и не принадлежит* $\operatorname{dom}\sigma$.

1) $e \Rightarrow x$, *где* x — *переменная. Тогда* $\mathrm{Cnt}(e,\sigma) = \tau$, *где* $\tau = \sigma \cup \{(u_e, \sigma x)\}$.

2) $e \Rightarrow c$, *где* c — *константа.* $\mathrm{Cnt}(e,\sigma) = \tau$, *где* $\tau = \sigma \cup \{(u_e, c)\}$.

3) $e \Rightarrow o(e_1, \ldots, e_n)$, *где* o — *некоторая операция, а* e_1, …, e_n — *выражения. Пусть вычисление* $\mathrm{Cnt}(e_1,\sigma)$ *завершается конфигурацией* σ^1, *вычисление* $\mathrm{Cnt}(e_2,\sigma^1)$ *завершается конфигурацией* σ^2, *и т. д., вычисление* $\mathrm{Cnt}(e_n,\sigma^{n-1})$ *завершается конфигурацией* σ^n. *Тогда вычисление* $\mathrm{Cnt}(e,\sigma)$ — *это следующая последовательность:*

$$\mathrm{Cnt}(e_1,\sigma), \mathrm{Cnt}(e_2,\sigma^1), \ldots, \mathrm{Cnt}(e_n,\sigma^{n-1}), \tau,$$

где $\tau = \sigma \cup \{(u_e, o(\sigma^n u_{e_1}, \ldots, \sigma^n u_{e_n}))\}$.

4) $e \Rightarrow \mathsf{apply}(f, e_1, \ldots, e_n)$, *где* f — *подпрограмма. Пусть вычисление* $\mathrm{Cnt}(e_1,\sigma)$ *завершается конфигурацией* σ^1, …, *вычисление* $\mathrm{Cnt}(e_n,\sigma^{n-1})$ *завершается конфигурацией* σ^n. *Пусть*

$$\delta = (f, \sigma^n u_{e_1}, \ldots, \sigma^n u_{e_n}).$$

Пусть $\hat{\sigma}$ — *состояние, которое получается из* σ *заменой всех имен переменных вида* v *на* $\hat{v}$, *которые не встречаются в* Π_f. *Пусть* ρ — *последняя конфигурация в* $\mathrm{Cnt}(\Pi_f, \sigma_\delta \cup \hat{\sigma})$. *Тогда*

вычисление $\mathrm{Cnt}(e, \sigma)$ *— это следующая последовательность:*

$$\mathrm{Cnt}(e_1, \sigma), \mathrm{Cnt}(e_2, \sigma^1), \ldots, \mathrm{Cnt}(e_n, \sigma^{n-1}),$$
$$\sigma_\delta \cup \hat{\sigma}, \mathrm{Cnt}(\Pi_f, \sigma_\delta \cup \hat{\sigma}), \mathrm{Cnt}(r_f, \rho), \tau,$$

где r_f *— выражение-результат подпрограммы* f, $\tau = \sigma \cup \{(u_e, \rho(r_f))\}$.

Определение 7.2 (Вычисление программы)**.** *Теперь определим, что такое вычисление программы* Π *на конфигурации* σ.

1) $\Pi \twoheadleftarrow$; *(пустой оператор). Тогда* $\mathrm{Cnt}(\Pi, \sigma) = \sigma$.

2) $\Pi \twoheadleftarrow x = e$; *где* x *— переменная,* e *— выражение. Пусть* $\mathrm{Cnt}(e, \sigma)$ *заканчивается* ρ. *Тогда*

$$\mathrm{Cnt}(\Pi, \sigma) = \mathrm{Cnt}(e, \sigma), \tau,$$

где

$$\tau = (\sigma \setminus \{(x, \sigma x)\}) \cup \{(x, \rho u_e)\}.$$

3) $\Pi \twoheadleftarrow \Pi_1 \Pi_2$. *Тогда, если* $\mathrm{Cnt}(\Pi_1, \sigma)$ *заканчивается* ρ, *то*

$$\mathrm{Cnt}(\Pi, \sigma) = \mathrm{Cnt}(\Pi_1, \sigma), \mathrm{Cnt}(\Pi_2, \rho).$$

4)
$$\Pi \twoheadleftarrow \left\{ \begin{array}{l} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array} \right.$$

Пусть $\mathrm{Cnt}(e, \sigma)$ *заканчивается* ρ. *Тогда*

$$\mathrm{Cnt}(\Pi, \sigma) = \left\{ \begin{array}{ll} \mathrm{Cnt}(e, \sigma), \mathrm{Cnt}(\Pi_1, \sigma), & \text{если } \rho u_e \neq 0; \\ \mathrm{Cnt}(e, \sigma), \sigma, & \text{иначе.} \end{array} \right.$$

5)

$$\Pi \Leftarrow \begin{cases} \textsf{if } e \textsf{ then} \\ \quad \Pi_1 \\ \textsf{else} \\ \quad \Pi_2 \\ \textsf{end;} \end{cases}$$

Пусть $\mathrm{Cnt}(e, \sigma)$ *заканчивается* ρ. *Тогда*

$$\mathrm{Cnt}(\Pi, \sigma) = \begin{cases} \mathrm{Cnt}(e, \sigma), \mathrm{Cnt}(\Pi_1, \sigma), & \textit{если } \rho u_e \neq 0; \\ \mathrm{Cnt}(e, \sigma), \mathrm{Cnt}(\Pi_2, \sigma), & \textit{иначе.} \end{cases}$$

6)

$$\Pi \Leftarrow \begin{cases} \textsf{while } e \textsf{ do} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

Пусть $(\sigma^i)_{i=0}^n$ *последовательность конфигураций такая, что* $\sigma^0 = \sigma$, $\sigma^i(e) \neq 0$ *тогда и только тогда, когда* $i < n$, $\sigma^{i+1} = \Pi_1(\sigma^i), i < n$. *Тогда вычислением* Π *на* σ *будет*

$$\mathrm{Cnt}(e, \sigma^0), \mathrm{Cnt}(\Pi_1, \sigma^0), \ldots \\ \ldots, \mathrm{Cnt}(e, \sigma^{n-1}), \mathrm{Cnt}(\Pi_1, \sigma^{n-1}), \mathrm{Cnt}(e, \sigma^n), \sigma^n.$$

Рассмотрим примеры.

Пример 7.1. *Возьмём программу:*

$$\Pi \Leftarrow \begin{cases} \textsf{if } x < y \textsf{ then} \\ \quad z = y; \\ \textsf{else} \\ \quad z = x; \\ \textsf{end;} \end{cases}$$

и конфигурацию $\sigma = \{(x, 1), (y, 2), (z, 0)\}$.

1) $\mathrm{Cnt}(x, \sigma) = \{(x, 1), (y, 2), (z, 0), (u_x, 1)\} = \sigma^1$.

2) $\mathrm{Cnt}(y, \sigma^1) = \{(x, 1), (y, 2), (z, 0), (u_x, 1), (u_y, 2)\} = \sigma^2$.

3) $\mathrm{Cnt}(x < y, \sigma) = \sigma^1, \sigma^2, \sigma^3$, *где*

$$\sigma^3 = \{(x,1),(y,2),(z,0),(u_{x<y},1)\}.$$

4) $\mathrm{Cnt}(y, \sigma) = \{(x,1),(y,2),(z,0),(u_y,2)\} = \sigma^4$.

5) $\mathrm{Cnt}(z = y, \sigma) = \sigma^4, \sigma^5$, *где*

$$\sigma^5 = \{(x,1),(y,2),(z,2)\}.$$

6) $\mathrm{Cnt}(\Pi, \sigma) = \sigma^1, \sigma^2, \sigma^3, \sigma^4, \sigma^5$.

Пример 7.2. *Рассмотрим другую программу:*

$$\left.\begin{array}{l} \left.\begin{array}{l} u = 0;\ v = y; \\ \mathsf{while}\ v < x\ \mathsf{do} \\ \quad \left.\begin{array}{l} u = \mathsf{succ}(u); \\ v = \mathsf{succ}(v); \end{array}\right\} \Pi_2 \\ \mathsf{end}; \end{array}\right\} \Pi_1 \end{array}\right\} \Pi$$

и конфигурацию $\sigma = \{(x,3),(y,2),(u,0),(v,0)\}$. *В дальнейшем* x *и* y *меняться не будут, поэтому мы будем опускать их.*

1) $\mathrm{Cnt}(0, \sigma) = \{(u,0),(v,0),(u_0,0)\} = \sigma^1$.

2) $\mathrm{Cnt}(u = 0;, \sigma) = \sigma^1, \sigma^2$, *где*

$$\sigma^2 = \{(u,0),(v,0)\}.$$

3) $\mathrm{Cnt}(y, \sigma^2) = \{(u,0),(v,0),(u_y,2)\} = \sigma^3$.

4) $\mathrm{Cnt}(v = y;, \sigma^2) = \sigma^3, \sigma^4$, *где*

$$\sigma^4 = \{(u,0),(v,2)\}.$$

5) $\mathrm{Cnt}(\Pi_3, \sigma) = \sigma^1, \sigma^2, \sigma^3, \sigma^4$.

6) $\mathrm{Cnt}(v, \sigma^4) = \{(u,0),(v,2),(u_v,2)\} = \sigma^5$.

7) $\mathrm{Cnt}(x, \sigma^5) = \{(u,0),(v,2),(u_v,2),(u_x,3)\} = \sigma^6$.

8) $\mathrm{Cnt}(v < x, \sigma^4) = \sigma^5, \sigma^6, \sigma^7$, *где*

$$\sigma^7 = \{(u,0),(v,2),(u_{v<x},1)\}.$$

9) $\mathrm{Cnt}(u, \sigma^4) = \{(u, 0), (v, 2), (u_u, 0)\} = \sigma^8$.

10) $\mathrm{Cnt}(\mathsf{succ}(u), \sigma^4) = \sigma^8, \sigma^9$, *где*

$$\sigma^9 = \{(u, 0), (v, 2), (u_{\mathsf{succ}(u)}, 1)\}.$$

11) $\mathrm{Cnt}(u = \mathsf{succ}(u);, \sigma^4) = \sigma^8, \sigma^9, \sigma^{10}$, *где*

$$\sigma^{10} = \{(u, 1), (v, 2)\}.$$

12) $\mathrm{Cnt}(v, \sigma^{10}) = \{(u, 1), (v, 2), (u_v, 2)\} = \sigma^{11}$.

13) $\mathrm{Cnt}(\mathsf{succ}(v), \sigma^{10}) = \sigma^{11}, \sigma^{12}$, *где*

$$\sigma^{12} = \{(u, 1), (v, 2), (u_{\mathsf{succ}(v)}, 3)\}.$$

14) $\mathrm{Cnt}(v = \mathsf{succ}(v);, \sigma^{10}) = \sigma^{11}, \sigma^{12}, \sigma^{13}$, *где*

$$\sigma^{13} = \{(u, 1), (v, 3)\}.$$

15) $\mathrm{Cnt}(\Pi_2, \sigma^4) = \sigma^8, \sigma^9, \sigma^{10}, \sigma^{11}, \sigma^{12}, \sigma^{13}$.

16) $\mathrm{Cnt}(v, \sigma^{13}) = \{(u, 1), (v, 3), (u_v, 3)\} = \sigma^{14}$.

17) $\mathrm{Cnt}(x, \sigma^{14}) = \{(u, 1), (v, 3), (u_v, 3), (u_x, 3)\} = \sigma^{15}$.

18) $\mathrm{Cnt}(v < x, \sigma^{13}) = \sigma^{14}, \sigma^{15}, \sigma^{16}$, *где*

$$\sigma^{16} = \{(u, 1), (v, 3), (u_{v<x}, 0)\}.$$

19) $\mathrm{Cnt}(\Pi_1, \sigma^4) = \sigma^5, \sigma^6, \ldots, \sigma^{16}, \sigma^{13}$.

20) $\mathrm{Cnt}(\Pi, \sigma) = \sigma^1, \ldots, \sigma^{16}, \sigma^{13}$.

Задача 7.3. Постройте вычисление для алгоритма *Euclid* на начальной конфигурации вызова $(\mathit{Euclid}, 3, 1)$, см. рис. 6.5 на стр. 235.

Лемма 7.2 (Лемма о вычислении выражений)**.** *Пусть e — выражение, σ — конфигурация и $\sigma(e)$ определено. Пусть ρ — последняя конфигурация в* $\mathrm{Cnt}(e, \sigma)$. *Тогда* $\rho = \sigma \cup \{(u_e, \sigma(e))\}$.

Доказательство. Индукция по построению выражения e.

Базис индукции. Для переменных и констант утверждение, очевидно, следует из определения вычисления.

Шаг индукции. Пусть $e \circ\!\!- o(e_1, \ldots, e_n)$. Тогда по индукции, $\mathrm{Cnt}(e_1, \sigma)$ заканчивается σ^1 и $\sigma^1(u_{e_1}) = \sigma(e_1)$. $\mathrm{Cnt}(e_2, \sigma^1)$ заканчивается σ^2, $\sigma^2(u_{e_2}) = \sigma^1(e_2) = \sigma(e_2)$ и $\sigma^2 u_{e_1} = \sigma^1 u_{e_1} = \sigma(e_1)$. В конце получаем: $\sigma^n(e_i) = \sigma u_{e_i}, i = 1, \ldots, n$. Но тогда по определению $\mathrm{Cnt}(e, \sigma)$ заканчивается τ и

$$\tau u_e = o(\sigma^n u_{e_1}, \ldots, \sigma^n u_{e_n}) = \\ = o(\sigma(e_1), \ldots, \sigma(e_n)) = \sigma(o(e_1, \ldots, e_n)). \quad \square$$

***Задача 7.4.** Обоснуйте индукционный шаг для выражений вида $f(\bar{e})$, где f — имя подпрограммы, $\bar{e}$ — выражения.

Лемма 7.3 (Лемма о вычислении программ)**.** *Пусть* Π *— программа,* σ *— конфигурация и* $\Pi(\sigma)$ *определено. Пусть* ρ *— последняя конфигурация в* $\mathrm{Cnt}(\Pi, \sigma)$*. Тогда* $\rho = \Pi(\sigma)$.

Доказательство. Индукция по построению программы Π.

Базис индукции. Для оператора присваивания $\Pi \circ\!\!- x = e$; по определению:

$$\mathrm{Cnt}(\Pi, \sigma) = \mathrm{Cnt}(e, \sigma), \tau$$

и $\tau x = \rho u_e$, где ρ — последняя конфигурация $\mathrm{Cnt}(e, \sigma)$. По предыдущему утверждению, $\rho u_e = \sigma(e)$. Следовательно, $\tau x = \sigma(e)$ и $\tau y = \sigma y$ для остальных переменных. Это точно соответствует $\Pi(\sigma)$.

Шаг индукции.

1)

$$\Pi \circ\!\!- \begin{cases} \mathsf{if}\ e\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

Пусть $\mathrm{Cnt}(e, \sigma)$ заканчивается ρ. Получаем, что $\sigma(e) \neq 0$ тогда и только тогда, когда $\rho u_e \neq 0$.

$$\mathrm{Cnt}(\Pi, \sigma) = \begin{cases} \mathrm{Cnt}(e, \sigma), \mathrm{Cnt}(\Pi_1, \sigma), & \text{если } \rho u_e \neq 0; \\ \mathrm{Cnt}(e, \sigma), \sigma, & \text{иначе.} \end{cases}$$

По индукции получаем, что последовательность $\mathrm{Cnt}(\Pi_1, \sigma)$ заканчивается $\Pi_1(\sigma)$. Поэтому $\mathrm{Cnt}(\Pi, \sigma)$ заканчивается $\Pi_1(\sigma)$, если $\sigma(e) \neq 0$, и σ в противном случае. То есть в любом случае $\mathrm{Cnt}(\Pi, \sigma)$ заканчивается $\Pi(\sigma)$.

2)

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{while}\ e\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

Пусть $(\sigma^i)_{i=0}^n$ последовательность конфигураций такая, что

$$\sigma^0 = \sigma, \quad \sigma^i(e) \neq 0 \iff i < n, \quad \sigma^{i+1} = \Pi_1(\sigma^i).$$

Тогда вычислением Π на σ будет

$$\mathrm{Cnt}(e, \sigma^0), \mathrm{Cnt}(\Pi_1, \sigma^0), \ldots \\ \ldots, \mathrm{Cnt}(e, \sigma^{n-1}), \mathrm{Cnt}(\Pi_1, \sigma^{n-1}), \mathrm{Cnt}(e, \sigma^n), \sigma^n.$$

Но, по определению, $\Pi(\sigma) = \sigma^n$.

□

***Задача 7.5.** Обоснуйте индукционный шаг для следования и полного ветвления.

Итак, мы доказали, что вычисление — один из способов задания семантики программ.

****Задача 7.6.** Сформулируйте понятие вычисления для программ с метками и докажите аналогичные утверждения.

§ 7.3. Время и память вычисления

Теперь мы в состоянии аксиоматически определить, что такое время и память вычисления.

Определение 7.3 (Время вычисления). *Временем вычисления программы* Π *на конфигурации* σ — $\mathrm{Tm}(\Pi, \sigma)$ — *будем называть длину последовательности* $\mathrm{Cnt}(\Pi, \sigma)$.

Определение 7.4 (Память состояния). *Памятью состояния* σ — $\mathrm{S}(\sigma)$ — *будем называть величину*

$$\mathrm{S}(\sigma) = \sum_{x \in \mathrm{dom}\, \sigma} \mathrm{S}(\sigma x),$$

где S — *описывает объём памяти для натуральных чисел.*

Определение 7.5 (Память вычисления). *Памятью вычисления программы* Π *на конфигурации* σ — $\mathrm{Sp}(\Pi, \sigma)$ — *будем называть величину*

$$\mathrm{Sp}(\Pi, \sigma) = \max\{\mathrm{S}(\sigma^1), \ldots, \mathrm{S}(\sigma^n)\},$$

где $\sigma^1, \ldots, \sigma^n$ — *вычисление* Π *на* σ.

Предположим, что $\mathrm{S}(x) = [\log_2 x] + 1$. Тогда, в приведённых выше примерах мы получим следующие результаты.

Пример 7.3. *См. пример 7.1 на стр. 291.*

$$\mathrm{Tm}(\Pi, \sigma) = 5; \quad \mathrm{Sp}(\Pi, \sigma) = \mathrm{S}(\sigma^2) = 7.$$

Рассмотрим какую-нибудь другую начальную конфигурацию τ. *Тогда* $\mathrm{Tm}(\Pi, \tau) = 5$ *по-прежнему.* $\mathrm{Sp}(\Pi, \tau)$ *будет равно*

$$\mathrm{S}(\sigma^2) = 2\mathrm{S}(\tau x) + 2\mathrm{S}(\tau y) + \mathrm{S}(\tau z).$$

Например, если

$$\tau = \{(x, 100); (y, 200)\},$$

то

$$\mathrm{Sp}(\Pi, \tau) = 2\mathrm{S}(100) + 2\mathrm{S}(200) + \mathrm{S}(0) = 2(6 + 1) + 2(7 + 1) + 1 = 31.$$

Пример 7.4. *См. пример 7.2 на стр. 292.*

$$\mathrm{Tm}(\Pi, \sigma) = 17; \quad \mathrm{Sp}(\Pi, \sigma) = \mathrm{S}(\sigma^{15}) = 11.$$

Для этой программы, очевидно, при изменении начальной конфигурации будет меняться не только Sp, *но и* Tm. *Последовательность конфигураций* $\sigma^5, \ldots, \sigma^{13}$ *будет повторена не один раз, а* $\tau x - \tau y$. *Следовательно, время работы можно определить по следующей формуле:*

$$\mathrm{Tm}(\Pi, \tau) = 8 + 9\lfloor \tau x - \tau y \rfloor.$$

Максимальная память в вычислении в любом случае будет приходиться на σ^{N-2}, *где* $N = \mathrm{Tm}(\Pi, \tau)$. *Следовательно,*

$$\mathrm{Sp}(\Pi, \tau) = \mathrm{S}(\tau x) + \mathrm{S}(\tau y) + \mathrm{S}(\tau x - \tau y) + \mathrm{S}(\tau x) + \mathrm{S}(\tau x) + \mathrm{S}(\tau x) = \\ = 4\mathrm{S}(\tau x) + \mathrm{S}(\tau y) + \mathrm{S}(\tau x - \tau y).$$

В частности, при

$$\tau = \{(x, 200); (y, 100)\}$$

получим:

$$\mathrm{Tm}(\Pi, \tau) = 908; \quad \mathrm{Sp}(\Pi, \tau) = 4 \times 8 + 7 + 7 = 46.$$

Такая ситуация, когда время вычисления превосходит память является наиболее типичной в программировании.

Заметим, что в данных определениях мы приняли некоторые упрощения. Например, мы считаем, что время выполнения любой операции всегда равно 1, и не зависит от величины операндов, что является абстракцией. Можно было бы, например, считать, что время выполнения каждой операции $o(x_1, \ldots, x_n)$ описывается некоторой функцией $T_o(\mathrm{S}(x_1), \ldots, \mathrm{S}(x_n))$.

Существует общая теория, которая изучает различные меры сложности вычислений. Время и память являются, конечно, наиболее полезными из мер сложности, но далеко не единственными. Вообще мера сложности — это какая-либо частичная функция: $\Phi : \Pi, \sigma \mapsto \varphi$, которая удовлетворяет следующим двум условиям:

1) $\Phi(\Pi, \sigma)$ определена тогда и только тогда, когда $\Pi(\sigma)$ определено.

2) Задача определения по заданным Π, σ и $\varphi \in \omega$ того, что $\Phi(\Pi, \sigma) \leqslant \varphi$, является алгоритмически разрешимой.

Можно доказать, что определенные нами функции Tm и Sp удовлетворяют этому условию, поэтому являются мерами сложности.

Определение 7.6 (Длина входа)**.** *Длиной входных данных вызова* $(A, a_1, \ldots, a_k)$ *называется сумма объемов значений входных переменных:*

$$\mathrm{S}(a_1) + \cdots + \mathrm{S}(a_k).$$

Обычно время и память вычисления рассматривают не как функции, зависящие от начальной конфигурации программы, а как функции от длины входных данных. Однако, тогда возможна вот какая ситуация: могут существовать две конфигурации σ и τ такие, что $\mathrm{S}(\sigma) = \mathrm{S}(\tau)$, но в то же время $\mathrm{Tm}(\Pi, \sigma) \neq \mathrm{Tm}(\Pi, \tau)$ и $\mathrm{Sp}(\Pi, \sigma) \neq \mathrm{Sp}(\Pi, \tau)$. Для примера можно рассмотреть программу из примера 7.2 на стр. 292 и конфигурации:

$$\tau = \{(x, 200), (y, 100)\};$$

$$\sigma = \{(x, 300), (y, 50)\}.$$

Очевидно, что

$$\mathrm{S}(\sigma) = 9 + 6 = \mathrm{S}(\tau) = 8 + 7 = 15.$$

В то же время

$$\mathrm{Tm}(\Pi, \tau) = 908; \quad \mathrm{Tm}(\Pi, \sigma) = 8 + 9 \times 250 = 2258;$$

$$\mathrm{Sp}(\Pi, \tau) = 46; \quad \mathrm{Sp}(\Pi, \sigma) = 4 \times 9 + 6 + 8 = 50.$$

Чтобы не возникало таких противоречий, введём дополнительные определения. Для простоты будем предполагать, что $\Pi(\sigma)$ определено для всех конфигураций σ.

Определение 7.7 (Максимальное время)**.** *Максимальным временем работы программы* Π *(временем работы в наихудшем случае) называется следующая функция* $\mathrm{Tm}_\Pi : \omega \to \omega$*:*

$$\mathrm{Tm}_\Pi(n) = \max\{\mathrm{Tm}(\Pi, \sigma) : \mathrm{S}(\sigma) = n\}.$$

Заметим, что это определение корректно, так как множество, по которому берется максимум конечно по определению функции S*. Можно сказать, что максимальное время работы программы показывает время работы в наиболее неблагоприятных обстоятельствах. Чтобы ни случилось, программа не будет работать дольше этого времени.*

Определение 7.8 (Среднее время)**.** *Среднее время работы программы* Π *— это функция* $\mathrm{Tm}^{cp}_\Pi : \omega \to \omega$*, определённая так:*

$$\mathrm{Tm}^{cp}_\Pi(n) = \frac{1}{N} \sum_{\mathrm{S}(\sigma)=n} \mathrm{Tm}(\Pi, \sigma).$$

В данном определении через N *обозначено количество конфигураций* σ*, для которых* $\mathrm{S}(\sigma) = n$*. Это определение корректно по той же причине. Среднее время определяет продолжительность работы программы в наиболее типичном случае.*

Аналогично определяются максимальная и средняя память.

Определение 7.9 (Максимальная память)**.**

$$\mathrm{Sp}_\Pi(n) = \max\{\mathrm{Sp}(\Pi, \sigma) : \mathrm{S}(\sigma) = n\}.$$

Определение 7.10 (Средняя память)**.**

$$\mathrm{Sp}_\Pi^{cp}(n) = \frac{1}{N} \sum_{\mathrm{S}(\sigma)=n} \mathrm{Sp}(\Pi, \sigma).$$

Рассмотрим примеры.

Пример 7.5. *См. пример 7.1 на стр. 291. Мы уже видели, что для нашей первой программы время работы не зависит от начальной конфигурации τ, следовательно,*

$$\mathrm{Tm}_\Pi(n) = \mathrm{Tm}_\Pi^{cp}(n) = 5.$$

Используемая память равна

$$2\mathrm{S}(\tau x) + 2\mathrm{S}(\tau y) + 1 = 2(\mathrm{S}(\tau x) + \mathrm{S}(\tau y)) + 1 = 2n + 1,$$

поэтому

$$\mathrm{Sp}_\Pi(n) = \mathrm{Sp}_\Pi^{cp}(n) = 2n + 1.$$

Пример 7.6. *См. пример 7.2 на стр. 292. Сначала найдём максимальные время и память.*

$$\mathrm{Tm}_\Pi(n) = \max\{\mathrm{Tm}(\Pi, \tau) : \mathrm{S}(\tau) = n\} = \\ = \max\{8 + 9(\tau x - \tau y) : [\log_2 x] + 1 + [\log_2 y] + 1 = n\}.$$

Очевидно, что для достижения максимума нужно взять конфигурацию τ такую, что $\tau y = 0$, а x выбрать максимальным при условии, что $[\log_2 \tau x] = n - 2$. Очевидно, что τx должно быть равно $2^{n-1} - 1$. Поэтому,

$$\mathrm{Tm}_\Pi(n) = 8 + 9(2^{n-1} - 1 - 0) = 4.5 \times 2^n - 1.$$

Память является максимальной при тех же условиях, поэтому

$$\mathrm{Sp}_\Pi(n) = 4(n-1) + 1 + n - 1 = 5n - 4.$$

Точные средние оценки в данном случае мы получать не будем ввиду их сложности. Получим приближённую оценку среднего времени. Очевидно, что

$$\mathrm{Tm}_{\Pi}^{cp}(n) = \frac{1}{N}\sum_{\mathrm{S}(\tau)=n}(8+9\lfloor \tau x - \tau y\rfloor) = 8 + \frac{9}{N}\sum_{\mathrm{S}(\tau)=n}\lfloor \tau x - \tau y\rfloor.$$

Чтобы найти последнюю сумму, заметим, что

$$\sum_{\mathrm{S}(\tau)=n}\lfloor \tau x - \tau y\rfloor = \sum_{\substack{\mathrm{S}(\tau)=n\\ \tau x \geqslant \tau y}} \tau x - \tau y \approx \sum_{\substack{\mathrm{S}(\tau)=n\\ \mathrm{S}(\tau x)\geqslant \mathrm{S}(\tau y)}} \tau x - \tau y.$$

Прежде, чем продолжать, выведем формулу для суммы всех чисел длины n:

$$\sum_{\mathrm{S}(x)=n} x = \sum_{x=2^{n-1}}^{2^n-1} x = 2^{n-1}\times 2^{n-1} + \sum_{x=0}^{2^{n-1}-1} x = \\ = 2^{2n-2} + \frac{2^{n-1}(2^{n-1}-1)}{2} = \frac{3}{2}2^{2n-2} - \frac{1}{2}2^{n-1}$$

Найдем нашу сумму, когда длины x и y зафиксированы: $\mathrm{S}(x)=i$, $\mathrm{S}(y)=n-i$.

$$\sum_{\substack{\mathrm{S}(x)=i\\ \mathrm{S}(y)=n-i}} x-y = 2^{n-i-1}\sum_{\mathrm{S}(x)=i} x - 2^{i-1}\sum_{\mathrm{S}(y)=n-i} y = \\ = 2^{n-i-1}\left(\frac{3}{2}2^{2i-2} - \frac{1}{2}2^{i-1}\right) - 2^{i-1}\left(\frac{3}{2}2^{2n-2i-1} - \frac{1}{2}2^{n-i-1}\right) = \\ = \frac{3}{2}2^{n+i-3} - \frac{1}{2}2^{n-2} - \frac{3}{2}2^{2n-i-3} + \frac{1}{2}2^{n-2} = \frac{3}{2}2^{n-3}\left(2^i - 2^{n-i}\right)$$

Теперь просуммируем по всевозможным i от $n/2$ до $n-1$ (то есть, когда $\mathrm{S}(x) \geqslant \mathrm{S}(y)$):

$$\sum_{i=n/2}^{n-1} \frac{3}{2} 2^{n-3}\bigl(2^i - 2^{n-i}\bigr) = \frac{3}{2} 2^{n-3}\left(\sum_{i=n/2}^{n-1} 2^i - 2^n \sum_{i=n/2}^{n-1} 2^{-i} \right) =$$

$$= \frac{3}{2} 2^{n-3}\left(2^{n/2}\bigl(2^{n/2}-1\bigr) - 2^n 2^{-n/2} \frac{1-2^{-n/2}}{1-2^{-1}}\right) =$$

$$= \frac{3}{2} 2^{n-3}\Bigl(2^n - 2^{n/2} - 2\bigl(2^{n/2}-1\bigr)\Bigr) =$$

$$= \frac{3}{2} 2^{n-3}\bigl(2^n - 3 \times 2^{n/2} + 2\bigr) =$$

$$= \frac{3}{2} 2^{2n-3}\bigl(1 - 3 \times 2^{-n/2} + 2^{-n+1}\bigr) \approx \frac{3}{2} 2^{2n-3}\bigl(1 - 3 \times 2^{-n/2}\bigr).$$

К этой сумме необходимо прибавить слагаемое, описывающее случай $y = 0$:

$$\frac{3}{2} 2^{2n-4} - \frac{1}{2} 2^{n-2} = \frac{3}{2} 2^{2n-3}\left(\frac{1}{2} - \frac{1}{3} 2^{-n+1}\right) \approx \frac{3}{2} 2^{2n-3} \times \frac{1}{2}.$$

Всего получается

$$\frac{3}{2} 2^{2n-3}\bigl(1 - 3 \times 2^{-n/2}\bigr) + \frac{3}{2} 2^{2n-3} \times \frac{1}{2} =$$

$$= \frac{3}{2} 2^{2n-3}\left(\frac{3}{2} - 3 \times 2^{-n/2}\right) = \frac{9}{8} 2^{2n-2}\bigl(1 - 2^{-n/2+1}\bigr).$$

Таким образом, мы получили приближенное равенство:

$$\sum_{\mathrm{S}(\tau)=n} \lfloor \tau x - \tau y \rfloor \approx \frac{9}{8} 2^{2n-2}\bigl(1 - 2^{-n/2+1}\bigr).$$

Теперь найдём общее количество начальных конфигураций длины n. Найдём количество z таких, что $\mathrm{S}(z) = i$. Имеем

$$[\log_2 z] + 1 = i, \quad 2^{i-1} \leqslant \log_2 z \leqslant 2^i - 1.$$

Общее количество z, следовательно, равно 2^{i-1}. Если $\mathrm{S}(x) = i, \mathrm{S}(y) = n - i$, то количество таких конфигураций есть

$$2^{i-1} \times 2^{n-i-1} = 2^{n-2}.$$

Поэтому общее количество конфигураций длины n есть $(n-1)2^{n-2}$. Прибавим ещё случаи, когда $x=0$ или $y=0$, получим: $(n+1)2^{n-2}$. Отсюда мы можем получить приближенную оценку для среднего времени:

$$\mathrm{Tm}_{\Pi}^{cp} \approx 8 + 9\frac{\frac{9}{8}2^{2n-2}(1-2^{-n/2+1})}{(n+1)2^{n-2}} = \\ = 8 + 9\frac{9}{8(n+1)}2^n\bigl(1-2^{-n/2+1}\bigr) = \\ = 8 + \frac{81}{8(n+1)}\bigl(2^n - 2^{n/2+1}\bigr).$$

Посмотрим, насколько точная оценка получилась. Пусть, например, $n=4$. Имеется два числа длины 1: 0 и 1, два числа длины 2: 2 и 3, и четыре числа длины 3: 4, 5, 6 и 7. Всего конфигураций τ таких, чтобы $\mathrm{S}(\tau x)+\mathrm{S}(\tau y)=4$, следовательно, имеется:

$$N = 2\times 4 + 2\times 2 + 4\times 2 = 20.$$

Найдём теперь сумму

$$\sum_{\mathrm{S}(\tau)=4} \lfloor \tau x - \tau y \rfloor = 4+5+6+7+3+4+5+6+1 = 41.$$

Поэтому точное среднее время для $n=4$ есть:

$$\mathrm{Tm}_{\Pi}^{cp}(4) = 8 + 9\times\frac{1}{20}\times 41 \approx 26.5.$$

Наша формула даёт:

$$\mathrm{Tm}_{\Pi}^{cp}(4) = 8 + \frac{81}{8\times 5}(16-8) = 24.2.$$

Точно так же можно проверить результат и для $n=5;6$. При $n=5$ точная оценка: 45.5, приближенная: 43. При $n=6$ имеем 79.5 и 77.5 соответственно. Как видим, оценка является очень точной даже для маленьких n.

***Задача 7.7.** Найдите оценку для средней памяти.

Как правило, для оценки времени и памяти алгоритмов используются «предельные» случаи, то есть когда длины входных данных

достаточно велики, чтобы не учитывать какие-то «мелкие» частности. Скажем, в предыдущем примере мы получили оценку среднего времени

$$8 + \frac{81}{8(n+1)}\bigl(2^n - 2^{n/2+1}\bigr).$$

При достаточно больших n слагаемые 8 и $\frac{81}{8(n+1)} \times 2^{n/2+1}$ ничтожны по сравнению с $\frac{81}{8(n+1)} \times 2^n$, поэтому можно считать, что среднее время работы равно $\frac{81}{8(n+1)} \times 2^n$.

Для следующих примеров предполагается знакомство с O-символикой Ландау из курса математического анализа. Мы пишем $f(n) = O(g(n))$, если существует вещественная константа c такая, что $|f(n)| \leqslant c|g(n)|$ для всех n, начиная с некоторого. Например,

$$100!^{100!}n^2 + 3n = O(n^2), \quad 3n^3 + n^2 \log_2 n = O(n^3).$$

Предполагается, что известны следующие два факта:

1) $\lim\limits_{n\to\infty} a^n/n^b = \infty$ при $a > 1$ (любая экспонента асимптотически превосходит любой полином);

2) $\lim\limits_{n\to\infty} n^a/\log_b n = \infty$ при $a > 0$ и $b > 1$ (любой полином асимптотически превосходит любой логарифм).

Пример 7.7. *См. пример 7.2 на стр. 292. Мы предполагаем, что x и y — входные переменные. Пусть общий размер входа равен $S(x) + S(y) = n$.*

Сначала оценим время. Первые два присваивания требуют некоторого фиксированного времени c_1. Можно проверить по определению вычисления, что $c_1 = 4$, но при оценке по порядку это не понадобится. Одна итерация тела цикла выполняется также за некоторую константу c_2: срабатывает одна проверка теста и два присваивания. На каждом шаге переменная v увеличивается на 1, а переменная x не меняется. Поэтому число итераций цикла не превосходит $\sigma x \leqslant 2^n$. Поэтому время работы цикла не превосходит $2^n c_2 + 3$. Значит, общее время не превосходит $c_1 + 3 + 2^n c_2 = O(2^n)$, так как слагаемым c_1 можно пренебречь.

```
algorithm F;
arguments x;
returns r;
    y = 1;
    r = 1;
    while y ⩽ x do
        t = 0;
        u = 0;
        while u < y do
            t = t + r;
            u = succ(u);
        end;
        r = t;
        y = succ(y);
    end;
end;
```

Рис. 7.1: Факториал числа, структурированная программа

Теперь оценим память. Для этого заметим, что максимальная память не превосходит $k \log_2 m$, *где* k — *число переменных в вычислении (то есть переменных программы и переменных для хранения промежуточных результатов), а* m — *максимальное значение, встречающееся в программе. Поскольку* k — *константа, то для определения максимальной памяти достаточно найти наибольшее значение переменной. Переменные* x *и* y *не изменяются, переменные* u *и* v *не превосходят* x. *Поэтому любая переменная не превосходит* 2^n, *а значит, максимальная память составляет* $O(n)$.

Пример 7.8. *Оценим максимальные время и память, которые требует структурированная программа для вычисления факториала числа* x *(рис. 7.1).*

Пусть размер входа равен n. Сначала оценим время работы внутреннего цикла. Как и в предыдущем примере, каждая итерация внутреннего цикла требует не более чем некоторую константу времени, а число итераций не превосходит y. Поскольку y не превосходит x, то число итераций внутреннего цикла есть величина порядка 2^n. Поэтому время работы внутреннего цикла составляет $O(2^n)$. Теперь оценим время работы внешнего цикла. Одна его итерация требует $O(2^n)$ (внутренний цикл) плюс константа (четыре присваивания). По порядку получается $O(2^n)$. Число итераций не превосходит 2^n, поэтому общее время составляет $O(2^n \times 2^n) = O(2^{2n})$. Первыми двумя присваиваниями можно пренебречь по сравнению с экспонентой, на асимптотическую оценку они не влияют.

Теперь оценим память. Сначала получим одно простое неравенство.

$$\log_2 k! = \log_2 \prod_{j=1}^{k} j = \sum_{j=1}^{k} \log_2 j \leqslant k \log_2 k$$

Итак, $\log_2 k! \leqslant k \log_2 k$. На самом деле можно доказать, что левая и правая часть есть величины одного порядка, но мы получаем грубую оценку. Максимальное значение принимает переменная r в конце работы алгоритма, это значение равно $\sigma x!$. Размер этого числа равен

$$[\log_2(\sigma x!)] + 1 \leqslant \sigma x \log_2 \sigma x + 1 = O(n2^n).$$

Поэтому и память составляет $O(n2^n)$. Память получилась такой большой, потому что число $\sigma x!$ весьма велико.

Пример 7.9. *Оценим максимальное время работы программы, которая проверяет число x на простоту (рис. 7.2 на следующей странице).*

Одна итерация цикла требует не более чем некоторую константу времени, а число итераций не превосходит 2^n. Все переменные не превосходят 2^n. Поэтому общее время составляет $O(2^n)$, а память — $O(n)$. Алгоритм можно несколько усовершенствовать. Первый вариант — заменить в цикле тест на $y < x/2$. Тогда число итераций уменьшится в два раза, поэтому асимптотическая оценка не изменится. Другой вариант — заменить тест на $y \leqslant [\sqrt{x}]$. Тогда число итераций не превосходит $\sqrt{2^n} = 2^{n/2}$, и общее время составляет $O(2^{n/2})$.

На первый взгляд может показаться, что этот алгоритм намного лучше. Однако на практике это не так. Например, в задачах кодирования

```
algorithm P;
arguments x;
returns f;
    y = 2;
    f = 1;
    while y < x do
        if x mod y == 0 then
            f = 0;
        end;
        y = succ(y);
    end;
end;
```

Рис. 7.2: Проверка числа на простоту, структурированная программа

приходится работать с числами, длина которых составляет 512 и более бит. В этом случае время «медленного» алгоритма составит

$$2^{512} > 2^{500} = (1024)^{50} > (10^3)^{50} = 10^{150},$$

а время «быстрого» алгоритма составит

$$2^{256} > 2^{250} = 1024^{25} > (10^3)^{25} = 10^{75}.$$

Реально ни один компьютер не в состоянии выполнить такое количество операций за приемлемое время. Для сравнения — возраст всей Вселенной оценивается примерно в 5×10^{17} секунд, быстродействие самых мощных современных компьютеров только приближается к границе 10^{18} операций в секунду.

Пример 7.10. *Оценим максимальное время алгоритма Евклида.*

Напомним, что через F_n обозначается n-е число Фибоначчи: $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_n + F_{n+1}$.

Утверждение 7.4. *Если $n \geqslant 2$, то $F_{n+5} > 10F_n$.*

```
algorithm Euclid;
arguments x, y;
returns x + y;
    if x < y then
        z = y;
        y = x;
        x = z;
    end;
    while x × y do
        z = x mod y;
        x = y;
        y = z;
    end;
end;
```

Рис. 7.3: Алгоритм Евклида

Доказательство. *По определению чисел Фибоначчи:*

$$F_{n+5} = F_{n+4} + F_{n+3} = 2F_{n+3} + F_{n+2} = \\ = 3F_{n+2} + 2F_{n+1} = 5F_{n+1} + 3F_n = 8F_n + 5F_{n-1} > \\ > 8F_n + 4F_{n-1} > 8F_n + 2F_n = 10F_n.$$

На предпоследнем шаге мы использовали следующее соотношение:

$$F_n = F_{n-1} + F_{n-2} < 2F_{n-1}. \qquad \square$$

Утверждение 7.5 (Теорема Ламе)**.** *Пусть значения переменных x и y равны a и b соответственно, причём $a > b > 0$. Тогда число делений в алгоритме Евклида не превосходит $5k$, где k — число цифр в десятичной записи числа b.*

Доказательство. *Пусть $a_0 = a$, $a_1 = b$. Тогда при выполнении алгоритма Евклида мы получим некоторую последовательность остатков*

$a_0, a_1, \ldots, a_{n-1}, a_n$:

$$\begin{cases} a_0 = a_1 q_1 + a_2, & 0 < a_2 < a_1, \\ a_1 = a_2 q_2 + a_3, & 0 < a_3 < a_2, \\ \ldots, & \\ a_{n-2} = a_{n-1} q_{n-1} + a_n, & 0 < a_n < a_{n-1}, \\ a_{n-1} = a_n q_n. & \end{cases}$$

Отметим, что $a_{i+1} < a_i$, так как a_{i+1} будет остатком от деления на a_i. Возвратной индукцией по i докажем, что $a_i \geqslant F_{n-i+2}$.
Базис индукции. Пусть $i = n$, тогда $a_n \geqslant 1 = F_2$. Пусть $i = n-1$, тогда $a_{n-1} > a_n \geqslant 1$. Следовательно, $a_{n-1} \geqslant 2 = F_3$.
Шаг индукции. По индукционному предположению $a_{i+1} \geqslant F_{n-i+1}$, $a_{i+2} \geqslant F_{n-i}$. Поэтому

$$a_i = a_{i+1} q_{i+1} + a_{i+2} \geqslant a_{i+1} + a_{i+2} \geqslant F_{n-i+1} + F_{n-i} = F_{n-i+2}.$$

При $i = 1$ получаем $b = a_1 \geqslant F_{n+1}$. Предположим теперь, что $n > 5k$. Тогда $b \geqslant F_{5k+2} > 10^k F_2 = 10^k$. Это противоречит тому, что количество десятичных цифр числа b равно k. □

Утверждение 7.6. *Время работы алгоритма Euclid на аргументах a и b составляет $O(n)$, где $n = S(a) + S(b)$ — общий размер входа.*

Доказательство. *Время работы ветвления ограничено константой. После ветвления значение переменной x не меньше значения переменной y. По предыдущему утверждению число итераций числа есть величина порядка $O(\log_{10} b) = O(n)$. Так как время одной итерации является константой, то общее время составляет $O(n)$.* □

Предметный указатель

$(a_i)_i$, 29
$(a_i)_{i=0}^{y}$, 29
$(a_i)_{i=x}^{y}$, 29
A^n, 27
$A^{<\omega}$, 27
$(a)_{e_1 \dots e_n}^{x_1 \dots x_n}$, 77
$=$, 26
$\equiv$, 38
fa_1, 24
$f(a_1, \dots, a_n)$, 24
$\bot$, 34
$\{\varphi\}\,\Pi\,\{\psi\}$, 150
L_φ, 168
$\mapsto$, 24
$\models$, 38, 149
$\not\models$, 150
$\bar{x}, \bar{y}$, 42
$\mathcal{P}(A)$, 28
Π_A, 48, 227
$(\Pi)_\beta^\alpha$, 118
$\restriction$, 24
σ_δ, 63
$\sigma(e)$, 53
Σ_Π^E, 108
Σ_Π, 55
$(\sigma)_{y_1 \dots y_n}^{x_1 \dots x_n}$, 79
$\multimap$, 42
$\top$, 34
$\mathfrak{V}$, 41
$\triangleright$, 149

Alg, 48
arg, 48

Cnt, 289, 290

do, 45
dom, 28

else, 45
end, 44

Fc, 239, 240, 243
Ff, 246
Fs, 243
Ft, 244
function, 227

if, 44

LVar, 46

Res, 63
ret, 48
rng, 28

S, 285, 296
Sp, 296, 300
Sp^{cp}, 300
Start, 107
succ, 40

then, 44
Tm, 296, 299
Tm^{cp}, 299

Var, 46

while, 45

Аксиома
  АКС, 154
  $\mathbf{АКС}_0$, 154
  ПРИСВ, 187
  ПРИСВ*, 188
  СПР, 204
  $\mathbf{СПР}_0$, 204
Аксиомы, 148
Алгоритм, 11, 48
  детерминированный, 16
  Евклида, 21
  итерационный, 21
  конечный, 17
  недетерминированный, 16
  полууниверсальный, 18
  простой, 87
  рекурсивный, 21
  универсальный, 17
  частный, 17
Блок-схема, 134
Верификация программы, 180
Вершина
  ветвления, 134
  внутренняя, 32
  графа, 30
  конца, 134
  начала, 134
  присваивания, 134
Время
  вычисления, 296
  максимальное, 299
  среднее, 299
Вывод, 148
Выводимость, 149
Вызов
  алгоритма, 63
  блок-схемы, 136
  подпрограммы, 227
  рекурсивный, 246
Выражение, 43, 228
  результат, 48
Вычисление
  выражения, 289
  программы, 290

Вычисление функции
алгоритмом, 64
Граф, 30
зависимости, 238
нагруженный, 31
размеченный, 31
Дерево, 32
вложенных вызовов, 244
Длина входа, 298
Допустимость, 187
Замена
метки, 117
переменных, 77
в состоянии, 79
Зацикливание, 57
Значение
выражения, 53, 228
переменной, 52
формулы, 35, 36
атомной, 35
функции, 24
Имя
алгоритма, 48
переменной, 41
подпрограммы, 227
Инвариант
программы, 164
частичный, 164
цикла, 166
частичный, 166
Информатика, 11
Исполнитель, 11
Истинность
условия, 149
Исчисление, 148
предусловий, 204
Хоара, 154
Квантор
всеобщности, 37
существования, 37
Композиция отображений, 26
Константа, 40
Конфигурация
блок-схемы, 135
выражения, 53
программы
структурированной, 55
программы с метками, 108
Корень дерева, 32
Корректность
полная, 151
программ, 158
частичная, 151
Лемма
о бесконечных деревьях
вызовов, 247
об обратимости замены,
80
об определённости
вложенных вызовов,
248
об убывающих цепях, 29

об удалении косвенной
рекурсии, 261
об упрощении
неполного ветвления, 89
полного ветвления, 88
цикла, 90
о вложенных вызовах, 243
о вычислении
выражений, 293
программ, 294
о замене метки, 118
о замене переменной
в выражениях, 80
в программах, 81
о кодировании пар чисел,
259
о минимальном объёме,
287
о неиспользуемых
переменных, 76
о полной корректности
цикла, 169
о представлении
предусловий, 217, 219
о разложении выражения,
87
о результате замены, 78
о сбалансированности, 49
о сохранении значения, 75
о суффиксе программы,
50
о существовании
предусловий, 219
о форме предусловий, 216
о частичной корректности
цикла, 167
Лес вложенных вызовов, 246
Лист дерева, 32
Ложность
условия, 150
Мера сложности, 298
Метка, 106
завершающая, 107
заключительная, 107
начальная, 107
оператора, 107
перехода, 107
Множество
вершин, 30
вложенных вызовов, 243
значений, 28
переменных, 46
изменяемых, 46
рёбер, 30
Начальная конфигурация
вызова, 63
Непротиворечивость
исчисления
предусловий, 204
Хоара, 155
Область определения, 28
Объём памяти, 285

Ограничение функции, 24
Ограничитель
  цикла, 168
Оператор, 43
  неполного ветвления, 44
  полного ветвления, 45
  присваивания, 43, 107
    левая часть, 43
    правая часть, 43
    простой, 87
  пустой, 44
  условного перехода, 107
  цикла, 45
Операция, 40
Ослабление постусловия, 154
Отображение, 23
  частичное, 25
Память
  вычисления, 296
  максимальная, 300
  состояния, 296
  средняя, 300
Параметры
  подпрограммы, 227
Переменная, 41
  входная
    алгоритма, 48
    блок-схемы, 134
    подпрограммы, 227
Поддерево, 33
Подпрограмма, 226
  рекурсивная, 239
Полнота исчисления, 214
  предусловий, 215
  Хоара, 215
Последовательность, 29
Построение программы
  простой, 92
  с метками, 120
  структурированной, 127
  функциональной, 251
Постусловие, 150
Правила вывода, 148
Правило
  ДОБ, 188, 276
  ДОБ$_1$, 277
  ЗАМ, 273
  ЗАМ$_1$, 273
  замены аргументов, 273
  НВТ, 155
  неполного ветвления, 155
  ОСП, 154
  ПВТ, 155
  полного ветвления, 155
  РЕК, 274
  рекурсии, 274
  СЛД, 155
  следование, 155
  СНВ, 205
  СПВ, 206
  ССЛ, 204
  СЦП, 206

СЭП, 204
УПР, 154
ЦКП, 170
ЦКЧ, 170
Предметная область, 39
Предусловие, 150
слабейшее, 203
полное, 203
Префикс, 42
собственный, 42
Программа, 13
основная, 238
простая, 87
с метками, 106
структурированная, 44
функциональная, 251
Произведение отображений, 26
Путь на графе, 32
Разрешимость, 15
Ребро графа, 30
Результат вызова, 63, 228
Рекурсия, 239
косвенная, 239
прямая, 239
Связки булевы, 35
Семантика, 12
блок-схемы, 135
неполного ветвления, 56
полного ветвления, 56
присваивания, 56
программы, 233
с метками, 108
структурированной, 55
пустого оператора, 55
следования, 56
цикла, 57
Символ кванторный, 35
Следование, 44
Следствие
об эквивалентности предусловий, 203
о двойном представлении программы, 52
о двух конфигурациях, 77
о результате, 77
упрощение присваиваний, 88
Совпадение графическое, 42
Состояние, 52
Спецификация программы, 180
Список, 29
вложенных вызовов, 239, 240, 243
входных переменных, 48
Суффикс, 42
собственный, 42
Тело
алгоритма, 48
подпрограммы, 227
Теорема

об удалении подпрограмм, 259
об удалении рекурсии, 262
о конечных деревьях, 33
о конечных деревьях вызовов, 248
о подстановке, 97
о полноте исчисления предусловий, 215
Хоара, 215
о представлении формул, 217
о представлении функций, 217
о простой программе, 92
о семантике следования, 62
о структуризации, 127
о существовании слабейших предусловий, 208
о цикле, 133
Трансляция, 12
Тройка Хоара, 150
Усиление предусловия, 154
Условие, 149
зацикливания, 57, 60, 109, 110, 249
Форма записи
инфиксная, 25
префиксная, 25
Формула логическая, 34
атомная, 34
Функция, 23
тотальная, 27
частичная, 25
Цепь убывающая, 29
Цикл, 32
Числа Фибоначчи, 110
Эквивалентность
алгоритмов, 72
формул, 38
Эффективное построение, 120
Язык естественный, 12
Язык программирования, 12
второго уровня, 16
декларативный, 18
императивный, 18
объектно-ориентированного, 19
первого уровня, 15
с метками, 19
специализированный, 18
структурный, 19
третьего уровня, 16
универсальный[1], 13
универсальный[2], 18
функционального, 19

Литература

[1] Агафонов В. Н. Математические основы обработки информации / Учебное пособие. — Новосибирск: Изд-во НГУ, 1982. — 92 с.

[2] Булос Дж. Вычислимость и логика / Дж. Булос, Р. Джеффри; пер. с англ. В. А. Душского и Е. Ю. Ногиной под ред. С. Н. Артемова. — М.: Мир, 1994. — 396 с., ил.

[3] Вирт Н. Алгоритмы + структуры данных = программы / пер. с англ. — М.: Мир, 1985. — 408 с.

[4] Грис Д. Наука программирования: Пер. с англ. / Д. Грис; пер. Н. Н. Непейвода, ред. пер. А. П. Ершов. — М.: Мир, 1984. — 416 с. : ил.

[5] Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон; пер. с англ. — М.: Мир, 1982. — 416 с.

[6] Ездаков А. Л. Функциональное и логическое программирование : учебное пособие / А. Л. Ездаков. — 2-е изд. — М.: БИНОМ. Лаборатория знаний, 2011. — 119 с. : ил.; 60x90/16. — ISBN 978-5-9963-0575-9.

[7] Кауфман В. Ш. Языки программирования. Концепции и принципы [Электронный ресурс] / В. Ш. Кауфман. — М.: ДМК Пресс,

2010. — 464 с.: ил. — ISBN 978-5-94074-622-5. — Режим доступа: http://znanium.com/bookread2.php?book=409077

[8] Кнут Д. Э. Искусство программирования: пер. с англ. / Д. Э. Кнут; под общ. ред. Ю. В. Козаченко. — 3-е изд. Т. 1 (2015): Основные алгоритмы. — Изд. испр. и доп. — М.: Вильямс, 2015. — 712 с.: ил.

[9] Кнут Д. Э. Искусство программирования: пер. с англ. / Д. Э. Кнут; под общ. ред. Ю. В. Козаченко. — 3-е изд. Т. 2: Получисленные алгоритмы — М.: Вильямс, 2007. — 828 с.

[10] Мейер Б. Методы программирования: [В 2-х т.] / Б. Мейер, К. Бодуэн; Перевод с фр. Ю. А. Первина. — М.: Мир, 1982. — [Т.] 1. — М. : Мир, 1982. — 368 с. : ил. [Т.] 2. — М. : Мир, 1982. — 368 с. : ил.

[11] Непомнящий В. А. Прикладные методы верификации программ / В. А. Непомнящий, О. М. Рякин; под ред. А. П. Ершова — М.: Радио и связь, 1988. — 256 с.: ил.

[12] Оре О. Теория графов / пер. с англ. И. Н. Врублевской под ред. Н. Н. Воробьева — М.: Наука, 1968. — 352 с. с ил.

[13] Прохорова О. В. Информатика [Электронный ресурс]: учебник/ Прохорова О. В. — Электрон. текстовые данные. — Самара: Самарский государственный архитектурно-строительный университет, ЭБС АСВ, 2013. — 106 с. — Режим доступа: http://www.iprbookshop.ru/20465.html.

[14] Трахтенброт Б. А. Алгоритмы и вычислительные автоматы. — М.: Советское радио, 1974. — 200 с. с ил.

[15] Требования и спецификации в разработке программ. Сборник статей / пер. с англ. под ред. В. Н. Агафонова. — М.: Мир, 1984. — 344 с., ил.

[16] Филд А. Функциональное программирование / А. Филд, П. Харрисон; пер. с англ. М. В. Горбатовой и др. под ред. В. А. Горбатова — М.: Мир, 1993. — 673 с., ил.

[17] Шень А. Программирование: теоремы и задачи (с1) 2-е изд., М.: МЦНМО, 2004. — 296 с. — Режим доступа: http://www.mccme.ru/free-books/shen/shen-progbook.pdf

Учебное издание

Дудаков Сергей Михайлович
Карлов Борис Николаевич

Математическое введение в информатику

Учебник

Усл. п. л. 20,0. Уч.-изд. л. 9,02.
Тираж 10 экз. [электр.].
Заказ № 597 от 04.12.2017.
Тверской государственный университет
Факультет прикладной математики и кибернетики
Адрес: 170100, г. Тверь, ул. Желябова, 33