Федеральное агентство по образованию
Российской Федерации

Тверской государственный
университет

С.М.Дудаков

МАТЕМАТИЧЕСКОЕ ВВЕДЕНИЕ В ИНФОРМАТИКУ

Рекомендовано учебно-методическим советом по прикладной математике и информатике УМО по классическому университетскому образованию в качестве учебного пособия для студентов высших учебных заведений обучающихся по направлению 010500 «Прикладная математика и информатика»

Тверь — 2007

УДК 519.681
ББК З81я731-1
Д 81

Тверской государственный университет
Факультет прикладной математики и кибернетики
http://pmkinfo.tversu.ru

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

Дудаков С.М. Математическое введение в информатику: Учеб. пособие. Тверь: Твер. гос. ун-т, 2003. — 221с.
ISBN 5-7609-0233-4

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

Учебное пособие адресовано, прежде всего, студентам младших курсов, обучающихся по направлению 010500 «Прикладная математика и информатика»

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

ISBN 5-7609-0233-4

Оглавление

Предисловие

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.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 структурных программ: присваивание, следование, ветвление и цикл. Такой набор, с одной стороны, является достаточным для того, чтобы построенный язык был универсальным и чтобы доказанные результаты 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 структурных программ.

Пятая глава посвящена доказательству корректности структурных 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 автору по адресу

Sergey.Dudakov@tversu.ru

Глава 1

Введение

§ 1.1. Основные понятия

С некоторым допущением можно сказать, что информатика — 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 вышеперечисленных 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 программой. Как правило, ЯП являются универсальными, 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 узкого круга задач (например, язык запросов к базам данных — SQL).

[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.

§ 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). Хотя слово «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, основыванный на построении таких треугольников. Там же был известен алгоритм решения квадратного уравнения. Древним грекам был известен алгоритм нахождения 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, то есть по произвольному утверждению о натуральных чисел (записанному, естественно, некоторым специальным образом) 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 частности, все современные ЯП, допускают трансляцию в любую из этих моделей, что является ещё одним подтверждением тезиса Чёрча.

Точная математическая формулировка понятий «алгоритм» и «р а з р е ш и м о с т ь» (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 (Я П п е р в о г о у р о в н я).

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

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

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 универсальным, если он приводит к правильному решению задачи при любых начальных условиях. Алгоритм называют конечным, 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 и памяти 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 конкретной 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 п о л у у н и в е р с а л ь н ы е, то есть рассчитанные на решение задачи не при любых начальных условиях, а при некоторых из них.

Языки программирования можно разделить на две большие группы — д е к л а р а т и в н ы е и и м п е р а т и в н ы е. 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 у н и в е р с а л ь н ы е ЯП — ЯП, которые не предназначены для решения какой-то специальной группы за-

дач, а годятся для решения их широкого круга[3] (C, C++, Pascal, Basic). Другая группа — специализированные ЯП, то есть, ориентированные на решение задач из достаточно узкой 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++, Pascal), ЯП ориентированные на метки (ранние версии Basic'a, 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:

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

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:

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

[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.

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* $b^2 - 4ac \geq 0$.

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 пункту 2.*

В отличие от предыдущего алгоритма, данный алгоритм является и т е р а ц и о н н ы м, *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 $\text{НОД}(x, y)$. Более совершенным является алгоритм Евклида, который основан на следующем утверждении:

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

Пример 1.5 (Алгоритм Евклида)**.**

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

Этот алгоритм является р е к у р с и в н ы м, *то есть для достижения результата используется тот же самый алгоритм, но с другими входными данными. Например, для вычисления* $\text{НОД}(6, 4)$, *нужно будет последовательно вычислить* $\text{НОД}(2, 4)$ *и* $\text{НОД}(2, 2)$.

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* $a \geq b + c$, *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 $n+1$ элементов 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 точно 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)**.** *Значение функции* f *на элементах* $a_1, \ldots, a_n$ — *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)**.** *Если* $A_1 \subseteq A$ *и* f — n*-местная функция, действующая из* A *в* B*, то функция* f_1 — *ограничение* f *на множество* A_1 *(записывается* $f \restriction A_1$*), если* f_1 — n*-местная функция, действующая из* A_1 *в* B *и для любых элементов* $a_1, \ldots, a_n \in A_1$ *выполнено* $f(a_1, \ldots, a_n) = f_1(a_1, \ldots, a_n)$*.*

Пример 2.3. *Рассмотрим множества* A*,* B *и функцию* f *из примера* 2.1*. 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.

Замечание 2.1. Для двухместных функций вместо рассмотренной выше записи значения (префиксная форма — так как символ функции стоит до её аргументов), используется инфиксная форма 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, 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-местная частичная функция (или частичное отображение), 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 не определена. Если такой элемент $b \in B$ существует, то он называется значением 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 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:*

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

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: $h = gf$. Точно так же как и для чисел можно определить степень отображения:

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

n — натуральное число. Считаем, что f^0 — тождественное отображение.

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 т о т а л ь н о й.

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

Определение 2.6. *Пусть A — множество, n — натуральное число. С помощью A^n мы обозначаем м н о ж е с т в о в с е в о з м о ж н ы х у п о р я д о ч е н н ы х н а б о р о в д л и н ы n, элементами которых являются элементы A (д е к а р т о в а с т е п е н ь 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:

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

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 — $b \in B$.

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. $\mathcal{P}(A)$ — *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 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

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

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.

Лемма 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 $x_i \leq x_0 - i$.

Базис индукции.

$x_0 = x_0 - 0$.

Шаг индукции.

Пусть для i доказано.

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

следовательно,

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

Получаем, что

$$x_{x_0-1} \leq x_0 - (x_0 - 1 + 1) = 0.$$

Следовательно, в последовательности не может быть больше x_0 элементов. □

§ 2.2. Графы

Определим, что такое граф и дерево.

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

Задача 2.6. На рисунке 6.5 изображён граф. Выпишите явным образом множество вершин и множество рёбер этого графа.

Обычно необходимо с вершинами и рёбрами графа связать некоторую информацию.

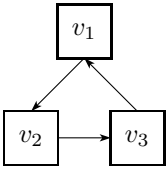


Рис. 2.1: Граф G из примера 2.9.

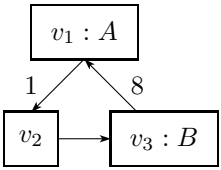


Рис. 2.2: Нагруженный граф (G, f_V, f_E) (пример 2.10).

Определение 2.12 (Нагруженный граф)**.** *Нагруженный (или размеченный) граф — тройка (G, f_V, f_E), где $G = (V, E)$ — граф, f_V и f_E — некоторые (возможно, частичные) функции, определённые на множестве 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 определяет объект, 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)**.** *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 циклом.*

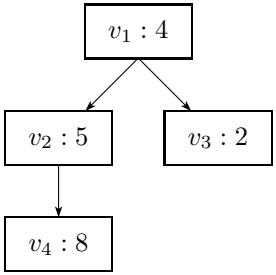


Рис. 2.3: Нагруженное дерево $((V, E), f_V, f_E)$ из примера 2.12.

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. 6.5.

Важную роль в математике играют специального вида графы — деревья.

Определение 2.14 (Дерево)**.** *Граф G называется д е р е в о м, если*

1) *Существует единственная вершина r — корень дерева — из которой имеется путь в любую другую вершину.*

2) *Для любых двух вершин существует не более одного пути из одной в другую.*

Вершины дерева, из которых не выходит ни одно ребро, называются л и с т ь я м и дерева, остальные вершины — в н у т р е н н и м и.

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

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

В этом дереве v_1 — корень.

Задача 2.8. На рисунке 6.6 изображён некоторый граф. Проверьте, что он является деревом. Является ли деревом граф на рисунке 6.5?

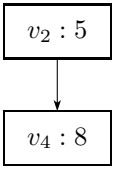


Рис. 2.4: Поддерево из примера 2.13.

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 $\left(V_v, E \cap V_v^2\right)$ 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 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. 6.6 всевозможные поддеревья.

Определение 2.16 (Лес)**.** *Л е с — граф, полученный объединением некоторого количества деревьев, никакие два из которых не имеют общих вершин.*

§ 2.3. Математическая логика

Мы будем рассматривать только формулы элементарной арифметики, и все определения будут относиться только к элементарной арифметике и её стандартной модели $(\omega, +, \times)$. 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 2.17 (Формула первого порядка)**.** *Пусть зафиксировано некоторое множество V для обозначения переменных. А т о м н ы е ф о р м у л ы — формулы вида*

$$P\left(e_1, \ldots, e_n\right),$$

где P — какое-либо отношение, $e_1, \ldots, e_n$ — выражения (термы)[1]*. Мы*

[1]Пока мы не уточняем, что такое выражение. Мы полагаем, что для понимания

будем записывать отношения в инфиксной форме для стандартных отношений (= или <). Если φ, ψ — формулы, $x \in V$ — переменная то

$$\begin{array}{cc} (\varphi \wedge \psi) & (\varphi \vee \psi) \\ (\varphi \to \psi) & (\neg\varphi) \\ (\exists x)(\varphi) & (\forall x)(\varphi) \end{array}$$

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

Значением формулы на состоянии будет ИСТИНА *(**И**)* или ЛОЖЬ *(**Л**). Для атомной формулы значением является* ИСТИНА*, если соответствующее отношение выполняется, и* ЛОЖЬ *— если нет.*

Значение формул, построенных с помощью булевых связок, определяются по таблице:

этого материала достаточно считать e_i арифметическими выражениями в интуитивном смысле, например, $(x + 3) \times y$ или $xy + z$.

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \to \psi$	$\neg\varphi$
Л	**Л**	**Л**	**Л**	**И**	**И**
Л	**И**	**Л**	**И**	**И**	**И**
И	**Л**	**Л**	**И**	**Л**	**Л**
И	**И**	**И**	**И**	**И**	**Л**

Значение формул, построенных с помощью кванторов, определяется так. Формула $(\exists x)(\varphi)$ *истинна на состоянии* σ*, если формула* φ *истинна на некотором состоянии* τ*, которое отличается от* σ *только значением на* x*. Формула* $(\forall x)(\varphi)$ *истинна на состоянии* σ*, если формула* φ *истинна на любом состоянии* τ*, которое отличается от* σ *только значением на* x*. 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* $(\exists x)$ *называют квантором существования по переменной* x*, а* $(\forall x)$ — *квантором всеобщности по переменной* x*.*

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

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

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

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

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* ЛОЖЬ*. Значит формула* $(\forall y)\ x + z^2 < y + zy$ *на состоянии* σ *ложна.*

Задача 2.12. Для каждой формулы из примера 2.14 постройте пример состояния, когда её значение отличается от того, которое было найдено в примере 2.15. 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 — полный квадрат.
3) Существует треугольник со сторонами σx, σy и σz.

Определение 2.19. *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)**.** *К о н с т а н т а в Я П — 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)**.** *О п е р а ц и я в Я П — 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 $+, -, \times, /$.

В нашем ЯП мы введем только одну операцию — succ, которая будет означать прибавление единицы: $\mathsf{succ}\,(0) = 1, \mathsf{succ}\,(3) = 4$. Как мы увидим, все остальные операции можно определить через succ, и можно написать любую программу, используя только эту операцию. Заметим, что операция succ определена для любого элемента нашей предметной области. Поэтому случай, когда результат операции неопределен (например, в случае деления на ноль) мы пока рассматривать не будем.

Кроме того, нам потребуется какой-то способ для запоминания промежуточных результатов вычисления. Для этого используется следующий элемент ЯП — п е р е м е н н ы е. Каждая переменная имеет некоторое имя. Мы будем считать, что всевозможные имена переменных составляют некоторое счётное множество $\mathfrak{V}$. Кроме того, мы 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 ЯП 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.

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 $\overline{x}$ или $\overline{y}$ соответственно.

Знаком $\circeq$ мы будем обозначать графическое совпадение синтаксических объектов (строк).

Пример 3.1. *Пусть*

$$m \circeq abc = def$$

$$n \circeq xyz \circ abd$$

Тогда

$$mn \circeq abc = defxyz \circ abd$$

Если a, b, c — синтаксические объекты и $a \circeq bc$, то b называют префиксом a, c — суффиксом a. Если префикс (суффикс) a отличается от самого a, то такой префикс (суффикс) называют собственным.

В любом языке структурного программирования имеется оператор, который позволяет сохранить в переменной результат. Чтобы определить такой оператор для нашего ЯП введем 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, $\mathsf{succ}(0)$, $\mathsf{succ}(\mathsf{succ}(x))$ — *выражения.*

Минимальным элементом ЯП, который означает выполнение каких-либо действий, является оператор. Для сохранения промежуточных результатов в нашем ЯП будет служить оператор присваивания.

Определение 3.5 (Оператор присваивания)**.** *Оператор присваивания — запись 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 имеется почти во всех языках программирования. Этот оператор будет служить нам «кирпичом», из которого мы будем строить все наши программы.

Пример 3.3. *Примеры операторов присваивания:*

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

Следующее наше понятие — тест.

Определение 3.6 (Тест)**.** *Тест — формула вида*

$$e_1 \;=\; e_2$$

или вида

$$e_1 \;<\; e_2,$$

где e_1 и e_2 — выражения. Часто, если конкретный знак сравнения не имеет для нас значения, то мы будем записывать тест в виде

$$e_1 \;\circ\; e_2$$

подразумевая, что вместо $\circ$ может стоять или $<$, или $=$.

Пример 3.4. *Примеры тестов:*

$$x \;<\; \mathsf{succ}\,(0)$$

$$\mathsf{succ}\,(x) \;=\; \mathsf{succ}\,(\mathsf{succ}\,(y))\,.$$

Замечание 3.1. Заметим, что оператор присваивания

$$x \;=\; e;$$

отличается от теста

$$x \;=\; e$$

наличием в конце точки с запятой (;).

Теперь мы можем определить, что такое структурированная программа.

Определение 3.7 (Структурированная программа)**.** *Как и в случае с выражениями мы будем давать определение структурированной (или структурной) программы по индукции.*

1. *Оператор присваивания* $x = e$; *является программой.*

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

3. *Неполное ветвление. Если* Π_1 *— программа, а* T *— тест, то*

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

   является программой.

4. *Полное ветвление. Если* Π_1 *и* Π_2 *— программы, а* T *— тест, то*

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

   является программой.

5. *Цикл. Если* Π_1 *— программа, а* T *— тест, то*

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

   является программой.

Рассмотрим несколько примеров.

Пример 3.5. *Операторы присваивания*

$$x \;=\; 0;$$

и

$$y \;=\; \mathsf{succ}\,(x)\,;$$

являются программами. Поэтому их соединение:

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

тоже является программой. Также программами являются и следующие конструкции, которые получены с помощью пунктов 3, 4 и 5 соответственно:

1. $$\begin{cases} \mathsf{if}\ x\ =\ 0\ \mathsf{then} \\ \quad x\ =\ 0; \\ \quad y\ =\ \mathsf{succ}\,(x)\,; \\ \mathsf{end}; \end{cases}$$

2. $$\begin{cases} \mathsf{if}\ x\ <\ \mathsf{succ}\,(y)\ \mathsf{then} \\ \quad x\ =\ \mathsf{succ}\,(y)\,; \\ \mathsf{else} \\ \quad y\ =\ 0; \\ \mathsf{end}; \end{cases}$$

3. $$\begin{cases} \mathsf{while}\ \mathsf{succ}\,(x)\ <\ y\ \mathsf{do} \\ \quad z\ =\ \mathsf{succ}\,(z)\,; \\ \quad y\ =\ \mathsf{succ}\,(x)\,; \\ \mathsf{end}; \end{cases}$$

Определение 3.8 (Множество переменных)**.** *Определим множество* $\mathbf{Var}\,(\alpha)$ — *множество переменных объекта* α *Для выражений:*

1) $\mathbf{Var}\,(x) = \{x\}$, *если* x — *переменная;*

2) $\mathbf{Var}\,(c) = \varnothing$, *если* c — *константа;*

3) $\mathbf{Var}\,(o\,(e_1, \ldots, e_n)) = \mathbf{Var}\,(e_1) \cup \cdots \cup \mathbf{Var}\,(e_n)$, *если* o — n-*местная операция,* $e_1, \ldots, e_n$ — *выражения.*

Для теста $T \backsimeq e_1 \circ e_2$*:*

$$\mathbf{Var}\,(T) = \mathbf{Var}\,(e_1) \cup \mathbf{Var}\,(e_2)\,.$$

Для программы Π *мы определим два множества* $\mathbf{Var}\,(\Pi)$ *и* $\mathbf{LVar}\,(\Pi)$ — *множество изменяемых переменных:*

1. $\Pi \backsimeq x = e;$. $\mathbf{Var}\,(\Pi) = \{x\} \cup \mathbf{Var}\,(e)$, $\mathbf{LVar}\,(\Pi) = \{x\}$.

2. $\Pi \backsimeq \Pi_1\Pi_2$
$$\mathbf{Var}\,(\Pi) = \mathbf{Var}\,(\Pi_1) \cup \mathbf{Var}\,(\Pi_2)\,,$$
$$\mathbf{LVar}\,(\Pi) = \mathbf{LVar}\,(\Pi_1) \cup \mathbf{LVar}\,(\Pi_2)\,.$$

3.
$$\Pi \multimap \begin{cases} \text{if } T \text{ then} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$
$\mathbf{Var}(\Pi) = \mathbf{Var}(T) \cup \mathbf{Var}(\Pi_1)$, $\mathbf{LVar}(\Pi) = \mathbf{LVar}(\Pi_1)$.

4.
$$\Pi \multimap \begin{cases} \text{if } T \text{ then} \\ \quad \Pi_1 \\ \text{else} \\ \quad \Pi_2 \\ \text{end;} \end{cases}$$
$$\mathbf{Var}(\Pi) = \mathbf{Var}(T) \cup \mathbf{Var}(\Pi_1) \cup \mathbf{Var}(\Pi_2),$$
$$\mathbf{LVar}(\Pi) = \mathbf{LVar}(\Pi_1) \cup \mathbf{LVar}(\Pi_2).$$

5.
$$\Pi \multimap \begin{cases} \text{while } T \text{ do} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$
$\mathbf{Var}(\Pi) = \mathbf{Var}(T) \cup \mathbf{Var}(\Pi_1)$, $\mathbf{LVar}(\Pi) = \mathbf{LVar}(\Pi_1)$.

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

Пример 3.6. *Для программ из примера 3.5:*

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

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.

Определение 3.9 (Алгоритм)**.** *Будем называть а л г о р и т м о м следующую запись:*

Alg *Имя алгоритма*;
arg *Список входных переменных*;
Тело алгоритма
end;

```
Alg Max;
arg x, y;
  if x < y then
    Max = y;
  else
    Max = x;
  end;
end;
```

Рис. 3.1: Наибольшее из двух чисел.

После Alg *указывается имя, которое мы даём алгоритму (и м я а л г о р и т м а), после* arg *— список разделенных запятыми переменных, в которых задаются начальные условия (в х о д н ы е п е р е м е н н ы е). Каждая переменная может появляться в этом списке не более одного раза. Т е л о а л г о р и т м а (программу) A, мы будем обозначать Π_A. Среди всех переменных мы будем выделять одну — в ы х о д н у ю п е р е м е н н у ю. Выходная переменная будет иметь такое же имя, как сам алгоритм. Выходная переменная не может являться частью выражения. Следовательно, она может использоваться только в левой части оператора присваивания. Выходная переменная не может быть входной переменной.*

Пример 3.7. *Рассмотрим алгоритм на рис. 3.1. В данном случае, имя алгоритма — Max, входные переменные — x и y. Тело алгоритма:*

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

Результат получается в выходной переменной Max. Заметим, что в данной программе нельзя использовать операторы вида

```
x = Max;
```

или

```
if Max < y then
  ⋮
end;
```

потому что в этом случае Max было бы выражением, а выходную переменную в выражениях использовать запрещено.

Лемма 3.1 (О сбалансированности)**.** *В любой программе* П *сумма количества 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, то $I(\Pi) = E(\Pi) = 0$.

Шаг индукции.

1. Если $\Pi \twoheadleftarrow \Pi_1\Pi_2$, 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 $\Pi_3 \twoheadleftarrow \Pi_1\Pi_4$, где Π_4 — префикс Π_2. По индукционному предположению $I(\Pi_1) = E(\Pi_1)$ и $I(\Pi_4) \geq E(\Pi_4)$. Следовательно,

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

2. Пусть

$$\Pi \twoheadleftarrow \begin{cases} \textsf{if } T \textsf{ then} \\ \quad \Pi_1 \\ \textsf{end;} \end{cases}$$

   Π_1 — программа. Тогда

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

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

$$\Pi_3 \twoheadleftarrow \textsf{if } T \textsf{ then } \Pi_4$$

   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* $\Pi \backsimeq \Pi_1\Pi_2$, Π *и* Π_1 — *программы,* $\Pi_1 \neq \Pi$, *то* Π_2 — *программа.*

Доказательство. Индукция по построению Π.

Базис индукции.

Пусть $\Pi \backsimeq x = e$; — оператор присваивания. 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 $\Pi_1 \backsimeq \Pi$. Противоречие. Следовательно, такой случай невозможен.

Шаг индукции.

1. Пусть

$$\Pi \backsimeq \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_3 \\ \mathsf{else} \\ \quad \Pi_4 \\ \mathsf{end;} \end{cases}$$

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:

$$\begin{aligned} \Pi_1 &\backsimeq \mathsf{if}\ T\ \mathsf{then}\ \Pi_3' \\ \Pi_2 &\backsimeq \Pi_3''\ \mathsf{else}\ \Pi_4\ \mathsf{end;} \end{aligned}$$

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

$$\begin{aligned} \Pi_1 &\backsimeq \mathsf{if}\ T\ \mathsf{then}\ \Pi_3\ \mathsf{else}\ \Pi_4' \\ \Pi_2 &\backsimeq \Pi_4''\ \mathsf{end;} \end{aligned}$$

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 в 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:

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

Если $\Pi_1 \circ\!\!- \Pi_3$, то, очевидно, $\Pi_2 \circ\!\!- \Pi_4$. Иначе, граница между Π_1 и Π_2 проходит внутри Π_3 или Π_4.

Первый случай:

$$\Pi \circ\!\!- \Pi_1\Pi_5\Pi_4$$

где $\Pi_1\Pi_5 \circ\!\!- \Pi_3$. Π_3 и Π_1 — программы и $\Pi_1 \not\equiv \Pi_3$. По индукционному предположению Π_5 — программа. Но тогда $\Pi_2 \circ\!\!- \Pi_5\Pi_4$ — программа.

Второй случай:

$$\Pi \circ\!\!- \Pi_3\Pi_5\Pi_2$$

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

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

Следствие 3.1 (О двойном представлении программы)**.** *Пусть* $\Pi, \Pi_1, \Pi_2, \Pi_3, \Pi_4$ *— программы,* $\Pi \circ\!\!- \Pi_1\Pi_2$, $\Pi \circ\!\!- \Pi_3\Pi_4$ *и* $\Pi_1 \not\equiv \Pi_3$. *Тогда существует программа* Π_5 *такая, что* $\Pi \circ\!\!- \Pi_3\Pi_5\Pi_2$ *или* $\Pi \circ\!\!- \Pi_1\Pi_5\Pi_4$.

*__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 3.10 (Состояние)**.** *Состояние — частичное отображение множества переменных в предметную область. Значение переменной* x *на состоянии* σ *—* σx.

Пример 3.8. *Например, если множество переменных*

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

то следующие отображения будут состояниями:

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

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

Определение 3.11 (Состояние программы)**.** *Состояние σ называется состоянием программы* Π*, если*

$$\operatorname{dom}\sigma \supseteq \mathbf{Var}(\Pi).$$

Иначе говоря, всем переменным, которые встречаются в программе Π*, состояние σ приписывает некоторые значения.*

В дальнейшем, говоря о каких-либо состоянии σ и программе Π мы всегда считаем, что σ — состояние Π.

Задача 3.5. Определите, будут ли состояния из примера 3.8 состояниями программ из примера 3.5.

Сначала определим семантику выражений.

Определение 3.12 (Значение выражения)**.** *Пусть e — выражение, σ — состояние. Значением выражения e на состоянии 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, $\sigma(\mathsf{succ}(e)) = \sigma(e) + 1$.

Рассмотрим пример.

Пример 3.9. *Будем рассматривать состояния σ_1 и σ_2 из примера 3.8.*

$$\sigma_1(0) = 0,$$
$$\sigma_1(x) = 1,$$
$$\sigma_2(\mathsf{succ}(y)) = \sigma_2(y) + 1 = 51,$$
$$\sigma_2(\mathsf{succ}(\mathsf{succ}(x))) = \sigma_2(\mathsf{succ}(x)) + 1 = \sigma_2(x) + 1 + 1 = 102.$$

Задача 3.6. Найдите значения выражений $\mathsf{succ}(z)$ и $\mathsf{succ}(\mathsf{succ}(z))$ на состояниях из примера 3.8.

Следующий наш шаг — определение семантики для тестов.

Определение 3.13 (Истинность теста)**.** *Пусть e_1 и e_2 — выражения. Истинность теста $e_1 = e_2$ на состоянии σ означает, что $\sigma(e_1) = \sigma(e_2)$. Истинность теста $e_1 < e_2$ на состоянии σ означает, что $\sigma(e_1) < \sigma(e_2)$. Истинность теста T на состоянии σ записывается как $\sigma \models T$.*

Ложность теста T на состоянии σ (обозначается $\sigma \not\models T$) — это отсутствие истинности.

Значение теста T (обозначается $\sigma(T)$) на состоянии σ — ИСТИНА, *если $\sigma \models T$, и* ЛОЖЬ, *в противном случае.*

Пример 3.10. *Рассмотрим те же состояния из примера 3.8. Тогда*

$$\sigma_1 \models x < y,$$
$$\sigma_1 \models \mathsf{succ}(x) = y,$$
$$\sigma_2 \not\models z = x.$$

Задача 3.7. Используя переменные x и y, напишите тесты, которые были бы истинны на состоянии σ тогда и только тогда, когда

1) $\sigma x \geq \sigma y$;
2) $\sigma x \geq \sigma y - 1$.

Теперь мы готовы к тому, чтобы определить семантику нашего ЯП. Пусть Σ_Π — множество всевозможных состояний программы Π.

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

Определение 3.14 (Семантика программы)**.** *Семантика программы Π определяется индукцией по построению программы.*

1. *Пусть*
$$\Pi \twoheadleftarrow x = e;$$
где x — переменная, e — выражение. Пусть σ — состояние. Тогда $\Pi(\sigma) = \tau$, где τ — состояние, которое 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 $\tau x = \sigma(e)$.

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

3. *Пусть*

$$\Pi \Leftarrow \begin{cases} \text{if } T \text{ then} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$

Тогда

1) *если* $\sigma \models T$, *то* $\Pi(\sigma) = \Pi_1(\sigma)$;
2) *если* $\sigma \not\models T$, *то* $\Pi(\sigma) = \sigma$.

4. *Пусть*

$$\Pi \Leftarrow \begin{cases} \text{if } T \text{ then} \\ \quad \Pi_1 \\ \text{else} \\ \quad \Pi_2 \\ \text{end;} \end{cases}$$

Тогда

1) *если* $\sigma \models T$, *то* $\Pi(\sigma) = \Pi_1(\sigma)$;
2) *если* $\sigma \not\models T$, *то* $\Pi(\sigma) = \Pi_2(\sigma)$.

5. *Для цикла семантика определяется несколько более сложным образом. Пусть*

$$\Pi \Leftarrow \begin{cases} \text{while } T \text{ do} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$

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

а) $\sigma^0 = \sigma$;
б) $\Pi_1\left(\sigma^i\right)$ *определено для* $i = 0, \ldots, n-1$ *и* $\sigma^{i+1} = \Pi_1\left(\sigma^i\right)$;
в) $\sigma^i \models T$ *тогда и только тогда, когда* $i < n$, *то есть* $\sigma^i \models T$, *если* $i < n$, *и* $\sigma^n \not\models T$.

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

Определение 3.15 (Зацикливание программы)**.** *Если* $\Pi(\sigma)$ *не определено для программы* Π *и состояния* σ*, то говорим, что программа* Π *на состоянии* σ *з а ц и к л и в а е т с я. В противном случае говорим, что программа о с т а н а в л и в а е т с я.*

Следствие 3.2 (Условие зацикливания)**.** *Пусть*

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

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

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*

$$\Pi \Leftarrow \begin{cases} \mathsf{if}\ x\ <\ y\ \mathsf{then} \\ \quad x\ =\ z; \\ \mathsf{end}; \end{cases}$$

Так как $\sigma \models x < y$*, то*

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

5.
$$\Pi \eqsim \begin{cases} \mathsf{if}\ z\ <\ y\ \mathsf{then} \\ \quad x\ =\ z; \\ \mathsf{end}; \end{cases}$$

Тогда $\sigma \not\models z < y$. *Поэтому* $\Pi(\sigma) = \sigma$.

6. *Для полного ветвления*

$$\Pi \eqsim \begin{cases} \mathsf{if}\ z\ <\ y\ \mathsf{then} \\ \quad x\ =\ z; \\ \mathsf{else} \\ \quad y\ =\ \mathsf{succ}(0); \\ \mathsf{end}; \end{cases}$$

Так как $\sigma \not\models z < y$, *то*

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

7. *Пусть*

$$\Pi \eqsim \begin{cases} \mathsf{while}\ x\ <\ z\ \mathsf{do} \\ \quad x\ =\ \mathsf{succ}(x); \\ \mathsf{end}; \end{cases}$$

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

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

Очевидно, что последовательность точно соответствует определению семантики для цикла. Далее, $\sigma^0 \models x < z$, $\sigma^1 \models x < z$, $\sigma^2 \not\models x < z$. *Следовательно,* $\Pi(\sigma) = \sigma^2$.

8. *Рассмотрим другую программу:*

$$\Pi \eqsim \begin{cases} \mathsf{while}\ x\ <\ z\ \mathsf{do} \\ \quad z\ =\ \mathsf{succ}(z); \\ \mathsf{end}; \end{cases}$$

Очевидно, что для состояния σ *последовательность* $\left(\sigma^i\right)_i$ *будет состоять из элементов*

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

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

Задача 3.9. Определите $\Pi(\sigma)$ для программ из примера 3.5 и состояний из примера 3.8.

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

Следствие 3.3. *Если программа* Π *не содержит циклов, то* $\Pi(\sigma)$ *определено для любого состояния* σ*.*

Задача 3.10. Докажите следствие.

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

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

Если для всякого состояния σ *из* $\sigma \models T$ *следует, что* $\Pi_1(\sigma) \models T$*, то* $\Pi(\sigma)$ *определено тогда и только тогда, когда* $\sigma \not\models T$*.*

Задача 3.11. Докажите следствие.

Лемма 3.3 (Условие зацикливания)**.** *Если для цикла*

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

на σ *последовательность состояний* $\left(\sigma^i\right)$ *содержит два одинаковых элемента* σ^k *и* σ^l $(k < l)$ *и* $\sigma^i \models T$ *для всех* $i \leq l$*, то* $\Pi(\sigma)$ *не определено.*

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

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

Кроме того, для всякого $i \in [k; l)$ выполняется $\sigma^i \models T$. Рассмотрим произвольное $j \geq k$. Число j может быть представлено в виде

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

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

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

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 $\sigma^j \models T$. Итак мы выяснили, что для всякого j имеет место $\sigma^j \models T$. Это означает, что $\Pi(\sigma)$ не определено. □

Теперь рассмотрим затруднение, о котором мы говорили, когда определяли семантику следования. Очевидно, что для 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 одно и то же? Мы докажем, что противоречия не возникает, пользуясь тем или иным представлением мы получим одно и то же определение семантики.

Теорема 3.1 (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 одна и та 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 $\Pi_1 \not\equiv \Pi_3$. По следствию 3.1 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.

Рассмотрим произвольное состояние 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 $\Pi \backsimeq \Pi_1\Pi_5\Pi_4$.

Определение 3.16 (Вызов алгоритма)**.** *Вызовом алгоритма A, имеющего n аргументов, мы будем называть набор*

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

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

Начальное состояние вызова δ — отображение $\mathbf{Var}(\Pi_A)$ *в предметную область такое, что*

$$\sigma_\delta(x) = \begin{cases} v_i, \text{если } x \text{ — } i\text{-ый аргумент } A, \\ 0, \text{иначе.} \end{cases}$$

Результат вызова δ (обозначается $\mathbf{Res}\,\delta$*) — это $\Pi_A(\sigma_\delta)(A)$, если $\Pi_A(\sigma_\delta)$ определено. Если $\Pi_A(\sigma_\delta)$ неопределено, то говорим, что результат вызова δ не определён.*

Задача 3.13. Напишите начальные состояния для вызовов алгоритмов на рис. 3.1 и 3.2:

$$(Max, 3, 6),$$
$$(Add, 2, 3).$$

Найдите результат этих вызовов.

Определение 3.17 (Вычисление функции алгоритмом)**.** *Пусть дан алгоритм*

Alg A;
arg $x_1, \ldots, x_n$;
Π
end;

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

1) $\mathbf{Res}\,\delta$ *определён тогда и только тогда, когда $\Phi_A(\overline{v})$ определено;*

```
Alg Add;
arg x, y;
  u = x; v = 0;
  while v < y do
    u = succ(u);
    v = succ(v);
  end;
  Add = u;
end;
```

Рис. 3.2: Сложение чисел.

2) *Если* $\mathbf{Res}\,\delta$ *определён, то* $\mathbf{Res}\,\delta = \Phi_A(\overline{v})$.

Рассмотрим несколько примеров.

Пример 3.12. *Алгоритм Max на рис. 3.1 вычисляет функцию* $f(x,y) = \max\{x,y\}$. *Например,*

$$\mathbf{Res}\,(Max,1,3) = 3;$$
$$\mathbf{Res}\,(Max,5,2) = 5.$$

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

Теперь напишем несколько алгоритмов для вычисления арифметических функций.

Пример 3.13. *Начнём со сложения (рис. 3.2).*

Прежде всего отметим, что для любого состояния τ *имеет место*

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

Рассмотрим произвольное состояние

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

После выполнения первых двух операторов получим новое состояние

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

Начнем строить последовательность состояний $\left(\sigma^i\right)_i$ *для цикла:*

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

```
Alg S;
arg x, y;
  u = y; v = 0;
  while u < x do
    u = succ(u);
    v = succ(v);   } Π2
  end;
  S = v;
end;
```

Рис. 3.3: Вычитание чисел.

Очевидно, что

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

Далее, $\sigma^i \models v < y$ *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 $\sigma^i \models v < y$. *Из этого следует, что* $\Pi_1(\sigma) = \sigma^{y_0}$ *и*

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

По определению семантики присваивания имеем

$$\Pi(\sigma)(Add) = \sigma x + \sigma y.$$

Это и означает, что наш алгоритм выполняет сложение натуральных чисел.

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

Пример 3.14. *Теперь вычитание (рис. 3.3). Напомним, что у нас нет отрицательных 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* $x \geq 0$, *и* $\lfloor x \rfloor = 0$, *если* $x \leq 0$.

Рассмотрим произвольное состояние

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

Перед началом цикла мы получим состояние

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

Построим последовательность состояний $\left(\sigma^i\right)_i$ *для цикла:*

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

Очевидно, что

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

Для теста имеем:

$$\sigma^i \models u < x \iff y_0 + i < x_0.$$

Если $x_0 \leq y_0$, *то последним элементом последовательности будет* σ^0, *так как для него уже будет* $\sigma^0 \not\models u < x$.

Если же $x_0 \geq y_0$, *то последним элементом будет первое* σ^j, *для которого* $\sigma^j \not\models u < x$, *то есть* $y_0 + j \geq x_0$. *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

$$\Pi(\sigma)(S) = \begin{cases} 0, & \text{если } \sigma x \leq \sigma y \\ \sigma x - \sigma y, & \text{если } \sigma x \geq \sigma y \end{cases}$$

Это точно соответствует тому, что мы хотели.

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

Пример 3.15. *Рассмотрим пример алгоритма, для вычисления НОД методом Евклида. См. рис.* 3.4.

В (1) *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.*

В нашем языке программирования нет теста, который проверял бы аргументы на неравенство. Поэтому мы включили два специальных фрагмента (2). *В результате их выполнения переменная* u *содержит результат сравнения.*

Строки (4) *содержат обычный шаг алгоритма Евклида, когда от большего числа отнимается меньшее. Так как вычитание в нашем языке отсутствует, то мы используем строки* (3) *для его выполнения.*

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

Задача 3.17. Напишите алгоритмы, которые выполняют:

1) умножение двух чисел;

```
Alg Euclid;
arg x, y;
  if x = 0 then                                                    ⎫
    Euclid = y;                                                    ⎪
  else                                                             ⎪
    if y = 0 then                                                  ⎪ (1)
      Euclid = x;                                                  ⎪
    else                                                           ⎭
      if x = y then                                                ⎫
        u = 0;                                                     ⎪
      else                                                         ⎪ (2)
        u = succ(0);                                               ⎪
      end;                                                         ⎭
      while u = succ(0) do                                         ⎫
        if x < y then                                              ⎪
          v = 0; w = x;                                            ⎪
          while w < y do                                           ⎫⎪
            v = succ(v); w = succ(w);                              ⎬⎪ (3)
          end;                                                     ⎭⎪
          y = v;                                                   ⎪
        else                                                       ⎪
          v = 0; w = y;                                            ⎪
          while w < x do                                           ⎫⎪
            v = succ(v); w = succ(w);                              ⎬⎪ (3)
          end;                                                     ⎭⎪
          x = v;                                                   ⎪ (4)
        end;                                                       ⎪
        if x = y then                                              ⎪
          u = 0;                                                   ⎪
        else                                                       ⎪ (2)
          u = succ(0);                                             ⎪
        end;                                                       ⎭
      end;
      Euclid = x;
    end;
  end;
end;
```

Рис. 3.4: Алгоритм Евклида.

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 3.18 (Эквивалентность алгоритмов)**.** *Алгоритмы A и B называются э к в и в а л е н т н ы м и, если они вычисляют одну и ту же (частичную) функцию:*

$$\Phi_A = \Phi_B.$$

Следствие 3.5. *Алгоритмы A и B эквивалентны тогда и только тогда, когда они имеют одинаковое число аргументов n и для любых $\overline{v} \in \omega^n$ или $\mathbf{Res}(A, \overline{v})$ и $\mathbf{Res}(B, \overline{v})$ одновременно неопределены, или одновременно определены и при этом*

$$\mathbf{Res}(A, \overline{v}) = \mathbf{Res}(B, \overline{v}).$$

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

Если $\Pi_A(\sigma) = \Pi_B(\sigma)$ для любого состояния σ, то A и B эквивалентны?

Верно ли обратное к нему утверждение? Докажите.

Пример 3.16. *Рассмотрим следующие алгоритмы.*

Alg A;	Alg B;	Alg C;
arg $x,\ y$;	arg $a,\ b$;	arg $u,\ v$;
$A\ =\ y$;	$B\ =\ b$;	$C\ =\ u$;
end;	end;	end;

Эти три алгоритма вычисляют двухместные функции:

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

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

§ 3.3. *Свойства структурных программ

Изучим некоторые свойства структурированных программ.

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

Лемма 3.4 (О сохранении значения)**.** *Если* Π *— программа,* x *— переменная* $x \notin \mathbf{LVar}(\Pi)$*,* σ*,* τ *— состояния и* $\Pi(\sigma) = \tau$*, то* $\sigma x = \tau x$*.*

Доказательство. Индукция по построению программы Π.

Базис индукции.

$\Pi \twoheadleftarrow y = e$; где y — переменная, e — выражение. Очевидно, что $y \neq x$, иначе $x \in \mathbf{LVar}(\Pi)$. 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 $\Pi \twoheadleftarrow \Pi_1 \Pi_2$. По индукции для любого состояния σ если $\Pi_1(\sigma) = \rho$, то $\sigma x = \rho x$. По индукции для любого состояния 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

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end;} \end{cases}$$

   По индукции для любого состояния σ

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

   Если $\sigma \models T$, 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

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end;} \end{cases}$$

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

$$\sigma^0 = \sigma,\ \sigma^{i+1} = \Pi_1\left(\sigma^i\right),\ \sigma^i \models T \iff i < n,\ \sigma^n = \tau.$$

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

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

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

□

Задача 3.19. Обоснуйте шаг для неполного ветвления.

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

Лемма 3.5 (О неиспользуемых переменных)**.** *Для любой программы* Π *и любого состояния* σ

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

Доказательство. Индукция по построению программы. □

Задача 3.20. Докажите лемму.

Задача 3.21. Докажите эту же лемму с заменой множества $\mathbf{Var}(\Pi)$ на произвольное множество переменных V такое, что $\mathbf{Var}(\Pi) \subseteq V$.

Следствие 3.6 (О двух состояниях)**.** *Если*

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

то

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

Следствие 3.7 (О результате)**.** *Если* $\Pi(\sigma)$ *определено, то*

$$\Pi(\sigma) = \Pi(\sigma \restriction \mathbf{Var}(\Pi)) \cup \sigma \restriction (\operatorname{dom}\sigma \setminus \mathbf{Var}(\Pi)).$$

Задача 3.22. Докажите оба следствия.

Определение 3.19 (Замена переменных)**.** *Если* a *— это выражение, тест или программа,* $x_1, \ldots, x_n$ *— переменные,* $e_1, \ldots, e_n$ *— выражения, то с помощью*

$$(a)\begin{smallmatrix} x_1 \ldots x_n \\ e_1 \ldots e_n \end{smallmatrix}$$

мы будем обозначать результат з а м е н ы в a *всех переменных* x_i *на выражения* e_i*.*

Пример 3.17. *Пусть* $e \circ\!\!- \mathsf{succ}(x)$. *Тогда* $(e)^{x}_{y} \circ\!\!- \mathsf{succ}(y)$.

Замену можно определить и индукцией по построению выражений, тестов и программ.

Определение 3.20 (Замена переменных)**.** *Для выражений:*

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

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

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

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

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

Для теста $T \circ\!\!- d_1 \circ d_2$, *где* d_1, d_2 — *выражения:*

$$(T)^{x_1\ldots x_n}_{e_1\ldots e_n} \circ\!\!- (d_1)^{x_1\ldots x_n}_{e_1\ldots e_n} \circ (d_2)^{x_1\ldots x_n}_{e_1\ldots e_n}$$

***Задача 3.23.** Дайте определение замены для программы индукцией по построению программы.

Лемма 3.6 (О результате замены)**.** *Если* $\overline{x}$ — *переменные,* $\overline{e}$ — *выражения, то*

1) *если* d — *выражение, то* $(d)^{\overline{x}}_{\overline{e}}$ — *выражение;*

2) *если* T — *тест, то* $(T)^{\overline{x}}_{\overline{e}}$ — *тест.*

Доказательство. Индукция по построению выражения d или теста T. □

***Задача 3.24.** 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 $x_i \in \mathbf{LVar}(\Pi)$.

Задача 3.25. Приведите пример, доказывающий необходимость этого условия.

Покажем, что это условие является одновременно и достаточным.

Лемма 3.7 (О результате замены)**.** *Пусть* Π — *программа,* $\overline{x}$ — *переменные,* $\overline{e}$ — *выражения, причём если* $x_i \in \mathbf{LVar}(\Pi)$, *то* e_i — *переменная. Тогда* $(\Pi)^{\overline{x}}_{\overline{e}}$ — *программа.*

Доказательство. Индукция по построению программы Π. □

***Задача 3.26.** 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 3.18. *Пусть* $T \multimap x < y$. *Тогда* $(T)^{xy}_{yz} \multimap y < z$, *то есть мы заменили* x *на* y, *а* y *на* z. *Если бы мы осуществляли замену не одновременно, а последовательно, то получили бы следующее*

$$\left((T)^{x}_{y}\right)^{y}_{z} \multimap (y < y)^{y}_{z} \multimap z < z.$$

Как видим, результат получился другой.

Введём понятие замены переменной для состояний.

Определение 3.21 (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 3.19. *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 3.8. *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 3.27. Докажите, лемму.

Задача 3.28. Приведите пример, доказывающий, что условие из леммы не является необходимым.

Отметим одно из очевидных свойств замены.

Лемма 3.9 (Обратимость замены)**.** *Пусть a — выражение, тест, программа или состояние. $x_1, \ldots, x_n$ — 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 x_i всюду появятся 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 3.29.** Докажите лемму индукцией по определению выражений, тестов и программ.

Следующие три леммы дают основные свойства замены переменных в выражениях, тестах и программах.

Лемма 3.10 (Замена переменных в выражениях)**.** *Пусть переменные $\overline{y}$ не входят в выражение e и состояние σ. Тогда*

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

для любых переменных $\overline{x}$.

Доказательство. Индукция по построению выражения e.

Базис индукции.

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

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

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

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

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

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

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

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

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

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

Далее,

$$(\sigma)^{\overline{x}}_{\overline{y}}((e)^{\overline{x}}_{\overline{y}}) = o((\sigma)^{\overline{x}}_{\overline{y}}((e_1)^{\overline{x}}_{\overline{y}}), \ldots, (\sigma)^{\overline{x}}_{\overline{y}}((e_m)^{\overline{x}}_{\overline{y}})) = \\ = o(\sigma(e_1), \ldots, \sigma(e_m)) = \sigma(e).$$

Лемма 3.11 (Замена переменных в тестах)**.** *Пусть переменные $\overline{y}$ не входят в состояние σ и тест T. Тогда*

$$(\sigma)^{\overline{x}}_{\overline{y}} \models (T)^{\overline{x}}_{\overline{y}} \iff \sigma \models T$$

для любых переменных $\overline{x}$.
Доказательство. По определению если $T \twoheadrightarrow e_1 \circ e_2$, e_1, e_2 — выражения, то

$$\rho \models T \iff \rho(e_1) \circ \rho(e_2).$$

Согласно лемме 3.10 имеем:

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

где $i = 1, 2$. Поэтому

$$(\sigma)^{\overline{x}}_{\overline{y}}\left((e_1)^{\overline{x}}_{\overline{y}}\right) \circ (\sigma)^{\overline{x}}_{\overline{y}}\left((e_2)^{\overline{x}}_{\overline{y}}\right) \iff \sigma(e_1) \circ \sigma(e_2).$$

Значит

$$(\sigma)^{\overline{x}}_{\overline{y}} \models (T)^{\overline{x}}_{\overline{y}} \iff \sigma \models T. \qquad \square$$

Лемма 3.12 (Замена переменных в программах)**.** *Пусть переменные $\overline{y}$ не входят в состояние σ и программу Π. Тогда*

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

для любых переменных $\overline{x}$.

Доказательство. Индукция по построению программы П.
Базис индукции.

$$\text{П} \circ\!\!-\!\!\circ\, z = e;$$

где z — переменная, e — выражение. Возможно два случая:

1. $z \neq x_1, \ldots, z \neq x_n$. Тогда $(\text{П})^{\overline{x}}_{\overline{y}} \circ\!\!-\!\!\circ\, z = (e)^{\overline{x}}_{\overline{y}}$; согласно лемме 3.10

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

Пусть $\text{П}\,(\sigma) = \tau$. По определению семантики присваивания $\tau u = \sigma u$, если $u \neq z$, и $\tau z = \sigma\,(e)$. Пусть

$$(\text{П})^{\overline{x}}_{\overline{y}} \left((\sigma)^{\overline{x}}_{\overline{y}} \right) = \rho.$$

По определению семантики присваивания $\rho u = (\sigma)^{\overline{x}}_{\overline{y}} u$, если $u \neq z$, и

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

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

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

Поскольку z отличается от $y_1, \ldots, y_n$ по условию леммы, то $z \neq (u)^{\overline{x}}_{\overline{y}}$. Значит,

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

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

2. $z \circ\!\!-\!\!\circ\, x_i$. Тогда $(\text{П})^{\overline{x}}_{\overline{y}} \circ\!\!-\!\!\circ\, y_i = (e)^{\overline{x}}_{\overline{y}}$. Согласно лемме 3.10

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

Пусть $\text{П}\,(\sigma) = \tau$. По определению семантики присваивания $\tau u = \sigma u$, если $u \neq x_i$, и $\tau x_i = \sigma\,(e)$. Пусть

$$(\text{П})^{\overline{x}}_{\overline{y}} \left((\sigma)^{\overline{x}}_{\overline{y}} \right) = \rho.$$

По определению семантики присваивания $\rho u = (\sigma)^{\overline{x}}_{\overline{y}} u$, если $u \neq y_i$, и

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

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

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

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

Шаг индукции.

1. Пусть $\Pi \multimap \Pi_1 \Pi_2$. Тогда, очевидно,

$$(\Pi)^{\overline{x}}_{\overline{y}} \multimap (\Pi_1)^{\overline{x}}_{\overline{y}} (\Pi_2)^{\overline{x}}_{\overline{y}}.$$

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

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

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

$$(\Pi_1)^{\overline{x}}_{\overline{y}} \left((\sigma)^{\overline{x}}_{\overline{y}} \right) = (\tau)^{\overline{x}}_{\overline{y}}.$$

Пусть $\Pi_2\,(\tau) = \rho$. Тогда

$$(\Pi_2)^{\overline{x}}_{\overline{y}} \left((\tau)^{\overline{x}}_{\overline{y}} \right) = (\rho)^{\overline{x}}_{\overline{y}}.$$

Итак,

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

2. Пусть

$$\Pi \multimap \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end;} \end{cases}$$

Тогда

$$(\Pi)_{\overline{y}}^{\overline{x}} \eqsim \begin{cases} \mathsf{if}\ \ (T)_{\overline{y}}^{\overline{x}}\ \mathsf{then} \\ \quad (\Pi_1)_{\overline{y}}^{\overline{x}} \\ \mathsf{else} \\ \quad (\Pi_2)_{\overline{y}}^{\overline{x}} \\ \mathsf{end;} \end{cases}$$

По определению семантики для полного ветвления имеем:

$$\Pi\,(\sigma) = \begin{cases} \Pi_1\,(\sigma)\,, & \text{если } \sigma \models T; \\ \Pi_2\,(\sigma)\,, & \text{если } \sigma \not\models T. \end{cases}$$

$$(\Pi)_{\overline{y}}^{\overline{x}}\left((\sigma)_{\overline{y}}^{\overline{x}}\right) = \begin{cases} (\Pi_1)_{\overline{y}}^{\overline{x}}\left((\sigma)_{\overline{y}}^{\overline{x}}\right), & \text{если } (\sigma)_{\overline{y}}^{\overline{x}} \models (T)_{\overline{y}}^{\overline{x}}\,; \\ (\Pi_2)_{\overline{y}}^{\overline{x}}\left((\sigma)_{\overline{y}}^{\overline{x}}\right), & \text{если } (\sigma)_{\overline{y}}^{\overline{x}} \not\models (T)_{\overline{y}}^{\overline{x}}\,. \end{cases}$$

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

$$(\Pi\,(\sigma))_{\overline{y}}^{\overline{x}} = \begin{cases} (\Pi_1\,(\sigma))_{\overline{y}}^{\overline{x}}, & \text{если } \sigma \models T; \\ (\Pi_2\,(\sigma))_{\overline{y}}^{\overline{x}}, & \text{если } \sigma \not\models T. \end{cases}$$

По лемме 3.11,

$$\sigma \models T \iff (\sigma)_{\overline{y}}^{\overline{x}} \models (T)_{\overline{y}}^{\overline{x}}.$$

По индукционному предположению

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

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

3. Для неполного ветвления доказывается аналогично.

4. Пусть

$$\Pi \eqsim \begin{cases} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end;} \end{cases}$$

Тогда

$$(\Pi)_{\overline{y}}^{\overline{x}} \eqsim \begin{cases} \mathsf{while}\ \ (T)_{\overline{y}}^{\overline{x}}\ \mathsf{do} \\ \quad (\Pi_1)_{\overline{y}}^{\overline{x}} \\ \mathsf{end;} \end{cases}$$

По индукционному предположению для любого состояния ρ имеет место:

$$(\Pi_1(\rho))^{\overline{x}}_{\overline{y}} = (\Pi_1)^{\overline{x}}_{\overline{y}}\left((\rho)^{\overline{x}}_{\overline{y}}\right).$$

Пусть $\left(\sigma^i\right)_i$ — последовательность состояний для программы Π на σ, а $\left(\tau^i\right)_i$ — для $(\Pi)^{\overline{x}}_{\overline{y}}$ на $(\sigma)^{\overline{x}}_{\overline{y}}$. Докажем индукцией по i, что $\left(\sigma^i\right)^{\overline{x}}_{\overline{y}} = \tau^i$.
Базис индукции: $i = 0$. Тогда

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

Индукционный шаг: пусть для i доказано. Тогда

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

Получаем, что

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

Итак, мы доказали, что $\left(\sigma^i\right)^{\overline{x}}_{\overline{y}} = \tau^i$ для всех i. Пусть $\Pi(\sigma)$ определено. Тогда существует n такое, что $\sigma^i \models T$ тогда и только тогда, когда $i < n$ и при этом $\Pi(\sigma) = \sigma^n$. Следовательно, по лемме 3.11

$$\left(\sigma^i\right)^{\overline{x}}_{\overline{y}} \models (T)^{\overline{x}}_{\overline{y}} \iff \sigma^i \models T \iff i < n.$$

Значит,

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

В обратную сторону. Пусть $(\Pi)^{\overline{x}}_{\overline{y}}\left((\sigma)^{\overline{x}}_{\overline{y}}\right)$ определено. По лемме 3.9

$$\left((\Pi)^{\overline{x}}_{\overline{y}}\right)^{\overline{y}}_{\overline{x}} \backsimeq \Pi,\quad \left((\sigma)^{\overline{x}}_{\overline{y}}\right)^{\overline{y}}_{\overline{x}} = \sigma.$$

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

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

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

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

□

Задача 3.30. Обоснуйте индукционный шаг для неполного ветвления.

Следствие 3.8. *Пусть дан алгоритм:*

$$\begin{array}{l}\mathsf{Alg}\ A;\\ \mathsf{arg}\ \overline{x};\\ \quad \Pi_A\\ \mathsf{end};\end{array}$$

Попарно различные переменные $\overline{y}$ *и* B *не встречаются в* Π_A *и в* $\overline{x}$*. Тогда алгоритм*

$$\begin{array}{l}\mathsf{Alg}\ B;\\ \mathsf{arg}\ \ (\overline{x})^{\overline{z}}_{\overline{y}};\\ \quad (\Pi_A)^{\overline{z}\,A}_{\overline{y}\,B}\\ \mathsf{end};\end{array}$$

эквивалентен исходному.

Задача 3.31. Докажите следствие.

§ 3.4. *Простые программы

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

Определение 3.22 (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) $x = o\,(y_1, \ldots, y_n)$;

а каждый тест — один из двух видов:

1) $y_1 = y_2$

2) $y_1 < y_2$

где $x, y_1, \ldots, y_n$ *— переменные,* c *— константа,* o *— операция. Операторы присваивания и тесты такого вида мы будем называть п р о с т ы м и.*

Пример 3.20. *Программы из примеров 3.13 и 3.14 являются простыми, а из примера 3.15 не является. Например, в программе Euclid есть тест* $u = \mathsf{succ}\,(0)$*, который в простой программе недопустим.*

Лемма 3.13 (О разложении выражения)**.** *Пусть d — выражение, y — переменная, σ, τ — состояния, $\tau = (y = d;)(\sigma)$. Тогда для любого выражения e*

$$\tau(e) = \sigma\left((e)_d^y\right).$$

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

Базис индукции.

1. $e \circ\!\!- y$. Тогда $\sigma\left((e)_d^y\right) = \sigma(d) = \tau y = \tau(e)$.
2. $e \circ\!\!- z$, z — переменная, $z \neq y$. Тогда $\sigma\left((e)_d^y\right) = \sigma z = \tau z = \tau(e)$.
3. $e \circ\!\!- c$, c — константа: $\sigma\left((e)_d^y\right) = \sigma c = \tau c = \tau(e)$.

Шаг индукции.

Пусть для выражений $e_1, \ldots, e_n$ лемма доказана и $e \circ\!\!- o(e_1, \ldots, e_n)$, где o — операция.

$$\sigma\left((e)_d^y\right) = \sigma\left(o\left((e_1)_d^y, \ldots, (e_n)_d^y\right)\right) =$$

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

Следствие 3.9 (О разложении теста)**.** *Пусть d — выражение, y — переменная, σ, τ — состояния, $\tau = (y = d;)(\sigma)$. Тогда для любого теста T*

$$\tau \models T \iff \sigma \models (T)_d^y.$$

Задача 3.32. Докажите следствие.

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

Следствие 3.10 (Упрощение присваиваний)**.** *Пусть x, y — переменные, d, e — выражения,*

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

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

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

$y \notin V$. Тогда для любого состояния 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 $\Pi_1(\sigma) = (x = e;)(\tau)$.

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

Лемма 3.14 (Упрощение полного ветвления)**.** *Пусть T — тест, Π' и Π'' — программы, y — переменная, d — выражение.*

$$\Pi \Leftarrow \begin{cases} \mathsf{if}\ (T)_d^y\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{else} \\ \quad \Pi'' \\ \mathsf{end}; \end{cases}$$

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

$y \notin V$. Пусть

$$\Pi_1 \Leftarrow \begin{cases} y\ =\ d; \\ \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{else} \\ \quad \Pi'' \\ \mathsf{end}; \end{cases}$$

Тогда для любого состояния σ выполнено

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

Доказательство. Пусть $T \Leftarrow e_1 \circ e_2$. Пусть $\tau = (y = d;)(\sigma)$.

$$\sigma \models (T)_d^y \iff \tau \models T.$$

Так как

$$\sigma \restriction V = \tau \restriction V,$$

то

$$\Pi'(\sigma) \restriction V = \Pi'(\tau) \restriction V,$$
$$\Pi''(\sigma) \restriction V = \Pi''(\tau) \restriction V.$$

Если $\sigma \models (T)_d^y$, то $\tau \models T$, и мы получаем, что $\Pi(\sigma) = \Pi'(\sigma)$ и $\Pi_1(\tau) = \Pi'(\tau)$. Если $\sigma \not\models (T)_d^y$, то $\tau \not\models T$, и, следовательно, $\Pi(\sigma) = \Pi''(\sigma)$ и $\Pi_1(\tau) = \Pi''(\tau)$. И в том, и в другом случае

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

Лемма 3.15 (Упрощение неполного ветвления)**.** *Пусть T — тест, Π' — программа, y — переменная, d — выражение.*

$$\Pi \Lsh \begin{cases} \mathsf{if}\ (T)_d^y\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{end;} \end{cases}$$

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

$y \notin V$. *Пусть*

$$\Pi_1 \Lsh \begin{cases} y\ =\ d; \\ \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi' \\ \mathsf{end;} \end{cases}$$

Тогда для любого состояния σ выполнено

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

***Задача 3.33.** Докажите лемму.

Лемма 3.16 (Упрощение цикла)**.** *Пусть T — тест, Π' — программа, y — переменная, d — выражение.*

$$\Pi \Lsh \begin{cases} \mathsf{while}\ (T)_d^y\ \mathsf{do} \\ \quad \Pi' \\ \mathsf{end;} \end{cases}$$

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

$y \notin V$. *Пусть*

$$\Pi_1 \Lsh \left\{ \begin{array}{l} y\ =\ d; \\ \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi' \\ \quad y\ =\ d; \\ \mathsf{end;} \end{array} \right\} \Pi_2$$

Тогда для любого состояния σ выполнено

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

Доказательство. Пусть $\tau = (y = d;)\,(\sigma)$. Рассмотрим последовательность состояний для Π на σ: $\sigma^0 = \sigma$ и $\sigma^{i+1} = \Pi'\left(\sigma^i\right)$. Рассмотрим последовательность состояний для Π_2 на τ: $\tau^0 = \tau$ и $\tau^{i+1} = (y = d;)\left(\Pi'\left(\tau^i\right)\right)$.

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

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

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

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

Далее,

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

Кроме того,

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

Так как d содержит только переменные из V, то

$$\Pi'\left(\tau^i\right)(d) = \Pi'\left(\sigma^i\right)(d).$$

То есть,

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

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

$$\tau^i \models T \iff (y = d;)\left(\sigma^i\right) \models T \iff \sigma^i \models (T)^y_d.$$

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

$$\sigma^i \models (T)^y_d \iff i < m; \quad \Pi(\sigma) = \sigma^m.$$

Следовательно,

$$\tau^i \models T \iff \sigma^i \models (T)^y_d \iff i < m.$$

Следовательно, $\Pi_2(\tau) = \tau^m$ 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

$$\tau^i \models T \iff i < m; \quad \Pi_2(\tau) = \tau^m.$$

Получаем, что

$$\sigma^i \models (T)_d^y \iff \tau^i \models T \iff i < m.$$

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

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

Теорема 3.2 (О простой программе)**.** *Для всякой программы* Π *существует простая программа* Π' *такая, что*

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

для любого состояния σ*.*

Доказательство. Для каждого a — выражения, теста 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 тестов: $w(e_1 \circ e_2) = w(e_1) + w(e_2)$, e_1, e_2 — выражения.

Для программ:

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

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

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

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

5) $w(\mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};) = w(\Pi_1) + 2^{w(T)} - 1$.

Здесь Π_1, Π_2 — программы, T — тест.

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 $e \circ\!\!- o(\overline{x})$, $\overline{x}$ — переменные;

3) $w(T) = 0$ тогда и только тогда, когда T — простой тест;

4) $w(\Pi) = 0$ 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

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

для всех состояний σ.

Базис индукции.

$\Pi \circ\!\!- x = e$; так как $w(\Pi) \geq 1$, то $w(e) \geq 2$. Следовательно, $e \circ\!\!- o(e_1, \ldots, e_n)$ и $w(e_i) \geq 1$ 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 3.10 следует, что

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

Шаг индукции.

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

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

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

$$\begin{aligned}\Pi_1(\sigma) \restriction \mathbf{Var}(\Pi) &= \Pi_4(\Pi_{31}(\sigma)) \restriction \mathbf{Var}(\Pi) = \\ &= \Pi_4(\Pi_{31}(\sigma) \restriction \mathbf{Var}(\Pi)) = \Pi_4(\Pi_3(\sigma) \restriction \mathbf{Var}(\Pi)) = \\ &= \Pi_4(\Pi_3(\sigma)) \restriction \mathbf{Var}(\Pi) = \Pi(\sigma) \restriction \mathbf{Var}(\Pi).\end{aligned}$$

Аналогично рассматривается и второй случай.

2. Пусть

$$\Pi \twoheadleftarrow \begin{cases}\mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_3 \\ \mathsf{else} \\ \quad \Pi_4 \\ \mathsf{end;}\end{cases}$$

Тогда $w(T) > 0$, $w(\Pi_3) > 0$ или $w(\Pi_4) > 0$. В последних двух случаях доказательство аналогично предыдущему. Если $w(T) > 0$, $T \twoheadleftarrow e_1 \circ e_2$, то $w(e_1) > 0$ или $w(e_2) > 0$. Пусть, например, $w(e_1) > 0$. Возьмём новую переменную y и построим Π_1:

$$\Pi_1 \twoheadleftarrow \begin{cases}y\ =\ e_1; \\ \mathsf{if}\ y\ \circ\ e_2\ \mathsf{then} \\ \quad \Pi_3 \\ \mathsf{else} \\ \quad \Pi_4 \\ \mathsf{end;}\end{cases}$$

По лемме 3.14

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

$$w(\Pi_1) = w(e_1) - 1 + w(e_2) + w(\Pi_3) + w(\Pi_4) = w(\Pi) - 1 < w(\Pi).$$

3. Для неполного ветвления аналогично.

4. Пусть

$$\Pi \twoheadleftarrow \begin{cases}\mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_3 \\ \mathsf{end;}\end{cases}$$

Если $w(\Pi_3) > 0$, то утверждение следует из индукционного предположения. Пусть $w(\Pi_3) = 0$. Это означает, что $2^{w(T)} > 1$ и $w(T) > 0$.

$T \vdash e_1 \circ e_2$ и $w(e_1) > 0$ или $w(e_2) > 0$. Пусть, например, $w(e_1) > 0$. Возьмём новую переменную y и построим Π_1:

$$\Pi_1 \vdash \begin{cases} y \;=\; e_1; \\ \mathsf{while}\ y \circ e_2\ \mathsf{do} \\ \quad \Pi_3 \\ \quad y \;=\; e_1; \\ \mathsf{end}; \end{cases}$$

По лемме 3.16

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

По определению веса

$$w(\Pi_1) = 2(w(e_1) - 1) + 2^{w(e_2)} - 1 + w(\Pi_3),$$

$$w(\Pi) = 2^{w(e_1)+w(e_2)} - 1 + w(\Pi_3).$$

Докажем, что

$$2(w(e_1) - 1) + 2^{w(e_2)} < 2^{w(e_1)+w(e_2)}.$$

Индукция по $w(e_1)$.
Базис: $w(e_1) = 1$.

$$2^{w(e_2)} < 2^{1+w(e_2)}.$$

Шаг: пусть для $w(e_1) = u$ доказано. Рассмотрим $w(e_1) = u + 1$:

$$2u + 2^{w(e_2)} = 2 + 2(u - 1) + 2^{w(e_2)}.$$

По индукционному предположению

$$2(u - 1) + 2^{w(e_2)} < 2^{u+w(e_2)},$$

получаем

$$2u + 2^{w(e_2)} < 2 + 2^{u+w(e_2)} < 2^{u+w(e_2)} + 2^{u+w(e_2)} = 2^{u+1+w(e_2)}.$$

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

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

для всех состояний 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

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

для всех i, то

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

*__Задача 3.34.__ Докажите все пропущенные места в доказательстве теоремы.

__Следствие 3.11.__ *Каждый алгоритм эквивалентен некоторому алгоритму, тело которого является простой программой.*

__Задача 3.35.__ Докажите следствие.

__Задача 3.36.__ Преобразуйте тело алгоритма *Euclid* (пример 3.15) 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 3.3__ (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:

```
Alg O;
arg x_1, ..., x_n;
  Π_O
end;
```

Рассмотрим произвольный алгоритм C на ЯП B:

$$\begin{array}{l}\mathsf{Alg}\ C;\\ \mathsf{arg}\ \overline{w};\\ \quad \Pi^B\\ \mathsf{end};\end{array}$$

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

$$u = o\,(u_1, \ldots, u_n)\,;$$

Заменим каждую переменную v (кроме выходной переменной O) в программе Π_O на новую переменную $\hat{v}$. Согласно лемме о замене переменных, алгоритм будет вычислять ту же самую функцию.

Теперь построим программу Π'_O следующим образом:

$$\begin{array}{l}\hat{x}_1 \ = \ y_1;\\ \vdots\\ \hat{x}_n \ = \ y_n;\\ \hat{z}_1 \ = \ 0;\\ \vdots\\ \hat{z}_m \ = \ 0;\\ \Pi_O\end{array}$$

Здесь $y_1, \ldots, y_n$ — это новые попарно различные переменные, $z_1, \ldots, z_m$ — все переменные Π_O, которые не являются аргументами. Тогда алгоритм

$$\begin{array}{l}\mathsf{Alg}\ O';\\ \mathsf{arg}\ y_1,\ \ldots,\ y_n;\\ \quad \Pi'_O\\ \mathsf{end};\end{array}$$

эквивалентен алгоритму O. В самом деле, рассмотрим результаты вызовов $\delta = (O, \overline{a})$ и $\delta' = (O', \overline{a})$. В результате выполнения первых $n+m$ строк Π'_O получим состояние τ' такое, 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 $\Pi_O\,(\sigma_\delta)\,(O) = \Pi_O\,(\tau')\,(O)$. Итак, O' действительно эквивалентен O.

Теперь заменим в программе Π^B каждый оператор присваивания вида

$$u = o\,(v_1, \ldots, v_n)\,;$$

на

$$(\Pi'_O)\,{}^{O\,y_1 \ldots y_n}_{u\;v_1 \ldots v_n}$$

Назовём полученную программу Π^A. Очевидно, что операция o в Π^A не встречается, то есть, Π^A — программа на языке программирования A. Осталось только доказать, что для всякого начального состояния σ выполнено

$$\Pi^B\,(\sigma)\,(C) = \Pi^A\,(\sigma)\,(C)\,.$$

Индукцией по построению программы Π^B докажем следующее утверждение: для всяких состояний σ и τ таких, что

$$\sigma \restriction \mathbf{Var}\left(\Pi^B\right) = \tau \restriction \mathbf{Var}\left(\Pi^B\right),$$

имеет место

$$\Pi^B\,(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^A\,(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Базис индукции.
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

   $$\Pi^A\,(\tau) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^B\,(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right),$$

   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

   $$\Pi^A\,(\tau) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^B\,(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right).$$

3. $\Pi^B \circ\!\!- x = p\,(z_1, \ldots, z_n)\,$; p — операция, $p \neq o$, $x, z_1, \ldots, z_n$ — переменные. Опять $\Pi^A \circ\!\!- \Pi^B$ и

   $$\Pi^A\,(\tau) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^B\,(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right).$$

4. $\Pi^B \circ\!\!- x = o\,(z_1, \ldots, z_n)\,$; $x, z_1, \ldots, z_n$ — переменные. Тогда $\Pi^A \circ\!\!- (\Pi'_O)\,{}^{O\,y_1 \ldots y_n}_{x\;z_1 \ldots z_n}$. Согласно лемме о замене переменных

   $$(\Pi'_O)\,{}^{O\,y_1 \ldots y_n}_{x\;z_1 \ldots z_n}\,(\sigma)\,(x) = o\,(\overline{z})\,.$$

Для любых остальных переменных из Π^B программа $(\Pi'_O)^{O\,y_1\ldots y_n}_{x\;z_1\ldots z_n}$ не изменяет значение, так как они не входят в неё в левой части оператора присваивания. Это и означает, что

$$\Pi^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Шаг индукции.
Возможны четыре варианта.

1. $\Pi^B \eqcirc \Pi_1^B \Pi_2^B$. Тогда $\Pi^A \eqcirc \Pi_1^A \Pi_2^A$ и по индукционному предположению

$$\Pi_1^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_1^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Пусть $\Pi_1^B(\sigma) = \sigma_1$ и $\Pi_1^A(\tau) = \tau_1$. По индукционному предположению

$$\Pi_2^B(\sigma_1) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_2^A(\tau_1) \restriction \mathbf{Var}\left(\Pi^B\right).$$

То есть

$$\Pi_2^B\left(\Pi_1^B(\sigma)\right) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_2^A\left(\Pi_1^A(\tau)\right) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Но по определению семантики следования

$$\Pi_2^B\left(\Pi_1^B(\sigma)\right) = \Pi^B(\sigma); \quad \Pi_2^A\left(\Pi_1^A(\tau)\right) = \Pi^A(\tau).$$

Следовательно,

$$\Pi^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

2.
$$\Pi^B \eqcirc \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1^B \\ \mathsf{else} \\ \quad \Pi_2^B \\ \mathsf{end}; \end{cases}$$

Тогда

$$\Pi^A \eqcirc \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1^A \\ \mathsf{else} \\ \quad \Pi_2^A \\ \mathsf{end}; \end{cases}$$

и по индукционному предположению

$$\Pi_1^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_1^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right),$$

$$\Pi_2^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_2^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Заметим, что поскольку тест T состоит только из переменных из $\mathbf{Var}\left(\Pi^B\right)$, то $\sigma \models T$ тогда и только тогда, когда $\tau \models T$. По определению семантики ветвления имеем:

$$\Pi^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right) = \left\{ \begin{array}{ll} \Pi_1^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right), & \text{если } \sigma \models T \\ \Pi_2^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right), & \text{если } \sigma \not\models T \end{array} \right\} =$$
$$= \left\{ \begin{array}{ll} \Pi_1^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right), & \text{если } \tau \models T \\ \Pi_2^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right), & \text{если } \tau \not\models T \end{array} \right\} = \Pi^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right).$$

3. Для неполного ветвления — аналогично.

4.
$$\Pi^B \Rightarrow \left\{ \begin{array}{l} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1^B \\ \mathsf{end;} \end{array} \right.$$

Тогда
$$\Pi^A \Rightarrow \left\{ \begin{array}{l} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1^A \\ \mathsf{end;} \end{array} \right.$$

и по индукционному предположению для любых состояний ρ и π таких, что

$$\rho \restriction \mathbf{Var}\left(\Pi^B\right) = \pi \restriction \mathbf{Var}\left(\Pi^B\right),$$

имеет место

$$\Pi_1^B(\rho) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_1^A(\pi) \restriction \mathbf{Var}\left(\Pi^A\right).$$

Рассмотрим последовательности состояний для Π^B и для Π^A:

$$\sigma^0 = \sigma, \quad \sigma^{i+1} = \Pi_1^B\left(\sigma^i\right);$$

$$\tau^0 = \tau, \quad \tau^{i+1} = \Pi_1^A\left(\tau^i\right).$$

Покажем, что

$$\sigma^i \restriction \mathbf{Var}\left(\Pi^B\right) = \tau^i \restriction \mathbf{Var}\left(\Pi^B\right)$$

для всех i. Для $i = 0$ это следует из условия. Пусть для i доказано. Тогда

$$\sigma^{i+1} \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi_1^B\left(\sigma^i\right) \restriction \mathbf{Var}\left(\Pi^B\right) = \\ = \Pi_1^A\left(\tau^i\right) \restriction \mathbf{Var}\left(\Pi^B\right) = \tau^{i+1} \restriction \mathbf{Var}\left(\Pi^B\right).$$

Так как тест T содержит только переменные из $\mathbf{Var}\left(\Pi^B\right)$, то $\sigma^i \models T$ тогда и только тогда, когда $\tau^i \models T$. Следовательно, последовательность $\left(\sigma^i\right)_i$ конечна тогда и только тогда, когда последовательность $\left(\tau^i\right)_i$ 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

$$\Pi^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right) = \tau^m \restriction \mathbf{Var}\left(\Pi^B\right) = \\ = \sigma^m \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right).$$

И, если $\Pi^B(\sigma)$ определено, то $\Pi^A(\tau)$ определено, и

$$\Pi^A(\tau) \restriction \mathbf{Var}\left(\Pi^B\right) = \Pi^B(\sigma) \restriction \mathbf{Var}\left(\Pi^B\right).$$

Построим алгоритм на ЯП A:

```
Alg C;
arg w;
  Π^A
end;
```

Поскольку выходная переменная C принадлежит множеству $\mathbf{Var}\left(\Pi^B\right)$, то для любого вызова $\delta = (C, \overline{v})$

$$\Pi^B(\sigma_\delta)(C) = \Pi^A(\sigma_\delta)(C).$$

Оба алгоритма вычисляют одну и ту же функцию, то есть эквивалентны. □

***Задача 3.37.** Обоснуйте индукционный шаг для неполного ветвления.

Смысл доказанной теоремы состоит в том, что если мы расширяем ЯП операциями, которые могут быть вычислены программами на этом языке, то множество функций, которые вычисляются программами этого языка не изменяется. Например, мы уже видели, что на нашем ЯП можно написать сложение и вычитание. Следовательно, если мы добавим операции

$+$ и $-$ к нашему языку, то его выразительная сила не увеличится. Точно так же, если мы напишем программы, 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.

Глава 4

Программы с метками

§ 4.1. Синтаксис

До сих пор мы рассматривали структурированные программы. Теперь мы рассмотрим иной подход к написанию программ. Исторически он является более ранним, однако, в настоящий момент его считают морально устаревшим, и он используется только в старых ЯП (таких как Фортран, Бейсик, Фокал) и в ЯП низкого уровня. Однако, основная конструкция этого подхода (переход к метке) остаётся практически во всех современных языках программирования.

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:*

1) *оператор присваивания:*

$$\alpha \;\; x \;=\; e;\;\; \beta$$

где x — переменная, e — выражение, α, β — метки;

2) *оператор условного перехода:*

$$\alpha \;\; \mathsf{if}\; T \;\mathsf{then}\; \beta \;\mathsf{else}\; \gamma$$

```
   Alg Min;
   arg x, y;
1  if x < y then 2 else 3
2  Min = x; 4
3  Min = y; 4
   end;
```

Рис. 4.1: Наименьшее из двух чисел.

где T — тест, 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, arg, ...*).*

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). В данном примере* **Start** $(\Pi_{Min}) = 1$. 4 *— заключительная метка.*

§ 4.2. Семантика

Как и раньше, состояние σ — частичное отображение множества переменных $\mathfrak{V}$ в предметную область.

Определение 4.3 (Расширенное состояние)**.** *Расширенное состояние программы с метками* П *— пара $(\sigma; \alpha)$, где σ — состояние программы* П*, а α — некоторая метка программы. Σ^E_Π — множество всевозможных расширенных состояний программы* П*.*

Пример 4.2. *Рассмотрим программу из примера 4.1. Следующие объекты будут расширенными состояниями:*

$$(\{(x,1);(y,4);(z,312)\},1),$$

$$(\{(x,2);(y,5);(z,32)\},3),$$

$$(\{(x,10);(y,8);(z,2)\},4).$$

Определение 4.4 (Семантика программы с метками)**.** *Сначала для всякой программы* Π *определим частичное отображение* $\Sigma^E_\Pi \to \Sigma^E_\Pi$. $\Pi(\sigma,\alpha)$ *определено и* $\Pi(\sigma,\alpha) = (\tau,\beta)$, *если существует натуральное число* n *и последовательность расширенных состояний* $\left(\sigma^i,\alpha_i\right)_{i=0}^n$ *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*

$$\alpha_i \;\; x \;=\; e;\;\; \gamma$$

то $\alpha_{i+1} = \gamma$, *и* $\sigma^{i+1} = (x = e;)\left(\sigma^i\right)$;

6) *если для* $i < n$ *в* Π *имеется оператор вида*

$$\alpha_i \;\; \mathsf{if}\ T\ \mathsf{then}\ \beta\ \mathsf{else}\ \gamma$$

то $\sigma^{i+1} = \sigma^i$, *а* α_{i+1} *равняется* β, *если* $\sigma^i \models T$, *и* γ, *если* $\sigma^i \not\models T$.

$\Pi(\sigma)$ *определено и* $\Pi(\sigma) = \tau$, *если*

$$\Pi(\sigma, \mathbf{Start}(\Pi)) = (\tau,\beta)$$

для некоторой метки β *(необходимо заключительной).*

Следствие 4.1. *Для каждой программы с метками* Π *существует программа с метками* Π', *которая содержит не более одной заключительной метки, и* $\Pi(\sigma) = \Pi'(\sigma)$ *для всякого состояния* σ.

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) *— расширенное состояние. Тогда* $\Pi(\sigma,\alpha)$ *определено тогда и только тогда, когда последовательность расширенных состояний, 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 расширенных состояний, 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. Пусть* $\sigma_1 = \{(x,1),(y,2),(z,3)\}$*. Тогда*

$$\Pi(\sigma_1, 3) = \left(\underbrace{\{(x,1),(y,2),(z,2)\}}_{\sigma_3}; 4 \right)$$

Последовательность: $(\sigma_1, 3), (\sigma_3, 4)$.

$$\Pi(\sigma_1, 2) = \left(\underbrace{\{(x,1),(y,2),(z,1)\}}_{\sigma_2}; 4 \right)$$

Последовательность: $(\sigma_1, 2), (\sigma_2, 4)$.

$\Pi(\sigma_1, 1) = (\sigma_2, 4)$*, потому что* $\sigma_1 \models x < y$*. 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 $i \geq 1$*. 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 изображён алгоритм для вычисления числа Фибоначчи по его номеру.*

Задача 4.4. Постройте последовательность расширенных состояний при вызове $(Fib, 3)$.

Докажем, что алгоритм Fib и в самом деле вычисляет числа Фибоначчи. Пусть начальное состояние σ и $\sigma i = i_0$.

Если $i_0 < 2$, то последовательность расширенных состояний содержит 3 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 $i_0 \geq 2$. Первые пять элементов последовательности расширенных состояний таковы: $(\sigma, 1)$, $(\sigma, 3)$, $(\sigma_3, 4)$, $(\sigma_4, 5)$, $(\sigma_5, 6)$, где:

```
    Alg Fib;
    arg i;
1     if i < succ(succ(0)) then 2 else 3
2     Fib = i; 100
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
15    Fib = f; 100
    end;
```

Рис. 4.2: Числа Фибоначчи.

– $\sigma_3 = \sigma$ 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 4.1. *В последовательности расширенных состояний 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 тест $t < p$ не выполняется.

Возьмем $l = k + 3\sigma^k p + 6$. 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:

$$\sigma^l p = \sigma^{l-4} f = \cdots = \sigma^k f$$
$$\sigma^l f = \sigma^{l-4} s = \sigma^{k+3\sigma^k p+2} s = \sigma^k f + \sigma^k p$$
$$\sigma^l j = \sigma^k j + 1$$

□

Следствие 4.4. *Для любого расширенного состояний вида* $(\sigma^k, 6)$ *такого, что* $\sigma^k f = F_{\sigma^k j}$ *и* $\sigma^k p = F_{\sigma^k j-1}$, *существует расширенное состояние* $(\sigma^l, 14)$ *такое, 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.

$$\sigma^l f = \sigma^k f + \sigma^k p = F_{\sigma^k j} + F_{\sigma^k j-1} = F_{\sigma^k j+1} = F_{\sigma^l j}.$$
$$\sigma^l p = \sigma^k f = F_{\sigma^k j} = F_{\sigma^l j-1}.$$

□

Утверждение 4.2. *Алгоритм Fib вычисляет* F_{i_0}.
Доказательство. Покажем следующее: для всякого $2 \le n \le i_0$ в последовательности расширенных состояний существует элемент $(\sigma^{l_n}, 14)$ такой, 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 $(\sigma^{l_n}, 14)$ идет $(\sigma^{l_n}, 6)$ и для него 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 $(\sigma^{l_n}, 14)$ идут $(\tau, 15)$ и $(\tau, 100)$. Следовательно,

$$\tau\,(Fib) = \sigma^{l_n} f = F_{i_0}.$$

□

Если сравнить это доказательство с доказательствами для структурированных программ, то нельзя не заметить, что оно является значительно

более сложным. Это связано с тем, что программы с метками не имеют ясно выраженных логических частей. В случае структурированной программы мы доказывали её правильность двигаясь от отдельных операторов и постепенно рассматривая всё более широкие 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 структурированными ЯП.

Задача 4.5. Напишите алгоритм, который подсчитывает сумму цифр заданного числа, и докажите его правильность. Используйте операции «+», «−», «×», «/» — целочисленное деление.

§ 4.3. Построение программ с метками

Мы рассмотрели два ЯП — 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 Π метки α на β определяется точно так же как замена одной переменной на другую. Результат такой замены обозначается аналогично:

$$(\Pi)_{\beta}^{\alpha}$$

То есть $(\Pi)_{\beta}^{\alpha}$ получается из Π заменой всюду метки α на метку β.

Замечание 4.1. Результат замены $(\Pi)_{\beta}^{\alpha}$ 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 $(\Pi)_{\beta}^{\alpha}$ — программа с метками.

Лемма 4.1 (О замене метки)**.** *Пусть метка β не встречается в программе Π. Тогда для любого состояния σ и для любой метки α имеет место $\Pi(\sigma) = (\Pi)_{\beta}^{\alpha}(\sigma)$.*

Доказательство. Докажем следующее утверждение: если $(\Pi)_{\beta}^{\alpha}(\sigma)$ определено, то $\Pi(\sigma)$ определено и $(\Pi)_{\beta}^{\alpha}(\sigma) = \Pi(\sigma)$.

Рассмотрим последовательность расширенных состояний $\left(\sigma^j, \lambda_j\right)_j$ для $(\Pi)^\alpha_\beta(\sigma)$. Так как $(\Pi)^\alpha_\beta(\sigma)$ определено, то последовательность расширенных состояний $\left(\sigma^j, \lambda_j\right)_j$ является конечной. Пусть (σ^m, λ_m) — её последний элемент. Это означает, что λ_m — заключительная метка программы $(\Pi)^\alpha_\beta$. Определим

$$\mu_i \backsimeq \begin{cases} \lambda_i, & \text{если } \lambda_i \not\backsimeq \beta, \\ \alpha, & \text{если } \lambda_i \backsimeq \beta. \end{cases}$$

Докажем, что последовательность расширенных состояний $\left(\sigma^j, \mu_j\right)_{j=0}^m$ является последовательностью расширенных состояний 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. $\lambda_0 \not\backsimeq \beta$ и $\mu_0 = \lambda_0$. Так как $\lambda_0 = \mathbf{Start}\left((\Pi)^\alpha_\beta\right)$, то $\mu_0 \backsimeq \lambda_0$ должна быть и начальной меткой программы Π, потому что мы заменяем только метку α на β.

2. $\lambda_0 \backsimeq \beta$. Тогда $\mu_0 \backsimeq \alpha$, потому что метки β в программе Π отсутствуют, они могут появиться в $(\Pi)^\alpha_\beta$ только в результате замены. Значит, $\mathbf{Start}(\Pi) \backsimeq \alpha \backsimeq \mu_0$.

Это доказывает базис утверждения.

Шаг индукции.

Пусть для i доказано, что последовательность расширенных состояний $\left(\sigma^j, \mu_j\right)_{j=0}^i$ является правильной и $i < m$. Рассмотрим $i + 1$. Возможны следующие варианты:

1. В программе $(\Pi)^\alpha_\beta$ есть оператор вида

$$\lambda_i \;\; x \;=\; e; \;\; \lambda_{i+1}$$

Тогда в программе Π есть оператор

$$\mu_i \;\; x \;=\; e; \;\; \mu_{i+1}$$

σ^{i+1} отличается от σ^i только тем, что $\sigma^{i+1}x = \sigma^i(e)$. Это относится и к $(\Pi)^\alpha_\beta$, и к Π. Следующей меткой для Π должна быть μ_{i+1}. Значит, $\left(\sigma^{i+1}, \mu_{i+1}\right)$ — очередной элемент.

2. В программе $(\Pi)^{\alpha}_{\beta}$ есть оператор

$$\lambda_i \quad \text{if } T \text{ then } \lambda_{i+1} \text{ else } \delta$$

и $\sigma^i \models T$. Тогда в программе Π есть оператор

$$\mu_i \quad \text{if } T \text{ then } \mu_{i+1} \text{ else } \delta'$$

Поэтому следующим элементом последовательности $\left(\sigma^j, \mu_j\right)_j$ будет $\left(\sigma^{i+1}, \mu_{i+1}\right)$, где $\sigma^{i+1} = \sigma^i$.

3. В программе $(\Pi)^{\alpha}_{\beta}$ есть оператор

$$\lambda_i \quad \text{if } T \text{ then } \delta \text{ else } \lambda_{i+1}$$

и $\sigma^i \not\models T$. Тогда в программе Π есть оператор

$$\mu_i \quad \text{if } T \text{ then } \delta' \text{ else } \mu_{i+1}$$

Следовательно, за элементом $\left(\sigma^i, \mu_i\right)$ должен идти $\left(\sigma^{i+1}, \mu_{i+1}\right)$, где $\sigma^{i+1} = \sigma^i$.

Рассмотрим μ_m. Если в Π существует метка оператора μ_m, то в программе $(\Pi)^{\alpha}_{\beta}$ 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 расширенных состояний $\left(\sigma^j, \mu_j\right)_j$ заканчивается элементом (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 $\Pi \eqsim \left((\Pi)^{\alpha}_{\beta}\right)^{\beta}_{\alpha}$ и при этом метка α не встречается в программе $(\Pi)^{\alpha}_{\beta}$. Согласно только что доказанному, из этого следует: если $\left((\Pi)^{\alpha}_{\beta}\right)^{\beta}_{\alpha}(\sigma)$ определено, то $(\Pi)^{\alpha}_{\beta}(\sigma)$ определено и $\left((\Pi)^{\alpha}_{\beta}\right)^{\beta}_{\alpha}(\sigma) = (\Pi)^{\alpha}_{\beta}(\sigma)$. А так как $\Pi \eqsim \left((\Pi)^{\alpha}_{\beta}\right)^{\beta}_{\alpha}$, то это и означает следование в обратную сторону. □

Определение 4.6 (Эффективное построение)**.** *Мы скажем, что по объекту A из некоторого класса можно эффективно построить объект B, если существует алгоритм, который выполняет такое построение для любого объекта A из этого класса.*

Пример 4.5. *Например, замена* $b \Leftarrow (a)_y^x$ *— 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 4.1 (О построении программы с метками)**.** *По всякой структурной программе* Π^S *можно эффективно построить программу с метками* Π^L *такую, что* $\Pi^S(\sigma) = \Pi^L(\sigma)$ *для любого состояния* σ*.*

Доказательство. Докажем теорему индукцией по построению Π^S.

Базис индукции.

Пусть

$$\Pi^S \Leftarrow x = e;$$

где x — переменная, e — выражение. Возьмём

$$\Pi^L \Leftarrow 1 \;\; x = e;\; 2.$$

Очевидно, что для всякого состояния σ имеет место $\Pi^S(\sigma) = \Pi^L(\sigma)$.

Шаг индукции.

Возможны четыре варианта.

1. $\Pi^S \Leftarrow \Pi_1^S \Pi_2^S$. По индукции для Π_1^S и для Π_2^S существуют программы Π_1^L и Π_2^L. 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 $\hat{\alpha}$. Согласно лемме о замене метки $\Pi_2^L(\tau) = \tilde{\Pi}_2^L(\tau)$ для всякого состояния τ. Пусть γ — заключительная метка Π_1^L, а δ — начальная метка $\tilde{\Pi}_2^L$. Пусть $\tilde{\Pi}_1^L \Leftarrow \left(\Pi_1^L\right)_\delta^\gamma$. Согласно лемме о замене метки $\Pi_1^L(\sigma) = \tilde{\Pi}_1^L(\sigma)$ для всякого состояния σ. Построим $\Pi^L \Leftarrow \tilde{\Pi}_1^L \tilde{\Pi}_2^L$. Докажем, что Π^L — искомая программа.

   Пусть $\Pi^S(\sigma)$ определено. Тогда $\Pi_1^S(\sigma)$ определено. Пусть $\Pi_1^S(\sigma) = \tau$. Согласно индукционному предположению $\tilde{\Pi}_1^L(\sigma) = \Pi_1^L(\sigma) = \tau$. Пусть $\Pi_2^S(\tau) = \rho$. Согласно индукционному предположению $\tilde{\Pi}_2^L(\tau) = \Pi_2^L(\tau) = \rho$. Рассмотрим последовательность расширенных состояний для $\Pi^L(\sigma)$. Так как $\tilde{\Pi}_1^L(\sigma)$ определено, то последовательность расширенных состояний $\left(\sigma^i, \lambda_i\right)_i$ 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 первым расширенным состоянием для $\tilde{\Pi}_2^L(\tau)$. Так как $\tilde{\Pi}_2^L(\tau)$ определено, то эта последовательность должна быть конечной: $\left(\sigma^i, \lambda_i\right)_{i=m}^{n}$. Очевидно, что $\sigma^n = \rho$. Рассмотрим последовательность $\left(\sigma^i, \lambda_i\right)_{i=0}^{n}$. Эта последовательность является последовательностью расширенных состояний 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, $\Pi^S(\sigma) = \Pi^L(\sigma)$.

Пусть теперь $\Pi^L(\sigma)$ определено. Рассмотрим последовательность расширенных состояний $\left(\sigma^i, \lambda_i\right)_{i=0}^{n}$. Тогда метка λ_n должна быть заключительной меткой программы Π^L. Но единственной заключительной меткой программы Π^L является заключительная метка программы $\tilde{\Pi}_2^L$. Но 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 $\tilde{\Pi}_1^L$ встречаться не может, потому что 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. 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 расширенных состояний должен быть элемент (σ^m, λ_m) такой, 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 $\left(\sigma^i, \lambda_i\right)_{i=0}^{m}$. Тогда она является последовательностью расширенных состояний 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 $\left(\sigma^i, \lambda_i\right)_{i=m}^{n}$. Тогда, она является последовательностью расширенных состояний 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

$$\Pi^S(\sigma) = \Pi_2^S\left(\Pi_1^S(\sigma)\right) = \Pi_2^S(\sigma^m) = \sigma^n.$$

таким образом, $\Pi^S(\sigma)$ определено и $\Pi^S(\sigma) = \Pi^L(\sigma)$.

2.

$$\Pi^S \twoheadleftarrow \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1^S \\ \mathsf{else} \\ \quad \Pi_2^S \\ \mathsf{end;} \end{cases}$$

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 $\tilde{\Pi}_1^L \twoheadleftarrow \left(\Pi_1^L\right)_\delta^\gamma$. Пусть α — новая метка, которая не встречается ни в $\tilde{\Pi}_1^L$ ни в $\tilde{\Pi}_2^L$. Возьмём

$$\Pi^L \twoheadleftarrow \begin{cases} \alpha\ \ \mathsf{if}\ T\ \mathsf{then}\ \mathbf{Start}\left(\tilde{\Pi}_1^L\right)\ \mathsf{else}\ \mathbf{Start}\left(\tilde{\Pi}_2^L\right) \\ \quad \tilde{\Pi}_1^L \\ \quad \tilde{\Pi}_2^L \end{cases}$$

Возможны два случая: либо $\sigma \models T$, либо $\sigma \not\models T$. 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 расширенных состояний $\left(\sigma^i, \lambda_i\right)_{i=0}^n$, для которой $\sigma^0 = \sigma$, $\lambda_0 = \mathbf{Start}\left(\tilde{\Pi}_1^L\right)$, $\sigma^n = \tau$ и $\lambda_n = \delta$. Рассмотрим последовательность расширенных состояний, полученную из предыдущей добавлением в начало расширенного состояния (σ, α): $(\sigma, \alpha), \left(\sigma^i, \lambda_i\right)_{i=0}^n$. Легко показать, что это — последовательность расширенных состояний 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 расширенных состояний $\left(\sigma^i, \lambda_i\right)_i$ для $\Pi^L(\sigma)$ является конечной. Предположим, она содержит $n+1$ элемент. Очевидно, что $\lambda_1 = \mathbf{Start}\left(\tilde{\Pi}_1^L\right)$. Последовательность $\left(\sigma^i, \lambda_i\right)_{i=1}^n$ является последовательностью расширенных состояний для $\tilde{\Pi}_1^L(\sigma)$, так как метки операторов $\tilde{\Pi}_2^L$ в $\tilde{\Pi}_1^L$ не встречаются. Тогда последовательность

$$\left(\sigma^i, \lambda_i\right)_{i=1}^{n-1}, \left(\sigma^i, \gamma\right)$$

является последовательностью расширенных состояний 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 $\sigma \not\models T$ 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

$$\Pi^S \twoheadleftarrow \begin{cases} \text{while } T \text{ do} \\ \quad \Pi_1^S \\ \text{end;} \end{cases}$$

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

$$\Pi^L \twoheadleftarrow \begin{cases} \gamma & \text{if } T \text{ then } \beta \text{ else } \alpha \\ & \Pi_1^L \end{cases}$$

Пусть $\Pi^S(\sigma)$ определено. Это означает, что существует последовательность состояний $\left(\sigma_S^i\right)_{i=0}^n$ такая, что

$$\sigma_S^0 = \sigma, \quad \sigma_S^{i+1} = \Pi_1^S\left(\sigma_S^i\right), \quad \sigma_S^i \models T \iff i < n.$$

Рассмотрим последовательность расширенных состояний $\left(\sigma_L^j, \lambda_j\right)_j$ 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 $\sigma_L^{j_i} = \sigma_S^i$.

Базис индукции.
$i = 0$. Тогда $j_0 = 0$ и $\sigma_L^{j_0} = \sigma_L^0 = \sigma = \sigma_S^0$.

Шаг индукции.
Пусть доказано, что $\sigma_L^{j_i} = \sigma_S^i$ и $i < n$. Это означает, что $\sigma_S^i \models T$ и $\sigma_L^{j_i} \models T$. Это означает, что вслед за расширенным состоянием $\left(\sigma_L^{j_i}, \gamma\right)$ должно идти расширенное состояние $\left(\sigma_L^{j_i}, \beta\right)$. Это является начальным расширенным состоянием для $\Pi_1^L\left(\sigma_L^{j_i}\right)$. Из этого следует, что

$$\Pi_1^L\left(\sigma_L^{j_i}\right) = \Pi_1^S\left(\sigma_L^{j_i}\right) = \Pi_1^S\left(\sigma_S^i\right) = \sigma_S^{i+1}.$$

Это означает, что последовательность расширенных состояний для $\Pi_1^L\left(\sigma_L^{j_i}\right)$ является конечной, и ее последним элементом является $\left(\sigma_S^{i+1}, \gamma\right)$. Заметим, что в этой последовательности метка γ встречается только один раз в конце.

Следовательно, в последовательности расширенных состояний для $\Pi^L(\sigma)$ после $\left(\sigma_L^{j_i}, \beta\right)$ должно встретиться состояние $\left(\sigma_S^{i+1}, \gamma\right)$ и это — первое место после j_i, где меткой является γ. Очевидно, что j_{i+1} — номер этого расширенного состояния, и $\sigma_L^{j_{i+1}} = \sigma_S^{i+1}$.

Итак, мы доказали, что $\sigma_L^{j_i} = \sigma_S^i$. Значит,

$$\sigma_L^{j_i} \models T \iff \sigma_S^i \models T \iff i < n.$$

То есть, $\sigma_L^{j_n} \not\models T$. Значит, после расширенного состояния $\left(\sigma_L^{j_n}, \gamma\right)$ будет идти $\left(\sigma_L^{j_n}, \alpha\right)$. 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 расширенных состояний $\left(\sigma_L^i, \lambda_i\right)_i$ 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, $\sigma_L^{j_i} \models T$ тогда и только тогда, когда $i < m$.

Рассмотрим последовательность состояний σ_S^i для $\Pi^S(\sigma)$. Аналогично доказывается, что $\sigma_L^{j_i} = \sigma_S^i$. То есть,

$$\sigma_S^i \models T \iff \sigma_L^{j_i} \models T \iff i < m.$$

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 структурная 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 — по программе с метками построим эквивалентную структурированную программу.

Теорема 4.2 (Структуризация)**.** *По всякой программе с метками* Π^L *можно эффективно построить структурную программу* Π^S *такую, что* $\mathbf{Var}\left(\Pi^L\right) \subseteq \mathbf{Var}\left(\Pi^S\right)$ *и*

$$\Pi^L(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right) = \Pi^S(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right)$$

для любого состояния 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 являются натуральными числами большими 0. Это означает, что программа Π^L имеет следующий вид:

$$\Pi^L \twoheadleftarrow \begin{cases} o_1 \\ o_2 \\ \vdots \\ o_m \end{cases}$$

Каждый из операторов o_k имеет один из видов:

1) $o_k \twoheadleftarrow \alpha_k\ x_k = e_k;\ \beta_k$
2) $o_k \twoheadleftarrow \alpha_k$ if T_k then β_k else γ_k

Пусть w и y — новые переменные, которые не встречаются в Π^L. Для каждого оператора o_k одного из этих видов определим структурную программу $\varphi(o_k)$ следующим образом:

1. $\varphi(o_k) \Leftarrow$
```
if w = succ(0) then
  if y = ᾱ_k then
    x_k = e_k;
    y = β̄_k;
    w = 0;
  end;
end;
```

2. $\varphi(o_k) \Leftarrow$
```
if w = succ(0) then
  if y = ᾱ_k then
    if T_k then
      y = β̄_k;
    else
      y = γ̄_k;
    end;
    w = 0;
  end;
end;
```

Здесь через $\overline{n}$ мы обозначаем следующее выражение:

$$\overline{n} \Leftarrow \underbrace{\mathsf{succ}(\mathsf{succ}(\ldots\mathsf{succ}(}_{n\text{ раз}}\,0\,\underbrace{)\ldots))}_{n\text{ раз}}$$

Очевидно, что $\sigma(\overline{n}) = n$ для любого состояния σ. Построим программу Π^S следующим образом:

$\Pi^S \Leftarrow$
```
y = ᾱ_1;
w = 0;
while w = 0 do
  w = succ(0);
  φ(o_1)
  φ(o_2)
  ⋮
  φ(o_k)
end;
```
} Π_1^S

Докажем, что $\Pi^S(\sigma)$ определено тогда и только тогда, когда $\Pi^L(\sigma)$ определено, для любого состояния σ и в этом случае

$$\Pi^S(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right) = \Pi^L(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right).$$

По определению семантики программ с метками мы должны рассматривать последовательность расширенных состояний $\left(\sigma_L^i, \lambda_i\right)_i$, для которой $\sigma_L^0 = \sigma$ и $\lambda_0 = \alpha_1$.

Теперь рассмотрим Π^S. Через σ'_S обозначим состояние, которое получается из σ применением первых двух операторов присваивания программы Π^S:

$$\sigma'_S = (y = \overline{\alpha_1};\ w = 0;)(\sigma).$$

Очевидно, что

$$\sigma'_S y = \alpha_1;\quad \sigma'_S w = 0;\quad \sigma'_S \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma \restriction \mathbf{Var}\left(\Pi^L\right),$$

поскольку никакие переменные из программы Π^L эти операторы не изменяют. Для оператора цикла мы должны строить последовательность состояний $\left(\sigma_S^i\right)_i$ такую, что $\sigma_S^0 = \sigma'_S$ и $\sigma_S^{i+1} = \Pi_1^S\left(\sigma_S^i\right)$.

Индукцией по i докажем следующее утверждение:

1) $\sigma_S^i y = \lambda_i$;

2) $\sigma_S^i w = 0$, если метка λ_{i-1} не является заключительной;

3) $\sigma_S^i \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_L^i \restriction \mathbf{Var}\left(\Pi^L\right)$.

Базис индукции.
Для $i = 0$ утверждение, очевидно, выполнено, так как $\sigma_S^0 = \sigma'_S$, а свойства σ'_S мы описали выше.
Шаг индукции.
Пусть для i утверждение доказано. Тогда рассмотрим $\Pi_1^S\left(\sigma_S^i\right) = \sigma_S^{i+1}$. По определению семантики следования имеем:

$$\sigma_S^{i+1} = \Pi_1^S\left(\sigma_S^i\right) = \varphi(o_m)\left(\ldots\varphi(o_1)\left((w = \mathsf{succ}(0);)\left(\sigma_S^i\right)\right)\ldots\right).$$

Пусть

$$\rho_S^i = (w = \mathsf{succ}(0);)\left(\sigma_S^i\right).$$

Тогда ρ_S^i отличается от σ_S^i только тем, что $\rho_S^i w = 1$. Пусть $\lambda_i = \alpha_{j_i}$.

Рассмотрим $\varphi(o_k)\left(\rho_S^i\right)$ при $k < j_i$. Очевидно, что

$$\rho_S^i \models w = \mathsf{succ}(0),$$

потому что $\rho_S^i w = 1$, но, в то же время,

$$\rho_S^i \not\models y = \overline{\alpha_k},$$

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 $\varphi(o_k)\left(\rho_S^i\right) = \rho_S^i$.

Рассмотрим $\varphi(o_k)\left(\rho_S^i\right)$ при $k = j_i$. Тогда

$$\rho_S^i \models w = \mathsf{succ}(0)$$

и

$$\rho_S^i \models y = \overline{\alpha_k},$$

так как

$$\rho_S^i y = \sigma_S^i y = \lambda_i = \alpha_{j_i} = \alpha_k.$$

Пусть $\pi_S^i = \varphi(o_k)\left(\rho_S^i\right)$.

1. Если

$$o_k \quad \circ\!\!- \quad \alpha_k\ x_k = e_k;\ \beta_k,$$

то по определению семантики присваивания и следования 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. Пусть теперь

$$o_k \quad \circ\!\!- \quad \alpha_k\ \mathsf{if}\ T_k\ \mathsf{else}\ \beta_k\ \mathsf{else}\ \gamma_k.$$

Тогда

$$\begin{aligned} \pi_S^i \restriction \mathbf{Var}\left(\Pi^L\right) &= \rho_S^i \restriction \mathbf{Var}\left(\Pi^L\right) = \\ &= \sigma_S^i \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_L^i \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_L^{i+1} \restriction \mathbf{Var}\left(\Pi^L\right). \end{aligned}$$

Кроме того, $\pi_S^i w = 0$ и

$$\pi_S^i y = \left\{ \begin{array}{ll} \beta_k, & \text{если } \rho_S^i \models T_k \\ \gamma_k, & \text{если } \rho_S^i \not\models T_k \end{array} \right\} = \left\{ \begin{array}{ll} \beta_k, & \text{если } \sigma_L^i \models T_k \\ \gamma_k, & \text{если } \sigma_L^i \not\models T_k \end{array} \right\} = \lambda_{i+1}.$$

Теперь рассмотрим $\varphi(o_k)\left(\pi_S^i\right)$ при $k > j_i$. Очевидно, что

$$\pi_S^i \not\models w = \mathsf{succ}(0),$$

поскольку $\pi_S^i w = 0$. Значит, $\varphi(o_k)\left(\pi_S^i\right) = \pi_S^i$.

Итак, мы доказали, что $\sigma_S^{i+1} = \Pi_1^S\left(\sigma_S^i\right) = \pi_S^i$. 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 заключительная.

Пусть $\Pi^L(\sigma)$ определено. Тогда последовательность расширенных состояний $\left(\sigma_L^i, \lambda_i\right)_i$ конечна. Предположим, что (σ_L^n, λ_n) — её последний элемент. Тогда λ_n должна быть заключительной меткой. То есть, $\lambda_n \neq \alpha_k$ для $k = 1, \ldots, m$. Это означает, что $\varphi(o_k)\left(\rho_S^i\right) = \rho_S^i$ для $k = 1, \ldots, m$. Но тогда $\sigma_S^{n+1} = \rho_S^n$ и

$$\sigma_S^{n+1} \not\models w = 0,$$

так как $\rho_S^n w = 1$. Это означает, что $\Pi^S(\sigma) = \sigma_S^{n+1}$. Но

$$\sigma_S^{n+1} \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_S^n \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_L^n \restriction \mathbf{Var}\left(\Pi^L\right).$$

Пусть теперь $\Pi^S(\sigma)$ определено. Тогда последовательность $\left(\sigma_S^i\right)_i$ конечна. Пусть σ_S^n — последний элемент. Это означает, что

$$\sigma_S^n \not\models w = 0.$$

Так как $\sigma_S^n = \Pi_1^S\left(\sigma_S^{n-1}\right)$, то $\Pi_1^S\left(\sigma_S^{n-1}\right)(w) \neq 0$. Если бы для некоторого $k = 1, \ldots, m$ имело бы место $\sigma_S^{n-1} y = \alpha_k$, то $\Pi_1^S\left(\sigma_S^{n-1}\right)(w) = 0$. Значит, $\alpha_k \neq \sigma_S^{n-1} y$ для всякого $k = 1, \ldots, m$. Согласно утверждению, $\sigma_S^{n-1} y = \lambda_{n-1}$. Это означает, что метка λ_{n-1} является заключительной и $\Pi^L(\sigma) = \sigma_L^{n-1}$. Далее, очевидно, что

$$\sigma_S^{n-1} \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_S^n \restriction \mathbf{Var}\left(\Pi^L\right).$$

Следовательно,

$$\begin{aligned} \Pi^L(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right) &= \sigma_L^{n-1} \restriction \mathbf{Var}\left(\Pi^L\right) = \\ &= \sigma_S^{n-1} \restriction \mathbf{Var}\left(\Pi^L\right) = \sigma_S^n \restriction \mathbf{Var}\left(\Pi^L\right) = \Pi^S(\sigma) \restriction \mathbf{Var}\left(\Pi^L\right). \end{aligned}$$

Задача 4.10. Структуризируйте программы Min с рис. 4.1 и Fib с рис. 4.2 методом, предложенным в теореме.

*__Задача 4.11.__ Модифицируйте доказательство теоремы так, что если исходная программа с метками является простой, то построенная структурная программа тоже будет простой.

Итак, мы доказали, что структурированные программы и программы с метками вычисляют одни и те же функции. Выведем следствие.

Следствие 4.5 (Теорема о цикле)**.** *Для всякой структурной программы* Π *существует структурная программа* Π_1*, которая содержит только один оператор цикла и*

$$\Pi\,(\sigma) \restriction \mathbf{Var}\,(\Pi) = \Pi_1\,(\sigma) \restriction \mathbf{Var}\,(\Pi)$$

для любого состояния σ *программы* Π_1*.*

Доказательство. Для Π существует программа с метками Π_2 такая, что

$$\Pi\,(\sigma) = \Pi_2\,(\sigma)$$

для любого состояния σ. По теореме о структуризации существует структурная программа Π_1, которая содержит один оператор цикла и

$$\Pi_1\,(\sigma) \restriction \mathbf{Var}\,(\Pi) = \Pi_2\,(\sigma) \restriction \mathbf{Var}\,(\Pi)$$

для любого состояния σ программы Π_1. Следовательно,

$$\Pi\,(\sigma) \restriction \mathbf{Var}\,(\Pi) = \Pi_1\,(\sigma) \restriction \mathbf{Var}\,(\Pi)$$

для любого состояния σ. □

*__Задача 4.12.__ Сформулируйте теорему о подстановке для программ с метками и, используя теоремы о построении программ с метками и о структуризации, докажите её.

*__Задача 4.13.__ 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.

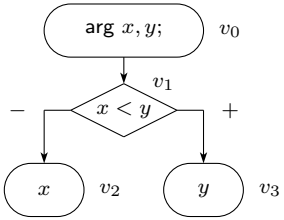


Рис. 4.3: Нахождение максимума.

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, тестов, имён переменных и записей вида*

$$\mathsf{arg}\ \overline{x};$$

для некоторых переменных $\overline{x}$*. Вершины, имеющие такие метки, мы будем называть вершинами присваивания, ветвления, конца и начала соответственно. Должны так же выполняться следующие условия:*

1. *Имеется ровно одна вершина начала. В эту вершину не ведёт ни одно ребро, а из нее исходит ровно одно ребро.*
2. *Ни из какой вершины конца не исходит ни одно ребро.*
3. *Из всякой вершины присваивания исходит ровно одно ребро.*
4. *Из всякой вершины ветвления исходят ровно два ребра, причём одно из них имеет метку «+», другое — «−».*

Традиционно, когда рисуют блок схемы, вершины присваивания обозначают прямоугольниками, вершины ветвления — ромбами, вершины начала и конца — овалами.

Пример 4.6. *Рассмотрим пример (рис. 4.3). Здесь вершина* v_0 *— вершина начала,* v_1 *— вершина теста,* v_2 *и* v_3 *— вершины конца.*

Определение 4.8 (Семантика блок-схем)**.** *Состояние блок-схемы* $\mathcal{B}$ *— пара* (v, σ)*, где* σ *— состояние,* v *— вершина графа. Результат применения блок-схемы к состоянию* (v, σ) *—* $\mathcal{B}(v, \sigma)$ *— новое*

состояние (w, τ), *если существует конечная последовательность состояний блок схемы* $\left(v^i, \sigma^i\right)$ *такая, что*

1. *вершины* $v^0, \ldots, v^n$ *образуют путь в* B, *причем* v^n — *вершина конца;*
2. $\left(v^0, \sigma^0\right) = (v, \sigma)$, $(w, \tau) = (v^n, \sigma^n)$;
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,*

$$L_V\left(v^i\right) \twoheadrightarrow x = e;$$

то $\sigma^{i+1} = (x = e;)\left(\sigma^i\right)$;

5. *если вершина* v^i, $i < n$ — *вершина теста,*

$$L_V\left(v^i\right) \twoheadrightarrow T;$$

то $\sigma^{i+1} = \sigma^i$, *и ребро, соединяющее* v^i *и* v^{i+1} *помечено «+», если* $\sigma^i \models T$, *помечено «−», если* $\sigma^i \not\models T$.

Пример 4.7. *Рассмотрим пример блок-схемы, изображенной на рис. 4.3 на противоположной странице. Возьмем состояние* (v_0, σ), *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,* $\sigma \models x < y$, *поэтому из двух ребер выбираем то, которое помечено «+». Заключительным будет состояние* (v_3, σ).

Аналогично определяются вызов блок-схемы, начальное состояние вызова, результат вызова и функция, вычисляемая блок-схемой.

Определение 4.9. *Вызов блок-схемы* $\mathcal{B}$ — *упорядоченная* n-*ка* $(\mathcal{B}, u_1, \ldots, u_n)$, *где* n *равно количеству аргументов блок-схемы. Начальное состояние вызова* — (v_0, σ), *где* v_0 — *вершина начала, а* $\sigma(x_i) = u_i$, *если* x_i — i-*ый аргумент блок-схемы. Результат вызова равен* t, *если для любого* σ — *начального состояния* — $\mathcal{B}(v_0, \sigma) = (w, \tau)$, *вершина конца* τ *помечена переменной* y, *и* $t = \tau y$. *Блок-схема вычисляет частичную функцию* f, *если* $f(\bar{t}) = \mathbf{Res}(\mathcal{B}, \bar{t})$ *для любых* $\bar{t}$.

Задача 4.14. Докажите, что блок-схема на рис. 4.3 на предыдущей странице вычисляет функцию максимума.

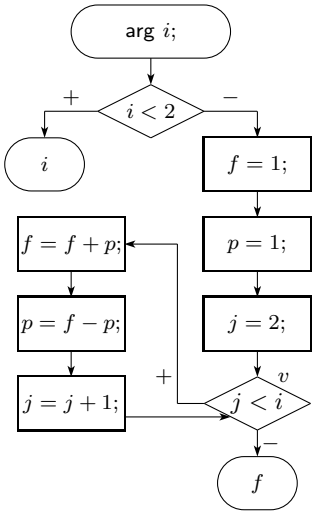


Рис. 4.4: Числа Фибоначчи.

Пример 4.8. *На рис. 4.4 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 периодически при переходе к v будет повторяться 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:

$$\sigma_{k+1} f = \sigma_k f + \sigma_k p = F_k + F_{k-1} = F_{k+1},$$

$$\sigma_{k+1} p = \sigma_{k+1} f - \sigma_k p = F_{k+1} - F_{k-1} = F_k.$$

При $k = i_0$ следующей за v окажется вершина конца, и результат вызова будет равен $\sigma_k f = F_{i_0}$.

Между блок-схемами и программами с метками существует тесная связь. Эту связь можно проследить, просто сравнив семантики программ с метками и блок-схем. В обоих случаях речь идет о последовтальностях

вида $(v_i, \sigma_i)_i$, где v_i тем или иным образом идентифицируют элементы программы или блок-схемы, причем правила построения этих последовательностей тоже схожи. Докажем эту эквивалентность.

Теорема 4.3. *По каждой программе с метками может быть эффективно построена эквивалентная блок-схема.*

Доказательство. Пусть П — программа с метками. Пусть L — множество меток программы П. Построим блок-схему $\mathcal{B}$. В качестве множества вершин нашей блок-схемы возьмем множество меток программы П.

Определим разметку вершин. Если λ — заключительная метка, то в блок-схеме $\mathcal{B}$ она будет отмечена некоторой переменной, следовательно, это будет вершина конца. Если в П имеется оператор $\beta\ x = e;\ \gamma$, то вершина β будет помечена оператором присваивания $x = e;$, а если есть оператор β if T then γ else δ, то — тестом T.

Теперь определим ребра и их разметку. Если в П имеется оператор $\beta\ x = e;\ \gamma$, то ребро из β будет идти в γ. Если в П имеется оператор β if T then γ else δ, то из β будет идти два ребра: в γ с меткой «+», в δ с меткой «−».

Поскольку мы в качестве множества вершин взяли множество меток программы П, то состояние блок-схемы (β, σ) можно рассматривать как расширенное состояние программы с метками П.

Продемонстрируем, что для любой метки β программы П и любого состояния σ выполнено $\Pi(\beta, \sigma) = \mathcal{B}(\beta, \sigma)$. Для этого покажем, что последовательность расширенных состояний $\left(\lambda_i, \sigma^i\right)_i$ программы П совпадет с последовательностью состояний $\left(\mu_i, \tau^i\right)_i$ блок-схемы $\mathcal{B}$. Используем индукцию по длине последовательности.

Базис индукции.

Первое состояние и в том, и в другом случае будет (β, σ).

Шаг индукции.

Пусть для i доказано: $\left(\lambda_i, \sigma^i\right) = \left(\mu_i, \tau^i\right)$.

Если λ_i — заключительная метка, то μ_i — вершина конца. Следовательно, и $\left(\lambda_i, \sigma^i\right)$, и $\left(\mu_i, \tau^i\right)$ будут последними элементами последовательностей.

Пусть в П есть оператор $\lambda_i\ x = e;\ \lambda_{i+1}$. Тогда следующим расширенным состоянием П будет $\left(\lambda_{i+1}, \sigma^{i+1}\right)$, где $\sigma^{i+1} = (x = e;)\left(\sigma^i\right)$. 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, следующее состояние блок-схемы также будет $\left(\lambda_{i+1}, \sigma^{i+1}\right)$.

Пусть в П есть оператор λ_i if T then γ else δ. Тогда следующим расши-

ренным состоянием Π будет $\left(\lambda_{i+1}, \sigma^i\right)$, где λ_{i+1} есть γ или δ в зависимости от того, выполняется T в σ^i или нет. 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 следующим состоянием блок-схемы будет $\left(\gamma, \sigma^i\right)$, если $\sigma^i \models T$, или $\left(\delta, \sigma^i\right)$ в противном случае. Таким образом, в обеих ситуациях получается новое состояние, совпадающее с $\left(\lambda_{i+1}, \sigma^i\right)$.

Чтобы достроить блок-схему, вычисляющую ту же функцию, что и алгоритм A, записанный в виде программы с метками, нужно к определению $\mathcal{B}$ добавить две детали. Во-первых, во всех вершинах конца написать имя выходной переменной A (то есть — имя алгоритма). Во-вторых нужно будет добавить вершину начала, помеченную аргументами алгоритма A, и добавить ребро, соединяющее A с вершиной — начальной меткой программы Π_A. □

Задача 4.15. Докажите, что блок-схема, построенная в конце доказательства будет вычислять ту же (частичную) функцию, что и исходный алгоритм.

Задача 4.16. Используя приведенный в доказательстве метод, постройте эквивалентную блок-схему для алгоритма на рис. 4.2 на с. 85.

Теорема 4.4. *По каждой блок-схеме может быть эффективно построена эквивалентная программа с метками.*

Доказательство. Это доказательство симметрично предыдущему. Пусть $\mathcal{B}$ — блок-схема, V — множество ее вершин. 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 ведет ребро в вершину w, то $\varphi(v)$ будет оператором $v\ x = e;\ w$. Если v — вершина теста, помеченная тестом T, из которой ведут ребра в вершины w_1 и w_2 с метками «+» и «−» соответственно, то $\varphi(v)$ будет оператором v if T then w_1 else w_2. Соединим все $\varphi(v)$ в одну программу.

Как и в предыдущем случае, каждое расширенное состояние (v, σ) полученной программы с метками Π можно рассматривать как состояние блок-схемы $\mathcal{B}$. Точно такими же рассуждениями, что и в предыдущей теореме доказывается, что для любого расширенного состояния (v, σ) будет выполнено $\Pi(v, \sigma) = \mathcal{B}(v, \sigma)$.

Для построения законченной записи алгоритма потребуется еще несколько действий. Пусть название алгоритма A отличается от всех, используемых в $\mathcal{B}$ переменных. Во-первых, соединяя все $\varphi(v)$ в одну программу с метками, следует позаботиться о том, чтобы начальной меткой полученной программы была метка v — вершина, в которую ведет ребро

из вершины начала. Во-вторых, возьмем новую заключительную метку α, а для любой старой вершины конца v, помеченной переменной x, добавим в программу оператор $v\ A = x;\ \alpha$. □

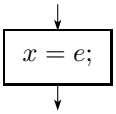
Задача 4.17. Докажите, что алгоритм, построенный в конце доказательства будет вычислять ту же (частичную) функцию, что и исходная блок-схема.

Задача 4.18. Постройте по блок-схеме на рис. 4.4 на с. 104 эквивалентную программу с метками.

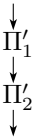
Таким образом, программы с метками и блок-схемы являются разными реализациями одной и той же концепции программирования.

Используя блок-схемы, можно проинтерпретировать доказательство теоремы 4.1 на с. 91 следующим образом. По каждой структурированной программе Π строится часть блок-схемы Π' с одним входом и одним выходом:

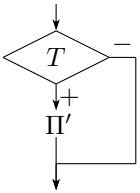
1. для присваивания $x = e;$:



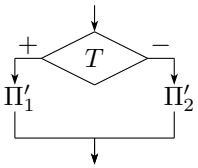
2. для следования $\Pi_1\Pi_2$:



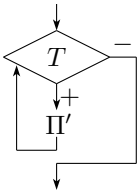
3. для неполного ветвления if T then Π end;:



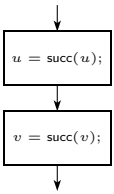
4. для полного ветвления if T then Π_1 else Π_2 end;:



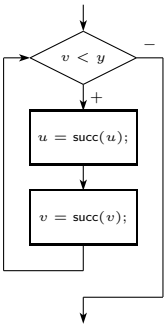
5. для цикла while T do Π end;:



Пример 4.9. *Построим таким образом блок-схему по структурной программе на рис. 3.2 на с. 50. Для тела цикла* Π_2 *мы получим:*



Для всего цикла будем иметь



Вся программа представлена в виде блок-схемы на рис. 4.5 на следующей странице.

Задача 4.19. Используя приведенный в примере метод, постройте эквивалентную блок-схему для алгоритма Евклида (рис. 3.4 на с. 53).

Задача 4.20. Докажите, что если тело алгоритма A — структурная программа Π не содержит циклов, то можно построить эквивалентную блок-схему в виде дерева.

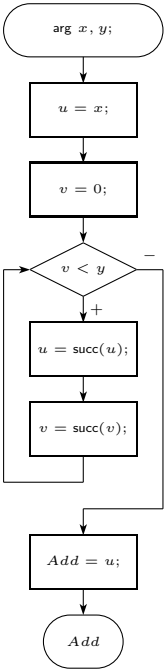


Рис. 4.5: Блок-схема для алгоритма сложения чисел

Глава 5

Доказательство корректности 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 состояний.

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

Определение 5.1 (Исчисление)**.** *Исчисление* $\mathfrak{I}$ *— это тройка* $(\mathfrak{S}, \mathfrak{A}, \mathfrak{R})$. $\mathfrak{S}$ *— множество синтаксических конструкций,* $\mathfrak{A} \subseteq \mathfrak{S}$ *— множество аксиом,* $\mathfrak{R}$ *— множество правил вывода. Правило вывода* r *— частичное отображение*

$$r : \mathfrak{S}^{<\omega} \to \mathcal{P}(\mathfrak{S}).$$

То есть по каждой конечной последовательности $s_1, \ldots, s_n$, $s_i \in S$ правило вывода даёт множество $\mathfrak{S}' \subseteq \mathfrak{S}$.

Определение 5.2 (Доказательство, вывод)**.** *Доказательство или вывод в исчислении $\mathfrak{I} = (\mathfrak{S}, \mathfrak{A}, \mathfrak{R})$ — это конечная последовательность $s_1, \ldots, s_n$, где $s_i \in \mathfrak{S}$ — синтаксические конструкции, и для каждого $i = 1, \ldots, n$ или $s_i \in \mathfrak{A}$ (то есть s_i — аксиома), или существует правило вывода $r \in \mathfrak{R}$ такое, что $s_i \in r\,(s_1, \ldots, s_{i-1})$ (то есть s_i получено из предыдущих элементов с помощью одного из правил вывода). Каждый вывод является выводом своей последней синтаксической конструкции.*

Определение 5.3 (Доказуемость, выводимость)**.** *Доказуемость (выводимость) в исчислении $\mathfrak{I}$ конструкции 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 доказуемость s в исчислении $\mathfrak{I}'$, полученном из $\mathfrak{I}$ добавлением аксиом $A_1, \ldots, A_n$.

Определение 5.4 (Условие)**.** *Условие для программы* П *— 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:

$$x < x^2 + y^2; \quad x < y \wedge y < z - x^3; \quad \exists v \left(x < v < y \wedge z^2 | v\right).$$

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) $\sigma \not\models \exists u \left(x < u < y \wedge z^2 | u\right)$, *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 3.8.

Определение 5.6 (Тройка Хоара)**.** *Тройка Хоара — запись вида* $\{\varphi\}\,\Pi\,\{\psi\}$, *где* Π — *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 писать предусловие, программу и постусловие в столбик, ограничивая его 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:

$$\left\{x < y + z^2\right\}\,\Pi\,\{x < y\}\,;$$

$$\{x < y\}\,\Pi\,\left\{\exists u \left(x < u < y \wedge z^2 | u\right)\right\};$$

$$\left\{\exists u \left(x < u < y \wedge z^2 | u\right)\right\}\,\Pi\,\left\{x < y + z^2\right\}.$$

Определение 5.7 (Частичная корректность)**.** *Тройка Хоара*

$$\{\varphi\}\,\Pi\,\{\psi\}$$

называется ч а с т и ч н о к о р р е к т н о й *на множестве состояний S, если для всякого состояния $\sigma \in S$ такого, что $\sigma \models \varphi$ и $\Pi(\sigma)$ определено, выполнено $\Pi(\sigma) \models \psi$. Тройка называется частично корректной, если она частично корректна на множестве всех состояний.*

Определение 5.8 (Полная корректность)**.** *Тройка Хоара*

$$\{\varphi\}\,\Pi\,\{\psi\}$$

называется п о л н о с т ь ю к о р р е к т н о й (и л и в п о л н е к о р р е к т н о й) *на множестве состояний S, если для любого состояния $\sigma \in S$ 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 состояний.*

Интуитивно это означает следующее. Предусловие — 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* $\sigma y \geq 5$, *то* $\Pi(\sigma)$ *не определено.*

Пусть

$$\varphi \Leftrightarrow x = y \wedge y < 3, \quad \psi \Leftrightarrow y^2 < 5.$$

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

$$\varphi \Leftrightarrow y - x = 4, \quad \psi \Leftrightarrow x + y = 1.$$

Тогда $\sigma \models x < y$. *Если* $\sigma y < 5$, *то* $\sigma y \leq 4$. *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* $\sigma \models \varphi$, $\Pi(\sigma)$ *не определено. Итак, тройка* $\{\varphi\}\,\Pi\,\{\psi\}$ *является частично корректной, но не является полностью корректной.*

Пусть

$$\varphi \Leftrightarrow x + y = 5, \quad \psi \Leftrightarrow x - y = 3.$$

Рассмотрим следующее состояние: $\sigma = \{(x, 2), (y, 3)\}$. *Очевидно,* $\sigma \models \varphi$. *Далее,* $\sigma \models x < y$ *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 3.5 напишите по три тройки Хоара, одна из которых должна быть полностью корректной, другая частично корректной, но не полностью корректной, третья — не частично корректной. Для всех ли программ это можно сделать?

Замечание 5.2. Если $\Pi(\sigma)$ определено для любого σ, то для любой тройки Хоара с программой Π полная и частичная корректности эквивалентны.

Замечание 5.3. Чтобы установить, что алгоритм

```
Alg A;
arg x_1, ..., x_n;
  Π_A
end;
```

вычисляет (частичную) функцию Φ_A достаточно доказать, что тройка

$$\{x_1 = v_1 \wedge \cdots \wedge x_n = v_n \wedge \overline{v} \in \operatorname{dom} \Phi_A\}\,\Pi_A\,\{A = \Phi_A(\overline{v})\}$$

полностью корректна. Предполагается, что переменные $\overline{v}$ не встречаются в Π_A.

Задача 5.4. Почему в предыдущем замечании необходимо, чтобы переменные $\overline{v}$ не встречались в Π_A? Приведите пример, который доказывает существенность этого условия.

§ 5.2. *Исчисление Хоара

Мы определим исчисление Хоара — $\mathfrak{IH}$[1]. Синтаксическими конструкциями будут тройки Хоара. Доказуемость тройки Хоара в этом исчислении будет означать полную и/или частичную корректность этой тройки.

У нас будут аксиомы вида:

АКС $\{(\varphi)^x_e\}\, x = e; \{\varphi\}$, где x — переменная, e — выражение.

Правила вывода исчисления $\mathfrak{IH}$:

УПР Усиление предусловия: если выводимо $\{\varphi\}\,\Pi\,\{\psi\}$ и формула $\theta \to \varphi$ истинна на предметной области, то выводимо $\{\theta\}\,\Pi\,\{\psi\}$.

ОП Ослабление постусловия: если выводимо $\{\varphi\}\,\Pi\,\{\psi\}$ и формула $\psi \to \theta$ истинна на предметной области, то выводимо $\{\varphi\}\,\Pi\,\{\theta\}$.

СЛ Правило следования: если выводимы тройки $\{\varphi\}\,\Pi_1\,\{\theta\}$ и $\{\theta\}\,\Pi_2\,\{\psi\}$, то выводимо $\{\varphi\}\,\Pi_1\Pi_2\,\{\psi\}$.

НВ Правило неполного ветвления: если выводимо $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ и формула $\varphi \wedge \neg T \to \psi$ истинна на предметной области, то выводимо

$$\{\varphi\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{end};\{\psi\}\,.$$

ПВ Правило полного ветвления: если выводимы тройки $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ и $\{\varphi \wedge \neg T\}\,\Pi_2\,\{\psi\}$, то выводимо

$$\{\varphi\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\psi\}\,.$$

Докажем, что с помощью аксиом можно доказать только вполне корректные тройки Хоара, а с помощью правил **УПР**–**ПВ** из корректных троек получать корректные. Говоря языком математической логики, докажем непротиворечивость исчисления Хоара.

[1]На самом деле мы определим два исчисления — одно для доказательства частичной корректности, другое — полной. Эти исчисления будут отличаться только правилом, которое используется для доказательства корректности цикла.

АКС Пусть $\Pi \Leftarrow x = e$; пусть $\sigma \models (\varphi)_e^x$, 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 $\{(\varphi)_e^x\}\,\Pi\,\{\varphi\}$.

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.

ОП 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.

СЛ 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.

НВ Пусть

$$\Pi \Leftarrow \text{if } T \text{ then } \Pi_1 \text{ end;}$$

Пусть тройка $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ корректна и формула $\varphi \wedge \neg T \to \psi$ истинна на предметной области. Пусть $\sigma \models \varphi$.

а) Частичная корректность. Пусть $\Pi(\sigma)$ определено.
Если $\sigma \models T$, то $\Pi(\sigma) = \Pi_1(\sigma)$ и $\sigma \models \varphi \wedge T$. Из-за частичной корректности $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ имеем $\Pi_1(\sigma) \models \psi$ и $\Pi(\sigma) \models \psi$.
Если $\sigma \not\models T$, то $\Pi(\sigma) = \sigma$. Кроме того, $\sigma \models \varphi \wedge \neg T$. 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 $\sigma \models T$, то $\Pi(\sigma) = \Pi_1(\sigma)$ и $\sigma \models \varphi \wedge T$. Так как $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ полностью корректна, то $\Pi_1(\sigma)$ определено и $\Pi_1(\sigma) \models \psi$. Следовательно, $\Pi(\sigma) \models \psi$.
Если $\sigma \not\models T$, то $\Pi(\sigma) = \sigma$. Кроме того, $\sigma \models \varphi \wedge \neg T$. 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.

ПВ Аналогично.

***Задача 5.5.** Докажите, что с помощью правила **ПВ** из частично (полностью) корректных троек можно получить только частично (полностью) корректные тройки.

****Замечание 5.4.** Обычно, когда вводят какие-либо исчисления, требуют, чтобы множество доказуемых конструкций было рекурсивно перечислимо. В нашем случае это не так, поскольку установить истинность формулы на алгебраической системе алгоритмически невозможно.

Можно ограничиться какой-либо рекурсивно 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**, **ОП** и **НВ** были не истинны на $(\omega, +, \times)$, а были теоремами арифметики Пеано. Это приведёт к сокращению множества доказуемых троек.

Вводя исчисление Хоара, мы преследовали цель сделать его не рекурсивно перечислимым, а удобным для доказательства корректности программ.

Замечание 5.5 (Корректность программ)**.** Если тройка

$$\{\text{ИСТИНА}\}\,\Pi\,\{\psi\}$$

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

$$\{\varphi\}\,\Pi\,\{\text{ЛОЖЬ}\}?$$

С помощью аксиомы **АКС** и правил **УПР**–**ПВ** можно доказывать корректность программ без циклов.

Замечание 5.6. Так как для любой программы Π без циклов, $\Pi(\sigma)$ определено для любого состояния σ, то для них полная и частичная корректности эквивалентны.

Пример 5.5. *Рассмотрим следующую программу:*

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

Докажем, что

$$\underset{\text{Э}\mathfrak{h}}{\triangleright}\ \{\text{ИСТИНА}\}\,\Pi\,\{z = \min\{x, y\}\}\,.$$

1. **АКС**. $\{x = \min\{x, y\}\}\ z = x;\{z = \min\{x, y\}\}$.
2. **УПР** : 1. *Так как формула*

$$x < y \wedge \text{ИСТИНА} \to x = \min\{x, y\}$$

истинна[2], *то*

$$\{x < y\}\ z = x;\{z = \min\{x, y\}\}\,.$$

3. **АКС**. $\{y = \min\{x, y\}\}\ z = y;\{z = \min\{x, y\}\}$.
4. **УПР** : 3. *Так как формула*

$$\neg x < y \wedge \text{ИСТИНА} \to y = \min\{x, y\}$$

истинна, то

$$\{\neg x < y\}\ z = y;\{z = \min\{x, y\}\}\,.$$

5. **ПВ** : 2, 4. $\{\text{ИСТИНА}\}\,\Pi\,\{z = \min\{x, y\}\}$.

[2]Здесь и дальше, говоря об истинности формулы, мы подразумеваем её истинность на нашей предметной области — множестве натуральных чисел со стандартными операциями.

Запись вида **АКС** *означает, что далее следует аксиома, а запись вида* **ПВ** : $2, 4$ *означает, что текущая тройка Хоара выводится из троек Хоара, полученных на шагах 2 и 4 с помощью правила* **ПВ**.

Пример 5.6. *Для следующей программы докажем полную корректность тройки*

$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y}\} \\ \Pi \\ \{x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}\} \end{array}\right|$$

то есть множество $\{x, y\}$ *не изменяется, но* x *становиться не больше* y.

$$\Pi \Leftarrow \begin{cases} \mathsf{if}\ y\ <\ x\ \mathsf{then} \\ \quad t\ =\ y; \\ \quad y\ =\ x; \\ \quad x\ =\ t; \\ \mathsf{end}; \end{cases}$$

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. **СЛ** : $1, 2$.
$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} \\ t = y; y = x; \\ \{y = \hat{x} \wedge t = \hat{y} \wedge t < y\} \end{array}\right|.$$
5. **СЛ** : $4, 3$.
$$\left|\begin{array}{c} \{x = \hat{x} \wedge y = \hat{y} \wedge y < x\} \\ t = y; y = x; x = t; \\ \{y = \hat{x} \wedge x = \hat{y} \wedge x < y\} \end{array}\right|.$$
6. **ОП** : 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. **НВ** : 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,
$$\{x = \hat{x} \wedge y = \hat{y}\}\, \Pi\, \{x = \min\{\hat{x}, \hat{y}\} \wedge y = \max\{\hat{x}, \hat{y}\}\}.$$

```
Alg Mid;
arg x, y, z;
  if x < y then
    if x < z then
      if y < z then
        Mid = y;
      else
        Mid = z;
      end;
    else
      Mid = x;
    end;
  else
    if x < z then
      Mid = x;
    else
      if y < z then
        Mid = z;
      else
        Mid = y;
      end;
    end;
  end;
end;
```

Рис. 5.1: Среднее из трёх чисел.

***Задача 5.7.** Докажите, что программа на рис. 5.1 находит среднее из трёх чисел.

Итак, мы научились доказывать корректность программ, не содержащих циклов. Прежде чем переходить к циклам дадим несколько определений.

Определение 5.9 (Инвариант программы)**.** *Условие φ — (частичный) инвариант программы Π, если тройка $\{\varphi\}\,\Pi\,\{\varphi\}$ является полностью (частично) корректной.*

Определение 5.10 (Инвариант цикла)**.** *Условие φ называется (частичным) инвариантом цикла*

$$\begin{array}{l}\mathsf{while}\ T\ \mathsf{do}\\ \quad \Pi\\ \mathsf{end};\end{array}$$

если тройка $\{\varphi \wedge T\}\,\Pi\,\{\varphi\}$ полностью (частично) корректна,

Очевидно, что каждый инвариант программы (цикла) является и частичным инвариантом программы (цикла). Обратное, естественно, неверно.

Пример 5.7. $\Pi \twoheadleftarrow t = x; x = y; y = t;$. *Пусть $\varphi \twoheadleftarrow \{x, y\} = \{u, v\}$. 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 5.8. *Покажем, что $u - v = w$ — инвариант следующего цикла:*

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

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 5.7. Если φ — инвариант программы Π, то φ — инвариант цикла

$$\textit{while}\ T\ \textit{do}\ \Pi\ \textit{end;}$$

для любого T.

Обратное неверно. Рассмотрим пример.

Пример 5.9. *Тот же цикл, но*

$$\varphi \twoheadrightarrow u - v = w \wedge u \leq x.$$

Пусть $\sigma \models \varphi$. *Мы уже знаем из предыдущего примера, что* $\Pi(\sigma) \models u - v = w$. *Так как* $\sigma \models u \leq x \wedge u < x$, *то* $\sigma \models u + 1 \leq x$. *Значит,*

$$\Pi(\sigma)(u) = \sigma u + 1 \leq \sigma x = \Pi(\sigma)(x).$$

То есть $\Pi(\sigma) \models u \leq x$. *Следовательно,* φ — *инвариант цикла.*

То, что φ *не является инвариантом тела цикла, легко убедиться, рассмотрев состояние* σ *такое, что* $\sigma u = \sigma x = \sigma v + \sigma w$. *Тогда*

$$\Pi(\sigma)(u) = \sigma u + 1 > \sigma x = \Pi(\sigma)(x),$$

то есть $\Pi(\sigma) \not\models \varphi$, *хотя* $\sigma \models \varphi$.

Определение 5.11 (Ограничитель цикла)**.** *Пусть дан цикл*

```
while T do
  Π
end;
```

и предусловие φ. *Ограничителем цикла при предусловии* φ *называется выражение* L_φ *такое, что*

1) $\sigma(L_\varphi) \in \omega$ *для всякого состояния* σ;

2) $\Pi(\sigma)(L_\varphi) < \sigma(L_\varphi)$ *для всякого состояния* σ *такого, что* $\sigma \models \varphi \wedge T$.

Если предусловие φ *истинно на предметной области, то такой ограничитель будет ограничителем и при любом другом предусловии. Мы будем называть его универсальным ограничителем.*

Пример 5.10. *Рассмотрим тот же цикл, что в примере 5.8. Пусть* $\varphi \twoheadrightarrow$ ИСТИНА *и* $L = \lfloor x - u \rfloor$. *Очевидно, что* $\sigma(L) \in \omega$ *для любого* σ. *Пусть* $\sigma \models \varphi \wedge T$, *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*

$$\sigma x - \sigma u > 0, \quad \sigma x - (\sigma u + 1) \geq 0.$$

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 5.1 (Полная корректность цикла)**.** *Пусть дан цикл*

$$\Pi \Leftarrow \begin{cases} \text{while } T \text{ do} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$

φ — инвариант цикла, L_φ — ограничитель. Тогда $\Pi(\sigma)$ определено и $\Pi(\sigma) \models \varphi \wedge \neg T$ для всякого состояния σ, для которого $\sigma \models \varphi$.

Доказательство. Пусть $\sigma \models \varphi$. Рассмотрим последовательность состояний цикла $\left(\sigma^i\right)_i$: $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma)$. Индукцией по i докажем, что если $\sigma^i \models T$, то $\Pi_1\left(\sigma^i\right)$ определено и $\sigma^{i+1} \models \varphi$.

Базис индукции.

Если $i = 0$, то $\sigma^0 \models \varphi$.

Шаг индукции.

Пусть для i доказано и $\sigma^i \models T$. Так как тройка $\{\varphi \wedge T\}\,\Pi_1\,\{\varphi\}$ полностью корректна, то $\Pi_1\left(\sigma^i\right)$ определено и $\sigma^{i+1} \models \varphi$.

Рассмотрим последовательность $\left(\sigma^i(L_\varphi)\right)_i$. Если предположить, что $\sigma^i \models T$ для всех $i \in \omega$, то, так как L_φ — ограничитель, и $\sigma^i \models \varphi \wedge T$, $\left(\sigma^i(L_\varphi)\right)_i$ — убывающая цепь натуральных чисел. Следовательно, она не может быть бесконечной, противоречие. Это означает, что должен существовать элемент σ^n такой, что $\sigma^n \not\models T$ и $\sigma^i \models T$ для $i < n$. По определению семантики цикла $\Pi(\sigma)$ определено и $\Pi(\sigma) = \sigma^n$. Получаем, что $\Pi(\sigma) \models \varphi \wedge \neg T$. □

Лемма 5.2 (Об ограничителе цикла)**.** *Если переменная z не входит в условия φ, ψ, тест T, выражение L и программу Π_1 и тройка*

$$\{\varphi \wedge T \wedge L = z\}\,\Pi_1\,\{\psi \wedge L < z\}$$

полностью корректна и $\sigma(L) \in \omega$ при любом состоянии σ, то L — ограничитель цикла

$$\text{while } T \text{ do } \Pi_1 \text{ end;}$$

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

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

ЦЧ Для частичной корректности. Если φ — частичный инвариант цикла Π, то $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$. То есть, из частичной корректности $\{\varphi \wedge T\}\,\Pi_1\,\{\varphi\}$ следует частичная корректность $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$.

ЦП Для полной корректности. Если φ — инвариант цикла Π и L_φ — ограничитель Π, то $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$. То есть, из полной корректности тройки

$$\{\varphi \wedge T \wedge L = z\}\,\Pi_1\,\{\varphi \wedge L < z\},$$

где z не входит в φ, T, L и Π_1, следует полная корректность тройки $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$. Предполагается, что выражение L должно принимать только натуральные значения.

Докажем, что с помощью этих правил из корректных троек получаются корректные. Для правила **ЦП** это следует из лемм 5.1 и 5.2.

Докажем это утверждение для правила **ЦЧ**. Пусть $\sigma \models \varphi$ и $\Pi(\sigma)$ определено. Тогда существует последовательность $\left(\sigma^i\right)_{i=0}^{n}$:

$$\sigma^0 = \sigma, \quad \sigma^{i+1} = \Pi_1\left(\sigma^i\right), \quad \sigma^i \models T \iff i < n, \quad \Pi(\sigma) = \sigma^n.$$

Так же как в лемме 5.1 доказывается, что $\sigma^i \models \varphi$. Следовательно, $\sigma^n \models \varphi \wedge \neg T$, то есть $\Pi(\sigma) \models \varphi \wedge \neg T$ и тройка $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$ частично корректна.

Рассмотрим пример.

Пример 5.11. *Мы докажем полную корректность тройки*

$$\{\text{ИСТИНА}\}\,\Pi\,\{u = \lfloor x - y \rfloor\}$$

для следующей программы:

$$\left.\begin{array}{l} u := 0;\ v := y; \\ \mathsf{while}\ v < x\ \mathsf{do} \\ \quad \left.\begin{array}{l} u := \mathsf{succ}(u); \\ v := \mathsf{succ}(v); \end{array}\right\} \Pi_3 \\ \mathsf{end}; \end{array}\right\} \Pi_2 \Bigg\} \Pi$$

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. **ОП** : 2. *Формула*

$$y = v \wedge u = 0 \rightarrow \\ \rightarrow \underbrace{\left(v = y \wedge x \leq y \wedge u = \lfloor x - y \rfloor \bigvee v \leq x \wedge v \geq y \wedge u = \lfloor v - y \rfloor\right)}_{\psi}$$

истинна, поэтому $\{y = y \wedge u = 0\}\, v = y; \{\psi\}$.

4. **СЛ** : 1, 3. $\{0 = 0 \wedge y = y\}\, u = 0; v = y; \{\psi\}$.
5. **УПР** : 4. *Очевидно, что формула*

$$\text{ИСТИНА} \rightarrow y = y \wedge 0 = 0$$

истинна. Поэтому

$$\{\text{ИСТИНА}\}\, u = 0; v = y; \{\psi\}\,.$$

- *Прежде, чем продолжать, преобразуем вторую часть* ψ*:*

$$v \leq x \wedge v \geq y \wedge u = \lfloor v - y \rfloor \equiv v \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor (v + 1) - y \rfloor.$$

Рассмотрим $\psi \wedge v < x$*:*

$$\psi \wedge v < x \equiv \Big(v = y \wedge x \leq y \wedge u = \lfloor x - y \rfloor \bigvee \\ \bigvee v \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor\Big) \wedge v < x \equiv \\ \equiv v = y \wedge x \leq y \wedge u = \lfloor x - y \rfloor \wedge v < x \bigvee \\ \bigvee v \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor \wedge v < x \equiv \\ \equiv \left\{ \begin{array}{c} v = y \wedge x \leq y \wedge v < x \equiv \text{ЛОЖЬ} \\ v \leq x \wedge v < x \equiv v < x \equiv v + 1 \leq x \end{array} \right\} \equiv \\ \equiv v + 1 \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor$$

6. **АКС**.

$$\left| \begin{array}{c} \{v + 1 \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor\} \\ u = succ\,(u)\,; \\ \{v + 1 \leq x \wedge v + 1 > y \wedge u = \lfloor v + 1 - y \rfloor\} \end{array} \right|.$$

7. **АКС**.

$$\left| \begin{array}{c} \{v + 1 \leq x \wedge v + 1 > y \wedge u = \lfloor v + 1 - y \rfloor\} \\ v = succ\,(v)\,; \\ \{v \leq x \wedge v > y \wedge u = \lfloor v - y \rfloor\} \end{array} \right|.$$

8. **ОП** : 7. *Формула*
$$v > y \wedge u = \lfloor v - u \rfloor \wedge v \leq x \to \psi$$
истинна. Значит
$$\{v + 1 \leq x \wedge v + 1 > y \wedge u = \lfloor v + 1 - y \rfloor\}\, v = succ\,(v)\,; \{\psi\}\,.$$
9. **СЛ** : 6, 8. $\{v + 1 \leq x \wedge v + 1 > y \wedge u + 1 = \lfloor v + 1 - y \rfloor\}\, \Pi_3\, \{\psi\}$.
10. **УПР** : 9. $\{\psi \wedge v < x\}\, \Pi_3\, \{\psi\}$.
11. **ЦП** : 10. ψ — *инвариант.* $L_\psi = x - v$ — *ограничитель. Значит* $\{\psi\}\, \Pi_2\, \{\psi \wedge \neg v < x\}$.

- *Преобразуем последнюю формулу:*

$$\psi \wedge \neg v < x \equiv \psi \wedge v \geq x \equiv$$
$$\equiv v = y \wedge x \leq y \wedge u = \lfloor x - y \rfloor \wedge v \geq x \bigvee$$
$$\bigvee v \leq x \wedge v \geq y \wedge u = \lfloor v - y \rfloor \wedge v \geq x \equiv$$
$$\equiv \left\{ \begin{array}{c} v \leq x \wedge v \geq x \equiv x = v \\ v = y \wedge x \leq y \wedge v \geq x \equiv v = y \wedge x \leq v \end{array} \right\} \equiv$$
$$\equiv v = y \wedge x \leq v \wedge u = \lfloor x - y \rfloor \bigvee v \geq y \wedge v = x \wedge u = \lfloor v - y \rfloor \equiv$$
$$\equiv \left\{ \; v = x \wedge u = \lfloor v - y \rfloor \equiv v = x \wedge u = \lfloor x - y \rfloor \; \right\} \equiv$$
$$\equiv u = \lfloor x - y \rfloor \wedge \Big(v = y \wedge x \leq v \bigvee v \geq y \wedge v = x \Big)$$

Это значит, что формула $\psi \wedge \neg v < x \to u = \lfloor x - y \rfloor$ *истинна.*

12. **ОП** : 11. $\{\psi\}\, \Pi_2\, \{u = \lfloor x - y \rfloor\}$.
13. **СЛ** : 5, 12. $\{\text{ИСТИНА}\}\, \Pi\, \{u = \lfloor x - y \rfloor\}$.

Мы формально доказали, что программа Π *действительно выполняет вычитание чисел.*

Мы часто в дальнейшем будем пользоваться следующей леммой.

Лемма 5.3 (О замене переменной)**.** *Если*
$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\, \Pi\, \{\psi\}\,,$$
то
$$\underset{\mathfrak{IH}}{\rhd} \left\{ (\varphi)^x_y \right\} (\Pi)^x_y \left\{ (\psi)^x_y \right\},$$
где y — *переменная, которая не встречается в доказательстве тройки* $\{\varphi\}\, \Pi\, \{\psi\}$ *(и, следовательно, в программе* Π*).*

Доказательство. Пусть $\mathcal{D}$ — доказательство $\{\varphi\}\,\Pi\,\{\psi\}$. Заменим всюду в $\mathcal{D}$ переменную x на y. Нужно теперь доказать, что полученная последовательность снова будет доказательством. Индукция по длине доказательства.

Базис индукции.

Аксиома: $\{(\varphi)^z_e\}\,z = e;\{\varphi\}$. Если $z \neq x$, то

$$((\varphi)^z_e)^x_y \multimap \left((\varphi)^x_y\right)^z_{(e)^x_y}$$

$$(z = e;)^x_y \multimap z = (e)^x_y$$

Получаем

$$\left\{\left((\varphi)^x_y\right)^z_{(e)^x_y}\right\} z = (e)^x_y\,;\left\{(\varphi)^x_y\right\}$$

что является аксиомой.

Если $z \multimap x$, то

$$((\varphi)^x_e)^x_y \multimap (\varphi)^x_{(e)^x_y} \multimap \left((\varphi)^x_y\right)^y_{(e)^x_y}$$

$$(x = e;)^x_y \multimap y = (e)^x_y$$

Получаем

$$\left\{\left((\varphi)^x_y\right)^y_{(e)^x_y}\right\} y = (e)^x_y\,;\left\{(\varphi)^x_y\right\}$$

что тоже является аксиомой.

Шаг индукции.

Индукционный шаг состоит в доказательстве того, что после замены переменных все применения правил останутся верными. Докажем два из них.

HB Пусть тройка

$$\{\varphi\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{end};\{\psi\}$$

получена из тройки $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ и истинности $\varphi \wedge \neg T \to \psi$. По индукционному предположению

$$\underset{\mathfrak{IS}}{\rhd} \left\{(\varphi)^x_y \wedge (T)^x_y\right\} (\Pi_1)^x_y \left\{(\psi)^x_y\right\}.$$

Формула $(\varphi)^x_y \wedge \neg\,(T)^x_y \to (\psi)^x_y$ так же будет истинна. По правилу **HB** получаем

$$\left\{(\varphi)^x_y\right\}\ \mathsf{if}\ \ (T)^x_y\ \ \mathsf{then}\ \ (\Pi_1)^x_y\ \ \mathsf{end};\left\{(\psi)^x_y\right\}.$$

ЦП Пусть из полной корректности тройки

$$\{\varphi \wedge T \wedge L = z\}\, \Pi_1\, \{\varphi \wedge L < z\}$$

получена полная корректность

$$\{\varphi\}\, \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};\{\varphi \wedge \neg T\}\,.$$

По индукционному предположению получаем доказуемость

$$\left\{(\varphi)^x_y \wedge (T)^x_y \wedge (L)^x_y = z\right\} (\Pi_1)^x_y \left\{(\varphi)^x_y \wedge (L)^x_y < z\right\}.$$

Так как y отличается в том числе и от z, то с помощью правила **ЦП** можно вывести

$$\left\{(\varphi)^x_y\right\} \mathsf{while}\ \ (T)^x_y\ \ \mathsf{do}\ \ (\Pi_1)^x_y\ \ \mathsf{end};\left\{(\varphi)^x_y \wedge \neg\,(T)^x_y\right\}. \qquad \square$$

****Задача 5.10.** Докажите, что из истинности формулы $\varphi \wedge \neg T \to \psi$ следует истинность $(\varphi)^x_y \wedge \neg\,(T)^x_y \to (\psi)^x_y$.

***Задача 5.11.** Докажите индукционный шаг для остальных правил.

Рассмотрим некоторые способы упростить формальные доказательства корректности.

Определение 5.12 (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 доказуемых конструкций.*

То есть аксиома или правило допустимы, если всё, что можно доказать с их использованием, можно доказать и без них. Введём некоторые новые аксиомы и правила, которые могут упростить доказательство, и докажем их допустимость.

ПРИСВ Аксиома:

$$\{\varphi\}\, x = e;\{\exists u\, ((\varphi)^x_u \wedge x = (e)^x_u)\}$$

при условии, что u не входит ни в φ, ни в e.

Доказательство. Аксиома **АКС**:

$$\left\{\exists u\left(((\varphi)^x_u)^x_e \wedge e = ((e)^x_u)^x_e\right)\right\} x = e;\{\exists u\, ((\varphi)^x_u \wedge x = (e)^x_u)\}$$

Так как u не входит ни в φ ни в e, то $\exists u\left(((\varphi)^x_u)^x_e \wedge e = (e)^x_u\right)$ 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:

$$\varphi \to \exists u \left(\left((\varphi)_u^x \right)_e^x \wedge e = \left((e)_u^x \right)_e^x \right).$$

ПРИСВ* Аксиома:

$$\{\varphi\}\, x = e; \{\varphi \wedge x = e\}$$

при условии, что x не входит ни в φ, ни в e.

Доказательство. Предыдущая аксиома:

$$\{\varphi\}\, x = e; \{\exists u \left((\varphi)_u^x \wedge x = (e)_u^x \right)\}.$$

Так как x не входит в e, то $(e)_u^x \circ\!\!- e$. Так как x не входит в φ, то $(\varphi)_u^x \circ\!\!- \varphi$. Так как u не входит в формулу под квантором, то этот квантор можно удалить. По правилу **ОП** получаем

$$\{\varphi\}\, x = e; \{\varphi \wedge x = e\}. \qquad \square$$

С помощью предыдущих аксиом и правила **СЛ** легко получить два правила:

ПРИСВ, **ПРИСВ*** Если выводимо $\{\varphi\}\, \Pi\, \{\psi\}$, то выводимы тройки

$$\{\varphi\}\, \Pi\, x = e; \{\exists u \left((\psi)_u^x \wedge x = (e)_u^x \right)\};$$

$$\{\varphi\}\, \Pi\, x = e; \{\psi \wedge x = e\}.$$

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.

ДОБ Правило вывода: если выводимо $\{\varphi\}\, \Pi\, \{\psi\}$ и $\mathbf{LVar}\,(\Pi) \cap \mathbf{Var}\,(\theta) = \varnothing$, то выводимо

$$\{\varphi \wedge \theta\}\, \Pi\, \{\psi \wedge \theta\}.$$

Доказательство. Пусть $\mathcal{D}$ — доказательство тройки $\{\varphi\}\, \Pi\, \{\psi\}$.

Прежде всего, в доказательстве могут быть использования правила **ЦП**, 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 доказательств, содержащих меньше n переходов, доказано, и $\mathcal{D}$ содержит их n штук. Пусть t — тройка, которая доказывается по правилу **ЦП**. 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 **ЦП** меньше чем в $\mathcal{D}$. Заменим в $\mathcal{D}_1$ все вхождения

переменной z на новую переменную $\hat{z}$. Получим новое доказательство $\hat{\mathcal{D}}_1$. Рассмотрим последовательность $\hat{\mathcal{D}}_1 t \mathcal{D}_1 \mathcal{D}_2$. Тогда эта последовательность является доказательством той же тройки, что и исходное доказательство, и оно содержит меньше переходов по правилу **ЦП** с использованием z.

Теперь можно использовать индукцию по доказательству $\{\varphi\}\,\Pi\,\{\psi\}$.

Базис индукции.
Если применяется аксиома **АКС**, то

$$\{(\varphi)^x_e\}\, x = e; \{\varphi\}\,.$$

Так как $x \in \mathbf{LVar}\,(\Pi)$, то $x \notin \mathbf{Var}\,(\theta)$. Следовательно, $(\theta)^x_e \leftrightarrows \theta$, поэтому:

$$\{(\varphi)^x_e \wedge \theta\}\, x = e; \{\varphi \wedge \theta\}$$

тоже является аксиомой.

Шаг индукции.
Рассмотрим, например, индукционный шаг для **ПВ**. В прежнем доказательстве тройка

$$\{\varphi\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\psi\}$$

получалась из троек $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ и $\{\varphi \wedge \neg T\}\,\Pi_2\,\{\psi\}$. По индукционному предположению тройки $\{\varphi \wedge \theta \wedge T\}\,\Pi_1\,\{\psi \wedge \theta\}$ и $\{\varphi \wedge \theta \wedge \neg T\}\,\Pi_2\,\{\psi \wedge \theta\}$ доказуемы. Следовательно, доказуема и тройка

$$\{\varphi \wedge \theta\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\psi \wedge \theta\}$$

по тому же правилу **ПВ**. □

***Задача 5.13.** Докажите, что последовательность $\hat{\mathcal{D}}_1 t \mathcal{D}_1 \mathcal{D}_2$, построенная в доказательстве, является выводом той же тройки, 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.

Пример 5.12. *Докажем корректность программы деления, приведённой на рис. 5.2.*

1. **ПРИСВ***. $\{\text{ИСТИНА}\}\, u = 0; \{u = 0\}$[3].

[3]На самом деле здесь использована аксиома **ПРИСВ*** и применено правило **ОП**, чтобы убрать из постусловия ИСТИНА. В дальнейшем, мы будем убирать ИСТИНА таким способом, не упоминая об этом специально.

```
Alg Div;
arg x, y;
  u = 0; v = 0;
  while u + y ≤ x do
    u = u + y;
    v = v + 1;
  end;
  Div = v;
end;
```

Π_1

Рис. 5.2: Деление чисел

2. **ПРИСВ***: 1. $\{\text{ИСТИНА}\}\, u = 0; v = 0; \{u = 0 \wedge v = 0\}$.
3. **ОП**: 2. *Формула*
$$u = 0 \wedge v = 0 \to vy = u \wedge u \leq x,$$
истинна, следовательно,
$$\{\text{ИСТИНА}\}\, u = 0; v = 0; \{vy = u \wedge u \leq x\}\,.$$
4. **ДОБ**: 3. $\{y > 0\}\, u = 0; v = 0; \{vy = u \wedge u \leq x \wedge y > 0\}$
5. **ПРИСВ**.
$$\left|\begin{array}{c} \{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x}\} \\ u = u + y; \\ \{\exists w\,(vy = w \wedge w \leq x \wedge w + y \leq x \wedge u = w + y \wedge x - w = \hat{x})\} \end{array}\right|$$
6. **ОП**: 5. *Формула*
$$\exists w\,(vy = w \wedge w \leq x \wedge w + y \leq x \wedge u = w + y \wedge x - w = \hat{x}) \to \\ \to vy = u - y \wedge u \leq x \wedge x - u = \hat{x} - y$$
истинна, поэтому
$$\left|\begin{array}{c} \{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x}\} \\ u = u + y; \\ \{vy = u - y \wedge u \leq x \wedge x - u = \hat{x} - y\} \end{array}\right|$$
7. **ПРИСВ**: 6.
$$\left|\begin{array}{c} \{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x}\} \\ u = u + y;\ v = v + 1; \\ \{\exists z\,(zy = u - y \wedge u \leq x \wedge x - u = \hat{x} - y \wedge v = z + 1)\} \end{array}\right|$$

8. **ОП**: 7. *Формула*

$$\exists z\,(zy = u - y \wedge u \leq x \wedge x - u = \hat{x} - y \wedge v = z + 1) \rightarrow$$

$$\rightarrow vy = u \wedge u \leq x \wedge x - u = \hat{x} - y$$

истинна, из чего следует

$$\left|\begin{array}{c}\{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x}\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leq x \wedge x - u = \hat{x} - y\}\end{array}\right|$$

9. **ДОБ**: 8.

$$\left|\begin{array}{c}\{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x} \wedge y > 0\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leq x \wedge x - u = \hat{x} - y \wedge y > 0\}\end{array}\right|$$

10. **ОП**: 9. *Формула*

$$vy = u \wedge u \leq x \wedge x - u = \hat{x} - y \wedge y > 0 \rightarrow$$

$$\rightarrow vy = u \wedge u \leq x \wedge x - u < \hat{x} \wedge y > 0$$

истинна, получаем

$$\left|\begin{array}{c}\{vy = u \wedge u \leq x \wedge u + y \leq x \wedge x - u = \hat{x} \wedge y > 0\} \\ u = u + y;\ v = v + 1; \\ \{vy = u \wedge u \leq x \wedge x - u < \hat{x} \wedge y > 0\}\end{array}\right|$$

11. **ЦП**: 10.

$$\left|\begin{array}{c}\{vy = u \wedge u \leq x \wedge y > 0\} \\ \Pi_1 \\ \{vy = u \wedge u \leq x \wedge y > 0 \wedge u + y > x\}\end{array}\right|$$

12. **ОП**: 11. *Формула*

$$vy = u \wedge u \leq x \wedge y > 0 \wedge u + y > x \rightarrow v = \left[\frac{x}{y}\right]$$

истинна, поэтому

$$\{vy = u \wedge u \leq x \wedge y > 0\}\,\Pi_1\left\{v = \left[\frac{x}{y}\right]\right\}$$

С помощью квадратных скобок мы обозначаем целую часть частного, например,

$$\left[\frac{10}{3}\right] = 3; \quad \left[\frac{6}{4}\right] = 1.$$

```
Alg Sum;
arg x;
  r = 0; y = x;
  while 0 < y do
    r = r + y mod 10;   } Π1
    y = y / 10;         }
  end;                  } Π2
  Sum = r;
end;
```

Рис. 5.3: Сумма цифр числа.

13. **СЛ**: 4, 12. $\{y > 0\}\, u = 0; v = 0; \Pi_1 \left\{v = \left[\frac{x}{y}\right]\right\}$
14. **АКС**. $\left\{v = \left[\frac{x}{y}\right]\right\} Div = v; \left\{Div = \left[\frac{x}{y}\right]\right\}$
15. **СЛ**: 13, 14. $\{y > 0\}\, \Pi_{Div} \left\{Div = \left[\frac{x}{y}\right]\right\}$

Пример 5.13. *Рассмотрим алгоритм нахождения суммы цифр числа, приведённый на рис. 5.3. Пусть $l(i)$ — последняя цифра числа i, а $s(i)$ — сумма его цифр.*

1. **ПРИСВ***. $\{\text{ИСТИНА}\}\, r = 0; \{r = 0\}$
2. **ПРИСВ*** : 1. $\{\text{ИСТИНА}\}\, r = 0; y = x; \{r = 0 \wedge y = x\}$.
3. **ОП** : 2. $\{\text{ИСТИНА}\}\, r = 0; y = x; \{s(x) = s(y) + r\}$.
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. **ОП** : 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. **СЛ** : 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. **ЦП** : 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. **ОП** : 9. $\{s\,(x) = s\,(y) + r\}\,\Pi_2\,\{s\,(x) = r\}$
11. **ПРИСВ*** : 10.
$$\left|\begin{array}{c}\{s\,(x) = s\,(y) + r\}\\ \Pi_2\ Sum = r;\\ \{s\,(x) = r \wedge Sum = r\}\end{array}\right|$$
12. **ОП** : 11. $\{s\,(x) = s\,(y) + r\}\,\Pi_2\ Sum = r; \{Sum = s\,(x)\}$
13. **СЛ** : 3, 11. $\{\text{ИСТИНА}\}\,\Pi_{Sum}\,\{Sum = s\,(x)\}$

Задача 5.16. Докажите корректность ранее написанных программ:

1) сложения чисел;
2) умножения чисел (используя сложение);
3) нахождения остатка (используя сложение и вычитание);

***Задача 5.17.** Напишите программы и докажите их корректность:

1) нахождение чисел Фибоначчи;
2) вычисление факториала;
3) нахождение двоичного логарифма.

Из операций разрешается использовать только *succ*.

Определение 5.13 (Спецификация программы)**.** *Спецификация программы — 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 в состоянии сформулировать, что эти части должны делать, следовательно, мы в состоянии определить предусловие и постусловие для каждой из них.

Определение 5.14 (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 наиболее сложные вещи: инварианты циклов, ограничители циклов и ограничители вызовов (для рекурсивных подпрограмм), а так же предусловия и постусловия для подпрограмм.

§ 5.3. *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 корректности программы. Поэтому интерес представляют, прежде всего, такие тройки, у которых предусловие описывает как можно больше ситуаций, то есть является как можно более слабым.

Определение 5.15 (Слабейшее предусловие)**.** *Пусть* Π *— программа,* ψ *— условие. Условие* φ *называется* *слабейшим предусловием для* Π*–*ψ*", если тройка Хоара* $\{\varphi\}\,\Pi\,\{\psi\}$ *частично корректна и для любого состояния* 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 5.16 (Полное слабейшее предусловие)**.** *Слабейшее предусловие* φ *для* $\Pi\,\{\psi\}$ *называется* *полным, если тройка* $\{\varphi\}\,\Pi\,\{\psi\}$ *является полностью корректной.*

Следствие 5.3. *Всякое полное слабейшее предусловие является слабейшим предусловием.*

Следствие 5.4. *Если тройка* $\{\text{ИСТИНА}\}\,\Pi\,\{\psi\}$ *частично (полностью) корректна, то* ИСТИНА *— (полное) слабейшее предусловие для* $\Pi\,\{\psi\}$*.*

Задача 5.18. Докажите оба следствия.

Следствие 5.5 (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 φ тогда и только тогда, когда ψ. Аналогично в обратную сторону. □

Пример 5.14. *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 — $\mathfrak{IP}$. Синтаксическими конструкциями $\mathfrak{IP}$ будут те же тройки Хоара, но доказуемость 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.

СПР Аксиома будет иметь тот же вид: $\{(\psi)^x_e\}\,x = e;\{\psi\}$, 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 $\sigma \models (\psi)^x_e$. □

СЭП Если $\{\varphi\}\,\Pi_1\,\{\psi\}$ и формула $\theta \leftrightarrow \varphi$ истинна, то $\{\theta\}\,\Pi_1\,\{\psi\}$.

Задача 5.19. Докажите, что с помощью правила **СЭП** можно получать только тройки с слабейшими предусловиями, если исходная тройка такова.

ССЛ Если выводимы $\{\varphi\}\,\Pi_1\,\{\theta\}$ и $\{\theta\}\,\Pi_2\,\{\psi\}$, то выводимо $\{\varphi\}\,\Pi_1\Pi_2\,\{\psi\}$.

Доказательство. Пусть $(\Pi_1\Pi_2)(\sigma) \models \psi$, то есть $\Pi_2(\Pi_1(\sigma)) \models \psi$. Следовательно, $\Pi_1(\sigma) \models \theta$, 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 $\{\varphi\}\,\Pi_1\Pi_2\,\{\psi\}$ следует из корректности $\{\varphi\}\,\Pi_1\,\{\theta\}$ и $\{\theta\}\,\Pi_2\,\{\psi\}$ по правилу **СЛ**. □

СНВ Пусть

$$\Pi \Leftrightarrow \mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{end};$$

Пусть φ — слабейшее предусловие для $\Pi_1\,\{\psi\}$. Тогда

$$\theta \Leftrightarrow \varphi \wedge T \vee \psi \wedge \neg T$$

слабейшее предусловие для $\Pi\,\{\psi\}$.
Доказательство. Предположим, что $\Pi\,(\sigma) \models \psi$.

а) Если $\sigma \models T$, то $\Pi\,(\sigma) = \Pi_1\,(\sigma)$. $\Pi_1\,(\sigma) \models \psi$, значит $\sigma \models \varphi$. Итак, $\sigma \models T$ и $\sigma \models \varphi$, следовательно, $\sigma \models \varphi \wedge T$ и $\sigma \models \theta$.

б) Если $\sigma \not\models T$, то $\sigma \models \neg T$ и $\Pi\,(\sigma) = \sigma$. То есть $\sigma \models \psi \wedge \neg T$ и $\sigma \models \theta$.

В любом случае, как мы видим, $\sigma \models \theta$.

Теперь докажем корректность тройки $\{\theta\}\,\Pi\,\{\psi\}$.

$$\theta \wedge T \Leftrightarrow (\varphi \wedge T \vee \psi \wedge \neg T) \wedge T \equiv \varphi \wedge T$$

$$\theta \wedge \neg T \Leftrightarrow (\varphi \wedge T \vee \psi \wedge \neg T) \wedge \neg T \equiv \psi \wedge \neg T$$

Корректность $\{\varphi \wedge T\}\,\Pi_1\,\{\psi\}$ получается из $\{\varphi\}\,\Pi_1\,\{\psi\}$ усилением предусловия. Формула $\theta \wedge \neg T \to \psi$ истинна. Следовательно, корректность тройки $\{\theta\}\,\Pi\,\{\psi\}$ получается по правилу **НВ**. □

СПВ Пусть

$$\Pi \Leftrightarrow \mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};$$

Пусть φ_1 — слабейшее предусловие для $\Pi_1\,\{\psi\}$, и φ_2 — слабейшее предусловие для $\Pi_2\,\{\psi\}$. Тогда

$$\varphi \Leftrightarrow \varphi_1 \wedge T \vee \varphi_2 \wedge \neg T$$

слабейшее предусловие для $\Pi\,\{\psi\}$.
Задача 5.20. Докажите, что φ — действительно слабейшее предусловие для $\Pi\,\{\psi\}$.

В частности, если $\varphi_1 \equiv \varphi_2$, то

$$\varphi \equiv \varphi_1 \wedge T \vee \varphi_1 \wedge \neg T \equiv \varphi_1 \wedge (T \wedge \neg T) \equiv \varphi_1 \equiv \varphi_2.$$

СЦП Пусть

$$\Pi \twoheadleftarrow \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};$$

Пусть $\varphi_0 \twoheadleftarrow \psi \wedge \neg T$, φ'_{i+1} — полное слабейшее предусловие для $\Pi_1\{\varphi_i\}$ и $\varphi_{i+1} \twoheadleftarrow \varphi'_{i+1} \wedge T$. 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 состояний $\left(\sigma^i\right)_{i=0}^{n}$ такие, что

$$\sigma^0 = \sigma, \quad \sigma^n = \tau, \quad \sigma^{i+1} = \Pi_1\left(\sigma^i\right), \quad \sigma^i \models T \iff i < n.$$

Если $n = 0$, то $\sigma = \tau$, $\sigma \models \psi \wedge \neg T$ 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 $\sigma^n = \tau$, $\tau \models \psi \wedge \neg T$, $\tau \models \varphi_0$.

Шаг индукции.
Пусть для i выполнено $\sigma^{n-i} \models \varphi_i$. Так как $\Pi_1\left(\sigma^{n-(i+1)}\right) = \sigma^{n-i}$, 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 $\sigma^{n-(i+1)} \models T$. 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

$$\{\varphi \wedge T \wedge L = z\}\,\Pi_1\,\{\varphi \wedge L < z\}$$

для некоторого L и переменной z, которая нигде больше не встречается. Возьмём $L \twoheadleftarrow \min\{i \in \omega : \varphi_i\}$. Заметим, что если $\sigma \models \varphi$, то $\sigma(L)$ определено.

Предположим, что $\sigma \models \varphi \wedge T \wedge L = z$. Пусть $i = \sigma(L)$. Следовательно, $\sigma \models \varphi_i$ и $\sigma \not\models \varphi_j$ для $j < i$. Если $i = 0$, то $\sigma \models \psi \wedge \neg T$, что противоречит тому, что $\sigma \models T$. Следовательно, $i > 0$. Тогда $\varphi_i \wedge T \equiv \varphi'_i \wedge T$. 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 $\sigma(L) < i$. Так как $z \notin \mathbf{LVar}(\Pi_1)$, то по лемме о сохранении значения

$$\sigma(L) < i = \sigma z = \Pi_1(\sigma)(z).$$

Следовательно, $\Pi_1(\sigma) \models \varphi \wedge L < z$.

По правилу **ЦП** получаем, что тройка $\{\varphi\}\,\Pi\,\{\varphi \wedge \neg T\}$ является полностью корректной. Формула $\varphi \wedge \neg T \to \psi$ 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 5.6. *Пусть* $\varphi_i \equiv \Phi(i)$ *для некоторой формулы* Φ *для всех* $i \geq 1$. *Тогда*

$$\varphi \equiv \psi \wedge \neg T \bigvee \exists i\,(i \geq 1 \wedge \Phi(i)).$$

Следствие 5.7. *Пусть* $\Phi(i) \equiv i = e \wedge \Phi'(i)$, *где выражение* e *не содержит* i. *Тогда*

$$\varphi \equiv \psi \wedge \neg T \vee \Phi'(e) \wedge e \geq 1.$$

****Задача 5.21.** 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 5.1 (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.

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 5.15. *Рассмотрим алгоритм на рис.* 5.4.

Очевидно, что постусловием должна быть формула $\psi \Leftarrow Fib = F_n$. *Для* $Fib = f;\{\psi\}$ *слабейшим предусловием очевидно будет* $f = F_n$.

```
Alg Fib;
arg n;
  f = 1; p = 0; i = 1;
  while i < n do
    i = succ(i);
    t = f + p;
    p = f;
    f = t;
  end;
  Fib = f;
end;
```

Рис. 5.4: Числа Фибоначчи.

Теперь найдём слабейшее предусловие для

$$\Pi_2 \left\{ \underbrace{f = F_n \wedge i \geq n}_{\varphi_0} \right\}.$$

f = t; $t = F_n \wedge i \geq n$.

p = f; $t = F_n \wedge i \geq n$.

t = f + p; $f + p = F_n \wedge i \geq n$.

i = succ(i); $f + p = F_n \wedge i + 1 \geq n$.

Получили, что

$$\varphi_1' \multimap f + p = F_n \wedge i + 1 \geq n,$$
$$\varphi_1 \multimap f + p = F_n \wedge i + 1 \geq n \wedge i < n \equiv$$
$$\equiv f + p = F_n \wedge i + 1 = n.$$

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*

$$\varphi_j \twoheadleftarrow F_{j+1}f + F_jp = F_n \wedge i + j = n.$$

Проверим его: найдём слабейшее предусловие для $\Pi_1\{\varphi_j\}$.

f = t; $F_{j+1}t + F_jp = F_n \wedge i + j = n.$

p = f; $F_{j+1}t + F_jf = F_n \wedge i + j = n.$

t = f + p; $F_{j+1}(f + p) + F_jf = F_n \wedge i + j = n.$ *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:

$$\varphi_j \equiv F_{j+1}f + F_jp = F_n \wedge j = n - i.$$

Таким образом, по правилу **СЦП** *(следствие 5.7) получаем, что полным слабейшим предусловием для* $\Pi_1\{\psi\}$ *будет*

$$f = F_n \wedge i \geq n \bigvee F_{n-i+1}f + F_{n-i}p = F_n \wedge n - i \geq 1.$$

Продолжим.

i = 1; $f = F_n \wedge 1 \geq n \bigvee F_nf + F_{n-1}p = F_n \wedge n - 1 \geq 1.$

p = 0; $f = F_n \wedge 1 \geq n \bigvee F_nf = F_n \wedge n \geq 2.$

f = 1; $1 = F_n \wedge 1 \geq n \bigvee F_n = F_n \wedge n \geq 2.$

Заметим, что

$$F_n = F_n \equiv \text{ИСТИНА},$$
$$1 = F_n \equiv n = 1 \vee n = 2,$$
$$1 = F_n \wedge 1 \geq n \equiv n = 1.$$

Поэтому, полным слабейшим предусловием для Π *будет*

$$n = 1 \vee n \geq 2 \equiv n \geq 1.$$

Итак, программа работает правильно, если $n \geq 1$.

Пример 5.16. *Рассмотрим программу, подсчитывающую количество простых чисел на промежутке* $[x, y)$, *на рис. 5.5. Мы считаем, что тест* $P(x)$ *истинен на* σ, *если* σx *— простое число. Пусть* $P(a, b)$ *— количество простых чисел на промежутке* $[a, b)$. *Постусловие должно выглядеть так:*

$$Primes = P(x, y).$$

```
Alg Primes;
arg x, y;
  u = x;
  v = 0;
  while u < y do
    if P(u) then
      v = v + 1;
    end;
    u = u + 1;
  end;
  Primes = v;
end;
```

Рис. 5.5: Подсчитывание простых чисел.

Primes = v; $v = P(x, y)$.

Для цикла:

$$\varphi_0 \multimap v = P(x, y) \wedge u \geq y$$

$$\Phi(i) \multimap v = P(x, u) \wedge i = y - u$$

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;

$$\Big(v = P(x, u+1) \wedge \neg P(u) \bigvee \bigvee v + 1 = P(x, u+1) \wedge P(u)\Big) \wedge i + 1 = y - u \equiv$$

$$\equiv v = P(x, u) \wedge i + 1 = y - u \multimap \Phi(i+1)$$

while . . . end; $v = P(x, y) \wedge u \geq y \vee v = P(x, u) \wedge y - u \geq 1$

v = 0; $0 = P(x, y) \wedge u \geq y \vee 0 = P(x, u) \wedge y - u \geq 1$

u = x; $0 = P(x, y) \wedge x \geq y \vee 0 = P(x, x) \wedge y - x \geq 1 \equiv$ ИСТИНА

Следовательно, слабейшим предусловием будет ИСТИНА, *и программа корректно работает всегда.*

Задача 5.22. Напишите программы и найдите для них слабейшие предусловия:

1) Вычисление факториала (используя умножение).
2) Определение простоты числа (используя умножение и целочисленное деление).
3) Нахождение количества делителей числа (используя умножение и целочисленное деление).

***Задача 5.23.** Напишите эти же программы и найдите слабейшие предусловия для них, используя только сложение и вычитание.

Ранее мы доказывали непротиворечивость исчислений $\mathfrak{IH}$ и $\mathfrak{IP}$. Непротиворечивость означала, что каждый доказуемый 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 5.2 (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 5.3 (Полнота исчисления Хоара)**.** *Пусть тройка $\{\varphi\}\,\Pi\,\{\psi\}$ вполне корректна. Тогда*

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\,\Pi\,\{\psi\}\,.$$

Доказательство. Докажем полноту исчисления Хоара индукцией по количеству операторов цикла в программе.

Базис индукции.
Если их нет, то по $\Pi\{\psi\}$ исчисление предусловий позволяет построить слабейшее предусловие φ', причём тройка $\{\varphi'\}\,\Pi\,\{\psi\}$ доказуема в $\mathfrak{IH}$. Докажем истинность формулы $\varphi \to \varphi'$, после чего доказуемость тройки $\{\varphi\}\,\Pi\,\{\psi\}$ будет следовать по правилу **УПР**.

Рассмотрим любое состояние $\sigma \models \varphi$. 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 $\sigma \models \varphi'$.

Шаг индукции.
Пусть для программ, которые содержат меньше чем n циклов, теорема доказана. Достаточно доказать теорему только для оператора цикла. Рассмотрим любую вполне корректную тройку

$$\{\varphi\}\underbrace{\mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};}_{\Pi}\{\psi\}\,,$$

где Π_1 содержит меньше n циклов. Исчисление предусловий позволяет построить формулу φ' — слабейшее предусловие для $\Pi\{\psi\}$. Тройка Хоара

$$\{\varphi' \wedge T \wedge L = z\}\,\Pi_1\,\{\varphi' \wedge L < z\}$$

(см. доказательство непротиворечивости **СЦП**) является вполне корректной. По индукционному предположению, она доказуема в $\mathfrak{IH}$. По правилу **ЦП** получаем доказуемость тройки

$$\{\varphi'\}\,\Pi\,\{\varphi' \wedge \neg T\}\,.$$

Формула $\varphi' \wedge \neg T \to \psi$ является истинной, следовательно, тройка $\{\varphi'\}\,\Pi\,\{\psi\}$ доказуема с помощью **ОП**. Точно также как и для базиса доказывается, что формула $\varphi \to \varphi'$ истинна, следовательно,

$$\underset{\mathfrak{IH}}{\rhd}\{\varphi\}\,\Pi\,\{\psi\}\,. \qquad \square$$

§ 5.4. **Существование слабейших предусловий

Прежде чем доказывать теорему о существовании полного слабейшего предусловия установим несколько лемм.

Лемма 5.4 (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*

$$(\psi)^{\overline{x}}_{\overline{e}} \wedge \theta',$$

где θ *и* θ' *— бескванторные формулы.*

Доказательство. Индукция по сложности программы. □

Задача 5.24. Докажите лемму.

Следующие две теоремы доказываются в теории алгоритмов.

Теорема 5.4 (О представлении функций)**.** *Для каждой рекурсивной функции*[4] f *существует формула арифметики* Ψ_f*, которая представляет* f*. То есть* $\Psi_f(\overline{x}, y)$ *истинна тогда и только тогда, когда* $f(\overline{x}) = y$*.*

Поскольку все программы работают с числами, то необходим некоторый способ представления логических формул и состояний натуральными числами. Пусть $[a]$ — номер a.

Теорема 5.5 (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 5.5 (О представлении предусловий)**.** *Для всякой формулы* ψ *существует арифметическая формула* Ψ_ψ*, представляющая истинность всех формул вида*

$$\varphi \leftrightarrows \bigvee_{j<m} (\psi)^{\overline{x}}_{\overline{e}_j} \wedge \theta_j,$$

на любом состоянии σ*, где* $m \in \omega$*,* $\overline{e}_j$ *— наборы выражений* θ_j *— бескванторные формулы.*

Доказательство. Пусть функции g и h таковы, что для любой формулы φ вышеуказанного вида

$$g([\varphi], j) = \left[(\psi)^{\overline{x}}_{\overline{e}_j}\right]$$

$$h([\varphi], j) = [\theta_j].$$

Если $j \geq m$, то

$$g([\varphi], j) = [0 = 1]$$

[4] Функция называется рекурсивной, если существует программа, которая её вычисляет.

$$h\left([\varphi], j\right) = [0 = 1].$$

То есть функции g и h разлагают формулу φ на составные части. Обе эти функции рекурсивны, по теореме обе эти функции представляются арифметическими формулами Ψ_g и Ψ_h соответственно. Очевидно, что формулы $(\psi)^{\overline{x}}_{\overline{e}_j}$ содержит столько же кванторов, сколько и ψ. Пусть это количество равно n. Тогда истинность всех формул $(\psi)^{\overline{x}}_{\overline{e}_j}$ и θ_j представляется арифметической формулой Ψ_n. Пусть

$$\Psi_\psi(y, x) \leftrightarrow$$
$$\leftrightarrow \exists j \exists u_\psi \exists u_\theta \left(\Psi_g(y, j, u_\psi) \wedge \Psi_h(y, j, u_\theta) \wedge \Psi_n(u_\theta, x) \wedge \Psi_n(u_\theta, x)\right).$$

Предположим, что $\sigma \models \varphi$. Тогда существует $j^0 < m$ такое, что $\sigma \models (\psi)^{\overline{x}}_{\overline{e}_{j^0}}$ и $\sigma \models \theta_{j^0}$. Пусть

$$u^0_\psi = \left[(\psi)^{\overline{x}}_{\overline{e}_{j^0}}\right]; \quad u^0_\theta = \left[\theta_{j^0}\right].$$

Тогда истинны формулы

$$\Psi_g\left([\varphi], j^0, u^0_\psi\right),\ \Psi_h\left([\varphi], j^0, u^0\theta\right),\ \Psi_n\left(u^0_\psi, [\sigma]\right),\ \Psi_n\left(u^0_\theta, [\sigma]\right).$$

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

$$\Psi_g\left([\varphi], j^0, u^0_\psi\right),\ \Psi_h\left([\varphi], j^0, u^0\theta\right),\ \Psi_n\left(u^0_\psi, [\sigma]\right),\ \Psi_n\left(u^0_\theta, [\sigma]\right).$$

Если $j^0 \geq m$, то по определению функции g имеем $g([\varphi], j) = [0 = 1]$. Так как Ψ_g представляет g и $\Psi_g\left([\varphi], j, u^0_\psi\right)$ 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

$$u^0_\psi = \left[(\psi)^{\overline{x}}_{\overline{e}_{j^0}}\right]; \quad u^0_\theta = \left[\theta_{j^0}\right].$$

Из истинности $\Psi_n\left(u^0_\psi, [\sigma]\right)$ и $\Psi_n\left(u^0_\theta, [\sigma]\right)$ следует, что $\sigma \models (\psi)^{\overline{x}}_{\overline{e}_{j^0}}$ и $\sigma \models \theta_{j^0}$. Следовательно, $\sigma \models (\psi)^{\overline{x}}_{\overline{e}_{j^0}} \wedge \theta_{j^0}$ и $\sigma \models \varphi$. □

Лемма 5.6 (О представлении предусловий)**.** *Пусть множество формул* $\{\varphi_i : i \in \omega\}$ *рекурсивно перечислимо по* i *и каждое* φ_i *является дизъюнкцией формул вида* $(\psi)_{\overline{e}}^{\overline{x}} \wedge \theta$*, 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 $\{\varphi_i : i \in \omega\}$ рекурсивно перечислимо, то существует рекурсивная 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 5.5, существует арифметическая формула Ψ_ψ, представляющая истинность всех формул φ_i. Пусть

$$\Phi \leftrightarrows \exists u_\varphi \left(\Psi_f(i, u_\varphi) \wedge \Psi_\psi(u_\varphi, x)\right).$$

Пусть $\sigma \models \varphi_i$. Возьмём $u_\varphi^0 = [\varphi_i]$. Тогда истинны формулы $\Psi_f\left(i, u_\varphi^0\right)$ и $\Psi_\psi\left(u_\varphi^0, [\sigma]\right)$. Следовательно, истинна и $\Phi(i, [\sigma])$.

Пусть истинна $\Phi(i, [\sigma])$. Это означает, что для некоторого u_φ^0 истинны и $\Psi_f\left(i, u_\varphi^0\right)$, и $\Psi_\psi\left(u_\varphi^0, [\sigma]\right)$. Из первого следует, что $u_\varphi^0 = [\varphi_i]$. Из второго — $\sigma \models \varphi_i$. □

Лемма 5.7 (О существовании предусловий)**.** *Пусть*

$$\Pi \leftrightarrows \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};$$

и Π_1 *не содержит циклов. Тогда для всякого постусловия* ψ *существует слабейшее полное предусловие* φ *для* $\Pi\{\psi\}$*.*

Доказательство. По лемме 5.4 полное слабейшее предусловие для $\Pi\{\psi\}$ может быть выражено бесконечной дизъюнкцией

$$\varphi \equiv \bigvee \{\varphi_i : i \in \omega\},$$

которая удовлетворяет условиям леммы 5.6. Это означает, что существует формула Φ такая, что $\Phi(i, [\sigma])$ истинна тогда и только тогда, когда $\sigma \models \varphi_i$.

Будем рассматривать состояния, определённые на множестве $\mathbf{Var}(\Pi)$. Пусть переменные из $\mathbf{Var}(\Pi)$ образуют набор $\overline{x}$ из n переменных. Пусть функция f по $\overline{v} \in \omega^n$ даёт номер состояния σ такого, что $x_i = v_i$. Тогда эта функция представляется формулой Ψ_f. Возьмём формулу

$$\Theta(i, \overline{x}) \leftrightarrows \exists u_\sigma \left(\Phi(i, u_\sigma) \wedge \Psi_f(\overline{x}, u_\sigma)\right).$$

Тогда

$$\bigvee \{\varphi_i : i \in \omega\} \equiv \exists i \Theta.$$

Следовательно, слабейшее предусловие для $\Pi\{\psi\}$ выражается формулой $\exists i \Theta$. □

****Задача 5.25.** Докажите, что

$$\bigvee \{\varphi_i : i \in \omega\} \equiv \exists i \Theta.$$

Теперь мы в состоянии 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

$$\Pi_2(\sigma) \restriction \mathbf{Var}(\Pi) = \Pi(\sigma) \restriction \mathbf{Var}(\Pi).$$

Кроме того, можно считать, что Π_2 имеет вид

$$\begin{array}{l} y \;=\; \alpha; \\ w \;=\; 0; \\ \left.\begin{array}{l} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{array}\right\} \Pi_3 \end{array}$$

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 $\Pi\{\psi\}$. □

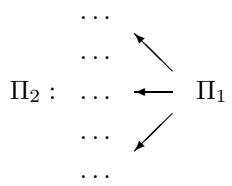
Ещё раз подчеркнём, что в доказательстве существенно используются свойства арифметики натуральных чисел. Если речь идёт о другой предметной области, то утверждение о существовании слабейших предусловий может быть неверным.

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. Это нерационально, так как от повторения написанного информации больше не становиться. 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. Описание подпрограмм следующее:



Определение 6.1 (Подпрограмма)**.** *Синтаксис подпрограмм:*

Sub *Имя подпрограммы*;
arg *Список переменных*;
Тело подпрограммы
end;

Переменные, которые стоят после слова arg *— это аргументы подпрограммы (или параметры подпрограммы). Тело подпрограммы (его мы будем обозначать* Π_A*, если* A *— имя подпрограммы) такое же как и тело алгоритмов.*

Пример 6.1. *Рассмотрим две подпрограммы:*

Sub A;
arg x;
  $A \;=\; \mathsf{succ}\,(\mathsf{succ}\,(x))\,;$
end;

Sub Max;
arg $x,\ y$;
1 if $x \;<\; y$ then 2 else 3
2 $Max \;=\; y;$ 4
3 $Max \;=\; x;$ 4
end;

В частном случае подпрограмма может не иметь аргументов, то есть список переменных после arg может быть пустым.

Вызов подпрограммы f, как и вызов алгоритма, — это

$$\delta = (f, \overline{v})\,,$$

где $\overline{v}$ — набор элементов предметной области длины равной количеству аргументов f. Точно так же определяется и результат вызова **Res** δ. Как и алгоритм, каждая подпрограмма A вычисляет некоторую частичную функцию Φ_A, определяемую так же, как для алгоритмов.

Чтобы использовать уже определённые подпрограммы расширим понятие выражения.

Определение 6.2 (Выражение)**.** *Кроме уже известных нам выражений мы будем считать выражением также следующую запись:*

$$f\,(e_1, \ldots, e_n)\,,$$

где f *— имя подпрограммы с* n *аргументами, а* $e_1, \ldots, e_n$ *— выражения. В частности, если* f *— подпрограмма без аргументов, то выражением будет запись* $f\,()$.

Пример 6.2. *Примеры выражений, в которых используются имена подпрограмм из примера 6.1:*

$$A\,(0)\,;$$

$$Max\,(x, \mathsf{succ}\,(y))\,;$$
$$Max\,(A\,(x)\,, \mathsf{succ}\,(0))\,.$$

Определим теперь, что такое значение выражения e на состоянии σ. Очевидно, что к уже существующему определению нужно добавить определение того, что такое $\sigma\,(f\,(e_1, \ldots, e_n))$, где f — имя подпрограммы.

Определение 6.3 (Значение выражения)**.** *Итак,* $\sigma\,(f\,(e_1, \ldots, e_n)) = v$, *если*

1) $\sigma\,(e_1)$, $\ldots$, $\sigma\,(e_n)$ *определены;*
2) $v = \mathbf{Res}\,(f, \sigma\,(e_1)\,, \ldots, \sigma\,(e_n))$.

Замечание 6.1. Пусть $\delta = (f, \sigma\,(e_1)\,, \ldots, \sigma\,(e_n))$. Чтобы определить $\sigma\,(f\,(\overline{e}))$ нужно определить $\Pi_f\,(\sigma_\delta)$. Однако, для определения $\Pi_f\,(\sigma_\delta)$ может снова потребоваться определить $\tau\left(f\left(\overline{d}\right)\right)$ и так далее. Если подобная цепочка оказывается бесконечной, то мы считаем, что $\sigma\,(f\,(\overline{e}))$ не определено.

Пример 6.3. *Рассмотрим выражения из предыдущего примера на состоянии* $\sigma = \{(x, 1)\,, (y, 2)\}$.

1) $A\,(0)$. *Очевидно, что* A *вычисляет функцию* $\Phi_A\,(x) = x+2$. *Следовательно,* $\sigma\,(A\,(0)) = 2$.
2) $Max\,(x, \mathsf{succ}\,(y))$. *Подпрограмма* Max *вычисляет функцию-максимум из двух аргументов. Поэтому*
$$\sigma\,(Max\,(x, \mathsf{succ}\,(y))) = \max\{1, 3\} = 3.$$
3) $\sigma\,(Max\,(A\,(x)\,, \mathsf{succ}\,(0))) = \max\{3, 1\} = 3$.

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* $x \geq y$ *и в этом случае* $\Pi_B\,(\tau)\,(B) = \tau x$. *Рассмотрим выражение* $B\,(y, \mathsf{succ}\,(a))$ *на том же состоянии* σ. $\mathbf{Res}\,(B, 1, 3)$ *не определён, следовательно,* $\sigma\,(B\,(y, \mathsf{succ}\,(a)))$ *тоже не определено.*

Пример 6.5. *Проиллюстрируем замечание 6.1.*

```
Sub C;
arg x;
  C = C (succ (x));
end;
```

Чтобы определить $\sigma\,(C\,(e))$ *нужно определить* $\mathbf{Res}\,\delta$, *где* $\delta = (C, \sigma\,(e))$. *Чтобы определить* $\mathbf{Res}\,\delta$, *нужно определить* $\Pi_C\,(\sigma_\delta)$, *а для этого снова нужно определять* $\sigma_\delta\,(C\,(\mathsf{succ}\,(x)))$ *и так далее. Следовательно, согласно замечанию, мы считаем, что* $\sigma\,(C\,(e))$ *не определено ни для каких* e *и* σ.

```
Sub B;
arg x, y;
  while x < y do
    y = succ(y);
  end;
  B = x;
end;
```

Рис. 6.1:

```
Sub D;
arg x;
  if x > 0 then
    D = D(x - 1) + 2;
  else
    D = 0;
  end;
end;
```

Рис. 6.2:

Пример 6.6. *Приведём ещё один пример (рис. 6.2 на противоположной странице). Здесь мы имеем схожую ситуацию — для определения* $\sigma(D(e))$ *нужно определить* $\sigma_\delta(D(x-1))$*, где* $\sigma_\delta = \{(x, \sigma(e)), (D, 0)\}$*. Однако, в этом случае последовательность не оказывается бесконечной. Индукцией по* σx *докажем, что* $\sigma(D(x)) = 2\sigma x$*.*

Базис индукции.

Если $\sigma x = 0$*, то* $\sigma_\delta x = 0$*, следовательно,*

$$\Pi_D(\sigma_\delta) = \{(x, \sigma_\delta x), (D, 0)\}.$$

Поэтому $\sigma(D(x)) = 0$*.*

Шаг индукции.

Пусть для $\sigma x < x_0$ *доказано и* $\sigma x = x_0$*. Тогда*

$$\Pi_D(\sigma_\delta) = \{(x, \sigma_\delta x), (D, \sigma_\delta(D(x-1)+2))\}.$$

```
Sub Add;
arg x, y;
  if x = 0 then
    Add = y;
  else
    Add = succ(Add(x − 1, y));
  end;
end;
```

Рис. 6.3: Сложение чисел.

По индукционному предположению, так как $\sigma_\delta(x-1) = x_0 - 1 < x_0$ *имеем*

$$\sigma_\delta(D(x-1)) = 2\sigma_\delta(x-1) = 2x_0 - 2.$$

Поэтому

$$\sigma(D(x)) = 2x_0 - 2 + 2 = 2x_0 = 2\sigma x.$$

***Задача 6.1.** Докажите, что подпрограмма на рис. 6.3 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 любого состояния. 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 6.4 (Семантика программы)**.**

1) *Тест* $T \multimap e_1 \circ e_2$ *определен на* σ *тогда и только тогда, когда* $\sigma(e_1)$ *и* $\sigma(e_2)$ *определены.*

2) $\Pi \multimap x = e$; *состояние* $\Pi(\sigma)$ *определено тогда и только тогда, когда* $\sigma(e)$ *определено.*

3) $\Pi \multimap$ if T then Π_1 end; *если* $\sigma(T)$ *не определено, то* $\Pi(\sigma)$ *тоже не определено.*

4) $\Pi \twoheadleftarrow \mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};$ *если* $\sigma(T)$ *не определено, то* $\Pi(\sigma)$ *тоже не определено.*

5) $\Pi \twoheadleftarrow \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};$ *состояние* $\Pi(\sigma)$ *определено, если существует последовательность состояний* $\sigma^0, \ldots, \sigma^n$, $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1(\sigma^i)$, $\sigma^i(T)$ *определено для* $i = 0, \ldots, n$ *и* $\sigma^i \models T$ *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* Π. *Пусть* $(\sigma^i, \alpha_i)_i$ *— последовательность расширенных состояний* Π. *Если для определения* σ^{i+1} *требуется определение значения выражения* e *на состоянии* 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. Если*

$$\Pi \twoheadleftarrow y = B(y, \mathsf{succ}(y));$$

то $\Pi(\sigma)$ *не определено.*

Рассмотрим логически законченный пример алгоритма с подпрограммами.

Пример 6.8. *Запишем алгоритм Евклида с использованием подпрограмм, см. рис. 6.4.*

Как видим, по сравнению с алгоритмом на рис. 3.4 текст программы стал намного легче для восприятия и короче за счёт того, что фрагменты, которые выполняли сравнение двух чисел — (2) на рис. 3.4 — и вычитание — (3) на рис. 3.4 — теперь вынесены в отдельные подпрограммы, то есть вместо двух раз они написаны по одному.

Задача 6.2. Напишите алгоритмы вычисления факториала и проверки числа на простоту, используя подпрограммы сложения, умножения и т.д.

Определение 6.5 (Основная программа)**.** *Поскольку подпрограммы — неотъемлемая часть алгоритма, то* а л г о р и т м о м *мы впредь будем называть всю совокупность подпрограмм и* о с н о в н о й п р о г р а м м ы *— программы, написанной после* Alg. *Таким образом, если подпрограмм нет, то понятия алгоритма и основной программы совпадают.*

```
Sub Neq;
arg x, y;
  if x = y then
    Neq = 0;
  else
    Neq = 1;
  end;
end;
```

```
Sub S;
arg x, y;
  u = 0; v = y;
  while v < x do
    u = succ(u);
    v = succ(v);
  end;
  S = u;
end;
```

```
Alg Euclid;
arg x, y;
  if x = 0 then z = y;
  else if y = 0 then z = x;
  else
    while Neq(x, y) = 1 do
        if x < y then
            y = S(y, x);
        else
            x = S(x, y);
        end;
    end;
    z = x;
  end;
  end;
  Euclid = z;
end;
```

Рис. 6.4: Алгоритм Евклида.

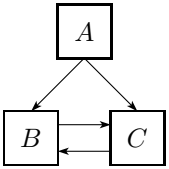


Рис. 6.5: Граф зависимости для примера 6.9

§ 6.2. Графы зависимости, списки и деревья вызовов

Рассмотрим произвольную совокупность подпрограмм A.

Определение 6.6 (Граф зависимости)**.** *Графом зависимости для множества подпрограмм A называется граф, множество вершин которого — A, а множество рёбер:*

$$\{(f, g) : \text{в } f \text{ встречаются выражения вида } g(\cdots)\}.$$

Пример 6.9. *Рассмотрим следующие подпрограммы:*

```
Sub A;          Sub B;          Sub C;
arg x;          arg x;          arg x;
  A = B(C(x));    B = C(x);       C = B(x);
end;            end;            end;
```

Граф зависимости для данного множества подпрограмм изображён на рисунке 6.5.

Задача 6.3. Постройте графы зависимости для подпрограмм из алгоритмов из задачи 6.2.

Определение 6.7 (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 6.10. *В предыдущем примере имеется рекурсия. B и C — рекурсивные подпрограммы. Рекурсия является косвенной.*

Определение 6.8 (Список вложенных вызовов)**.** *Список вложенных вызовов для выражения e и состояния σ ($\mathbf{Fc}(\sigma, e)$) определяется по индукции:*

1) *$e \Rightarrow x$ или $e \Rightarrow c$, x — переменная, c — константа. Тогда $\mathbf{Fc}(\sigma, e)$ — пуст:* $\mathbf{Fc}(\sigma, e) = ()$.

2) *$e \Rightarrow o(e_1, \ldots, e_n)$, где o — операция. Тогда*

$$\mathbf{Fc}(\sigma, e) = \mathbf{Fc}(\sigma, e_1), \ldots, \mathbf{Fc}(\sigma, e_n).$$

Если $\sigma(e_i)$ неопределено, то часть, следующая за $\mathbf{Fc}(\sigma, e_i)$ отбрасывается:

$$\mathbf{Fc}(\sigma, e) = \mathbf{Fc}(\sigma, e_1), \ldots, \mathbf{Fc}(\sigma, e_i).$$

3) *$e \Rightarrow f(e_1, \ldots, e_n)$, где f — подпрограмма. Тогда*

$$\mathbf{Fc}(\sigma, e) = \mathbf{Fc}(\sigma, e_1), \ldots, \mathbf{Fc}(\sigma, e_n), (f, \sigma(e_1), \ldots, \sigma(e_n)).$$

Если $\sigma(e_i)$ не определено, то часть, следующая за $\mathbf{Fc}(\sigma, e_i)$ считается пустой:

$$\mathbf{Fc}(\sigma, e) = \mathbf{Fc}(\sigma, e_1), \ldots, \mathbf{Fc}(\sigma, e_i).$$

Рассмотрим пример.

Пример 6.11. *Пусть f и g — подпрограммы с одним и двумя аргументами соответственно. Пусть*

$$e \Rightarrow g(f(x), g(x, y))$$

Рассмотрим состояние $\sigma = \{(x, 1), (y, 2)\}$. Предположим, что $\mathbf{Res}(f, 1) = 3$ и $\mathbf{Res}(g, 1, 2) = 5$. Тогда

$$\begin{aligned}
\mathbf{Fc}(\sigma, x) &= () \\
\mathbf{Fc}(\sigma, f(x)) &= ((f, 1)) \\
\mathbf{Fc}(\sigma, y) &= () \\
\mathbf{Fc}(\sigma, g(x, y)) &= ((g, 1, 2)) \\
\mathbf{Fc}(\sigma, e) &= ((f, 1), (g, 1, 2), (g, 3, 5))
\end{aligned}$$

Определение 6.9 (Список вложенных вызовов)**.** *Для теста $T \Rightarrow e_1 \circ e_2$ список $\mathbf{Fc}(\sigma, T)$ определяется аналогично двухместной операции:*

$$\mathbf{Fc}(\sigma, T) = \mathbf{Fc}(\sigma, e_1), \mathbf{Fc}(\sigma, e_2).$$

Если $\sigma(e_1)$ не определено, то

$$\mathbf{Fc}(\sigma, T) = \mathbf{Fc}(\sigma, e_1).$$

Определение 6.10 (Список вложенных вызовов)**.** *Для программы* Π *список вложенных вызовов* $\mathbf{Fc}(\sigma, \Pi)$ *строится индукцией по построению программы* Π*:*

1) $\Pi \doteq x = e$; *в этом случае* $\mathbf{Fc}(\sigma, \Pi) = \mathbf{Fc}(\sigma, e)$.

2) $\Pi \doteq \Pi_1 \Pi_2$. *Тогда*

$$\mathbf{Fc}(\sigma, \Pi) = \mathbf{Fc}(\sigma, \Pi_1), \mathbf{Fc}(\sigma, \Pi_2).$$

Если $\Pi_1(\sigma)$ *неопределено, то*

$$\mathbf{Fc}(\sigma, \Pi) = \mathbf{Fc}(\sigma, \Pi_1).$$

3) *Неполное ветвление:*

$$\Pi \doteq \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

в этом случае

$$\mathbf{Fc}(\sigma, \Pi) = \begin{cases} \mathbf{Fc}(\sigma, T), \mathbf{Fc}(\sigma, \Pi_1), & \text{если } \sigma \models T; \\ \mathbf{Fc}(\sigma, T), & \text{иначе.} \end{cases}$$

Как и раньше, если $\sigma(T)$ *не определено, то* $\mathbf{Fc}(\sigma, \Pi_1)$ *из определения исчезает:*

$$\mathbf{Fc}(\sigma, \Pi) = \mathbf{Fc}(\sigma, T).$$

4) *Полное ветвление:*

$$\Pi \doteq \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end}; \end{cases}$$

в этом случае

$$\mathbf{Fc}(\sigma, \Pi) = \begin{cases} \mathbf{Fc}(\sigma, T), \mathbf{Fc}(\sigma, \Pi_1), \text{если } \sigma \models T; \\ \mathbf{Fc}(\sigma, T), \mathbf{Fc}(\sigma, \Pi_2), \text{иначе.} \end{cases}$$

Если $\sigma(T)$ *не определено, то*

$$\mathbf{Fc}(\sigma, \Pi) = \mathbf{Fc}(\sigma, T).$$

5) *Цикл:*

$$\Pi \backsimeq \begin{cases} \text{while } T \text{ do} \\ \quad \Pi_1 \\ \text{end;} \end{cases}$$

Пусть $\left(\sigma^i\right)_i$ *— последовательность состояний для цикла:* $\sigma^0 = \sigma$, $\sigma^{i+1} = \Pi_1\left(\sigma^i\right)$. *Тогда*

$$\mathbf{Fc}\left(\sigma, \Pi\right) = \mathbf{Fc}\left(\sigma^0, T\right), \mathbf{Fc}\left(\sigma^0, \Pi_1\right), \mathbf{Fc}\left(\sigma^1, T\right), \mathbf{Fc}\left(\sigma^1, \Pi_1\right), \ldots$$

Заметим, что в этом случае последовательность вызовов $\mathbf{Fc}\left(\sigma, \Pi\right)$ *может быть бесконечной, если программа зацикливается. Если последовательность* $\left(\sigma^i\right)_i$ *конечна, то количество частей вида* $\mathbf{Fc}\left(\sigma, \cdots\right)$ *тоже конечно. Оно может быть конечным в двух случаях:*

- $\sigma^n \not\models T$ *для некоторого* n. *В этом случае последней частью* $\mathbf{Fc}\left(\sigma, \Pi\right)$ *будет* $\mathbf{Fc}\left(\sigma^n, T\right)$.
- $\Pi_1\left(\sigma^n\right)$ *не определено для некоторого* n. *Тогда последняя часть* $\mathbf{Fc}\left(\sigma, \Pi\right)$ — $\mathbf{Fc}\left(\sigma^n, \Pi_1\right)$.

Определение 6.11 (Список вложенных вызовов)**.** *Если*

$$\delta = \left(f, v_1, \ldots, v_n\right)$$

является вызовом, то с п и с к о м в л о ж е н н ы х в ы з о в о в *для* δ *будет* $\mathbf{Fc}\left(\sigma_\delta, \Pi_f\right)$*:*

$$\mathbf{Fc}\left(\delta\right) = \mathbf{Fc}\left(\sigma_\delta, \Pi_f\right).$$

Определение 6.12 (Множество вложенных вызовов)**.** *Если* $\mathbf{Fc}\left(\ldots\right)$ — *список вложенных вызовов, то с помощью* $\mathbf{Fs}\left(\ldots\right)$ *мы будем обозначать множество, которое образуют элементы этого списка.*

***Задача 6.4.** Дайте определение **Fs** по индукции (аналогично **Fc**).

Задача 6.5. Постройте $\mathbf{Fc}\left(\mathit{Euclid}, 6, 4\right)$ и $\mathbf{Fs}\left(\mathit{Euclid}, 6, 4\right)$, см. рис. 6.4.

Задача 6.6. Постройте примеры списков и множеств вложенных вызовов, используя алгоритмы из задачи 6.2.

Лемма 6.1 (О вложенных вызовах)**.** *Если* e, d — *выражения,* x — *переменная,* $x \in \mathbf{Var}\left(e\right)$, σ — *состояние,* $\tau = \left(x = d;\right)\left(\sigma\right)$ *и* $\sigma\left(\left(e\right)_d^x\right)$ *определено, то*

$$\mathbf{Fs}\left(\sigma, \left(e\right)_d^x\right) = \mathbf{Fs}\left(\sigma, d\right) \cup \mathbf{Fs}\left(\tau, e\right).$$

Доказательство. Индукция по сложности выражения e. Очевидно, что τ отличается от σ только тем, что $\tau x = \sigma(d)$.

Базис индукции.

Пусть $e \circ\!\!- x$ (в остальных случаях не выполнено условие $x \in \mathbf{Var}(e)$). Тогда $\mathbf{Fs}(\tau, e) = \varnothing$ и $(e)_d^x \circ\!\!- d$, следовательно,

$$\mathbf{Fs}(\sigma, (e)_d^x) = \mathbf{Fs}(\sigma, d) \cup \varnothing.$$

Шаг индукции.

Допустим, что $e \circ\!\!- o(e_1, \ldots, e_n)$, где o — операция. Тогда

$$\mathbf{Fs}(\tau, e) = \mathbf{Fs}(\tau, e_1) \cup \cdots \cup \mathbf{Fs}(\tau, e_n),$$
$$\mathbf{Fs}(\sigma, (e)_d^x) = \mathbf{Fs}(\sigma, (e_1)_d^x) \cup \cdots \cup \mathbf{Fs}(\sigma, (e_n)_d^x).$$

По индукционному предположению

$$\mathbf{Fs}(\sigma, (e_i)_d^x) = \mathbf{Fs}(\tau, e_i) \cup \mathbf{Fs}(\sigma, d).$$

Следовательно,

$$\mathbf{Fs}(\sigma, (e)_d^x) = (\mathbf{Fs}(\tau, e_1) \cup \mathbf{Fs}(\sigma, d)) \cup \cdots \cup (\mathbf{Fs}(\tau, e_n) \cup \mathbf{Fs}(\sigma, d)) =$$
$$= (\mathbf{Fs}(\tau, e_1) \cup \cdots \cup \mathbf{Fs}(\tau, e_n)) \cup \mathbf{Fs}(\sigma, d) = \mathbf{Fs}(\tau, e) \cup \mathbf{Fs}(\sigma, d).$$

Случай, когда $e \circ\!\!- f(e_1, \ldots, e_n)$, где f — подпрограмма, рассматривается аналогично. □

Задача 6.7. Обоснуйте индукционный шаг для подпрограмм.

Следствие 6.1. *Если d — выражение, T — тест, x — переменная, $x \in \mathbf{Var}(T)$, σ — состояние, $\tau = (x = d;)(\sigma)$, то*

$$\mathbf{Fs}(\sigma, (T)_d^x) = \mathbf{Fs}(\sigma, d) \cup \mathbf{Fs}(\tau, T).$$

Задача 6.8. Докажите следствие.

Определение 6.13 (Дерево вложенных вызовов)**.** *Если*

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

вызов, то дерево вложенных вызовов $\mathbf{Ft}(\delta)$ — это дерево с корнем δ, сыновьями каждой вершины γ которого являются вызовы $\mathbf{Fc}(\gamma)$, идущие в том же порядке слева направо.

Следствие 6.2. *В дереве $\mathbf{Ft}(\delta)$ каждое поддерево с корнем γ является деревом $\mathbf{Ft}(\gamma)$.*

*__Задача 6.9.__ Докажите следствие.

Рассмотрим пример.

Пример 6.12. *Рассмотрим следующие подпрограммы:*

```
Sub f;
arg x;
  if x < 2 then
    f = x + 1;
  else
    f = g(f(x − 1), x);
  end;
end;
```

```
Sub g;
arg x, y;
  if x < y then
    g = g(y − x, f(x));
  else
    g = f(x − y);
  end;
end;
```

```
Sub h;
arg x;
  while 1 < x do
    x = g(2, x);
  end;
  h = x;
end;
```

Рассмотрим вызов $\delta = (h, 4)$. *Дерево вложенных вызовов* $\mathbf{Ft}\,(\delta)$ *изображено на рисунке* 6.6.

Задача 6.10. Постройте $\mathbf{Ft}\,(Euclid, 5, 3)$, см. рис. 6.4.

Задача 6.11. Постройте примеры деревьев вложенных вызовов, используя алгоритмы из задачи 6.2.

Определение 6.14 (Лес вложенных вызовов)**.** *Рассмотрим произвольный объект* a *— выражение, тест или программу. Пусть* σ *— состояние. Построим* $\mathbf{Fs}\,(\sigma, a)$ *и для каждого элемента* $\delta \in \mathbf{Fs}\,(\sigma, a)$ *— дерево вызовов* $\mathbf{Ft}\,\delta$. *Множество этих деревьев мы будем называть л е с о м в ы з о в о в* $\mathbf{Ff}\,(\sigma, a)$.

Пример 6.13. *Рассмотрим подпрограммы из примера* 6.12. *Лес вызовов* $\mathbf{Ff}\,(\sigma_\delta, \Pi_h)$, *будет состоять из двух деревьев — поддеревьев* $\mathbf{Ft}\,\delta$, *см. рис.* 6.6, *корнями которых будут* $(g, 2, 4)$ *и* $(g, 2, 2)$ *(вторая строка).*

Определение 6.15 (Рекурсивный вызов)**.** *Пусть* $\mathbf{Ft}\,\delta$ *— дерево вложенных вызовов для* $\delta = (f, \overline{v})$. *Вершина* ε, *отличная от корня и имеющая вид* $\varepsilon = (f, \overline{v}_1)$, *называется р е к у р с и в н ы м в ы з о в о м для* δ.

Пример 6.14. *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 $\delta = (f, \overline{v})$ есть рекурсивные вызовы, то f — рекурсивная подпрограмма.

Обратное неверно.

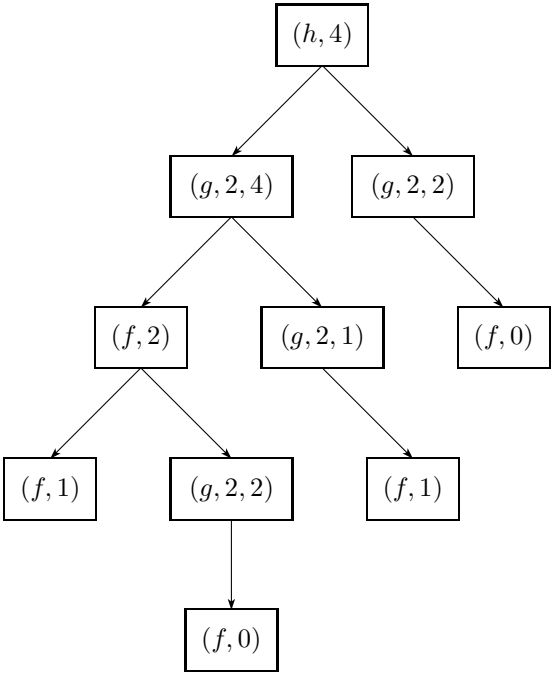


Рис. 6.6: Дерево вызовов программы из примера 6.12.

Пример 6.15. *Рассмотрим две подпрограммы:*

```
Sub f;                    Sub g;
arg x, y;                 arg x, y;
  if x < y then             if x < y then
    f = 2;                    g = f(x, y);
  else                      else
    f = g(x, y);              g = 2;
  end;                      end;
end;                      end;
```

Очевидно, что обе подпрограммы рекурсивны (косвенная рекурсия). Однако, рекурсивные вызовы ни для какой из них невозможны.

Задача 6.12. Докажите, что в предыдущем примере рекурсивные вызовы невозможны.

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

Лемма 6.2 (О конечных деревьях)**.** *Дерево конечно тогда и только тогда, когда количество сыновей каждой вершины конечно и каждая ветвь дерева конечна.*

Докажем следующее свойство дерева вызовов:

Лемма 6.3 (О бесконечных деревьях вызовов)**.** *Если дерево* $\mathbf{Ft}\,\delta$ *бесконечно, то результат* $\mathbf{Res}\,\delta$ *не определён.*

Доказательство. Предположим, что $\delta = (f, \overline{v})$. Прежде всего нужно доказать, для того чтобы $\Pi(\sigma_\delta)$ было определено необходимо чтобы

1) список $\mathbf{Fc}(\sigma_\delta, \Pi_f)$ был конечным;

2) для всех вызовов γ из списка $\mathbf{Fc}(\sigma, \Pi)$ было определено $\mathbf{Res}\,\gamma$.

Предположим, что дерево $\mathbf{Ft}\,\delta$ бесконечно, но $\mathbf{Res}\,\delta$ определено (следовательно, определено $\Pi_f(\sigma_\delta)$). Индукцией по глубине вершин дерева $\mathbf{Ft}\,\delta$ легко доказывается, что условия (1) и (2) выполняются для всех вершин дерева. Значит, каждая вершина имеет конечно много сыновей. По лемме о конечных деревьях 6.2 в дереве $\mathbf{Ft}\,\delta$ должна существовать бесконечная ветвь. Так как подпрограмм конечно много, то вызовы некоторой подпрограммы будут на этой ветви повторяться бесконечно часто. Согласно замечанию 6.1 мы должны считать, $\mathbf{Res}\,\delta$ неопределён, что противоречит предположению. □

***Задача 6.13.** Восполните все пропущенные места в доказательстве леммы.

Обратное, очевидно, неверно:

Пример 6.16. *Если программа не содержит подпрограмм, то дерево вызовов состоит из одного корня, но вовсе необязательно, что программа останавливается:*

```
Sub A;
arg x;
while 0 < 1 do
  A = x;
end;
```

Очевидно, что результат $\mathbf{Res}\,(A, x)$ *не определён для всех* $x \in \omega$.

Однако, для программ без циклов обратное верно. Прежде всего докажем следующую лемму.

Лемма 6.4 (Об определённости вложенных вызовов)**.** *Если для программы без циклов каждый элемент* $\mathbf{Fc}\,\delta$ *определён, то* $\mathbf{Res}\,\delta$ *определён.*

Доказательство. Сначала индукцией по построению выражения e доказывается, что если в $\mathbf{Fc}\,(\sigma, e)$ каждый элемент определён, то значение e определено. Затем то же самое доказывается для тестов и программ. □

***Задача 6.14.** Докажите лемму полностью.

Лемма 6.5 (О конечных деревьях вызовов)**.** *Если дерево* $\mathbf{Ft}\,\delta$ *конечно для программы без циклов, то* $\mathbf{Res}\,\delta$ *определён.*

Доказательство. Индукция по высоте дерева $\mathbf{Ft}\,\delta$.

Базис индукции.

Если δ — лист дерева, то список вложенных вызовов $\mathbf{Fc}\,\delta$ пуст, и все его элементы определены. По лемме $\mathbf{Res}\,\delta$ определено.

Шаг индукции.

Пусть для всех сыновей вершины δ утверждение доказано. Так как каждый вложенный вызов определён, то $\mathbf{Res}\,\delta$ тоже определено. □

Лемма 6.6 (Условие зацикливания)**.** *Если в дереве* $\mathbf{Ft}\,\delta$ *есть вершина* $\varepsilon = \delta$*, то* $\mathbf{Res}\,\delta$ *не определён.*

Задача 6.15. Докажите лемму.

§ 6.3. *Аппликативное и функциональное программирование

В этом параграфе мы докажем, что вызовы подпрограмм могут заменить и оператор присваивания, и оператор цикла. Но прежде чем делать это в общем случае рассмотрим пример.

```
Sub S;
arg x, y;
  S = S_1 (x, y, 0);
end;
```

```
Sub S_1;
arg x, y, z;
  if y < x then
    S_1 = S_1 (x, succ (y), succ (z));
  else
    S_1 = z;
  end;
end;
```

Рис. 6.7: Вычитание чисел.

Пример 6.17. *Изменим подпрограмму, вычисляющую разность, и введём вспомогательную подпрограмму, см. рис. 6.7.*

Как видим, ни одна из подпрограмм не содержит ни операторов цикла, ни операторов присваивания, за исключением специального случая, который используется для изменения выходной переменной результата вычисления подпрограммы. Докажем, что подпрограмма S действительно вычисляет разность чисел.

Утверждение 6.1. *для любого состояния σ выполнено*

$$\Pi_{S_1}(\sigma)(S_1) = \lfloor \sigma x - \sigma y \rfloor + \sigma z.$$

Доказательство. Докажем это индукцией по $\lfloor \sigma x - \sigma y \rfloor$.

Базис индукции.

$\lfloor \sigma x - \sigma y \rfloor = 0$. То есть $\sigma y \geq \sigma x$. Тогда

$$\Pi_{S_1}(\sigma) = (S_1 = z;)(\sigma) = \tau,$$

где τ отличается от σ только тем, что

$$\tau(S_1) = \sigma z = \lfloor \sigma x - \sigma y \rfloor + \sigma z.$$

Следовательно,

$$\Pi_{S_1}(\sigma)(S_1) = \lfloor \sigma x - \sigma y \rfloor + \sigma z.$$

Шаг индукции.

Пусть для $\lfloor \sigma x - \sigma y \rfloor < n$ утверждение доказано, $n > 0$ и $\lfloor \sigma x - \sigma y \rfloor = n$. Тогда $\sigma x > \sigma y$, следовательно,

$$\Pi_{S_1}(\sigma) = (S_1 = S_1(x, \mathsf{succ}(y), \mathsf{succ}(z));)(\sigma).$$

Чтобы определить значение выражения $S_1\left(x, \mathsf{succ}\left(y\right), \mathsf{succ}\left(z\right)\right)$ на состоянии σ, рассмотрим τ такое, что

$$\begin{aligned}\tau x &= \sigma x,\\ \tau y &= \sigma(\mathsf{succ}(y)) = \sigma y + 1,\\ \tau z &= \sigma(\mathsf{succ}(z)) = \sigma z + 1.\end{aligned}$$

Определим $\Pi_{S_1}\left(\tau\right)$. Заметим, что

$$\lfloor \tau x - \tau y \rfloor = \lfloor \sigma x - \sigma y - 1 \rfloor < \lfloor \sigma x - \sigma y \rfloor = n.$$

По индукционному предположению:

$$\begin{aligned}\Pi_{S_1}\left(\tau\right)\left(S_1\right) = \lfloor \tau x - \tau y \rfloor + \tau z = \\ = \lfloor \sigma x - \sigma y - 1 \rfloor + \sigma z + 1 = \lfloor \sigma x - \sigma y \rfloor + \sigma z.\end{aligned}$$

Следовательно, $\Pi_{S_1}\left(\sigma\right)\left(S_1\right) = \Pi_{S_1}\left(\tau\right)\left(S_1\right) = \lfloor \sigma x - \sigma y \rfloor + \sigma z$. □

Следствие 6.3. *Если* $\sigma z = 0$, *то* $\Pi_{S_1}\left(\sigma\right)\left(S_1\right) = \lfloor \sigma x - \sigma y \rfloor$.

Следствие 6.4. $\Pi_S\left(\sigma\right)\left(S\right) = \lfloor \sigma x - \sigma y \rfloor$.

Итак, мы доказали, что рассмотренный пример корректен.

Теперь рассмотрим, как избавиться от присваиваний и циклов в общем случае.

Определение 6.16 (Аппликативная программа)**.** *Программа называется а п п л и к а т и в н о й, если она не содержит операторов присваивания (кроме присваиваний выходной переменной).*

Определение 6.17 (Функциональная программа)**.** *Аппликативная программа называется ф у н к ц и о н а л ь н о й, если она не содержит циклов.*

Соответственно, алгоритм мы будем называть аппликативным (функциональным), если он состоит исключительно из аппликативных (функциональных) программ.

Обычный оператор присваивания часто называют также оператором разрушающего присваивания, поскольку он уничтожает значение переменной, которое было до его выполнения. Аппликативная программа (apply — применять) называется так, потому что разрушения не происходит, и все полученные результаты хранятся до конца.

Функциональная программа так называется, потому что единственный разрешённый оператор — ветвление. В конце параграфа мы покажем, что

если бы у нас была специальная операция **IF** (x, y, z), которая бы вычисляла первый аргумент и в зависимости от того истинен он или ложен возвращала значение второго или третьего аргумента, то можно было бы обойтись и без этого оператора, то есть программы содержали бы только вызовы подпрограмм. Более того, существует теория функционального программирования, где доказывается, что можно обойтись и без такой операции, если изменить обычные понятия истинности и ложности.

Итак, перейдём к доказательству основных теорем.

Теорема 6.1 (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 П мы выделим в отдельную подпрограмму, например, A'. То есть основную программу

Alg A;
arg $x_1,\ \ldots,\ x_n$;
П
end;

заменяем на

Sub A';
arg $x_1,\ \ldots,\ x_n$;
$(\text{П})^{A}_{A'}$
end;

Alg A;
arg $x_1,\ \ldots,\ x_n$;
$A\ =\ A'(x_1,\ \ldots,\ x_n)$;
end;

Ясно, что полученный алгоритм эквивалентен исходному.

Будем считать, что в теле любой подпрограммы f оператор $f = e$; встречается один раз и идёт последним. Если это не так, то этого можно добиться, введя новую переменную u, заменив всюду оператор $f = e$; на оператор $u = e$; и добавив в конце тела f оператор $f = u$;

Мы будем предполагать, что все переменные любой подпрограммы (кроме, конечно, выходной переменной) являются её аргументами. Этого тоже нетрудно добиться, достаточно включить их в список после arg, а в выражениях с использованием подпрограмм подставить для них нули.

Заметим, что при соблюдении этих условий выполняется такое утверждение. Если $\overline{x}$ — аргументы подпрограммы f, то для любого состояния σ

$$\sigma\left(f\left(\overline{x}\right)\right) = \Pi_f\left(\sigma\right)\left(f\right).$$

С каждой программой Π свяжем некоторое натуральное число $w\left(\Pi\right)$ — вес. Это будет мера «испорченности программы» по сравнению с функциональной. Определим его индуктивно для тела подпрограммы f:

1) $\Pi \backsimeq x = e$; $w\left(\Pi\right) = 1$, если $x \neq f$, и $w\left(\Pi\right) = 0$, если $x \backsimeq f$.

2) $\Pi \backsimeq \Pi_1\Pi_2$. Если Π_1 не является следованием, то $w\left(\Pi\right) = w\left(\Pi_1\right) + 2w\left(\Pi_2\right) + 1$.

3) $\Pi \backsimeq$ if T then Π_1 end; $w\left(\Pi\right) = 2w\left(\Pi_1\right) + 1$.

4) $\Pi \backsimeq$ if T then Π_1 else Π_2 end; $w\left(\Pi\right) = 2\left(w\left(\Pi_1\right) + w\left(\Pi_2\right)\right)$.

5) $\Pi \backsimeq$ while T do Π_1 end; $w\left(\Pi\right) = 2w\left(\Pi_1\right) + 2^{w(\Pi_1)+1} + 2$.

Если алгоритм A состоит из подпрограмм $f_1, \ldots, f_n$ и основной программы Π_A, то $w\left(A\right) = w\left(\Pi_A\right) + w\left(\Pi_{f_1}\right) + \cdots + w\left(\Pi_{f_n}\right)$.

Нетрудно заметить, что если вес алгоритма равен 0, то он является функциональным, потому что может состоять только из полных ветвлений и присваивания значений выходным переменным.

Теперь покажем, что если вес алгоритма больше 0, то он может быть уменьшен.

Пусть тело некоторой подпрограммы f не удовлетворяет определению функциональной программы. Возможны четыре варианта:

1) $\Pi_f \backsimeq x_i = e; \Pi'$

2) $\Pi_f \backsimeq$ if T then Π' end; Π''

3) $\Pi_f \backsimeq$ if T then Π' else Π'' end; Π'''

4) $\Pi_f \backsimeq$ while T do Π' end; Π'' $f = u$;

В последнем случае мы без ограничения общности предполагаем, что u является переменной. Выше мы сказали, что этого можно добиться. Рассмотрим, как разобраться с каждым из этих случаев.

1. Создадим новую подпрограмму f_1 и заменим подпрограмму f всюду подпрограммой $\tilde{f}$:

Sub $\tilde{f}$;	Sub f_1;
arg $x_1,\ \ldots,\ x_n$;	arg $x_1,\ \ldots,\ x_n$;
$\tilde{f}\ =\ f_1\left(x_1,\ldots,x_{i-1},e,x_{i+1},\ldots,x_n\right)$;	$(\Pi')^f_{f_1}$
end;	end;

Рассмотрим $\sigma\left(\tilde{f}\left(e_1,\ldots,e_n\right)\right)$. По определению

$$\sigma\left(\tilde{f}\left(e_1,\ldots,e_n\right)\right)=\Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right),$$

где $\tau x_j=\sigma\left(e_j\right)$. Далее,

$$\Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right)=\tau\left(f_1\left(x_1,\ldots,x_{i-1},e,x_{i+1},\ldots,x_n\right)\right)=\Pi_{f_1}\left(\rho\right)\left(f_1\right),$$

где $\rho x_j=\tau x_j$ кроме того, что $\rho x_i=\tau\left(e\right)$ и $\rho f_1=0$. Но тогда

$$\Pi_{f_1}\left(\rho\right)\left(f_1\right)=\Pi_{f_1}\left(\left(x_i=e;\right)\left(\tau\right)\right)\left(f_1\right),$$

по определению семантики присваивания и, потому что переменная f_1 встречается в Π_{f_1} один раз в конце, а переменная $\tilde{f}$ не встречается вовсе. Используем лемму о замене переменной:

$$\begin{aligned}\Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right)&=\Pi_{f_1}\left(\left(x_i=e;\right)\left(\tau\right)\right)\left(f_1\right)=\\&=\left(\Pi'\right)^f_{f_1}\left(\left(x_i=e;\right)\left(\tau\right)\right)\left(f_1\right)=\Pi'\left(\left(x_i=e;\right)\left(\tau\right)\right)\left(f\right)=\Pi_f\left(\tau\right)\left(f\right).\end{aligned}$$

Из этого следует, что

$$\sigma\left(\tilde{f}\left(e_1,\ldots,e_n\right)\right)=\sigma\left(f\left(e_1,\ldots,e_n\right)\right).$$

Рассмотрим, как в ходе такого преобразования изменяется вес:

$$w\left(\Pi_f\right)\geq 2w\left(\Pi'\right)+1,$$

$$w\left(\Pi'\right)+w\left(\Pi_{\tilde{f}}\right)=w\left(\Pi'\right)<w\left(\Pi_f\right).$$

2. Аналогично следующему пункту.

3. Построим три новые подпрограммы f_1, f_2 и f_3, а f заменим всюду на $\tilde{f}$:

```
Sub f̃;
arg x̄;
  if T then
    f̃ = f1 (x̄);
  else
    f̃ = f2 (x̄);
  end;
end;
```

```
Sub f1;
arg x̄;
  Π′
  f1 = f3 (x̄);
end;
```

```
Sub f2;
arg x̄;
  Π″
  f2 = f3 (x̄);
end;
```

```
Sub f3;
arg x̄;
  (Π‴)^f_{f3}
end;
```

Итак, рассмотрим $\sigma\left(\tilde{f}\left(\overline{e}\right)\right) = \Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right)$, где $\tau x_i = \sigma\left(e_i\right)$. По определению,

$$\Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right) = \begin{cases} \tau\left(f_1\left(\overline{x}\right)\right), & \text{если } \tau \models T; \\ \tau\left(f_2\left(\overline{x}\right)\right), & \text{если } \tau \not\models T; \\ \text{не определено}, & \text{если } \tau\left(T\right) \text{ не определено.} \end{cases}$$

Рассмотрим, например, $\tau\left(f_1\left(\overline{x}\right)\right)$.

$$\tau\left(f_1\left(\overline{x}\right)\right) = \Pi_{f_1}\left(\tau\right)\left(f_1\right) = \left(\Pi' f_1 = f_3\left(\overline{x}\right);\right)\left(\tau\right)\left(f_1\right) = \Pi'\left(\tau\right)\left(f_3\left(\overline{x}\right)\right).$$

Для всякого состояния ρ имеем

$$\rho\left(f_3\left(\overline{x}\right)\right) = \Pi_{f_3}\left(\rho\right)\left(f_3\right) = \Pi'''\left(\rho\right)\left(f\right).$$

Следовательно, для $\rho = \Pi'\left(\tau\right)$ получаем

$$\Pi'\left(\tau\right)\left(f_3\left(\overline{x}\right)\right) = \Pi'''\left(\Pi'\left(\tau\right)\right)\left(f\right).$$

Итак,

$$\tau\left(f_1\left(\overline{x}\right)\right) = \Pi'''\left(\Pi'\left(\tau\right)\right)\left(f\right).$$

Аналогичным образом можно доказать, что

$$\tau\left(f_2\left(\overline{x}\right)\right) = \Pi'''\left(\Pi''\left(\tau\right)\right)\left(f\right).$$

Поэтому,

$$\Pi_{\tilde{f}}\left(\tau\right)\left(\tilde{f}\right) = \begin{cases} \Pi'''\left(\Pi'\left(\tau\right)\right)\left(f\right), & \text{если } \tau \models T; \\ \Pi'''\left(\Pi''\left(\tau\right)\right)\left(f\right), & \text{если } \tau \not\models T; \\ \text{не определено}, & \text{если } \tau\left(T\right) \text{ не определено.} \end{cases}$$

Подойдём с другой стороны: $\sigma\left(f\left(\overline{e}\right)\right) = \Pi_f\left(\tau\right)\left(f\right)$. По определению семантики для ветвления и для следования, получаем:

$$\Pi_f\left(\tau\right)\left(f\right) = \begin{cases} \Pi'''\left(\Pi'\left(\tau\right)\right)\left(f\right), & \text{если } \tau \models T; \\ \Pi'''\left(\Pi''\left(\tau\right)\right)\left(f\right), & \text{если } \tau \not\models T; \\ \text{не определено}, & \text{если } \tau\left(T\right) \text{ не определено}. \end{cases}$$

Как видим, получилось то же самое, то есть

$$\sigma\left(f\left(\overline{e}\right)\right) = \sigma\left(\tilde{f}\left(\overline{e}\right)\right)$$

для всякого состояния σ.

Вес был равен

$$w = 2\left(w\left(\Pi'\right) + w\left(\Pi''\right)\right) + 2w\left(\Pi'''\right) + 1,$$

стал равен:

$$w\left(\Pi'\right) + w\left(\Pi''\right) + w\left(\Pi'''\right).$$

4. Заменим всюду f на $\tilde{f}$:

```
Sub f̃;
arg x̄;
  if T then
    Π′
    u = f̃(x̄);
  else
    Π″
  end;
  f̃ = u;
end;
```

Пусть $\tau\left(f\left(\overline{e}\right)\right)$ определено. Пусть

$$\delta = \left(f, \tau\left(e_1\right), \ldots, \tau\left(e_n\right)\right).$$

Значит последовательность состояний цикла $\left(\sigma^i\right)_{i=0}^{n}$: $\sigma^0 = \sigma_\delta$, $\sigma^{i+1} = \Pi'\left(\sigma^i\right)$ является конечной, $\Pi'\left(\sigma^i\right)$ определено для всех $i < n$ и $\sigma^i \models T$ тогда и только тогда, когда $i < n$. Следовательно,

$$\tau\left(f\left(\overline{e}\right)\right) = \Pi''\left(\sigma^n\right)\left(u\right) = \Pi_f\left(\sigma^n\right)\left(f\right),$$

так как $\sigma^n \not\models T$.

Индукцией по $n - i$ докажем, что $\Pi_{\tilde{f}}\left(\sigma^i\right)$ определено и

$$\Pi_{\tilde{f}}\left(\sigma^i\right)\left(\tilde{f}\right) = \Pi_f\left(\sigma^n\right)(f).$$

Базис индукции.
Если $n = i$, то $\sigma^i \not\models T$ и, следовательно,

$$\Pi_{\tilde{f}}\left(\sigma^i\right)\left(\tilde{f}\right) = \Pi''\left(\sigma^i\right)(u) = \Pi_f\left(\sigma^i\right)(u) = \Pi_f\left(\sigma^n\right)(f).$$

Шаг индукции.
Пусть для $i+1$ утверждение доказано и $i < n$. Тогда $\sigma^i \models T$.

$$\Pi_{\tilde{f}}\left(\sigma^i\right) = \Pi'\left(\sigma^i\right)\left(\tilde{f}\left(\overline{x}\right)\right) = \sigma^{i+1}\left(\tilde{f}\left(\overline{x}\right)\right) = \Pi_{\tilde{f}}\left(\sigma^{i+1}\right)\left(\tilde{f}\right).$$

По индукционному предположению получаем, что

$$\Pi_{\tilde{f}}\left(\sigma^{i+1}\right)\left(\tilde{f}\right) = \Pi_f\left(\sigma^n\right)(f).$$

Итак, мы доказали, что

$$\Pi_{\tilde{f}}\left(\sigma^0\right)\left(\tilde{f}\right) = \Pi_f\left(\sigma^n\right)(f).$$

Но $\sigma_0 = \sigma_\delta$, поэтому

$$\tau\left(\tilde{f}\left(\overline{e}\right)\right) = \Pi_{\tilde{f}}\left(\sigma_\delta\right)\left(\tilde{f}\right) = \Pi_f\left(\sigma^n\right)(f) = \tau\left(f\left(\overline{e}\right)\right).$$

Пусть $\tau\left(\tilde{f}\left(\overline{e}\right)\right)$ определено. Пусть

$$\delta = \left(f, \tau\left(e_1\right), \ldots, \tau\left(e_n\right)\right).$$

Рассмотрим последовательность состояний цикла $\left(\sigma^i\right)_i$: $\sigma^0 = \sigma_\delta$, $\sigma^{i+1} = \Pi'\left(\sigma^i\right)$.

Индукцией по i покажем, что если $\sigma^j \models T$ для $j \le i$, то $\Pi_{\tilde{f}}\left(\sigma^i\right)$ определено и

$$\Pi_{\tilde{f}}\left(\sigma^i\right)\left(\tilde{f}\right) = \Pi_{\tilde{f}}\left(\sigma_\delta\right)\left(\tilde{f}\right).$$

Базис индукции.
Для $i = 0$ это следует из определённости $\tau\left(\tilde{f}\left(\overline{e}\right)\right)$.

Шаг индукции.
Пусть для i утверждение доказано. Пусть $\sigma^j \models T$ для $j \leq i+1$. Рассмотрим $\Pi_{\tilde{f}}\left(\sigma^i\right)$. Так как $\sigma^i \models T$, то

$$\Pi_{\tilde{f}}\left(\sigma^i\right)\left(\tilde{f}\right) = \left(\tilde{f} = u;\right)\left(\left(u = \tilde{f}\left(\overline{x}\right);\right)\left(\Pi'\left(\sigma^i\right)\right)\right)\left(\tilde{f}\right) = \\ = \sigma^{i+1}\left(\tilde{f}\left(\overline{x}\right)\right) = \Pi_{\tilde{f}}\left(\sigma^{i+1}\right)\left(\tilde{f}\right),$$

что и требовалось. Более того, мы доказали, что если $\sigma^i \models T$, то для определения $\Pi_{\tilde{f}}\left(\sigma^i\right)$ необходимо определить $\Pi_{\tilde{f}}\left(\sigma^{i+1}\right)$.

Так как $\Pi_{\tilde{f}}\left(\sigma^0\right)$ определено, то такая цепь не может быть бесконечной, следовательно, $\sigma^n \not\models T$ для некоторого n. Тогда

$$\Pi_f\left(\sigma_\delta\right)(f) = \Pi''\left(\sigma^n\right)(u).$$

С другой стороны

$$\Pi_{\tilde{f}}\left(\sigma_\delta\right)(f) = \Pi''\left(\sigma^n\right)(u),$$

то есть $\Pi_f\left(\sigma_\delta\right)$ должно быть определено. Из этого следует, что

$$\tau\left(\tilde{f}\left(\overline{e}\right)\right) = \tau\left(f\left(\overline{e}\right)\right).$$

Вес был:

$$w\left(\Pi_f\right) = 2 + 2w\left(\Pi'\right) + 2^{w\left(\Pi'\right)+1} + 2w\left(\Pi''\right) + 1 + 1,$$

стал:

$$w\left(\Pi_{\tilde{f}}\right) \leq 1 + 2\left(w\left(\Pi'\right) + 1 + 2^{w\left(\Pi'\right)}\right) + 2w\left(\Pi''\right) < w\left(\Pi_f\right).$$

Будем производить преобразования 1–4, пока они приводят к уменьшению $w(A)$.

Если $w(A) = 0$, то тело каждой подпрограммы f либо является присваиванием вида $f = e$; либо является полным ветвлением, частями которого снова являются или присваивания такого вида, или операторы полного ветвления. Следовательно, такой алгоритм является функциональным. □

*__Задача 6.16.__ Покажите, как убрать неполное ветвление и докажите правильность предложенного построения.

Как мы видим, передача аргументов заменяет присваивание, а рекурсивный вызов — оператор цикла.

Задача 6.17. Пользуясь методом из теоремы, по алгоритмам из задачи 6.2 постройте эквивалентные им функциональные.

Лемма 6.7. *Если у нас есть операция* $\mathbf{IF}(x, y, z)$*, аргументами которой являются* x *— тест,* y, z *— выражения, такая, что для всякого состояния* σ

$$\sigma(\mathbf{IF}(x, y, z)) = \begin{cases} \sigma(y), \text{если } \sigma \models x, \\ \sigma(z), \text{если } \sigma \not\models x, \\ \text{не определено, если } \sigma(x) \text{ не определено,} \end{cases}$$

то программа вида

```
if T then
  f = e1;
else
  f = e2;
end;
```

эквивалентна программе

$$f \;=\; \mathbf{IF}(T,\; e_1,\; e_2);$$

Задача 6.18. Докажите лемму.

Теперь мы можем показать, что при наличии операции **IF**, функциональная программа оправдывает своё название:

Следствие 6.5. *Если у нас есть операция* $\mathbf{IF}(x, y, z)$ *(см. предыдущую лемму), то каждый алгоритм эквивалентен функциональному, в котором каждая подпрограмма и основная программа имеет тело вида:*

$$f \;=\; e;$$

где f *— имя подпрограммы или алгоритма соответственно.*

Задача 6.19. Докажите следствие.

Замечание 6.3. Покажем, что изменив понятия истинности и ложности можно избавиться и от **IF**. Будем считать, что истина — это двухместная операция, возвращающая первый аргумент, а ложь — двухместная операция, возвращающая второй аргумент. Тогда $\mathbf{IF}(x, y, z)$ можно записать в виде $(x)(y, z)$. В самом деле, если тест x истинен, то

$$(x)(y, z) = \text{ИСТИНА}(y, z) = y = \mathbf{IF}(x, y, z),$$

а если ложен, то

$$(x)(y,z) = \text{ЛОЖЬ}(y,z) = z = \mathbf{IF}(x,y,z).$$

§ 6.4. **Удаление подпрограмм

Теперь докажем обратную теорему.

Теорема 6.2 (Об удалении подпрограмм)**.** *По каждому алгоритму можно построить эквивалентный алгоритм без подпрограмм.*

Доказательство. Докажем эту теорему для случая, когда нет рекурсии, а затем докажем, что каждый алгоритм можно преобразовать к нерекурсивному. В этом случае должна быть подпрограмма f, в которой не встречаются никакие другие подпрограммы (в противном случае имелась бы рекурсия). Пользуясь теоремой о подстановке можно удалить подпрограмму f из всех остальных подпрограмм и основной программы. Ввиду того, что внутри f другие подпрограммы не встречались, то после подстановки рекурсии не будет. Пользуясь этой процедурой можно исключить по очереди все подпрограммы. □

Такой метод не годится, если присутствует рекурсия, так подставляя вместо выражений вида $f(\overline{e})$ 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 6.8 (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 $z > 0$ существуют единственные числа a и b такие, что b — нечётное, a — степень двойки и $z = ab$.

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 6.20.** Завершите доказательство леммы.

Задача 6.21. Напишите подпрограммы для вычисления функций ν, ν_1, ν_2 из леммы без использования рекурсии.

Очевидными свойствами функции ν являются следующие:

Следствие 6.6. $\nu(x, y) > x$, $\nu(x, y) > y$, $\nu(x, y) > 0$.

Следующий наш шаг — доказать, что всякую косвенную рекурсию можно заменить прямой.

Лемма 6.9 (О косвенной рекурсии)**.** *Каждый алгоритм эквивалентен некоторому алгоритму, в котором присутствует только одна подпрограмма.*

Доказательство. Прежде всего, будем считать, что все подпрограммы имеют одинаковое количество аргументов. Если это не так всегда можно добавить фиктивные параметры, которые нигде не используются.

Пусть алгоритм содержит m подпрограмм $f_1, \dots, f_m$, каждая из которых имеет n аргументов. Можно считать, что все подпрограммы $f_1, \dots, f_m$ имеют аргументы $x_1, \dots, x_n$. Рассмотрим следующую подпрограмму f:

```
Sub f;
arg x, x_1, ..., x_n;
if x = 1 then
  (Π_{f_1}) f_1(·) ... f_m(·) / f(1, ·) ... f(m, ·)
end;
⋮
if x = m then
  (Π_{f_m}) f_1(·) ... f_m(·) / f(1, ·) ... f(m, ·)
end;
```

Здесь x — новая переменная. Каждое выражение вида $f_i(\overline{e})$ мы заме-

няем выражением вида $f\,(i, \overline{e})$. Для каждого состояния σ будем иметь

$$\sigma\,(f_i\,(\overline{e})) = \sigma\,(f\,(i, \overline{e}))\,.$$

Проделав точно такую же замену $f_i\,(\overline{e})$ на $f\,(i, \overline{e})$ в основной программе мы получим алгоритм, эквивалентный исходному. Так как теперь в алгоритме присутствует только одна подпрограмма, то рекурсия может быть только прямой. □

***Задача 6.22.** Докажите, что для любого состояния σ

$$\sigma\,(f_i\,(\overline{e})) = \sigma\,(f\,(i, \overline{e}))$$

для любого $i = 1, \ldots, m$, и любых выражений $\overline{e}$.

Теорема 6.3 (Об удалении рекурсии)**.** *По каждому алгоритму можно построить эквивалентный алгоритм без рекурсии.*

Доказательство. Итак, согласно предыдущим утверждениям, мы можем считать, что у нас есть одна подпрограмма f с n аргументами. 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

$$u \;=\; f\,(u_1,\; \ldots,\; u_n)\,;$$

Более того, можно считать, что переменная u во всех таких операторах одна и та же.

Во-вторых, переделаем тело подпрограммы f в программу с метками. Будем считать, что метки — натуральные положительные числа. γ — заключительная метка. Заметим, что все проделанные преобразования привели к эквивалентной подпрограмме.

Итак, подпрограмма f сейчас имеет вид

Sub f;
arg $\overline{x}$;
o_1
o_2
$\vdots$
o_N
end;

Каждый из операторов o_k имеет один из видов:

1) $o_k \eqcirc \alpha_k\ x_k = e_k;\ \beta_k,$

2) $o_k \eqcirc \alpha_k$ if T_k then β_k else $\gamma_k,$

3) $o_k \eqcirc \alpha_k\ u = f\,(u_1, \ldots, u_n);\ \beta_k,$

причём e_k и T_k не содержат f.

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 4.2. Добавим новые переменные w, y, s и R. Заменим всюду оператор присваивания $f = e$; на $R = e$; 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:

$$\varphi\,(o_k) \eqcirc \begin{cases} \text{if } w \;=\; \mathsf{succ}\,(0) \text{ then} \\ \quad \text{if } y \;=\; \overline{\alpha_k} \text{ then} \\ \qquad s \;=\; \nu\,(s,\ z_1);\ \ldots\ s \;=\; \nu\,(s,\ z_m); \\ \qquad s \;=\; \nu\,(s,\ x_1);\ \ldots\ s \;=\; \nu\,(s,\ x_n); \\ \qquad s \;=\; \nu\,\big(s,\ \overline{\beta_k}\big); \\ \qquad x_1 \;=\; u_1;\ \ldots\ x_n \;=\; u_n; \\ \qquad z_1 \;=\; 0;\ \ldots\ z_m \;=\; 0; \\ \qquad y \;=\; \overline{\alpha_1}; \\ \qquad w \;=\; 0; \\ \quad \text{end}; \\ \text{end}; \end{cases}$$

Здесь мы предполагаем, что $z_1, \ldots, z_m$ — все переменные программы Π_f (исключая y, w, s, но включая R), которые не являются её аргументами,

а $x_1, \ldots, x_n$ — аргументы f. Пусть

$$\varphi_1 \Leftarrow \begin{cases} \mathsf{if}\ w\ =\ \mathsf{succ}\,(0)\ \mathsf{then} \\ \quad \mathsf{if}\ 0\ <\ s\ \mathsf{then} \\ \quad\quad y\ =\ \nu_2\,(s);\ s\ =\ \nu_1\,(s); \\ \quad\quad x_n\ =\ \nu_2\,(s);\ s\ =\ \nu_1\,(s); \\ \quad\quad\quad\quad \vdots \\ \quad\quad x_1\ =\ \nu_2\,(s);\ s\ =\ \nu_1\,(s); \\ \quad\quad z_m\ =\ \nu_2\,(s);\ s\ =\ \nu_1\,(s); \\ \quad\quad\quad\quad \vdots \\ \quad\quad z_1\ =\ \nu_2\,(s);\ s\ =\ \nu_1\,(s); \\ \quad\quad u\ =\ R; \\ \quad\quad w\ =\ 0; \\ \quad \mathsf{end}; \\ \mathsf{end}; \end{cases}$$

Теперь мы готовы построить нерекурсивную подпрограмму f_1:

$$\begin{array}{l} \mathsf{Sub}\ f_1; \\ \mathsf{arg}\ \overline{x}; \\ \quad y\ =\ \overline{\alpha_1};\ w\ =\ 0;\ s\ =\ 0; \\ \quad \mathsf{while}\ w\ =\ 0\ \mathsf{do} \\ \quad\quad w\ =\ \mathsf{succ}\,(0)\,; \\ \quad\quad \varphi\,(o_1) \\ \quad\quad\quad \vdots \\ \quad\quad \varphi\,(o_N) \\ \quad\quad \varphi_1 \\ \quad \mathsf{end}; \\ \quad f_1\ =\ R; \\ \mathsf{end}; \end{array}$$

Пусть $\Pi_f\,(\sigma)$ определено. Следовательно, последовательность расширенных состояний $\left(\sigma^i, \lambda_i\right)_i$ конечна. Рассмотрим последовательность состояний цикла Π_{f_1}: $\left(\sigma_1^j\right)_j$, и последовательность расширенных состояний Π_f: $\left(\sigma^i, \lambda_i\right)_i$. Пусть $j_0 = 0$. Индукцией по высоте леса вложенных вызовов $\mathbf{Ff}\,(\sigma, \Pi_f)$ мы докажем следующее утверждение:

Утверждение 6.2. *Для всякого i существует j_i такое, что*

1) $\sigma_1^{j_i} s = \sigma_1^{j_0} s$,

2) $\sigma_1^{l} s \geq \sigma_1^{j_0} s$ *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 $\mathbf{Ff}(\sigma, \Pi_f)$ имеет высоту $h+1$. Используем индукцию по длине последовательности $\left(\sigma^i, \lambda_i\right)_i$. Для операторов первых двух видов доказательство повторяет доказательство теоремы о структуризации.

Рассмотрим i-ый элемент последовательности: $\left(\sigma^i, \lambda_i\right)$. Пусть $\lambda_i = \alpha_k$, а сам оператор имеет вид

$$\alpha_k \quad u = f\left(\overline{u}\right); \quad \beta_k.$$

Пусть δ — вызов, служащий, для вычисления $\sigma^i\left(f\left(\overline{u}\right)\right)$.

В последовательности $\left(\sigma_1^j\right)_i$ следующим за $\sigma_1^{j_i}$ будет состояние $\sigma_1^{j_i+1}$ такое, что $\sigma_1^{j_i+1} y = \alpha_1$, $\sigma_1^{j_i+1} w = 0$,

$$\sigma_1^{j_i+1} s = \nu\left(\nu^{j_i}\left(\nu^{j_i}\left(s, \overline{z}\right), \overline{x}\right), \beta_k\right),$$

и $\sigma_1^{j_i+1} z = \sigma_\delta z$ для остальных переменных z.

Здесь мы обозначаем

$$\nu^l\left(s, \overline{w}\right) = \nu\left(\nu\left(\ldots \nu\left(\sigma_1^l s, \sigma_1^l w_1\right)\ldots\right), \sigma_1^l w_N\right).$$

Заметим, что $\sigma_1^{j_i+1} s > \sigma_1^{j_i} s \geq 0$.

Высота леса $\mathbf{Ff}\left(\sigma_\delta, \Pi_f\right)$ не больше h. Так как вызов δ определён, то последовательность расширенных состояний для $\Pi_f\left(\sigma_\delta\right)$ конечна, и заканчивается она расширенным состоянием 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 тесты $y = \overline{\alpha_k}$ в операторах $\varphi(o_k)$ окажутся ложными. Следовательно, тесты $w = succ(0)$ и $0 < s$ в φ_1 будут истинны. 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 $\left(\sigma^{i+1}, \lambda_{i+1}\right)$. Утверждение доказано.

Последовательность расширенных состояний $\left(\sigma^i, \lambda_i\right)_i$ оканчивается расширенным состоянием (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 6.3. *Если* $k < l$,

$$\sigma_1^k s = \sigma_1^l s, \quad \sigma_1^j s \geq \sigma_1^k s \;\; \text{для } j \in [k, l],$$

то в последовательности расширенных состояний для $\Pi_f\left(\sigma^0, \lambda_0\right)$ *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* $\left(\sigma^{l'}, \lambda_{l'}\right)$ *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 $\sigma_1^j s > \sigma_1^k s$.

Базис индукции.

Если таких j нет, то доказательство повторяет доказательство теоремы о структуризации.

Шаг индукции.

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 $\sigma_1^i s \geq \sigma_1^j s$ для всех $i \in [j, p]$.

Последовательности состояний

$$\sigma^k, \ldots, \sigma^{j-1};$$

$$\sigma^j, \ldots, \sigma^p;$$

$$\sigma^{p+1}, \ldots, \sigma^l$$

удовлетворяют условиям индукционного предположения. Применим его к этим трём случаям.

В последовательности расширенных состояний для $\Pi_f\left(\sigma^0, \lambda_0\right)$ 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 $\left(\sigma^{j'}, \lambda_{j'}\right)$ 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

$$\lambda_{j'}\ u = f\left(\overline{u}\right);\ \lambda_{j'+1}.$$

Для вычисления $\sigma^{j'}\left(f\left(\overline{u}\right)\right)$ используется вызов

$$\delta = \left(f, \sigma^{j'} u_1, \ldots, \sigma^{j'} u_n\right).$$

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 все тесты $y = \overline{\alpha}$ являются ложными на состоянии σ^p. Это означает, что $\sigma^p y = \gamma$ — заключительная метка.

Таким образом, в последовательности расширенных состояний для $\Pi_f\left(\sigma^0, \lambda_0\right)$ 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, имеется элемент $\left(\sigma^{p'}, \lambda_{p'}\right)$ такой, что

1) $\gamma = \lambda_{p'}$,

2) $\sigma_1^p R = \sigma^{p'} f$.

Из этого следует, что

$$\mathbf{Res}\,\delta = \sigma_1^p R.$$

Заметим, что σ_1^{p+1} совпадает с σ_1^{j-1} кроме того, что

$$\sigma_1^{p+1} u = \sigma_p^1 R = \mathbf{Res}\,\delta = \sigma^{j'}\left(f\left(\overline{u}\right)\right).$$

Это означает, что σ_1^{p+1} соответствует $\left(\sigma^{j'+1}, \lambda_{j'+1}\right)$ в последовательности расширенных состояний.

В последовательности расширенных состояний для $\Pi_f\left(\sigma^{j'+1}, \lambda_{j'+1}\right)$ 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 $\left(\sigma^{l'}, \lambda_{l'}\right)$ 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 $\left(\sigma_1^i\right)_i$ 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 расширенных состояний для $\Pi_f\left(\sigma^0, \lambda_0\right)$, где $\lambda_0 = \alpha_1$, $\sigma^0 f = 0$ и $\sigma^0 z = \sigma_\delta z$ для остальных переменных z, имеется элемент $\left(\sigma^l, \lambda_l\right)$ такой, что $\lambda_l = \gamma$. То есть $\Pi_f(\sigma)$ определено. □

Задача 6.23. Пользуясь предложенным методом удалите рекурсию из программы из примера 6.17.

§ 6.5. *Доказательство корректности подпрограмм

Очевидно, что для доказательства корректности алгоритмов с подпрограммами правил, которые мы изучали раньше, недостаточно. В самом деле, если f — подпрограмма, то мы не можем использовать аксиому

$$\left\{(\varphi)^x_{f(\overline{y})}\right\} x = f\left(\overline{y}\right); \{\varphi\},$$

```
Sub S;
arg x, y, z;
  if x < succ(y) then
    S = z;
  else
    S = S(x, succ(y), succ(z));
  end;
end;
```

Рис. 6.8: Вычитание.

потому что мы не можем сказать, что означает $f(\overline{y})$ в условии. Истинность условия на состоянии должна определяться самим условием и состоянием, но не программой, для которой это условие написано.

Определение 6.18 (Ограничитель вызовов)**.** *Ограничитель вызовов подпрограммы f при условии φ — выражение L_φ такое, что*

1) $\sigma(L_\varphi) \in \omega$ *для любого состояния* σ*;*

2) *Для любого вызова δ и рекурсивного для δ вызова ε, если $\sigma_\delta \models \varphi$, то*

$$\sigma_\varepsilon(L_\varphi) < \sigma_\delta(L_\varphi).$$

Пример 6.18. *Рассмотрим подпрограмму на рис. 6.8. Пусть $L \Leftarrow \lfloor x - y \rfloor$. Рассмотрим произвольный вызов*

$$\delta = (S, a, b, c)$$

и рекурсивный для него вызов

$$\varepsilon = (S, d, e, f).$$

Очевидно, что $a > b$ (иначе $\mathbf{Fc}\,\delta = ()$). Пусть $\zeta = (S, a, b+1, c+1)$, тогда

$$\mathbf{Fc}\,\delta = (\zeta) = ((S, a, b+1, c+1)),$$

$$\sigma_\zeta(L) = \lfloor a - b - 1 \rfloor < \lfloor a - b - 1 \rfloor = \sigma_\delta(L).$$

По индукции получаем, что

$$\sigma_\varepsilon(L) < \sigma_\delta(L).$$

Для подпрограмм у нас будет два правила вывода.

ЗП Замена переменной (для присваиваний) Пусть x не входит в выражение e и условие φ. Пусть x обязательно входит в выражение d, если e содержит имена подпрограмм. Тогда из частичной (полной) корректности троек

$$\{\varphi\}\, x = e; \{\psi\} \quad \{\psi\}\, y = d; \{\theta\}$$

следует частичная (полная) корректность тройки

$$\{\varphi\}\, y = (d)_e^x\,; \{\exists x\, (\theta \wedge \exists y \psi)\}\,.$$

ЗП Замена переменной (для тестов) Будем использовать три разновидности этого правила. Пусть x не входит в условия φ и ψ, программы Π_1 и Π_2. Пусть x обязательно входит в тест T. Тогда из частичной (полной) корректности каждой из троек

$$\{\varphi\}\ \ x = e;\ \mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{end}; \{\psi\}$$

$$\{\varphi\}\ \ x = e;\ \mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end}; \{\psi\}$$

$$\{\varphi\}\ \ x = e;\ \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ x\ =\ e;\ \mathsf{end}; \{\psi\}$$

следует частичная (полная) корректность троек

$$\{\varphi\}\ \ \mathsf{if}\ \ (T)_e^x\ \ \mathsf{then}\ \Pi_1\ \mathsf{end}; \{\psi\}$$

$$\{\varphi\}\ \ \mathsf{if}\ \ (T)_e^x\ \ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end}; \{\psi\}$$

$$\{\varphi\}\ \ \mathsf{while}\ \ (T)_e^x\ \ \mathsf{do}\ \Pi_1\ \mathsf{end}; \{\psi\}$$

соответственно.

ВП Вызов подпрограммы Тройка

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n)\,; \left\{(\psi)_x^f\right\}$$

выводима, если

1) x не встречается среди $z_1, \ldots, z_n$;
2) $z_1, \ldots, z_n$ — аргументы подпрограммы f;

3)
$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)_x^f\right\} \underset{\mathfrak{IH}}{\rhd} \{\varphi'\}\, \Pi_f\, \{\psi'\}$$

Здесь
$$\varphi' \twoheadleftarrow \hat{\varphi} \wedge z_1 = \hat{z}_1 \wedge \cdots \wedge z_n = \hat{z}_n;$$
$$\psi' \twoheadleftarrow \hat{\psi}.$$

$\hat{\varphi}$ и $\hat{\psi}$ получаются из φ и ψ заменой каждой переменной u (кроме f) на переменную $\hat{u}$, не встречающуюся в Π_f.

4) для полной корректности требуется существование ограничителя вызова $L_{\varphi'}$.

Пункты 3 и 4 можно объединить в один: потребовать, чтобы
$$\{\varphi \wedge L < u\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)_x^f\right\} \underset{\mathfrak{IH}}{\rhd} \left\{\varphi' \wedge \hat{L} = u\right\} \Pi_f\, \{\psi'\},$$
где u — новая переменная.

Пункт 3 в правиле **ВП** означает следующее: мы берём
$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)_x^f\right\}$$
в качестве новой аксиомы, используя её и другие аксиомы и правила вывода доказываем тройку $\{\varphi'\}\, \Pi_f\, \{\psi'\}$, и если это удаётся, то тройку
$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n)\,; \left\{(\psi)_x^f\right\}$$
мы считаем доказанной.

****Задача 6.24.** Докажите, что с помощью каждого из правил **ЗП** из частично (полностью) корректных троек получаются частично (полностью) корректные.

Доказательство аналогичного утверждения для правила **ВП** является технически сложной задачей и оно вынесено в отдельный параграф.

Рассмотрим некоторые правила, которые допустимы в нашем исчислении.

ДОБ Докажем допустимость правила **ДОБ** в новом исчислении. Точно так же заменим в доказательстве переменные, которые используются в правилах **ЗП**, **ВП** и **ЦП** на новые, которые не встречаются ни доказательстве ни в θ.

Доказательство будет проводиться такой же индукцией, как и раньше. Только базисом индукции кроме аксиом **АКС** будут применения правила **ВП**, а также аксиомы, которые используются для этого правила.

Пусть доказуема тройка $\{\varphi\}\, x = f\,(\overline{z})\,; \{\psi\}$. Пусть u, v — новые переменные. Тройка

$$\{\varphi \wedge \theta\}\, u = v; \{\varphi \wedge \theta\}$$

является аксиомой **АКС**. Тройка

$$\{\varphi \wedge \theta\}\, x = f\,(\overline{z})\,; \{\psi\}$$

получается по правилу **УПР**. По правилу **ЗП** получаем

$$\{\varphi \wedge \theta\}\, x = f\,(\overline{z})\,; \{\exists u\,(\psi \wedge \exists x\,(\varphi \wedge \theta))\}\,.$$

Так как u не входит в $\psi \wedge \exists x\,(\varphi \wedge \theta)$, а x не входит в θ, то формула

$$\exists u\,(\psi \wedge \exists x\,(\varphi \wedge \theta)) \to \psi \wedge \theta$$

истинна. По правилу **ОП** получаем

$$\{\varphi \wedge \theta\}\, x = f\,(\overline{z})\,; \{\psi \wedge \theta\}\,.$$

ВП* Пусть φ из переменных содержит только $z_1, \ldots, z_n$, ψ — $z_1, \ldots, z_n, f$, и $z_1, \ldots, z_n \notin \mathbf{LVar}\,(\Pi_f)$. Тогда для доказуемости тройки

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n)\,; \left\{(\psi)^f_x\right\}$$

достаточно потребовать, чтобы

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)^f_x\right\} \underset{\mathfrak{IH}}{\rhd} \{\varphi\}\, \Pi_f\, \{\psi\}\,.$$

Доказательство. Пусть

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)^f_x\right\} \underset{\mathfrak{IH}}{\rhd} \{\varphi\}\, \Pi_f\, \{\psi\}\,.$$

Пусть

$$\theta \twoheadleftarrow z_1 = \hat{z}_1 \wedge \cdots \wedge z_n = \hat{z}_n.$$

Так как

$$\mathbf{Var}\,(\theta) \cap \mathbf{LVar}\,(\Pi_f) = \varnothing,$$

то по правилу **ДОБ** получаем:

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)^f_x\right\} \underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge \theta\}\, \Pi_f\, \{\psi \wedge \theta\}\,.$$

Так как φ не содержит переменных кроме $\overline{z}$, а ψ кроме $\overline{z}$ содержит лишь f, то

$$\varphi' \multimap \hat{\varphi} \wedge \theta; \quad \hat{\varphi} \wedge \theta \equiv \varphi \wedge \theta;$$
$$\psi' \multimap \hat{\psi}; \quad \hat{\psi} \Leftarrow \hat{\psi} \wedge \theta; \quad \hat{\psi} \wedge \theta \equiv \psi \wedge \theta.$$

По правилам **УПР** и **ОП** получаем:

$$\{\varphi\}\, x = f\,(z_1, \ldots, z_n) \left\{(\psi)^f_x\right\} \underset{\mathfrak{IH}}{\rhd} \{\varphi'\}\, \Pi_f\, \{\psi'\},$$

что и требовалось. □

ЗАМ Пусть

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\, x = e; \{\psi\},$$

y не входит ни в ψ, ни в одно из выражений $\overline{e}$, все выражения $\overline{e}$ не содержат подпрограмм, $\overline{z}$ — набор не совпадающих с x попарно различных переменных. Тогда

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{x\overline{z}}_{y\overline{e}}\right\} y = (e)^{x\overline{z}}_{y\overline{e}}; \left\{(\psi)^{x\overline{z}}_{y\overline{e}}\right\}.$$

Доказательство. Пусть u — новая переменная и

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi\}\, x = e; \{\psi\}.$$

Сначала докажем, что

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^z_u\}\, x = (e)^z_u; \{(\psi)^z_u\},$$

где z — любая переменная, отличная от x. Аксиома **ПРИСВ***:

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^z_u\}\, z = u; \{(\varphi)^z_u \wedge z = u\},$$

по правилу **ОП**:

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^z_u\}\, z = u; \{\varphi \wedge z = u\},$$

по правилу **УПР**:

$$\underset{\mathfrak{IH}}{\rhd} \{\varphi \wedge z = u\}\, x = e; \{\psi\}.$$

По правилу **ЗП**:

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^z_u\}\, x = (e)^z_u; \{\exists z\,(\exists x\,(\varphi \wedge z = u) \wedge \psi)\},$$

Формула

$$\exists z\,(\exists x\,(\varphi \wedge z = u) \wedge \psi) \to (\psi)^z_u$$

истинна, следовательно, по правилу **ОП**:

$$\underset{\mathfrak{IH}}{\rhd} \{(\varphi)^z_u\}\, x = (e)^z_u\,;\{(\psi)^z_u\}\,.$$

Применив эту процедуру несколько раз мы докажем, что

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{\overline{z}}_{\overline{u}}\right\} x = (e)^{\overline{z}}_{\overline{u}}; \left\{(\psi)^{\overline{z}}_{\overline{u}}\right\}.$$

Так как ψ не содержит y, то $\left((\psi)^{x\overline{z}}_{y\overline{u}}\right)^y_x \leftrightarrows (\psi)^{\overline{z}}_{\overline{u}}$. Согласно аксиоме **АКС**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\psi)^{\overline{z}}_{\overline{u}}\right\} y = x; \left\{(\psi)^{x\overline{z}}_{y\overline{u}}\right\}.$$

Теперь по правилу **ЗП**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{\overline{z}}_{\overline{u}}\right\} y = (e)^{\overline{z}}_{\overline{u}}; \left\{\exists x \left(\exists y \left((\psi)^{\overline{z}}_{\overline{u}}\right) \wedge (\psi)^{x\overline{z}}_{y\overline{u}}\right)\right\}.$$

Так как $(\psi)^{x\overline{z}}_{y\overline{u}}$ не содержит x, то по правилу **ОП**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{\overline{z}}_{\overline{u}}\right\} y = (e)^{\overline{z}}_{\overline{u}}; \left\{(\psi)^{x\overline{z}}_{y\overline{u}}\right\}.$$

Далее **АКС**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{x\overline{z}}_{y\overline{u}}\right\} x = y; \left\{(\varphi)^{\overline{z}}_{\overline{u}}\right\}.$$

По правилу **ЗП**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{x\overline{z}}_{y\overline{u}}\right\} y = (e)^{x\overline{z}}_{y\overline{u}}; \left\{\exists x \left(\exists y \left((\varphi)^{\overline{z}}_{\overline{u}}\right) \wedge (\psi)^{x\overline{z}}_{y\overline{u}}\right)\right\}.$$

И снова по правилу **ОП**:

$$\underset{\mathfrak{IH}}{\rhd} \left\{(\varphi)^{x\overline{z}}_{y\overline{u}}\right\} y = (e)^{x\overline{z}}_{y\overline{u}}; \left\{(\psi)^{x\overline{z}}_{y\overline{u}}\right\}.$$

Теперь осталось только $\overline{u}$ заменить на $\overline{e}$. Это делается точно так же, как $\overline{z}$ на $\overline{u}$. □

*__Задача 6.25.__ Завершите доказательство допустимости правила **ЗАМ**.

```
Alg mcc1;
arg x, y, z;
  if z mod x = 0 then
    if z mod y = 0 then
      mcc1 = z;
    else
      mcc1 = mcc1(x, y, z + 1);
    end;
  else
    mcc1 = mcc1(x, y, z + 1);
  end;
end;
```

Π_1

```
Alg mcc;
arg x, y;
  mcc = mcc1(x, y, 1);
end;
```

Рис. 6.9: Наименьшее общее кратное.

Пример 6.19. *Рассмотрим пример — подпрограмму для вычисления наименьшего общего кратного, см. рис. 6.9.*

1. *Предположим, что*

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}(x,y) \wedge xy - z < u\}\\ m = mcc_1(x,y,z);\\ \{m = \mathrm{HOK}(x,y)\}\end{array}\right|$$

2. **ЗАМ** : 1.

$$\left|\begin{array}{c}\{0 < z+1 \le \mathrm{HOK}(x,y) \wedge xy - (z+1) < u\}\\ mcc_1 = mcc_1(x,y,z+1);\\ \{mcc_1 = \mathrm{HOK}(x,y)\}\end{array}\right|$$

3. **АКС**. $\{z = \mathrm{HOK}(x,y)\}\, mcc_1 = z; \{mcc_1 = \mathrm{HOK}(x,y)\}$

4. **УПР** : 3.

$$0 < z \le \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y = 0 \wedge xy - z = u \rightarrow$$

$$\rightarrow z = \mathrm{HOK}(x,y)$$

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}(x,y) \wedge z \bmod x = 0 \wedge z \bmod y = 0 \wedge xy - z = u\}\\ mcc_1 = z;\\ \{mcc_1 = \mathrm{HOK}(x,y)\}\end{array}\right|$$

5. **УПР** : 2.

$$0 < z \le \mathrm{HOK}\,(x,y) \wedge z \bmod x = 0 \wedge \neg z \bmod y = 0 \wedge xy - z = u \rightarrow$$

$$\rightarrow 0 < z+1 \le \mathrm{HOK}\,(x,y) \wedge xy - (z+1) < u$$

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}\,(x,y) \wedge z \bmod x = 0 \wedge \neg z \bmod y = 0 \wedge xy - z = u\} \\ mcc_1 = mcc_1\,(x,y,z+1)\,; \\ \{mcc_1 = \mathrm{HOK}\,(x,y)\}\end{array}\right|$$

6. **ПВ** : 4, 5.

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}\,(x,y) \wedge z \bmod x = 0 \wedge xy - z = u\} \\ \Pi_1 \\ \{mcc_1 = \mathrm{HOK}\,(x,y)\}\end{array}\right|$$

7. **УПР** : 2.

$$0 < z \le \mathrm{HOK}\,(x,y) \wedge \neg z \bmod x = 0 \wedge xy - z = u \rightarrow$$

$$\rightarrow 0 < z+1 \le \mathrm{HOK}\,(x,y) \wedge xy - (z+1) < u$$

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}\,(x,y) \wedge \neg z \bmod x = 0 \wedge xy - z = u\} \\ mcc_1 = mcc_1\,(x,y,z+1)\,; \\ \{mcc_1 = \mathrm{HOK}\,(x,y)\}\end{array}\right|$$

8. **ПВ** : 6, 7.

$$\left|\begin{array}{c}\{0 < z \le \mathrm{HOK}\,(x,y) \wedge xy - z = u\} \\ \Pi_{mcc_1} \\ \{mcc_1 = \mathrm{HOK}\,(x,y)\}\end{array}\right|$$

9. **ВП*** : 1 – 8.

$$\{0 < z \le \mathrm{HOK}\,(x,y)\}\; m = mcc_1\,(x,y,z)\,; \{m = \mathrm{HOK}\,(x,y)\}$$

10. **ЗАМ** : 9.

$$\{0 < 1 \le \mathrm{HOK}\,(x,y)\}\; m = mcc_1\,(x,y,1)\,; \{m = \mathrm{HOK}\,(x,y)\}$$

11. **ВП*** : 1 – 10.

$$\{0 < 1 \le \mathrm{HOK}\,(x,y)\}\; m = mcc\,(x,y)\,; \{m = \mathrm{HOK}\,(x,y)\}$$

12. **УПР** : 11.

$$x \ne 0 \wedge y \ne 0 \rightarrow 0 < 1 \le \mathrm{HOK}\,(x,y)$$

$$\{x \ne 0 \wedge y \ne 0\}\; m = mcc\,(x,y)\,; \{m = \mathrm{HOK}\,(x,y)\}$$

Пример 6.20. *Рассмотрим подпрограмму, вычисляющую факториал, см. рис.* 6.10.

```
Sub F;
arg x;
  if x < 2 then
    F = 1;
  else
    F = F(x - 1) × x;
  end;
end;
```

Рис. 6.10: Факториал.

1. *Предположим, что*

$$\{\text{ИСТИНА}\}\, y = F(x)\,; \{y = x!\}$$

2. **ПРИСВ***. $\{x < 2\}\, F = 1; \{x < 2 \wedge F = 1\}$
3. **ОП** : 2. $\{x < 2\}\, F = 1; \{F = x!\}$
4. **ЗАМ** : 1. $\{\text{ИСТИНА}\}\, y = F(x-1)\,; \{y = (x-1)!\}$
5. **ПРИСВ***. $\{y = (x-1)!\}\, F = y \times x; \{y = (x-1)! \wedge F = yx\}$
6. **ОП** : 5. $\{y = (x-1)!\}\, F = y \times x; \{F = x!\}$
7. **ЗП** : 6, 4.

$$\left| \begin{array}{c} \{\text{ИСТИНА}\} \\ F = F(x-1) \times x; \\ \{\exists y\, (\exists F\, (y = (x-1)!) \wedge F = x!)\} \end{array} \right|$$

8. **ОП** : 7. $\{\text{ИСТИНА}\}\, F = F(x-1) \times x; \{F = x!\}$
9. **УПР** : 8. $\{\neg x < 2\}\, F = F(x-1) \times x; \{F = x!\}$
10. **ПВ** : 2, 8. $\{\text{ИСТИНА}\}\, \Pi_F\, \{F = x!\}$
11. **ВП*** : 1 – 10. *Ограничитель вызовов — x, следовательно, тройка*

$$\{\text{ИСТИНА}\}\, y = F(x)\,; \{y = x!\}$$

вполне корректна.

Пример 6.21. *Рассмотрим алгоритм Евклида на рис. 6.11.*

Чтобы каждый раз не переписывать $x > 0 \wedge y > 0$, мы предполагаем, что это всегда так.

```
Sub mcd;
arg x, y;
  if x = y then
    mcd = x;
  else
    if x < y then
      mcd = mcd(x, y − x);
    else
      mcd = mcd(x − y, y);
    end;
  end;
end;
```

Π_1

Рис. 6.11: Алгоритм Евклида.

1. Предположим, что

$$\{\text{ИСТИНА}\}\, z = mcd\,(x, y)\,; \{z = \text{НОД}\,(x, y)\}\,.$$

2. **ПРИСВ*.** $\{x = y\}\, mcd = x; \{mcd = x \wedge x = y\}$

3. **ОП** : 2. $\{x = y\}\, mcd = x; \{mcd = \text{НОД}\,(x, y)\}$

4. **ЗАМ, ДОБ** : 1.

$$\left|\begin{array}{c} \{\neg x = y \wedge x < y\} \\ mcd = mcd\,(x, y - x)\,; \\ \{\neg x = y \wedge x < y \wedge mcd = \text{НОД}\,(x, y - x)\} \end{array}\right|$$

5. **ЗАМ, ДОБ** : 1.

$$\left|\begin{array}{c} \{\neg x = y \wedge \neg x < y\} \\ mcd = mcd\,(x - y, y)\,; \\ \{\neg x = y \wedge \neg x < y \wedge mcd = \text{НОД}\,(x - y, y)\} \end{array}\right|$$

6. **ОП** : 4.

$$\left|\begin{array}{c} \{\neg x = y \wedge x < y\} \\ mcd = mcd\,(x, y - x)\,; \\ \{mcd = \text{НОД}\,(x, y)\} \end{array}\right|$$

7. **ОП** : 5.
$$\left|\begin{array}{c}\{\neg x = y \wedge \neg x < y\}\\ mcd = mcd\,(x-y,y)\,;\\ \{mcd = \text{НОД}\,(x,y)\}\end{array}\right|$$
8. **ПВ** : 6, 7. $\{\neg x = y\}\,\Pi_1\,\{mcd = \text{НОД}\,(x,y)\}$.
9. **ПВ** : 3, 8. $\{\text{ИСТИНА}\}\,\Pi_{mcd}\,\{mcd = \text{НОД}\,(x,y)\}$.
10. **ВП*** : 1 – 9. *Ограничитель* — $|x-y|$, *следовательно*
$$\{\text{ИСТИНА}\}\,z = mcd\,(x,y)\,;\{z = \text{НОД}\,(x,y)\}\,.$$

§ 6.6. **Корректность правила ВП

Чтобы доказать непротиворечивость **ВП** введём новое понятие.

Определение 6.19 (Дерево троек-состояний)**.** *Пусть дано $\mathcal{D}$ — некоторое доказательство тройки t для программы Π и некоторое состояние σ этой программы. Деревом троек-состояний (ДТС) для доказательства $\mathcal{D}$ и состояния σ назовём дерево* $\mathbf{Ct}\,(\mathcal{D},\sigma)$, *каждой вершине которого приписана метка (s,τ), где s — некоторый элемент доказательства $\mathcal{D}$, τ — некоторое состояние. Корнем дерева будет (t,σ). Для каждой вершины (s,τ) должны выполняться следующие свойства:*

1) *Каждая аксиома доказательства, которая не используется в правилах* **ВП**, *является листом.*

2) *Если тройка s получена из троек*
$$s_1 \ldots \{\ldots\}\,\Pi_1\,\{\ldots\}$$
и s_2 соответственно по правилу **СЛ** *или* **ЗП** *(для присваиваний), то сыновьями (s,τ) будут (s_1,τ) и $(s_2,\Pi_1\,(\tau))$. Если $\Pi_1\,(\tau)$ не определено, то второй сын отсутствует.*

3) *Если*
$$s \ldots \{\ldots\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{end};\{\ldots\}$$
получена из s_1 по правилу **НВ**, *то единственным сыном (s,τ) будет (s_1,τ), если $\tau \models T$. Иначе вершина не имеет сыновей.*

4) *Если*
$$s \ldots \{\ldots\}\,\mathsf{if}\ T\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2\ \mathsf{end};\{\ldots\}$$
получено из s_1 и s_2 по правилу **ПВ**, *то сыном (s,τ) будет (s_1,τ), если $\tau \models T$, и (s_2,τ) в противном случае.*

5) *Если*

$$s \circ\!\!- \{\dots\}\,\mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};\{\dots\}$$

получено из s_1 *по правилу* **ЦП** *или* **ЦЧ**, *то в соответствии с определением семантики цикла имеем последовательность состояний* $\left(\tau^i\right)_i$ *конечную или бесконечную. Тогда сыновьями* (s,τ) *будут те* $\left(s_1,\tau^i\right)$, *для которых* $\tau^i \models T$.

6) *Пусть*

$$s \circ\!\!- \{\dots\}\,x = f\,(z_1,\dots,z_n)\,;\{\dots\}$$

по правилу **ВП** *или же* s *является аксиомой, из которой в дальнейшем доказывается тройка*

$$s_1 \circ\!\!- \{\dots\}\,\Pi_f\,\{\dots\}$$

(для последующего использования правила **ВП**). *Тогда сыном* (s,τ) *будет* $(s_1,\sigma_\delta \cup \hat{\tau})$ *где* δ — *вызов* $\delta = (f,\tau z_1,\dots,\tau z_n)$. $\hat{\tau}$ *получено заменой в* τ *всех переменных* u *новыми переменными* $\hat{u}$.

7) *В остальных случаях, если* s *получено из* s_1, *то сыном* (s,τ) *будет* (s_1,τ).

Следствие 6.7. *Каждое поддерево ДТС является ДТС.*

Лемма 6.10 (О конечных ДТС)**.** *Пусть* $\sigma \models \varphi$ *и* $\mathcal{D}$ — *доказательство тройки* $\{\varphi\}\,\Pi\,\{\psi\}$. *Пусть ДТС* $\mathbf{Ct}\,(\mathcal{D},\sigma)$ *конечно. Тогда для всякой вершины дерева*

$$(\{\theta\}\,\Pi_1\,\{\chi\}\,,\tau)$$

выполнено $\tau \models \theta$, $\Pi_1\,(\tau)$ *определено и* $\Pi_1\,(\tau) \models \chi$.

Доказательство. Индукция по высоте дерева.

Если дерево состоит из одной вершины, то эта вершина — (a,σ), где a — аксиома **АКС**. В этом случае утверждение следует из полной корректности аксиом **АКС**.

Пусть для всех ДТС высоты меньшей h утверждение доказано. Доказательство индукционного шага, следовательно, состоит в разборе случаев применения правил вывода для корня. Рассмотрим несколько из них.

ЦЧ Пусть корень:

$$\left(\{\varphi\}\underbrace{\mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};}_{\Pi}\{\varphi\wedge\neg T\}\,,\sigma\right).$$

Сыновьями корня будут вершины вида

$$\left(\{\varphi \wedge T\}\, \Pi_1\, \{\varphi\}\, , \sigma^i\right).$$

где $\left(\sigma^i\right)_i$ — последовательность состояний для цикла. Так как дерево конечно, то эта последовательность конечна, допустим, она содержит n элементов.

Докажем, что $\sigma^i \models \varphi$ для всех $i \geq n$. В самом деле, пусть для $j < i$ утверждение выполнено. Тогда $\sigma^{i-1} \models \varphi \wedge T$. По индукционному предположению, имеем: $\Pi_1\left(\sigma^{i-1}\right)$ определено и $\Pi_1\left(\sigma^{i-1}\right) \models \varphi$. Но $\Pi_1\left(\sigma^{i-1}\right) = \sigma^i$.

Тогда $\Pi(\sigma)$ определено и $\Pi(\sigma) = \sigma^n$. Если бы было $\sigma^n \models T$, то корень имел бы не n, а $n+1$ сына. Следовательно, $\sigma^n \models \varphi \wedge \neg T$.

ВП Пусть корень

$$\left(\{\varphi\} \underbrace{x = f\left(\overline{z}\right);}_{\Pi} \left\{(\psi)_x^f\right\}, \sigma\right).$$

Сыном корня будет вершина

$$\left(\{\varphi'\}\, \Pi_f\, \{\psi'\}\, , \sigma_\delta \cup \hat{\sigma}\right).$$

Напомним, что

$$\varphi' \twoheadleftarrow \hat{\varphi} \wedge z_1 = \hat{z}_1 \wedge \cdots \wedge z_n = \hat{z}_n,$$
$$\delta = \left(f, \sigma z_1, \ldots, \sigma z_n\right),$$

Очевидно, что $\sigma_\delta \cup \hat{\sigma} \models \varphi'$. По индукционному предположению $\Pi_f\left(\sigma_\delta \cup \hat{\sigma}\right)$ определено и $\Pi_f\left(\sigma_\delta \cup \hat{\sigma}\right) \models \psi'$. Следовательно, $\Pi(\sigma)$ определено. Так как все переменные с «крышками» не входят в Π_f, то

$$\Pi_f\left(\sigma_\delta \cup \hat{\sigma}\right)(f) = \Pi_f\left(\sigma_\delta\right)(f) = \sigma\left(f\left(\overline{z}\right)\right).$$

Следовательно, $\Pi(\sigma) \models (\psi)_x^f$.

СЛ Пусть корень:

$$\left(\{\varphi\} \underbrace{\Pi_1 \Pi_2}_{\Pi} \{\psi\}\, , \sigma\right).$$

Сыновьями корня будут вершины

$$\left(\{\varphi\}\, \Pi_1\, \{\theta\}\, , \sigma\right),$$

$$(\{\theta\}\,\Pi_2\,\{\psi\},\Pi_1\,(\sigma))\,.$$

Вторая вершина существует, так как $\sigma \models \varphi$ и по индукционному предположению $\Pi_1\,(\sigma)$ определено и $\Pi_1\,(\sigma) \models \theta$. По индукционному предположению $\Pi_2\,(\Pi_1\,(\sigma))$ определено и $\Pi_2\,(\Pi_1\,(\sigma)) \models \psi$. Но $\Pi\,(\sigma) = \Pi_2\,(\Pi_1\,(\sigma))$, следовательно, $\Pi\,(\sigma)$ определено и $\Pi\,(\sigma) \models \psi$. □

Следствие 6.8. *Для выполнения условия $\tau \models \theta$ достаточно, чтобы конечной была часть дерева, лежащая левее указанной вершины.*

***Задача 6.26.** Докажите следствие.

***Задача 6.27.** Докажите лемму полностью.

Лемма 6.11 (О множествах вызовов)**.** *Для каждой вершины (s,τ) ДТС такой, что $s \Rightarrow \{\dots\}\,\Pi\,\{\dots\}$ получено по правилу отличному от* **ВП** *и $\Pi\,(\tau)$ определено, имеет место*

$$\mathbf{Fs}\,(\tau,\Pi) = \bigcup\{\mathbf{Fs}\,(\tau',\Pi') : (\{\dots\}\,\Pi'\,\{\dots\},\tau') \text{ — сын } (s,\tau)\}\,.$$

Доказательство. Рассмотрим, например, правило **ЗП**.

$$s \Rightarrow \{\varphi\}\,y = (e')_e^x\,;\{\exists x\,(\theta \wedge \exists y\psi)\}$$

получено из $\{\varphi\}\,x = e;\{\psi\}$ и $\{\psi\}\,y = e';\{\theta\}$. По лемме 6.1

$$\mathbf{Fs}\left(\tau, y = (e')_e^x\,;\right) = \mathbf{Fs}\left(\tau,(e')_e^x\right) = \mathbf{Fs}\,(\tau,e) \cup \mathbf{Fs}\,(\rho,e')$$

где $\rho = (x = e;)\,(\tau)$. Это в точности соответствует требуемому.

Рассмотрим то же правило для цикла. То есть

$$s \Rightarrow \{\varphi\}\underbrace{\mathsf{while}\ \ (T)_e^x\ \ \mathsf{do}\ \Pi_1\ \mathsf{end};}_{\Pi}\{\psi\}$$

получено из

$$\{\varphi\}\underbrace{x = e;\ \mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \ x\ \ =\ \ e;\ \mathsf{end};}_{\Pi_2}\{\psi\}$$

где x не входит в условия φ и ψ, программу Π_1 и входит в тест T.

По определению, имеем последовательности состояний: $\tau^0 = \tau$, $\tau^{i+1} = \Pi_1\,(\tau)$ — для первого цикла, и $\tau_2^0 = (x = e;)\,(\tau)$, $\tau_2^{i+1} = (\Pi_1 x = e;)\left(\tau_2^i\right)$ — для второго. Аналогично лемме 3.16 доказывается, что $\tau_2^i = (x = e;)\left(\tau^i\right)$.

Так как x не встречается в Π_1, то $\mathbf{Fs}\left(\tau^i,\Pi_1\right) = \mathbf{Fs}\left(\tau_2^i,\Pi_1\right)$. По следствию 6.1

$$\mathbf{Fs}\left(\tau^i,(T)_e^x\right) = \mathbf{Fs}\left(\tau^i,e\right) \cup \mathbf{Fs}\left((x = e;)\left(\tau^i\right),T\right).$$

По определению **Fs**:

$$\mathbf{Fs}\left(\tau,\Pi\right)=\mathbf{Fs}\left(\tau^0,(T)_e^x\right)\cup\mathbf{Fs}\left(\tau^0,\Pi_1\right)\cup\mathbf{Fs}\left(\tau^1,(T)_e^x\right)\cup\\ \cup\mathbf{Fs}\left(\tau^1,\Pi_1\right)\cup\mathbf{Fs}\left(\tau^1,(T)_e^x\right)\cup\ldots$$

$$\mathbf{Fs}\left(\tau,\Pi_2\right)=\mathbf{Fs}\left(\tau,x=e;\right)\cup\mathbf{Fs}\left(\tau_2^0,T\right)\cup\\ \cup\mathbf{Fs}\left(\tau_2^0,\Pi_1\right)\cup\mathbf{Fs}\left(\left(\Pi_1\right)\left(\tau_2^0\right),x=e;\right)\cup\mathbf{Fs}\left(\tau_2^1,T\right)\cup\\ \cup\mathbf{Fs}\left(\tau_2^1,\Pi_1\right)\cup\mathbf{Fs}\left(\left(\Pi_1\right)\left(\tau_2^1\right),x=e;\right)\cup\mathbf{Fs}\left(\tau_2^2,T\right)\cup\ldots$$

*__Задача 6.28.__ Докажите лемму для остальных случаев.

Лемма 6.12. *Если* $\Pi\left(\sigma\right)$ *определено для некоторой вершины*

$$\left(\{\cdots\}\,\Pi\,\{\cdots\},\sigma\right)$$

ДТС, то то же самое выполняется и для всех её сыновей.

*__Задача 6.29.__ Докажите лемму.

Лемма 6.13 (О конечности ДТС)**.** *Если* $\Pi\left(\sigma\right)$ *определено, то ДТС с корнем* $\left(\{\ldots\}\,\Pi\,\{\ldots\},\sigma\right)$ *конечно.*

Доказательство. Пусть $\Pi\left(\sigma\right)$ определено. Усечённым ДТС (УДТС) будем называть дерево, полученное по правилам ДТС 1–5 и 7. То есть листьями УДТС будут также вершины вида

$$\left(\left\{\theta\right\}x=f\left(\overline{z}\right);\left\{\left(\psi\right)_x^f\right\},\tau\right),$$

а, используя лемму 6.11, легко доказать, что $\mathbf{Fs}\left(\sigma,\Pi\right)$ состоит из всех элементов вида

$$\left(f,\tau z_1,\ldots,\tau z_n\right).$$

Заметим, что УДТС конечно.

Доказательство будем проводить индукцией по высоте леса $\mathbf{Ff}\left(\sigma,\Pi\right)$. Если он пуст (то есть вложенные вызовы отсутствуют), то ДТС совпадает с УДТС и конечно.

Пусть для всех лесов высоты меньше h доказано, а наш лес имеет высоту h. Будем рассматривать все листья УДТС, соответствующие вызовам подпрограмм. Рассмотрим вершину, лежащую в ДТС под

$$W=\left(\left\{\theta\right\}x=f\left(\overline{z}\right);\left\{\left(\psi\right)_x^f\right\},\tau\right).$$

Она имеет вид

$$\left(\left\{\theta'\right\}\Pi_f\left\{\psi'\right\},\hat{\tau}\cup\sigma_\delta\right).$$

Очевидно, что $\Pi_f\left(\hat{\tau}\cup\sigma_\delta\right)$ определено и лес $\mathbf{Ff}\left(\hat{\tau}\cup\sigma_\delta,\Pi_f\right)$ имеет меньшую высоту. По индукционному предположению ДТС ниже W конечно. А так как само УДТС конечно, то и ДТС должно быть конечным. □

*__Задача 6.30.__ Докажите, что если $\Pi(\sigma)$ определено, то УДТС конечно.

Лемма 6.14 (О корректности **ВП**). *С помощью правила* **ВП** *из доказательств частично (полностью) корректных троек можно получить только частично (полностью) корректные.*

Доказательство. Пусть $\sigma\models\varphi$ и тройка $\{\varphi\}\,\Pi\,\{\psi\}$ доказуема. Построим ДТС. Если речь идёт о частичной корректности, то дерево конечно из-за леммы 6.13.

Докажем, что для полной корректности дерево не может быть бесконечным. Предположим, что это не так. Тогда в дереве есть бесконечная ветвь или какая-то вершина имеет бесконечно много сыновей.

Допустим, что в дереве нет ни одной бесконечной ветви. Значит должны быть вершины с бесконечным количеством сыновей. Рассмотрим какую-либо вершину (s,τ), имеющую бесконечно много сыновей, каждый потомок которой — $\left(s_i,\tau^i\right)$ — имеет конечно много сыновей. Такие вершины должны существовать из-за отсутствия бесконечных ветвей. Бесконечно много сыновей может иметь только вершины с циклом. Следовательно,

$$s\circledcirc\{\varphi\}\,\mathsf{while}\ T\ \mathsf{do}\ \Pi_1\ \mathsf{end};\{\psi\}$$

Так как мы доказываем полную корректность, то используется правило **ЦП**. По лемме 6.10, $\tau\models\varphi$. Следовательно, существует ограничитель L цикла. Тогда $\tau^{i+1}(L)<\tau^i(L)$. Следовательно, $\tau^i(L)$ образуют убывающую последовательность натуральных чисел. По лемме об убывающих цепях мы получаем, что последовательность не может быть бесконечной. Противоречие. Следовательно, в дереве должны быть бесконечные ветви.

Итак, в дереве есть хотя бы одна бесконечная ветвь. Возьмём самую левую из них. Тогда для каждой вершины на этой ветви левее её лежит конечно много вершин. Так как само доказательство конечно, то некоторое использование правила **ВП** для подпрограммы g на этой ветви встречается бесконечно часто. Тройка $\left\{\varphi'\wedge\hat{L}=v\right\}\Pi_g\left\{\psi'\right\}$ доказывается по правилу **ВП** с помощью аксиомы $\left\{\varphi\wedge L<v\right\}y=g\left(\overline{u}\right);\left\{(\psi)_y^g\right\}$. По следствию 6.8, в каждой вершине выполняется предусловие. Переменная v — новая,

она не встречается в программе. Следовательно, если вершина

$$(\{\dots\}\, y = g\,(\overline{u})\,;\{\dots\}\,;\tau_1)$$

лежит под вершиной

$$(\{\dots\}\, y = g\,(\overline{u})\,;\{\dots\}\,;\tau_2)$$

то $\tau_1 v = \tau_2 v$ и $\tau_2\,(L) < \tau_1\,(L)$. По лемме об убывающих цепях, таких вершин может быть лишь конечно много. Значит, наше предположение неверно. Дерево должно быть конечным.

Итак, мы получили конечное ДТС. Его листьями являются вершина вида (a, τ), где a — аксиома из множества $\mathbf{Ax}$. Их корректность мы уже знаем. По лемме 6.10, для корня дерева (s, σ) тройка s является корректной на множестве $\{\sigma\}$. □

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 $\mathbf{S} : \omega \to \omega$ 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 $\mathbf{S}(x) \geq 1$ 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 $\{x \in \omega : \mathbf{S}(x) \leq \alpha\}$ является конечным.

3. Мы хотим, чтобы функция $\mathbf{S}$ была монотонной, то есть чем больше число, тем больше места требуется для его хранения. Это значит, что если $x > y$, то $\mathbf{S}(x) \geq \mathbf{S}(y)$.

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. Потребуем, чтобы для всякой такой функции f существовала константа $c \in \omega$, для которой было выполнено

$$\mathbf{S}(f(x,y)) \geq \mathbf{S}(x) + \mathbf{S}(y) - c$$

для любых x и y.

Рассмотрим пример. Пусть $\mathbf{S}(x) = [\log_k x] + 1$, $[x]$ — целая часть x. Эта функция описывает длину числа в его k-ичном представлении. Очевидно, что $\mathbf{S}$ положительна, монотонна, и множество $\{x \in \omega : [\log_2 x] + 1 \leq \alpha\}$ конечно для каждого $\alpha \in \omega$. Чтобы доказать последнее свойство, заметим, что для любой монотонной разнозначной нумерующей функции f выполнено $f(x,y) \geq xy + x + y$. Это следует из того, что для всяких x' и y' таких, что $x' \leq x$ и $y' \leq y$, должно быть $f(x',y') \leq f(x,y)$. 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

$$f(x,y) \geq (x+1)(y+1) - 1 = xy + x + y.$$

Итак,

$$\begin{aligned}\mathbf{S}(f(x,y)) = [\log_k f(x,y)] + 1 &\geq [\log_k (xy + x + y)] + 1 \geq \\ &\geq [\log_k xy] + 1 \geq [\log_k x] + [\log_k y] + 1 = \\ &= ([\log_k x] + 1) + ([\log_k y] + 1) - 1 = \mathbf{S}(x) + \mathbf{S}(y) - 1.\end{aligned}$$

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* $\mathbf{S}$*, удовлетворяющей условиям 1–4, существует константа* $k \in \omega$ *такая, что* $\mathbf{S}(x) \geq [\log_k x] + 1$ *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:

$$\mathbf{S}\left(f\left(x,y\right)\right) \geq \mathbf{S}\left(x\right) + \mathbf{S}\left(y\right) - c$$

для любых $x, y \in \omega$. Пусть $x = y$. Тогда

$$f\left(x,x\right) = \frac{2x\left(2x+1\right)}{2} + x = 2x^2 + 2x.$$

Итак,

$$\mathbf{S}\left(2x^2 + 2x\right) \geq 2\,\mathbf{S}\left(x\right) - c.$$

Выберем k настолько большим, чтобы

$$\mathbf{S}\left(k\right) \geq c + 6, \quad k^3 \geq 3k^2 \quad \text{и} \quad k^2 \geq 2k.$$

Следовательно, $\mathbf{S}\left(k\right) \geq \log_k k + c + 5$.

Если $k \leq x < 3k^2$, то

$$\mathbf{S}\left(x\right) \geq \log_k k + c + 5 \geq \log_k k + \log_k k^2 + c + 3 = \\ = \log_k k^3 + c + 3 \geq \log_k x + c + 3.$$

Пусть для $x \leq 3k^{2n}$ доказано, что

$$\mathbf{S}\left(x\right) \geq \log_k x + c + 3.$$

Отметим, что

$$\mathbf{S}\left(3k^{2n}\right) \geq 2\,\mathbf{S}\left(k^n\right) - c \geq 2\left(\log_k k^n + c + 3\right) - c = 2n + c + 6$$

Если $3k^{2n} \leq x < 3k^{2n+2}$, то

$$\mathbf{S}\left(x\right) \geq 2n + c + 6 = \log_k k^{2n+3} + c + 3 \geq \log_k x + c + 3.$$

Итак, для всех $x \geq k$ имеет место неравенство

$$\mathbf{S}\left(x\right) \geq \log_k x + c + 3 \geq \log_k + 1.$$

Для $x < k$ имеем:

$$\mathbf{S}\left(x\right) \geq 1 = \log_k x + 1.$$

□

***Задача 7.1.** Докажите, что функция

$$f\left(x,y\right) = \frac{\left(x+y\right)\left(x+y+1\right)}{2} + x$$

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 программы на состоянии σ. Будем называть вычислением последовательность состояний. Вычисление на состоянии σ будем обозначать $\mathbf{Cnt}(a, \sigma)$, где a — выражение, тест или программа. Вычисление определено, только если определено $\sigma(a)$ для выражений и тестов 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 является переменной σ.*

1. $e \twoheadrightarrow x$, *где* x — *переменная. Тогда* $\mathbf{Cnt}(e, \sigma) = \tau$, *где* $\tau = \sigma \cup \{(u_e, \sigma x)\}$.

2. $e \twoheadrightarrow c$, *где* c — *константа.* $\mathbf{Cnt}(e, \sigma) = \tau$, *где* $\tau = \sigma \cup \{(u_e, c)\}$.

3. $e \twoheadrightarrow o(e_1, \ldots, e_n)$, *где* o — *некоторая операция, а* e_1, …, e_n — *выражения. Пусть вычисление* $\mathbf{Cnt}(e_1, \sigma)$ *завершается состоянием* σ^1, *вычисление* $\mathbf{Cnt}\left(e_2, \sigma^1\right)$ *завершается состоянием* σ^2, *и т.д., вычисление* $\mathbf{Cnt}\left(e_n, \sigma^{n-1}\right)$ *завершается состоянием* σ^n. *Тогда вычисление* $\mathbf{Cnt}(e, \sigma)$ *это следующая последовательность:*

$$\mathbf{Cnt}(e_1, \sigma), \mathbf{Cnt}\left(e_2, \sigma^1\right), \ldots, \mathbf{Cnt}\left(e_n, \sigma^{n-1}\right), \tau,$$

где $\tau = \sigma \cup \{(u_e, o(\sigma^n u_{e_1}, \ldots, \sigma^n u_{e_n}))\}$.

4. $e \twoheadrightarrow f(e_1, \ldots, e_n)$, *где* f — *подпрограмма. Пусть вычисление* $\mathbf{Cnt}(e_1, \sigma)$ *завершается состоянием* σ^1, …, *вычисление* $\mathbf{Cnt}\left(e_n, \sigma^{n-1}\right)$ *завершается состоянием* 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 — *последнее состояние в* $\mathbf{Cnt}(\Pi_f, \sigma_\delta \cup \hat{\sigma})$. *Тогда вычисление* $\mathbf{Cnt}(e, \sigma)$ *это следующая последовательность:*

$$\mathbf{Cnt}(e_1, \sigma), \mathbf{Cnt}\left(e_2, \sigma^1\right), \ldots \\ \ldots, \mathbf{Cnt}\left(e_n, \sigma^{n-1}\right), \sigma_\delta \cup \hat{\sigma}, \mathbf{Cnt}(\Pi_f, \sigma_\delta \cup \hat{\sigma}), \tau,$$

где $\tau = \sigma \cup \{(u_e, \rho f)\}$.

Аналогично можно определить, что такое вычисление теста, если считать тест двухместной операцией, результатом которой является 1, если условие, задаваемое тестом, выполняется, и 0 в противном случае.

Определение 7.2 (Вычисление теста)**.**

$$\mathbf{Cnt}\left(e_1 \circ e_2, \sigma\right) = \mathbf{Cnt}\left(e_1, \sigma\right), \mathbf{Cnt}\left(e_2, \sigma^1\right), \tau,$$

где σ^1 *— последнее состояние в* $\mathbf{Cnt}\left(e_1, \sigma\right)$, σ^2 *— последнее состояние в* $\mathbf{Cnt}\left(e_2, \sigma^1\right)$, *а*

$$\tau = \sigma \cup \begin{cases} \{(u_T, 1)\}, & \text{если } \sigma^2\left(u_{e_1}\right) \circ \sigma^2\left(u_{e_2}\right), \\ \{(u_T, 0)\}, & \text{иначе.} \end{cases}$$

Определение 7.3 (Вычисление программы)**.** *Теперь определим, что такое вычисление программы* Π *на состоянии* σ*.*

1. $\Pi \twoheadleftarrow x = e$; *где* x *— переменная,* e *— выражение. Пусть* $\mathbf{Cnt}\left(e, \sigma\right)$ *заканчивается* ρ*. Тогда*

$$\mathbf{Cnt}\left(\Pi, \sigma\right) = \mathbf{Cnt}\left(e, \sigma\right), \tau,$$

где

$$\tau = \left(\sigma \setminus \{(x, \sigma x)\}\right) \cup \{(x, \rho u_e)\}.$$

2. $\Pi \twoheadleftarrow \Pi_1\Pi_2$*. Тогда, если* $\mathbf{Cnt}\left(\Pi_1, \sigma\right)$ *заканчивается* ρ*, то*

$$\mathbf{Cnt}\left(\Pi, \sigma\right) = \mathbf{Cnt}\left(\Pi_1, \sigma\right), \mathbf{Cnt}\left(\Pi_2, \rho\right).$$

3.

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end;} \end{cases}$$

Пусть $\mathbf{Cnt}\left(T, \sigma\right)$ *заканчивается* ρ*. Тогда*

$$\mathbf{Cnt}\left(\Pi, \sigma\right) = \begin{cases} \mathbf{Cnt}\left(T, \sigma\right), \mathbf{Cnt}\left(\Pi_1, \sigma\right), & \text{если } \rho u_T = 1; \\ \mathbf{Cnt}\left(T, \sigma\right), \sigma, & \text{иначе.} \end{cases}$$

4.

$$\Pi \twoheadleftarrow \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{else} \\ \quad \Pi_2 \\ \mathsf{end;} \end{cases}$$

Пусть $\mathbf{Cnt}(T,\sigma)$ *заканчивается* ρ. *Тогда*

$$\mathbf{Cnt}(\Pi,\sigma)=\begin{cases}\mathbf{Cnt}(T,\sigma),\mathbf{Cnt}(\Pi_1,\sigma), & \text{если } \rho u_T=1;\\ \mathbf{Cnt}(T,\sigma),\mathbf{Cnt}(\Pi_2,\sigma), & \text{иначе.}\end{cases}$$

5.

$$\Pi \Rightarrow \begin{cases}\mathsf{while}\ T\ \mathsf{do}\\ \quad \Pi_1\\ \mathsf{end;}\end{cases}$$

Пусть $\left(\sigma^i\right)_{i=0}^n$ *последовательность состояний такая, что* $\sigma^0=\sigma$, $\sigma^i\models T$ *тогда и только тогда, когда* $i<n$, $\sigma^{i+1}=\Pi_1\left(\sigma^i\right), i<n$. *Тогда вычислением* Π *на* σ *будет*

$$\mathbf{Cnt}\left(T,\sigma^0\right),\mathbf{Cnt}\left(\Pi_1,\sigma^0\right),\ldots \\ \ldots,\mathbf{Cnt}\left(T,\sigma^{n-1}\right),\mathbf{Cnt}\left(\Pi_1,\sigma^{n-1}\right),\mathbf{Cnt}\left(T,\sigma^n\right),\sigma^n.$$

Рассмотрим примеры.

Пример 7.1. *Возьмём программу:*

$$\Pi \Rightarrow \begin{cases}\mathsf{if}\ x\ <\ y\ \mathsf{then}\\ \quad z\ =\ y;\\ \mathsf{else}\\ \quad z\ =\ x;\\ \mathsf{end;}\end{cases}$$

и состояние $\sigma=\{(x,1),(y,2),(z,0)\}$.

1. $\mathbf{Cnt}(x,\sigma)=\{(x,1),(y,2),(z,0),(u_x,1)\}=\sigma^1$.
2. $\mathbf{Cnt}\left(y,\sigma^1\right)=\{(x,1),(y,2),(z,0),(u_x,1),(u_y,2)\}=\sigma^2$.
3. $\mathbf{Cnt}(x<y,\sigma)=\sigma^1,\sigma^2,\sigma^3$, *где*

$$\sigma^3=\{(x,1),(y,2),(z,0),(u_{x<y},1)\}.$$

4. $\mathbf{Cnt}(y,\sigma)=\{(x,1),(y,2),(z,0),(u_y,2)\}=\sigma^4$.
5. $\mathbf{Cnt}(z=y,\sigma)=\sigma^4,\sigma^5$, *где*

$$\sigma^5=\{(x,1),(y,2),(z,2)\}.$$

6. $\mathbf{Cnt}(\Pi,\sigma)=\sigma^1,\sigma^2,\sigma^3,\sigma^4,\sigma^5$.

Пример 7.2. *Рассмотрим другую программу:*

$$\left.\begin{array}{l} \left.\begin{array}{l} u \;=\; 0; \\ v \;=\; y; \end{array}\right\}\Pi_3 \\ \textsf{while } v \;<\; x \textsf{ do} \\ \quad \left.\begin{array}{l} u \;=\; \textsf{succ}\,(u)\,; \\ v \;=\; \textsf{succ}\,(v)\,; \end{array}\right\}\Pi_2 \\ \textsf{end;} \end{array}\right\}\Pi_1 \Bigg\}\Pi$$

и состояние $\sigma = \{(x,3),(y,2),(u,0),(v,0)\}$. *В дальнейшем* x *и* y *меняться не будут, поэтому мы будем опускать их в записи состояний.*

1. $\mathbf{Cnt}\,(0,\sigma) = \{(u,0),(v,0),(u_0,0)\} = \sigma^1$.
2. $\mathbf{Cnt}\,(u=0;,\sigma) = \sigma^1,\sigma^2$, *где*
$$\sigma^2 = \{(u,0),(v,0)\}.$$
3. $\mathbf{Cnt}\left(y,\sigma^2\right) = \{(u,0),(v,0),(u_y,2)\} = \sigma^3$.
4. $\mathbf{Cnt}\left(v=y;,\sigma^2\right) = \sigma^3,\sigma^4$, *где*
$$\sigma^4 = \{(u,0),(v,2)\}.$$
5. $\mathbf{Cnt}\,(\Pi_3,\sigma) = \sigma^1,\sigma^2,\sigma^3,\sigma^4$.
6. $\mathbf{Cnt}\left(v,\sigma^4\right) = \{(u,0),(v,2),(u_v,2)\} = \sigma^5$.
7. $\mathbf{Cnt}\left(x,\sigma^5\right) = \{(u,0),(v,2),(u_v,2),(u_x,3)\} = \sigma^6$.
8. $\mathbf{Cnt}\left(v<x,\sigma^4\right) = \sigma^5,\sigma^6,\sigma^7$, *где*
$$\sigma^7 = \{(u,0),(v,2),(u_{v<x},1)\}.$$
9. $\mathbf{Cnt}\left(u,\sigma^4\right) = \{(u,0),(v,2),(u_u,0)\} = \sigma^8$.
10. $\mathbf{Cnt}\left(succ\,(u),\sigma^4\right) = \sigma^8,\sigma^9$, *где*
$$\sigma^9 = \left\{(u,0),(v,2),\left(u_{succ(u)},1\right)\right\}.$$
11. $\mathbf{Cnt}\left(u=succ\,(u)\,;,\sigma^4\right) = \sigma^8,\sigma^9,\sigma^{10}$, *где*
$$\sigma^{10} = \{(u,1),(v,2)\}.$$
12. $\mathbf{Cnt}\left(v,\sigma^{10}\right) = \{(u,1),(v,2),(u_v,2)\} = \sigma^{11}$.
13. $\mathbf{Cnt}\left(succ\,(v),\sigma^{10}\right) = \sigma^{11},\sigma^{12}$, *где*
$$\sigma^{12} = \left\{(u,1),(v,2),\left(u_{succ(v)},3\right)\right\}.$$

14. $\mathbf{Cnt}\left(v = succ\left(v\right);, \sigma^{10}\right) = \sigma^{11}, \sigma^{12}, \sigma^{13}$, где

$$\sigma^{13} = \left\{\left(u, 1\right), \left(v, 3\right)\right\}.$$

15. $\mathbf{Cnt}\left(\Pi_2, \sigma^4\right) = \sigma^8, \sigma^9, \sigma^{10}, \sigma^{11}, \sigma^{12}, \sigma^{13}$.
16. $\mathbf{Cnt}\left(v, \sigma^{13}\right) = \left\{\left(u, 1\right), \left(v, 3\right), \left(u_v, 3\right)\right\} = \sigma^{14}$.
17. $\mathbf{Cnt}\left(x, \sigma^{14}\right) = \left\{\left(u, 1\right), \left(v, 3\right), \left(u_v, 3\right), \left(u_x, 3\right)\right\} = \sigma^{15}$.
18. $\mathbf{Cnt}\left(v < x, \sigma^{13}\right) = \sigma^{14}, \sigma^{15}, \sigma^{16}$, где

$$\sigma^{16} = \left\{\left(u, 1\right), \left(v, 3\right), \left(u_{v<x}, 0\right)\right\}.$$

19. $\mathbf{Cnt}\left(\Pi_1, \sigma^4\right) = \sigma^5, \sigma^6, \ldots, \sigma^{16}, \sigma^{13}$.
20. $\mathbf{Cnt}\left(\Pi, \sigma\right) = \sigma^1, \ldots, \sigma^{16}, \sigma^{13}$.

Задача 7.3. Постройте вычисление для алгоритма *Euclid* на начальном состоянии вызова $\left(Euclid, 3, 1\right)$, см. рис. 6.4.

Лемма 7.2 (Лемма о вычислении выражений)**.** *Пусть e — выражение, σ — состояние и $\sigma\left(e\right)$ определено. Пусть ρ — последнее состояние в* $\mathbf{Cnt}\left(e, \sigma\right)$. *Тогда* $\rho = \sigma \cup \left\{\left(u_e, \sigma\left(e\right)\right)\right\}$.

Доказательство. Индукция по построению выражения e.

Базис индукции.

Для переменных и констант утверждение, очевидно, следует из определения вычисления.

Шаг индукции.

Пусть $e \twoheadleftarrow o\left(e_1, \ldots, e_n\right)$. Тогда по индукции, $\mathbf{Cnt}\left(e_1, \sigma\right)$ заканчивается σ^1 и $\sigma^1\left(u_{e_1}\right) = \sigma\left(e_1\right)$. $\mathbf{Cnt}\left(e_2, \sigma^1\right)$ заканчивается σ^2, $\sigma^2\left(u_{e_2}\right) = \sigma^1\left(e_2\right) = \sigma\left(e_2\right)$ и $\sigma^2 u_{e_1} = \sigma^1 u_{e_1} = \sigma\left(e_1\right)$. В конце получаем: $\sigma^n\left(e_i\right) = \sigma u_{e_i}, i = 1, \ldots, n$. Но тогда по определению $\mathbf{Cnt}\left(e, \sigma\right)$ заканчивается τ и

$$\tau u_e = o\left(\sigma^n u_{e_1}, \ldots, \sigma^n u_{e_n}\right) = \\ = o\left(\sigma\left(e_1\right), \ldots, \sigma\left(e_n\right)\right) = \sigma\left(o\left(e_1, \ldots, e_n\right)\right).$$

***Задача 7.4.** Обоснуйте индукционный шаг для выражений вида $f\left(\overline{e}\right)$, где f — имя подпрограммы, $\overline{e}$ — выражения.

***Задача 7.5.** Сформулируйте и докажите аналогичное утверждение для тестов.

Лемма 7.3 (Лемма о вычислении программ)**.** *Пусть Π — программа, σ — состояние и $\Pi\left(\sigma\right)$ определено. Пусть ρ — последнее состояние в* $\mathbf{Cnt}\left(\Pi, \sigma\right)$. *Тогда* $\rho = \Pi\left(\sigma\right)$.

Доказательство. Индукция по построению программы Π.
Базис индукции.
Для оператора присваивания $\Pi \equiv x = e$; по определению:

$$\mathbf{Cnt}\left(\Pi, \sigma\right) = \mathbf{Cnt}\left(e, \sigma\right), \tau$$

и $\tau x = \rho u_e$, где ρ — последнее состояние $\mathbf{Cnt}\left(e, \sigma\right)$. По предыдущему утверждению, $\rho u_e = \sigma\left(e\right)$. Следовательно, $\tau x = \sigma\left(e\right)$ и $\tau y = \sigma y$ для остальных переменных. Это точно соответствует $\Pi\left(\sigma\right)$.
Шаг индукции.

1.
$$\Pi \equiv \begin{cases} \mathsf{if}\ T\ \mathsf{then} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

Пусть $\mathbf{Cnt}\left(T, \sigma\right)$ заканчивается ρ. Получаем, что $\sigma \models T$ тогда и только тогда, когда $\rho u_T = 1$.

$$\mathbf{Cnt}\left(\Pi, \sigma\right) = \begin{cases} \mathbf{Cnt}\left(T, \sigma\right), \mathbf{Cnt}\left(\Pi_1, \sigma\right), & \text{если } \rho u_T = 1; \\ \mathbf{Cnt}\left(T, \sigma\right), \sigma, & \text{иначе.} \end{cases}$$

По индукции, $\mathbf{Cnt}\left(\Pi_1, \sigma\right)$ заканчивается $\Pi_1\left(\sigma\right)$. Поэтому $\mathbf{Cnt}\left(\Pi, \sigma\right)$ заканчивается $\Pi_1\left(\sigma\right)$, если $\sigma \models T$, и σ в противном случае. То есть в любом случае $\mathbf{Cnt}\left(\Pi, \sigma\right)$ заканчивается $\Pi\left(\sigma\right)$.

2.
$$\Pi \equiv \begin{cases} \mathsf{while}\ T\ \mathsf{do} \\ \quad \Pi_1 \\ \mathsf{end}; \end{cases}$$

Пусть $\left(\sigma^i\right)_{i=0}^{n}$ последовательность состояний такая, что

$$\sigma^0 = \sigma, \quad \sigma^i \models T \iff i < n, \quad \sigma^{i+1} = \Pi_1\left(\sigma^i\right).$$

Тогда вычислением Π на σ будет

$$\mathbf{Cnt}\left(T, \sigma^0\right), \mathbf{Cnt}\left(\Pi_1, \sigma^0\right), \ldots \\ \ldots, \mathbf{Cnt}\left(T, \sigma^{n-1}\right), \mathbf{Cnt}\left(\Pi_1, \sigma^{n-1}\right), \mathbf{Cnt}\left(T, \sigma^n\right), \sigma^n.$$

Но, по определению, $\Pi\left(\sigma\right) = \sigma^n$.

□

*__Задача 7.6.__ Обоснуйте индукционный шаг для следования и полного ветвления.

Итак, мы доказали, что вычисление — один из способов задания семантики программ.

**__Задача 7.7.__ 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 7.4 (Время вычисления)**.** *Временем вычисления программы* Π *на состоянии* σ — $\mathbf{Tm}(\Pi, \sigma)$ — *будем называть длину последовательности* $\mathbf{Cnt}(\Pi, \sigma)$.

Определение 7.5 (Память состояния)**.** *Памятью состояния* σ — $\mathbf{S}(\sigma)$ — *будем называть величину*

$$\mathbf{S}(\sigma) = \sum_{x \in \operatorname{dom} \sigma} \mathbf{S}(\sigma x),$$

где $\mathbf{S}$ — *описывает объём памяти для натуральных чисел.*

Определение 7.6 (Память вычисления)**.** *Памятью вычисления программы* Π *на состоянии* σ — $\mathbf{Sp}(\Pi, \sigma)$ — *будем называть величину*

$$\mathbf{Sp}(\Pi, \sigma) = \max\left\{\mathbf{S}\left(\sigma^1\right), \ldots, \mathbf{S}\left(\sigma^n\right)\right\},$$

где $\sigma^1, \ldots, \sigma^n$ — *вычисление* Π *на* σ.

Предположим, что $\mathbf{S}(x) = [\log_2 x] + 1$. Тогда, в приведённых выше примерах мы получим следующие результаты.

Пример 7.3. *См. пример 7.1.*

$$\mathbf{Tm}(\Pi, \sigma) = 5; \quad \mathbf{Sp}(\Pi, \sigma) = \mathbf{S}\left(\sigma^2\right) = 7.$$

Рассмотрим какое-нибудь другое начальное состояние τ. *Тогда* $\mathbf{Tm}(\Pi, \tau) = 5$ *по-прежнему.* $\mathbf{Sp}(\Pi, \tau)$ *будет равно*

$$\mathbf{S}\left(\sigma^2\right) = 2\,\mathbf{S}(\tau x) + 2\,\mathbf{S}(\tau y) + \mathbf{S}(\tau z).$$

Например, если

$$\tau = \{(x, 100); (y, 200)\},$$

то

$$\mathbf{Sp}(\Pi, \tau) = 2\,\mathbf{S}(100) + 2\,\mathbf{S}(200) + \mathbf{S}(0) = 2(6+1) + 2(7+1) + 1 = 31.$$

Пример 7.4. *См. пример 7.2.*

$$\mathbf{Tm}\,(\Pi,\sigma)=17;\quad \mathbf{Sp}\,(\Pi,\sigma)=\mathbf{S}\left(\sigma^{15}\right)=11.$$

Для этой программы, очевидно, при изменении входного состояния будет меняться не только **Sp**, *но и* **Tm**. *Последовательность состояний* $\sigma^5,\ldots,\sigma^{13}$ *будет повторена не один раз, а* $\tau x-\tau y$. *Следовательно, время работы можно определить по следующей формуле:*

$$\mathbf{Tm}\,(\Pi,\tau)=8+9\lfloor\tau x-\tau y\rfloor.$$

Максимальная память состояния в вычислении в любом случае будет приходиться на σ^{N-2}, *где* $N=\mathbf{Tm}\,(\Pi,\tau)$. *Следовательно,*

$$\begin{aligned}\mathbf{Sp}\,(\Pi,\tau)=\mathbf{S}\,(\tau x)+\mathbf{S}\,(\tau y)+\mathbf{S}\,(\tau x-\tau y)+\mathbf{S}\,(\tau x)+\mathbf{S}\,(\tau x)+\mathbf{S}\,(\tau x)=\\ =4\,\mathbf{S}\,(\tau x)+\mathbf{S}\,(\tau y)+\mathbf{S}\,(\tau x-\tau y).\end{aligned}$$

В частности, при

$$\tau=\{(x,200);(y,100)\}$$

получим:

$$\mathbf{Tm}\,(\Pi,\tau)=908;\quad \mathbf{Sp}\,(\Pi,\tau)=4\cdot 8+7+7=46.$$

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 $T_o\,(\mathbf{S}\,(x_1),\ldots,\mathbf{S}\,(x_n))$.

Существует общая теория, которая изучает различные меры сложности вычислений. Время и память являются, конечно, наиболее полезными из мер сложности, но далеко не единственными. Вообще м е р а с л о ж н о с т и — 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 $\Phi\,(\Pi,\sigma)\leq\varphi$, является алгоритмически разрешимой.

Можно доказать, что определенные нами функции **Tm** и **Sp** удовлетворяют этому условию, поэтому являются мерами сложности.

Определение 7.7 (Длина входа)**.** *Если σ — начальное состояние, то $\mathbf{S}(\sigma)$ будем называть длиной входных данных.*

Обычно время и память вычисления рассматривают не как функции, зависящие от начального состояния программы, а как функции от длины входных данных. Однако, тогда возможна вот какая ситуация: могут существовать два состояния σ и τ такие, что $\mathbf{S}(\sigma) = \mathbf{S}(\tau)$, но в то же время $\mathbf{Tm}(\Pi, \sigma) \neq \mathbf{Tm}(\Pi, \tau)$ и $\mathbf{Sp}(\Pi, \sigma) \neq \mathbf{Sp}(\Pi, \tau)$. Для примера можно рассмотреть программу из примера 7.2 и состояния:

$$\tau = \{(x, 200), (y, 100)\};$$

$$\sigma = \{(x, 300), (y, 50)\}.$$

Очевидно, что

$$\mathbf{S}(\sigma) = 9 + 6 = \mathbf{S}(\tau) = 8 + 7 = 15.$$

В то же время

$$\mathbf{Tm}(\Pi, \tau) = 908; \quad \mathbf{Tm}(\Pi, \sigma) = 8 + 9 \cdot 250 = 2258;$$

$$\mathbf{Sp}(\Pi, \tau) = 46; \quad \mathbf{Sp}(\Pi, \sigma) = 4 \cdot 9 + 6 + 8 = 50.$$

Чтобы не возникало таких противоречий, введём дополнительные определения. Для простоты будем предполагать, что $\Pi(\sigma)$ определено для всех состояний σ.

Определение 7.8 (Максимальное время)**.** *Максимальным временем работы программы Π (временем работы в наихудшем случае) будем называть следующую функцию $\mathbf{Tm}_\Pi : \omega \to \omega$:*

$$\mathbf{Tm}_\Pi(n) = \max\{\mathbf{Tm}(\Pi, \sigma) : \mathbf{S}(\sigma) = n\}.$$

Заметим, что это определение корректно, так как множество, по которому берется максимум конечно по определению функции $\mathbf{S}$. 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 7.9 (Среднее время)**.** *Среднее время работы программы Π — это функция $\mathbf{Tm}^{cp}_\Pi : \omega \to \omega$, определённая так:*

$$\mathbf{Tm}^{cp}_\Pi(n) = \frac{1}{N} \sum_{\mathbf{S}(\sigma)=n} \mathbf{Tm}(\Pi, \sigma).$$

В данном определении через N обозначено количество состояний σ, для которых $\mathbf{S}(\sigma) = n$. Это определение корректно по той же причине. Среднее время определяет продолжительность работы программы в наиболее типичном случае.

Аналогично определяются максимальная память средняя память.

Определение 7.10 (Максимальная память)**.**

$$\mathbf{Sp}_\Pi(n) = \max\{\mathbf{Sp}(\Pi, \sigma) : \mathbf{S}(\sigma) = n\}.$$

Определение 7.11 (Средняя память)**.**

$$\mathbf{Sp}_\Pi^{cp}(n) = \frac{1}{N} \sum_{\mathbf{S}(\sigma)=n} \mathbf{Sp}(\Pi, \sigma).$$

Рассмотрим примеры.

Пример 7.5. *См. пример 7.1. Мы уже видели, что для нашей первой программы время работы не зависит от начального состояния τ, следовательно,*

$$\mathbf{Tm}_\Pi(n) = \mathbf{Tm}_\Pi^{cp}(n) = 5.$$

Используемая память равна

$$2\,\mathbf{S}(\tau x) + 2\,\mathbf{S}(\tau y) + 1 = 2\,(\mathbf{S}(\tau x) + \mathbf{S}(\tau y)) + 1 = 2n + 1,$$

поэтому

$$\mathbf{Sp}_\Pi(n) = \mathbf{Sp}_\Pi^{cp}(n) = 2n + 1.$$

Пример 7.6. *См. пример 7.2. Сначала найдём максимальные время и память.*

$$\begin{aligned}\mathbf{Tm}_\Pi(n) &= \max\{\mathbf{Tm}(\Pi, \tau) : \mathbf{S}(\tau) = n\} = \\ &= \max\{8 + 9\,(\tau x - \tau y) : [\log_2 x] + 1 + [\log_2 y] + 1 = n\}.\end{aligned}$$

Очевидно, что для достижения максимума нужно взять состояние τ такое, 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,

$$\mathbf{Tm}_\Pi(n) = 8 + 9\left(2^{n-1} - 1 - 0\right) = 4.5 \cdot 2^n - 1.$$

Память является максимальной при тех же условиях, поэтому

$$\mathbf{Sp}_\Pi(n) = 4\,(n-1) + 1 + n - 1 = 5n - 4.$$

Точные средние оценки в данном случае мы получать не будем ввиду их сложности. Получим приближённую оценку среднего времени. Очевидно, что

$$\mathbf{Tm}_\Pi^{cp}(n) = \frac{1}{N} \sum_{\mathbf{S}(\tau)=n} (8 + 9\lfloor \tau x - \tau y \rfloor) = 8 + \frac{9}{N} \sum_{\mathbf{S}(\tau)=n} \lfloor \tau x - \tau y \rfloor.$$

Чтобы найти последнюю сумму, заметим, что

$$\sum_{\mathbf{S}(\tau)=n} \lfloor \tau x - \tau y \rfloor = \sum_{\substack{\mathbf{S}(\tau)=n \\ \tau x \geq \tau y}} \tau x - \tau y \approx \sum_{\substack{\mathbf{S}(\tau)=n \\ \mathbf{S}(\tau x) \geq \mathbf{S}(\tau y)}} \tau x - \tau y.$$

Прежде, чем продолжать, выведем формулу для суммы всех чисел длины n:

$$\sum_{\mathbf{S}(x)=n} x = \sum_{x=2^{n-1}}^{2^n-1} x = 2^{n-1} \cdot 2^{n-1} + \sum_{x=0}^{2^{n-1}-1} x =$$

$$= 2^{2n-2} + \frac{2^{n-1}\left(2^{n-1}-1\right)}{2} = \frac{3}{2}2^{2n-2} - \frac{1}{2}2^{n-1}$$

Найдем нашу сумму, когда длины x и y зафиксированы: $\mathbf{S}(x) = i$, $\mathbf{S}(y) = n - i$.

$$\sum_{\mathbf{S}(x)=i;\mathbf{S}(y)=n-i} x - y = 2^{n-i-1} \sum_{\mathbf{S}(x)=i} x - 2^{i-1} \sum_{\mathbf{S}(y)=n-i} y =$$

$$= 2^{n-i-1}\left(\frac{3}{2}2^{2i-2} - \frac{1}{2}2^{i-1}\right) - 2^{i-1}\left(\frac{3}{2}2^{2n-2i-1} - \frac{1}{2}2^{n-i-1}\right) =$$

$$= \frac{3}{2}2^{n+i-3} - \frac{1}{2}2^{n-2} - \frac{3}{2}2^{2n-i-3} + \frac{1}{2}2^{n-2} = \frac{3}{2}2^{n-3}\left(2^i - 2^{n-i}\right)$$

Теперь просуммируем по всевозможным i от $n/2$ до $n-1$ (то есть, когда $\mathbf{S}(x) \geq \mathbf{S}(y)$):

$$\sum_{i=n/2}^{n-1} \frac{3}{2}2^{n-3}\left(2^i - 2^{n-i}\right) = \frac{3}{2}2^{n-3}\left(\sum_{i=n/2}^{n-1} 2^i - 2^n \sum_{i=n/2}^{n-1} 2^{-i}\right) =$$

$$= \frac{3}{2}2^{n-3}\left(2^{n/2}\left(2^{n/2}-1\right) - 2^n 2^{-n/2}\frac{1-2^{-n/2}}{1-2^{-1}}\right) =$$

$$= \frac{3}{2}2^{n-3}\left(2^n - 2^{n/2} - 2\left(2^{n/2}-1\right)\right) =$$

$$= \frac{3}{2}2^{n-3}\left(2^n - 3\cdot 2^{n/2} + 2\right) =$$

$$= \frac{3}{2}2^{2n-3}\left(1 - 3\cdot 2^{-n/2} + 2^{-n+1}\right) \approx \frac{3}{2}2^{2n-3}\left(1 - 3\cdot 2^{-n/2}\right).$$

К этой сумме необходимо прибавить слагаемое, описывающее случай $y = 0$:

$$\frac{3}{2}2^{2n-4} - \frac{1}{2}2^{n-2} = \frac{3}{2}2^{2n-3}\left(\frac{1}{2} - \frac{1}{3}2^{-n+1}\right) \approx \frac{3}{2}2^{2n-3}\cdot\frac{1}{2}.$$

Всего получается

$$\frac{3}{2}2^{2n-3}\left(1-3\cdot 2^{-n/2}\right)+\frac{3}{2}2^{2n-3}\cdot\frac{1}{2}=$$
$$=\frac{3}{2}2^{2n-3}\left(\frac{3}{2}-3\cdot 2^{-n/2}\right)=$$
$$=\frac{9}{8}2^{2n-2}\left(1-2^{-n/2+1}\right).$$

Таким образом, мы получили приближенное равенство:

$$\sum_{\mathbf{S}(\tau)=n}\lfloor\tau x-\tau y\rfloor\approx\frac{9}{8}2^{2n-2}\left(1-2^{-n/2+1}\right).$$

Теперь найдём общее количество начальных состояний длины n. Найдём количество z таких, что $\mathbf{S}(z)=i$. Имеем

$$[\log_2 z]+1=i,\quad 2^{i-1}\le\log_2 z\le 2^i-1.$$

Общее количество z, следовательно, равно 2^{i-1}. Если $\mathbf{S}(x)=i,\mathbf{S}(y)=n-i$, то количество таких состояний есть

$$2^{i-1}\cdot 2^{n-i-1}=2^{n-2}.$$

Поэтому общее количество состояний длины n есть $(n-1)\,2^{n-2}$. Прибавим еще случаи, когда $x=0$ или $y=0$, получим: $(n+1)\,2^{n-2}$. Отсюда мы можем получить приближенную оценку для среднего времени:

$$\mathbf{Tm}_{\Pi}^{cp}\approx 8+9\frac{\frac{9}{8}2^{2n-2}\left(1-2^{-n/2+1}\right)}{(n+1)\,2^{n-2}}=$$
$$=8+9\frac{9}{8\,(n+1)}2^n\left(1-2^{-n/2+1}\right)=$$
$$=8+\frac{81}{8\,(n+1)}\left(2^n-2^{n/2+1}\right).$$

Посмотрим, насколько точная оценка получилась. Пусть, например, $n=4$. Имеется 2 числа длины 1: 0 и 1, 2 числа длины 2: 2 и 3, и 4 числа длины 3: 4, 5, 6 и 7. Всего состояний τ таких, чтобы $\mathbf{S}(\tau x)+\mathbf{S}(\tau y)=4$, следовательно, имеется:

$$N=2\cdot 4+2\cdot 2+4\cdot 2=20.$$

Найдем теперь сумму

$$\sum_{\mathbf{S}(\tau)=4}\lfloor\tau x-\tau y\rfloor=4+5+6+7+3+4+5+6+1=41.$$

Поэтому точное среднее время для $n = 4$ *есть:*

$$\mathbf{Tm}_{\Pi}^{cp}(4) = 8 + 9 \cdot \frac{1}{20} \cdot 41 \approx 26.5.$$

Наша формула даёт:

$$\mathbf{Tm}_{\Pi}^{cp}(4) = 8 + \frac{81}{8 \cdot 5}(16 - 8) = 24.2.$$

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 7.8.** Найдите оценку для средней памяти.

Предметный указатель

Литература

[1] В.Н.Агафонов. Математические основы обработки информации. Новосибирск, Изд-во НГУ, 1982.

[2] Дж. Булос, Р.Джеффри. Вычислимость и логика. М. Мир, 1994.

[3] Н.Вирт. Алгоритмы и структуры данных. М. Мир, 1984.

[4] Н.И.Вьюкова, В.А.Галатенко, А.Б.Ходулев. Систематический подход к программированию. М. Наука, 1988.

[5] Д.Грис. Наука программирования. М. Мир, 1984.

[6] М.Гэри, Д.Джонсон. Вычислительные машины и труднорешаемые задачи. М.Мир, 1982.

[7] Д.Кнут. Исскуство программирования для ЭВМ (три тома). М. Мир, 1978.

[8] Б.Мейер, К.Бодуэн. Методы программирования (два тома). М. Мир, 1982.

[9] В.А.Непомнящий, О.М.Рякин. Прикладные методы верификации программ. М. Радио и связь, 1988.

[10] О.Оре. Теория графов. М. Наука, 1968.

[11] Б.А.Трахтенброт. Алгоритмы и вычислительные автоматы. М. Советское радио, 1974.

[12] Требования и спецификации в разработке программ. Сборник статей. М. Мир, 1984.

[13] А.Филд, П.Харрисон. Функциональное программирование. М. Мир, 1993.

Учебное издание

Дудаков Сергей Михайлович

Математическое введение в информатику

Учебное пособие

Отпечатано в авторской редакции
Подписано в печать 28.08.2003. Формат 60 × 84 1/16. Бумага типографская №1. Печать офсетная. Усл.п.л.14,0. Уч.-изд.л.9,85. Тираж 150 экз. Заказ 382.
Тверской государственный университет
Факультет прикладной математики и кибернетики
Адрес: 170000, г.Тверь, пер.Садовый, 35.