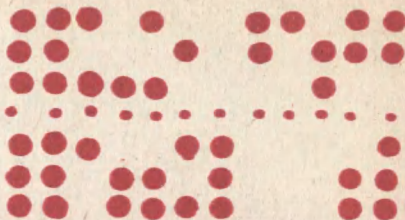


**БИБЛИОТЕЧКА  
ПРОГРАММИСТА**



С. А. АБРАМОВ

# Элементы анализа программ



БИБЛИОТЕЧКА  
ПРОГРАММИСТА

---

С. А. АБРАМОВ

# ЭЛЕМЕНТЫ АНАЛИЗА ПРОГРАММ

ЧАСТИЧНЫЕ ФУНКЦИИ  
НА МНОЖЕСТВЕ  
СОСТОЯНИЙ



МОСКВА «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
1986

ББК 22.18

A16

УДК 519.6

Абрамов С. А. Элементы анализа программ. Частичные функции на множестве состояний.— М.: Наука. Гл. ред. физ.-мат. лит., 1986.— 128 с.

В книге описываются различные подходы к установлению свойств программ и доказательству теорем о программах. Цель книги — дать единую теоретико-множественную схему: с одной стороны, наиболее распространенные традиционные методы, с другой стороны, новые методы анализа программ.

Для студентов и аспирантов математических специальностей, а также научных работников, интересующихся проблемами теоретического программирования.

Библиогр. 17 назв.

Рецензент

доктор физико-математических наук Д. Б. Подшивалов

*Сергей Александрович Абрамов*

ЭЛЕМЕНТЫ АНАЛИЗА ПРОГРАММ

Частичные функции  
на множестве состояний

Серия «Библиотечка программиста»

Редактор П. И. Воронина

Художественный редактор Т. Н. Кольченко

Технический редактор Е. В. Морозова

Корректоры Е. Ю. Рычагова, М. Г. Живоговская

ИБ № 12666

Сдано в набор 06.08.85. Подписано к печати 26.05.86. Т-11529 Формат 84×108/32. Бумага тип. № 2. Гарнитура обыкновенная. Печать высокая. Усл. печ. л. 6,72. Усл. кр.-отт. 6,93. Уч.-изд. л. 7,16. Тираж 21 000 экз. Заказ № 867. Цена 45 коп.

Ордена Трудового Красного Знамени издательство «Наука»

Главная редакция физико-математической литературы

117071 Москва В-71, Ленинский проспект, 15

4-я типография издательства «Наука»

630077 г. Новосибирск-77, Станиславского, 25

A  $\frac{1702070000-099}{053(02)-86}$  28-86

© Издательство «Наука». Главная редакция физико-математической литературы, 1986

## ПРЕДИСЛОВИЕ

Анализ программ как одно из направлений теории программирования берет начало от работ П. Наура и Р. Флойда по доказательству правильности (верификации) программ. Флойдом была высказана идея приписывания точке программы так называемого индуктивного, или промежуточного, утверждения и указана возможность доказательства частичной правильности программы (т. е. соответствия друг другу ее предусловия и постусловия; вопрос о завершимости при этом не рассматривается), построенного на установлении согласованности индуктивных утверждений. Затем Ч. Хоором были высказаны идеи проведения доказательства частичной правильности программы в виде вывода в некоторой логической системе. Э. Дейкстра ввел понятие слабейшего предусловия, позволяющее, в принципе, одновременно доказывать как соответствие друг другу предусловия и постусловия, так и завершимость. В дальнейшем идеи этих авторов легли в основу многочисленных исследований; преобладающим явился формально-логический подход.

Методы доказательства правильности программ принесли определенную пользу программированию. Неоднократно отмечалось, что эти методы указывают способ рассуждения о ходе выполнения программ, дают удобную систему комментирования программ и, главное, устанавливают взаимосвязи между конструкциями языков программирования и их семантикой. Указывалась ситуация, когда полный анализ программы должен проводиться вычислительной машиной — эта ситуация может возникать в процессе автоматизированного обучения языкам программирования. Однако нельзя не обратить внимания на то, что у программиста-практика требование проведения доказательства правильности написанной

программы вызывает сопротивление, так как он обычно располагает иного рода доводами в пользу достаточной надежности своей программы. Если принять более широкое толкование термина «анализ программ», подразумеваемая доказательство разнообразных свойств программ или доказательство теорем о программах, то ценность методов анализа станет более ясной. Программист может заинтересоваться такими свойствами своей программы (правильность которой он считает вполне обоснованной), о которых он совершенно не думал в ходе составления и отладки программы. Его может, например, заинтересовать вопрос о характере изменения заключительного значения некоторой числовой переменной при возрастании исходного значения другой числовой переменной (подобный пример разбирается в книге). Может быть замечено, что некоторая переменная, вспомогательная по замыслу, к концу выполнения программы принимает некоторое практически интересное значение. Может возникнуть очень важный вопрос о количестве операций, затрачиваемых при выполнении программы (отметим, что с точки зрения теории доказательства правильности программ этот вопрос не традиционен), и т. д. Методы анализа программ могут быть средством доказательства предположений, касающихся этих вопросов.

Какими могут быть доказательства теорем о программах? Формальное доказательство в виде вывода в некоторой логической системе вполне надежно, но, во-первых, логические системы иногда оказываются неполными: с этим уже столкнулись и в связи с доказательствами правильности программ — М. Вэнд показал, что система Хоора над языком первого порядка может быть неполной (последний результат развивается в книге). Во-вторых, формальные доказательства в виде выводов оказываются очень длинными и необозримыми. По поводу последнего обстоятельства в ряде исследований высказываются сходные мысли о том, что практические доказательства должны со временем стать достаточно короткими и обозримыми и приобрести сходство с публикуемыми доказательствами математических теорем (С. С. Лавров, Р. Андерсон, В. Турский и др.). Упростить же доказательства, сделав их менее формальными и более близкими математической традиции, позволяет интенсивное привлечение теоретико-множественных понятий и методов. Важные шаги в этом направлении сделаны денотационной семантикой.

В предлагаемой книге тоже развивается именно теоретико-множественный подход. Главной ее целью является, во-первых, разработка достаточно общих теоретико-множественных методов анализа программ и, во-вторых, изучение, с позиций предлагаемых методов, возможностей наиболее распространенных известных методов. Программа  $S$  описывает преобразование  $s$  (возможно частичное, т. е. определенное не всюду) множества состояний памяти  $V$  в себя, при этом под состоянием понимается набор значений переменных, участвующих в программе. Многие задачи анализа программы  $S$  — это различные вопросы, касающиеся устройства преобразования  $s$ . Для этих целей в математике часто используется прием, состоящий в рассмотрении на  $V$  множества функций со значениями в фиксированном множестве  $M$  и изучении преобразований, которые индуцируются в этом множестве функций преобразованием  $s$ . Когда множество  $M$ , а следовательно, и множество функций имеют удобную алгебраическую структуру, этот прием может дать хорошие результаты.

Оказывается, что этот общематематический подход к анализу преобразований множеств охватывает подходы Дейкстры и Хоора к анализу программ. Используя этот подход, можно построить новые, более общие методы анализа программ. Почти во всех разделах этой книги рассматриваются более сложные, чем предикаты, функциональные конструкции — частичные функции на множестве состояний, а также преобразования этих функций, различными способами индуцированные программой\*). Это дает новые семантики основных конструкций языков программирования, их можно назвать семантиками в функциональных множествах. Методы анализа, основанные на этих семантиках, позволяют доказывать такие свойства программ, которые иногда невозможно, если не прибегать к изменению программ, даже сформулировать в рамках систем Хоора и Дейкстры. Ряд же доказательств, которые могут быть проведены в этих рамках, упрощается.

---

\*) Выписывая индуцированные программами преобразования, мы, как правило, будем обосновывать их вид исходя из интуитивной операционной семантики программ. Во многих случаях можно было бы воспользоваться более четко определенной денотационной семантикой [5, 9] или семантикой Мак-Карти (семантикой операционных определений) [5].

В этом отношении примечателен, в частности, § 4 гл. I. Программам сопоставляются специальные преобразования частичных функций — операционные смещения. Должным образом описанные операционные смещения помогают оценивать затрачиваемое при выполнении программы число операций, или, например, исследовать последовательность выводимых с помощью инструкции *output* значений. В § 6 гл. I рассматриваются множество функций и операционное смещение, полезное для исследования неустранимой и вычислительной погрешностей описываемых программой вычислений. Исследуются последствия незавершающегося выполнения программ. Предлагается, таким образом, единый взгляд на многие внешне несхожие проблемы и дается инструмент для изучения таких последствий выполнения программ, которые не определяются одними только заключительными значениями переменных. Описание преобразований частичных функций, т. е. преобразований, задающих семантики в функциональных множествах, составляет главный результат гл. I. В последнем параграфе этой же главы приводится все необходимое для исследования в гл. II возможностей традиционного метода Хоора. В этих исследованиях, в частности, показывается, что логическая неполнота системы Хоора над языками первого порядка имеет более серьезный, чем принято считать, характер; устанавливается также различие между индуктивными утверждениями Флойда и инвариантами Хоора.

В гл. III рассматриваются специальные вопросы, касающиеся недетерминированных программ. Источником недетерминированности могут служить определенные Дейкстрой инструкции *if* и *do* (инструкция выбора и инструкция повторения)

$$\text{if } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{ fi,}$$

$$\text{do } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{ od,}$$

или, например, инструкция ввода *input*. Недетерминированность программ дает значительную свободу для определения понятия предусловия. В гл. III приведены разнообразные варианты этого понятия. Эти варианты предусловий предопределяют характер выполнения программы на всех или некоторых путях ее выполнения. Выбор варианта должен производиться в зависимости от содержательной постановки задачи анализа (в частности, нас, с одной стороны, могут интересовать ситуации,

которые обязательно возникают в ходе выполнения программы, а с другой стороны — ситуации, которые могут возникнуть, а могут и нет; традиционный метод Дейкстры не позволяет отличить последние от незавершимости). При этом оказывается, что некоторые задачи не могут решаться путем построения сильнейшего постусловия по данному предусловию, так как по самому смыслу этих задач сильнейшего постусловия не существует. В конце гл. III рассматривается задача анализа недетерминированных программ, выполнение которых носит вероятностный характер. Предлагается метод анализа в среднем, также основанный на идее введения некоторых индуцированных преобразований в множестве функций (числовых). Метод применяется для построения оптимального в среднем алгоритма одновременного нахождения наибольшего и наименьшего элементов в массиве чисел (сопоставление алгоритмам решения подобных задач некоторых модельных недетерминированных алгоритмов основано на идеях И. Пола). Решение конкретной задачи сортировки и поиска — единственный сложный и нетривиальный пример, приводимый в книге. В остальных примерах рассматриваются короткие программы. Но и в этих простых примерах, как правило, хорошо просматривается преодоление трудностей анализа благодаря предлагаемой технике.

Язык, на котором пишутся программы, включает присваивания, а также условные, составные и циклические инструкции. Этот упрощенный язык является моделью многих реальных языков программирования. В гл. III дополнительно подключаются, как уже говорилось, недетерминированные инструкции Дейкстры *if* и *do*. В некоторых случаях рассматриваются инструкции ввода и вывода. Программа — это инструкция выбранного языка, чаще всего составная. Поэтому без термина «программа», строго говоря, можно было бы обойтись. Однако этот термин указывает на смысловую законченность текста на выбранном языке программирования. В то же время термин «инструкция» употребляется, как правило, тогда, когда речь идет всего лишь о фрагменте настоящей программы.



## ГЛАВА I

# ПРОГРАММЫ И ИНДУЦИРУЕМЫЕ ИМИ ПРЕОБРАЗОВАНИЯ МНОЖЕСТВ ФУНКЦИЙ (ДЕТЕРМИНИРОВАННЫЙ СЛУЧАЙ)

В этой главе объясняется, в чем состоит сущность методов Хора и Дейкстры с точки зрения теорий множеств и функций, затем эти методы обобщаются за счет привлечения более сложных, чем предикаты, функциональных конструкций.

### § 1. Преобразования предикатов; сопряженные задачи

Многие виды семантики языков программирования сопоставляют каждой отдельной инструкции языка некоторое преобразование предикатов на множестве состояний памяти. Состояние памяти при этом рассматривается как набор значений переменных, участвующих в программе. (Далее будем употреблять для краткости термины «состояние» и «множество состояний». Множество состояний будет, как правило, обозначаться через  $V$ .) Предикаты на множестве состояний понимаются как логические функции на этом множестве. Если  $a, b, \dots, x$  суть участвующие в программе переменные и  $a, b$  принимают числовые значения, то, например, под  $a > b$  надо понимать предикат, принимающий значение И на тех и только тех состояниях, каждому из которых соответствуют такие значения переменных  $a$  и  $b$ , что значение  $a$  больше значения  $b$ .

Хорошо известная семантика Дейкстры основывается на понятии слабейшего предусловия\*). Если заданы

---

\*) Эта семантика, а также все результаты и выводы Дейкстры, которые упоминаются далее в тексте, изложены в [4].

программа  $P$  и являющееся предикатом на  $V$  постусловие  $\psi$ , то слабое предусловие для  $\psi$  относительно  $P$  — это предикат  $\text{wr}(P, \psi)$ , описывающий самое слабое условие, которому достаточно подчинить начальное состояние для того, чтобы выполнение программы  $P$  завершилось и дало заключительное состояние, удовлетворяющее  $\psi$ ; таким образом  $\text{wr}(P, \psi)$  выражает необходимое и достаточное условие того, что выполнение программы  $P$  завершится и приведет к состоянию, удовлетворяющему  $\psi$ .

Займемся подробнее выяснением математического смысла понятия слабого предусловия.

Программе  $P$  как тексту специального вида соответствует преобразование  $p$  множества  $V$  в себя, т. е.  $V \xrightarrow{p} V$  (в дальнейшем программы будут обозначаться большими буквами латинского алфавита, а одноименными малыми буквами будут обозначаться соответствующие им преобразования множества  $V$ ). Если зафиксировать  $P$ , то слабое предусловие  $\text{wr}(P, \psi)$  станет преобразованием множества предикатов на  $V$  в себя. Пусть, сначала, программа  $P$  такова, что  $p$  всюду определено. Тогда для любого  $v \in V$

$$\text{wr}(P, \psi)(v) = \psi(p(v)), \quad (1)$$

т. е.  $\text{wr}(P, \psi)$  определяется через  $p$  следующим образом:

$$\text{wr}(P, \psi) = p^*(\psi), \quad p^*(\psi)(v) = \psi(p(v)). \quad (2)$$

Определенное с помощью последней формулы преобразование  $p^*$  будем называть *преобразованием, сопряженным к  $p$*  (такое употребление этого термина не противоречит принятому в теории линейных пространств). Уточняя область значений тех функций, к которым применяется  $p^*$ , содержание формул (1), (2) можно охарактеризовать так: при фиксированной программе  $P$  преобразование  $\text{wr}(P, \psi)$  является сопряженным к  $p$  в множестве предикатов на  $V$ .

Для того чтобы сохранить в силе (1), (2) и отождествить понятие слабого предусловия с сопряженным преобразованием в случае частичного  $p$ , достаточно принять точку зрения на предикат (в тех случаях, когда он выступает в роли предусловия или постусловия) как на частичную функцию, принимающую значение И всюду, где она определена. Таким образом, значение Л заменяется неопределенностью.

Множества  $V^*$  и  $W^*$ , состоящие из функций, определенных на множествах  $V$  и  $W$  и принимающих значения в некотором фиксированном множестве  $M$ , преобразуются под действием преобразования  $f^*$ , сопряженного к  $V \xrightarrow{f} W$ , в обратную сторону:  $W^* \xrightarrow{f^*} V^*$ . Поэтому удобство семантики, основанной на слабейшем предусловии, т. е. на обратном преобразовании предикатов, объясняется естественными причинами. Итак, мы видим, что по своей сути понятие слабейшего предусловия не является исключительным достоянием дисциплины программирования.

Слабейшее предусловие — это основной инструмент для работы с программами, используемый Дейкстрой. Разбирая и комментируя примеры построения («вывода») программ, Дейкстра объясняет сущность своей техники построения в таком духе, что доказательство правильности программы строится одновременно с программой или даже несколько раньше программы: «...выбрав форму доказательства правильности, мы пишем программу так, чтобы она удовлетворяла требованиям доказательства». Это объяснение выглядит несколько таинственным. Прочитированные слова могут быть переведены на обычный математический язык. Выбор «формы доказательства» — это решение сопряженной в множестве предикатов задачи, а написание программы так, «чтобы она удовлетворяла требованиям доказательства», — это переход от решения сопряженной задачи к решению основной.

**Примечание 1.** Разбираемая фраза и в изложении Дейкстры, и в переводе на математический язык — это довольно сильная идеализация того, что имеется в действительности. На самом деле в книге Дейкстры демонстрируется смешанная техника, и обе задачи — сопряженная в множестве предикатов и основная — рассматриваются одновременно, т. е. привлекаются и интуитивные операционные соображения относительно того, как выполняются те или иные инструкции, и слабейшие предусловия. Такая техника представляется более полезной, нежели провозглашенная в прочитированной фразе. Рассмотрение сразу двух задач часто дает возможность, с одной стороны, избежать формальных сложностей, и, с другой стороны, лишний раз проконтролировать решение (получается нечто вроде счета с контролем при решении вычислительных задач).

Рассмотрим теперь прямое преобразование предикатов, т. е. семантику, основанную на сильнейшем постусловии, которое будем обозначать через  $sp(P, \varphi)$ . Такая семантика оказывается более сложной, чем wr-семанти-

ка. Описать  $\text{sp}(P, \varphi)$  с помощью простой формулы (суперпозиции) через  $p, p^*, \varphi$  не удастся. Если для некоторых  $v_1, v_2 \in V$  и предиката  $\varphi(v)$  имеет место  $\varphi(v_1) \neq \varphi(v_2)$ ,  $p(v_1) = p(v_2) = v_3$ , то для того, чтобы  $\text{sp}(P, \varphi)$  было сильнейшим постуловием, нужно считать  $\text{sp}(P, \varphi)(v_3) = \text{И}$ . Вопрос о том, чему равно значение  $\text{sp}(P, \varphi)(v)$ , когда  $\neg \varphi(v)$ , в вычислительном отношении решается существенно сложнее, чем для  $p^*(\varphi)(v)$ : надо проверить все состояния  $v'$ , для которых  $p(v) = p(v')$ , и выяснить, нет ли среди них такого, для которого  $\varphi(v') = \text{И}$ .

Семантика инструкции  $x := f(x)$  в общем случае достаточно сложна из-за возможных интерпретаций, при которых  $f$  необратима; эту семантику нельзя получить с помощью суперпозиции  $f, s, \varphi$ : для этого потребуется квантор. В этом отношении показательна семантика Флойда (фактически  $\text{sp}$ -семантика) инструкции присваивания  $x := f(a, b, \dots, x)$ , согласно которой

$$\begin{aligned} \text{sp}(x := f(a, b, \dots, x), \varphi(a, b, \dots, x)) &\Rightarrow \\ &= \exists x' (x := f(a, b, \dots, x') \wedge \varphi(a, b, \dots, x')). \quad (3) \end{aligned}$$

Преобразование предикатов  $\text{sp}(P, \varphi)$  лишено важных свойств  $\text{wp}(P, \varphi)$ : вообще говоря, неверно, что  $\text{sp}(P, \neg \varphi) = \neg \text{sp}(P, \varphi)$ ,  $\text{sp}(P, \varphi_1 \wedge \varphi_2) = \text{sp}(P, \varphi_1) \wedge \text{sp}(P, \varphi_2)$ . Отметим, однако, что соотношение  $\text{sp}(P, \varphi_1 \vee \varphi_2) = \text{sp}(P, \varphi_1) \vee \text{sp}(P, \varphi_2)$  сохраняется. Потеря важных свойств объясняется возможностью отсутствия обратного к  $p$  преобразования. Это же служит препятствием к построению прямого преобразования произвольных функций на  $V$ . В работах по денотационной семантике [8] предикат иногда рассматривается не как функция на  $V$ , а как подмножество  $V$ ; прямое и обратное преобразования предиката — это полный образ и прообраз подмножества. Но и переход от функций к подмножествам не дает определения прямого преобразования функций с более обширным, чем у булевых функций, множеством значений, в то время как  $p^*(g)$  определено для произвольной функции  $g$ . В дальнейшем будут обстоятельно изучаться дополнительные возможности, которые открывает выход за пределы булевых функций на  $V$ , а сейчас мы лишь бегло отметим, что привлечение, например, функций с неотрицательными целочисленными значениями оказывается совершенно естественным в задаче установления завершенности выполнения

циклической программы: выбирается такая функция со значениями указанного типа, которая убывает при переходе к новому состоянию в результате выполнения одного этапа цикла.

Интересно отметить, что в некоторых работах (см., например, [2]) семантика присваивания (хооровская подстановка) уже определена к тому времени, когда обсуждается вопрос о завершимости программ, однако характер видоизменения конкретных целочисленных функций в результате выполнения последовательности присваиваний устанавливается частными рассуждениями, не прибегая к уже сформулированному правилу.

Факт идентичности слабейшего предусловия и сопряженного преобразования в множестве булевых функций приводит к предположению о том, что должны существовать достаточно универсальные методы анализа программ, построенные на применении сопряженных преобразований в произвольных множествах функций. Для детерминированных программ такие методы будут предложены в следующих параграфах этой главы. Настоящий же параграф завершим рядом общих предварительных замечаний.

Если в программе  $P$  участвуют переменные  $a, b, \dots, x$  и  $p^*$  применимо к функциям со значениями тех типов, к которым относятся эти переменные, то  $p^*$  обеспечивает нам знание самого преобразования  $p$ . Можно рассматривать  $a, b, \dots, x$  как функции состояния и применить к ним преобразование  $p^*$ .

Пусть  $A, B, \dots, X$  — значения переменных до выполнения  $P$ , а  $A^P, B^P, \dots, X^P$  — после выполнения. Поскольку, очевидно, справедливы равенства

$$A^P = p^*(a)(A, B, \dots, X), \quad B^P = p^*(b)(A, B, \dots, X), \dots \\ \dots, X^P = p^*(x)(A, B, \dots, X),$$

то, действительно, известно само преобразование  $p$ .

Вообще же смысл  $p^*(g)$ , где  $g(a, b, \dots, x)$  — некоторая функция того типа, к которому может быть применено  $p^*$ , таков:

$$g(A^P, B^P, \dots, X^P) = p^*(g)(A, B, \dots, X), \quad (4)$$

т. е. функция  $g$  на измененных значениях переменных определяется через неизмененные значения: в частности, (4) верно и для предикатов ( $p^* = \text{wp}$ ).

Примечание 2. Сильнейшее постусловие и его возможные аналоги подобным свойством не обладают: зная функцию (например, предикат)  $f(a, b, \dots, x)$  и программу  $P$ , невозможно, вообще говоря, указать функцию  $g(a, b, \dots, x)$  такую, что для любых значений  $A, B, \dots, X$  переменных  $a, b, \dots, x$  выполнено  $f(A, B, \dots, X) = g(A^P, B^P, \dots, X^P)$ . Пусть переменная  $a$  может принимать любые неотрицательные целые значения и предикат  $\varphi(a)$  определен как  $3|a$  (т. е. 3 делит  $a$ ). Ясно, что после выполнения инструкции  $a := \text{остаток}(a, 2)$  невозможно восстановить прежнее значение предиката  $\varphi$ .

Пусть в программе  $P$  участвует логическая переменная  $x$ ; можно рассмотреть  $x$  в качестве постусловия (интерпретируя  $x$  в этом случае как частичную функцию на  $V$ , принимающую значения в множестве, состоящем из одного элемента  $I$ ). Тогда сама программа  $P$  как бы представляет собой формулу для соответствующего слабейшего предусловия. Но приведенные Дейкстрой правила вычисления слабейшего предусловия дадут другой вид  $\text{wp}(P, x)$ , т. е.  $p^*(x)$ . Если смотреть на  $p^*$  как на преобразование текстов в фиксированном языке, то этим преобразованием, как видно из приведенного примера, решается некоторая задача трансляции. Заметим, что обобщение  $\text{wp}$ -семантики на множества функций с произвольными областями значений в еще большей степени, чем  $\text{wp}$ -семантика, может служить связкой между различными языками.

Языки программирования дают богатые средства для краткого описания алгоритмов, но именно из-за этого возникают трудности с доказательствами свойств программ средствами интуитивной операционной семантики. Многие языки математических теорий (рекурсивных функций, исчисления предикатов и т. д.), формализованные в той или иной мере, обладают большой выразительной силой и обходятся при этом крайне скудным набором выразительных средств. Эти языки придуманы не для того, чтобы на них было легко и удобно «программировать», а для того, чтобы было удобно доказывать свойства описываемых объектов. Семантика типа  $\text{wp}$ -семантики может играть роль связки между языками этих двух видов. Когда множество  $M$ , в котором принимают значения рассматриваемые функции, содержит в себе множества, соответствующие предусматриваемым языком программирования типа данных, обобщения  $\text{wp}$ -семантики играют роль еще одной связки: между операторным языком и функциональным (можно сказать — между

операторным программированием и функциональным программированием).

Изложение теории частичных функций на множестве состояний в дальнейшем будет таким: поскольку система Хоора возникла исторически раньше системы Дейкстры, мы первоначально рассмотрим именно систему Хоора, беря функции произвольной природы в качестве предусловий и постусловий. Связь системы Хоора со слабым предусловием (сопряженным преобразованием) проявляется, например, в том, что модели реальных языков программирования, рассмотренные Хоором и Дейкстрой, имеют изначальным своим элементом инструкцию присваивания, а для этой инструкции правило Хоора (подстановка) описывает получение именно слабого предусловия по заданному постусловию \*).

В самой системе Хоора каждое свойство  $\{\varphi\}P\{\psi\}$  рассматривается как формула, в которой  $\varphi$  и  $\psi$  — предикатные формулы некоторого фиксированного языка (чаще всего языка первого порядка, реже — второго). Правила Хоора — это правила вывода формул. Однако при исследовании конкретных свойств программ ничто не мешает рассматривать  $\varphi$  и  $\psi$  содержательно — как логические функции на  $V$ .

Множество состояний  $V$  во всех рассуждениях, проводимых в этой книге, будет играть значительную роль. В тех программах, которые приводятся в качестве примеров, участвуют лишь переменные сравнительно простых типов, и поэтому в этих примерах не возникает никаких недоразумений в связи с понятием переменной и множества состояний. Но хорошо известно, что общие понятия переменной, значения переменной и связи переменной с ее значением — это довольно сложные понятия программирования, оперирование которыми требует большой аккуратности. В. Н. Редько рассмотрел некоторые специальные множества и функции, с помощью которых во многих сложных случаях оказывается возможным описание и множества состояний, и преобразований состояний при выполнении разнообразных инструкций языков программирования; допускаются нетривиальные типы данных, многократная косвенная адресация и т. д. Эти множества и функции названы именными множествами и функциями [13].

---

\*) Идеи Хоора и являющиеся их прототипом идеи Флойда к настоящему времени подробно изложены уже во многих монографиях и учебниках (см., например, [1]).

## § 2. Обобщение хооровских свойств и правил Хоора

В силу причин, о которых будет сказано далее, во многих случаях оказывается удобным рассматривать в качестве предусловий и постусловий программ даже не частичные функции с некоторой областью знаний  $M$ , а бинарные отношения — подмножества  $V \times M$ . На эти бинарные отношения мы иногда будем смотреть как на частичные многозначные функции, определенные на  $V$  и принимающие значения в  $M$ . Утверждая для некоторой многозначной функции  $t$ , что  $t(v) = m$  (или что  $t(v) \neq m$ ), мы будем подразумевать, что  $m$  — одно из значений функции  $t$  для значения аргумента, равного  $v$  (или что среди этих значений функции  $t$  нет  $m$ ); это же самое можно было бы записывать как  $vtm$  (или соответственно как  $\neg(vtm)$ ).

В этом параграфе без специальных оговорок считается, что  $P, Q, R, S$  — это программы (соответственно  $p, q, r, s$  — преобразования множества  $V$ ) и  $f, g, h \subseteq V \times M$ .

**Определение 1.** Имеет место свойство  $\{f\}P\{g\}$ , если всякий раз, когда для некоторых  $v_0, v_1 \in V$  и  $m \in M$  выполнено  $p(v_0) = v_1$  и  $f(v_0) = m$ , выполнено и  $g(v_1) = m$ . В выражении  $\{f\}P\{g\}$  бинарное отношение  $f$  называется *предусловием* (для  $g$  относительно  $P$ ), а  $g$  — *постусловием* (для  $f$  относительно  $P$ ).

Для определенных таким образом свойств программ можно сформулировать аналоги правил Хоора.

1) *Инструкция присваивания.* Справедливо

$$\{h(d(a, b, \dots, x), b, \dots, x)\}a := d(a, b, \dots, x)\{h(a, b, \dots, x)\},$$

здесь  $a, b, \dots, x$  — переменные, участвующие в программе  $P$ ,  $d$  — некоторая однозначная функция.

2) *Составная инструкция.* Если имеют место свойства  $\{f\}P\{g\}$  и  $\{g\}Q\{h\}$ , то имеет место и  $\{f\}P; Q\{h\}$ .

3) *Условная инструкция.* Если имеют место свойства  $\{\text{if } \rho \text{ then } f \text{ fi}\}P\{g\}$  и  $\{\text{if } \neg \rho \text{ then } f \text{ fi}\}Q\{g\}$ , то имеет место и  $\{\text{if } \rho \text{ then } P \text{ else } Q \text{ fi}\}g$ . Здесь  $\text{if } \rho \text{ then } f \text{ fi}$  и  $\text{if } \neg \rho \text{ then } f \text{ fi}$  задают функции (возможно, многозначные), которые определены только при  $\rho$  и  $\neg \rho$  соответственно. Таков аналог конъюнкций  $\rho \wedge f$  и  $\neg \rho \wedge f$ , и мы часто будем использовать обозначения вида



$\rho \wedge f$  — это вполне корректно, так как  $(\rho \wedge \sigma) \wedge f = \rho \wedge (\sigma \wedge f)$ . Итак, если имеют место  $\{\rho \wedge f\} P \{g\}$  и  $\{\neg \rho \wedge f\} P \{g\}$ , то имеет место

$$\{f\} \text{ if } \rho \text{ then } P \text{ else } Q \text{ fi } \{g\}.$$

4) *Циклическая инструкция.* Если имеет место свойство  $\{\rho \wedge f\} P \{f\}$ , то имеет место  $\{f\} \text{ while } \rho \text{ do } P \text{ od } \{\neg \rho \wedge f\}$ ;  $f$  называется *инвариантом*.

5) *Пустая инструкция.* Имеет место  $\{f\}\{f\}$ .

6) *Ослабление свойства.* Если имеют место свойство  $\{f\}P\{g\}$  и включения  $f_1 \subseteq f$ ,  $g \subseteq g_1$ , то имеет место и  $\{f_1\}P\{g_1\}$ . Включение, как обычно, дает аналог импликации — знаку импликации  $\supset$  соответствует при этом знак включения  $\subseteq$ .

По поводу правила 1) отметим следующее. Инструкция присваивания — это текст, и правая часть инструкции присваивания — не функция, а выражение. Строго говоря, правило 1) следовало бы записать иначе:

$$\{h(d(a, b, \dots, x), b, \dots, x)\} x := D\{h(a, b, \dots, x)\},$$

где  $d(a, b, \dots, x)$  — однозначная функция, определяемая выражением  $D$ . Надо помнить, что и в записи условной инструкции после символа *if* располагается логическое выражение (предикатная формула), а не предикат (логическая функция). Мы будем позволять себе в тех случаях, когда это не приводит к недоразумениям, обозначать одной и той же буквой формулу и соответствующую функцию.

В качестве иллюстрации применения правил проведем подробное «доказательство правильности» программы  $S$  вычисления факториала, используя для этого предложенный аппарат. Займемся свойством

$$\{n!\} k := 1; \text{ while } n \neq 0 \text{ do } k := k \cdot n; \\ n := n - 1 \text{ od } \{k\}. \quad (1)$$

Здесь множеством состояний будет множество пар неотрицательных целых чисел  $(n, k)$ , множеством  $M$  — множество неотрицательных целых чисел. В качестве предусловия и постусловия привлечены однозначные функции.

По правилу 1) можем написать  $\{n!\} k := 1 \{(k = 1) \wedge \wedge n !\}$  поэтому для доказательства (1) достаточно

установить

$\{(k = 1) \wedge n!\} \text{ while } n \neq 0 \text{ do}$

$$k := k \cdot n; \quad n := n - 1 \text{ od } \{k\}. \quad (2)$$

Докажем вначале свойство

$\{n! k\} \text{ while } n \neq 0 \text{ do}$

$$k := k \cdot n; \quad n := n - 1 \text{ od } \{(n = 0) \wedge n! k\}. \quad (3)$$

Для этого воспользуемся правилом 4) и покажем, что  $n!k$  — инвариант, а именно проверим

$$\{(n \neq 0) \wedge n! k\} k := k \cdot n; \quad n := n - 1 \{n! k\}. \quad (4)$$

В силу правила 1) можно записать

$$\{(n - 1)! k\} n := n - 1 \{n! k\};$$

$$\{(n - 1)! kn\} k := k \cdot n \{(n - 1)! k\};$$

а в силу правила 2) имеем

$$\{(n - 1)! kn\} k := k \cdot n; \quad n := n - 1 \{n! k\};$$

поскольку  $(n \neq 0) \wedge n! k \subseteq (n - 1)! kn$ , то, по правилу 6), получается (4). Теперь для завершения доказательства (2) достаточно сослаться на очевидные включения

$$(k = 1) \wedge n! \subseteq n! k, \quad (n = 0) \wedge n! k \subseteq k$$

и применить правило 6).

Сравним это доказательство с традиционным доказательством хооровского типа, которое могло бы заключаться, например, в установлении при помощи булева инварианта одного из свойств

$$\{m = n\} S \{m! = k\}; \quad (5)$$

$$\{m = n!\} S \{m = k\} \quad (6)$$

программы  $S$  или свойства

$$\{I\} m := n; \quad S \{m! = k\} \quad (7)$$

измененной программы. Одно из отличий — переход от импликаций, доказательство которых может осуществляться, например, средствами логического вывода, к включениям. Различные средства доказательства включений такого вида доставляет, например, теория неподвижной точки программ — мы коснемся этих вопросов позднее. Второе (качественное) отличие связано с привлечением более естественного (числового, а не булева)

инварианта цикла. Весь характер задачи указывает на естественность такого инварианта: используется то, что хотя в ходе выполнения программы значения  $n$  и  $k$  изменяются, но  $n!k$  остается при этом неизменным; в начале выполнения программы значение инварианта совпадает с факториалом заданного значения  $n$ , в конце выполнения значение инварианта совпадает со значением  $k$ . Такова суть приведенного выше подробного доказательства. В этом примере ясно просматривается аналогия между инвариантами циклов и первыми интегралами обыкновенных дифференциальных уравнений или некоторыми величинами, характеризующими физический процесс, относительно которых справедливы законы сохранения.

Перед тем как рассмотреть еще одно выгодное отличие приведенного доказательства от традиционного, заметим следующее. Свойство  $\{f\}P\{g\}$  представляет собой совокупность обычных хоровских свойств  $\{f_{(m)}\}P\{g_{(m)}\}$ ,  $m \in M$ , где  $f_{(m)}$ ,  $g_{(m)}$  — предикаты на  $V$ , эквивалентные соответственно  $f(v) = m$  и  $g(v) = m$ . Обычное хоровское свойство, с другой стороны, может быть записано с помощью бинарных отношений, для которых множество  $M$  одноэлементно. Если множество  $M$  бесконечно, то рассмотрение одного-единственного свойства, записанного с помощью бинарных отношений, может заменить рассмотрение бесконечного числа хоровских свойств.

Разобранный выше пример программы  $S$  вычисления факториала имеет к этому прямое отношение. Как говорилось, задачу установления правильности программы  $S$  считают обычно эквивалентной задаче доказательства какого-нибудь свойства программы  $S$  вроде свойств (5), (6) или свойства (7) измененной программы. Но на самом деле доказательство этих свойств еще не дает полного решения задачи: дополнительно надо убедиться, что переменная  $m$  сохранила свое значение при выполнении  $S$ . Но в хоровском свойстве программы факт неизменности значения некоторой переменной не отражается. Замечание такого рода, что  $m$  не входит в инструкции программы и, следовательно, не изменяется, есть привлечение интуитивной операционной семантики, а это разрушает замкнутость рассматриваемой методики. То же самое относится к предпринимаемому иногда явному разделению *входных* (неизменяемых), *временных* и *выходных* переменных. Немаловажно и то, что реальные программы пишутся без этого разделения.

Возможны ситуации, когда переменная, неизменность значения которой надо установить, входит в инструкции программы. Рассмотрим программу  $S'$  вычисления факториала

$k := 1; \quad i := 0; \quad \text{while } i \neq n \text{ do } i := i + 1; \quad k := k \cdot i \text{ od.}$

Здесь переменная  $n$  используется в циклической инструкции. Возможность доказательства свойства  $\{I\}S'\{k = n!\}$  ни о чем еще не говорит: точно так же может быть доказано и свойство  $\{I\}k := 1; \quad n := 1\{k = n!\}$ , однако вывод, что выполнение программы  $k := 1; \quad n := 1$  приводит к вычислению факториала для любого неотрицательного  $n$ , был бы неправильным. Ссылка на неизменность значений некоторых переменных совершенно невозможна в тех случаях, когда программа содержит такие инструкции, которые изменяют значения всех переменных, участвующих в программе: гипотетически можно представить себе, например, инструкцию уменьшения значений всех переменных на 1.

Если не прибегать к операционным соображениям, то вместо одного свойства мы вынуждены будем иметь дело с бесконечной последовательностью хооровских свойств; пусть в примере программы  $S$  значения переменных суть неотрицательные целые числа, тогда этими свойствами будут

$\{0 = n!\}S\{0 = k\}, \{1 = n!\}S\{1 = k\}, \{2 = n!\}S\{2 = k\}, \dots$

Привлекая бинарные отношения (которые в данном случае оказываются однозначными функциями с неотрицательными целыми значениями), мы можем записать всю последнюю серию свойств в виде одного —  $\{n!\}S\{k\}$ .

Отметим, что совпадение значений переменной  $m$  до и после выполнения программы  $S$  изображается свойством  $\{m\}S\{m\}$ .

Запись последовательности хооровских свойств в виде одного свойства может потребовать привлечения и многозначных функций. Если бы мы интересовались более слабым свойством программы  $S$  (выполнение  $S$  приводит к тому, что  $k$  приобретает значение, не меньшее  $n!$ ), то вместо  $\{m = n!\}S\{m \leq k\}$  можно было бы написать

$$\{n!\}S\{[0, k]\}. \quad (8)$$

Это можно пояснить так. Как обычно, предусловие и постусловие являются функциями на  $V$  — в данном

случае на множестве пар  $(n, k)$  неотрицательных целых чисел. Обозначим предусловие через  $f(n, k)$ , а постусловие — через  $g(n, k)$ , тогда значением функции  $f(n, k)$  является  $n!$ , а значением функции (многозначной)  $g(n, k)$  является каждое неотрицательное целое число из отрезка  $[0, k]$ . Согласно определению 1 свойство (8) означает, что если для исходного значения переменной  $n$  вычислить значение  $n!$ , то это значение будет неотрицательным и не превзойдет заключительного значения переменной  $k$ .

Еще один пример. Доказательство завершимости выполнения цикла **while**  $P$  **do**  $P$  **od** проводится, по Дейкстре, подбором однозначной функции  $t$ , принимающей целочисленные неотрицательные значения, и, при условии завершимости выполнения  $P$ , последующим доказательством свойства

$$\{t(a, b, \dots, x) \leq t_0\} P \{t(a, b, \dots, x) < t_0\},$$

где  $t_0$  — некоторая дополнительная переменная. Чтобы избежать введения дополнительной переменной, можно рассмотреть свойство

$$\{[t(a, b, \dots, x), \infty)\} P \{(t(a, b, \dots, x), \infty)\}.$$

Простой пример: для доказательства завершимости выполнения программы  $S$  докажем свойство

$$\{[n, \infty)\} k := k \cdot n; \quad n := n - 1 \{(n, \infty)\}$$

(это соответствует рассмотрению функции  $t(n, k) = n$ ). Для доказательства последнего достаточно установить включение  $[n, \infty) \subseteq (n-1, \infty)$ . Дадим общее правило исключения дополнительной переменной  $m$  из хооровского свойства

$$\{\varphi(v, m)\} P \{\psi(v, m)\}; \quad (9)$$

от (9) можно перейти к свойству, предусловием и постусловием, в котором будут соответственно  $\{m | m \in M, \varphi(v, m)\}$  и  $\{m | m \in M, \psi(v, m)\}$ , при этом  $m$  может быть заменена любой буквой и все свойство целиком может быть переписано, например, так:

$$\{\{l | l \in M, \varphi(v, l)\}\} P \{\{m | m \in M, \psi(v, m)\}\}. \quad (10)$$

Позднее будет рассмотрен способ перехода от неявного задания многозначной функции к явному. Итак, одно из преимуществ предлагаемого подхода перед традиционным состоит в его большей логической замкну-

тости. Можно отметить еще некоторое эстетическое преимущество: этот подход позволяет не писать лишнего в доказательствах. В самом деле, введение переменной  $t_0$  в разобранный выше примере не нужно ни для чего, кроме удовлетворения желания записать предусловие и постусловие в виде предикатов (переменная  $t_0$  не будет и не может преобразовываться при использовании правил Хоора).

Таким образом, множество хооровских свойств, параметризованное элементами некоторого множества  $M$ , можно рассматривать как одно свойство. Природа множества  $M$  может быть различной. Например, это может быть множество предикатов на  $V$ . Рассмотрим некоторую циклическую программу **while**  $\rho$  **do**  $P$  **od**; в виде одного-единственного свойства можно записать утверждение, что для любого предиката  $\varphi$  имеет место хооровское свойство  $\{\varphi\}$  **while**  $\rho$  **do**  $P$  **od**  $\{\neg\rho\}$ ; в качестве предусловия возьмем многозначную функцию  $f$ , определяемую так:  $f(v) = \varphi$ , если истинно  $\rho(v)$ ; а в качестве постусловия возьмем  $g(v)$ , определяемую так:  $g(v) = \varphi$ , если истинно  $\neg\rho(v)$ . В качестве инварианта можно взять  $h$  такую, что  $h(v) = \varphi$  для любых  $v$ ,  $\varphi$ .

В следующем примере в роли множества  $M$  вновь выступает множество состояний  $V$ . Пусть мы сравниваем две программы  $P$  и  $Q$  и хотим доказать, что  $p \sqsubseteq q$ . Укажем такое свойство программы  $P$ , которое эквивалентно этому включению. Ясно, что достаточно определить предусловие так, чтобы в случае завершения выполнения программы  $P$  для некоторого начального состояния этому начальному состоянию сопоставлялся результат выполнения  $Q$ , а в противном случае ему сопоставлялись бы все состояния. При этом постусловие должно сопоставлять любому состоянию это же самое состояние. Пусть  $a, b, \dots, x$  — переменные, участвующие в программах  $P$  и  $Q$ . Пусть  $a$  принимает значение в множестве  $V_a$ ,  $b$  — в  $V_b$ ,  $\dots$ ,  $x$  — в  $V_x$ , и, таким образом,  $V = V_a \times V_b \times \dots \times V_x$ . В  $Q$  заменим переменные  $a, b, \dots, x$  на  $\bar{a}, \bar{b}, \dots, \bar{x}$ . Тогда, в силу сказанного, включению  $p \sqsubseteq q$  соответствует следующее свойство программы  $P$ :

$$\{(A, B, \dots, X) \mid A \in V_a, B \in V_b, \dots, X \in V_x,$$

$$\{\bar{a} = a \wedge \bar{b} = b \wedge \dots \wedge \bar{x} = x\}$$

$$Q \{\bar{a} = A \wedge \bar{b} = B \wedge \dots \wedge \bar{x} = X\} \} P \{(a_1, b_1, \dots, x)\}. \quad (11)$$

Пусть  $P$  — это  $a := a + a$ ,  $Q$  — это  $\bar{a} := 2 \cdot \bar{a}$ , тогда (11) переписывается в виде

$$\{\{A \mid A \in \mathbb{N}, \{\bar{a} = a\} \bar{a} := 2\bar{a} \{\bar{a} = A\}\} \} a := a + a\{a\}, \quad (12)$$

где  $\mathbb{N}$  — множество неотрицательных целых чисел.

Докажем выписанное свойство. Предусловие относительно программы  $a := a + a$  можно переписать в виде

$$\{A \mid A \in \mathbb{N}, (\bar{a} = a) \supset (\bar{a} = A)\}.$$

При фиксированном значении  $a$  существует одно-единственное число  $A$ , удовлетворяющее выписанному условию, а именно  $2a$ . Свойство (12) переписывается в виде  $\{2a\} a := a + a\{a\}$ . Включение  $2a \subseteq a + a$ , которому равносильно последнее свойство, очевидно.

Если в момент завершения выполнения программы принимать во внимание значения не всех, а только некоторых переменных (заранее указанных) и в этом смысле понимать значения  $p$  и  $q$ , то включение  $p \subseteq q$  тоже может изображаться свойством программы  $P$ , сходным с (10). Пусть программы  $P$  и  $Q$  выглядят соответственно так:

$m := 1; i := 2; \text{ while } i \neq k \text{ do } m := m \cdot i; i := i + 1 \text{ od},$

$\bar{m} := 1; \text{ while } \bar{k} \neq 0 \text{ do } \bar{m} := \bar{m} \cdot \bar{k}; \bar{k} := \bar{k} - 1 \text{ od}.$

Пусть мы хотим доказать, что если для некоторых значений переменных  $m, i, k$  выполнение программы  $P$  завершается, то для значений переменных  $\bar{m}, \bar{k}$ , равных значениям  $m, k$ , завершается выполнение  $Q$  и при этом заключительные значения  $m$  и  $\bar{m}$  совпадают. Последнее изображается свойством программы  $P$

$$\{\{M \mid M \in \mathbb{N}, \{(\bar{m} = m) \wedge (\bar{k} = k)\} Q \{\bar{m} = M\}\} \} P \{m\}.$$

Рассмотрим предусловие этого свойства. В качестве  $M$  может быть взято только  $k!$ ; то, что  $k!$  действительно подходит, может быть установлено доказательством свойства

$$\{(\bar{m} = m) \wedge (\bar{k} = k)\} Q \{\bar{m} = k!\}$$

с помощью инварианта  $k! = \bar{m} \bar{k}!$ ; то, что другие значения не подходят, можно показать, установив завершенность выполнения программы  $Q$  для любых принадлежащих  $\mathbb{N}$  начальных значений переменных этой программы. Свойство  $\{k!\} P \{m\}$  доказывается несложно. Включение  $q \subseteq p$  места не имеет, при попытке доказать

его описанным методом в качестве предусловия относительно программы  $Q$  возникла бы многозначная функция  $f(\bar{m}, \bar{k})$ , значением которой является каждое неотрицательное целое число при  $\bar{k} = 0$  и  $\bar{k}!$  при  $\bar{k} > 0$ . Потребовалось бы доказать неверное включение  $f(\bar{m}, \bar{k}) \subseteq \bar{k}!$

В обзоре работ по денотационной семантике [8] отмечено, что анализ некоторых программ с рекурсивными процедурами требует рассмотрения бесконечного множества предикатов на множестве состояний  $V$  или единичных предикатов на расширенном множестве состояний. Подобные конструкции могут быть заменены бинарными отношениями, т. е. многозначными функциями на  $V$  со значениями в некотором множестве  $M$ .

Этот параграф закончим замечанием о многозначных частичных функциях. Разумеется, само понятие многозначной функции является довольно расплывчатым. Как отмечалось в начале параграфа, можно было бы, говоря об обобщенных хооровских свойствах, пользоваться аппаратом и обозначениями алгебры бинарных отношений. В следующих параграфах к многозначным частичным функциям часто будут применяться операции алгебры бинарных отношений. Функциональные обозначения привлекались для того, чтобы, по возможности, формулы теории Хоора оставались формулами с тем же смыслом в рамках развиваемого обобщения этой теории.

В некоторых случаях окажется более удобным и естественным другой подход, заключающийся в рассмотрении бинарных отношений как однозначных частичных функций, значениями которых служат непустые подмножества множества  $M$ .

### § 3. Обобщение слабейшего предусловия и слабейшего свободного предусловия

Столь же полезными, как и обобщения правил Хоора, были бы обобщения формул Дейкстры, описывающих слабейшее предусловие (сопряженное преобразование) применительно к основным инструкциям алгоритмических языков — присваиванию, условной инструкции, циклу и т. д. Но сами формулы Дейкстры уже в случае условной инструкции содержат логические связки, а в случае циклической инструкции еще и квантор существования и поэтому не могут непосредственно применяться к постусловиям, являющимся произвольными функциями.



Однако возможно обобщение этих формул — оно и будет описано в этом параграфе для детерминированного случая. Для аналога слабейшего предусловия сохраняется принятое обозначение  $\text{wp}(P, g)$ , где  $P$  — программа (инструкция),  $g$  — постусловие. В тех случаях, когда в качестве функций предусловий и постусловий используются предикаты, они по-прежнему обозначаются малыми греческими буквами и рассматриваются как частичные функции, принимающие для тех аргументов, для которых они определены, значение И.

Пусть дана программа  $P$ , которой соответствует преобразование  $p$  (может быть частичное) множества состояний  $V$  в себя. Пусть  $g(v)$  — некоторая функция, рассматриваемая как постусловие. Наша задача — описание  $\text{wp}(P, g)(v) = (p^*(g))(v) = g(p(v))$ .

Для инструкции присваивания и составной инструкции все остается таким же, как в системе Дейкстры. Более того, формула  $\text{wp}(a := f(a, b, \dots, x), g(a, b, \dots, x)) = g(f(a, b, \dots, x), b, \dots, x)$  сохраняет силу и для частичной (однозначной) функции  $f(a, b, \dots, x)$ ; при традиционном взгляде на предикаты охватить случай частичной  $f$  удастся только за счет усложнения формулы (Дейкстра использует для этого случая специальную операцию **cand** — условную конъюнкцию).

Рассмотрим условную инструкцию  $P$

**if  $\beta$  then  $R$  else  $S$  fi;**

$\text{wp}(P, g)$ , очевидно, может быть определено условным выражением

**if  $\beta$  then  $\text{wp}(R, g)$  else  $\text{wp}(S, g)$  fi.**

Рассмотрим  $P$  вида

**while  $\beta$  do  $S$  od.**

Напомним, что при описании  $\text{wp}(P, \psi)$  в случае циклической  $P$  и логического постусловия  $\psi$  строится последовательность приближений к  $\text{wp}(P, \psi)$  — последовательность предикатов, которые мы обозначим через  $\text{wp}(P, \psi)_0$ ,  $\text{wp}(P, \psi)_1, \dots$ ; предикат  $\text{wp}(P, \psi_j)(v)$  означает, что для начального состояния  $v$  выполнение циклической инструкции завершается не позднее чем через  $j$  этапов и в итоге получается состояние, удовлетворяющее  $\psi$ . Имеет место рекуррентное соотношение

$$\text{wp}(P, \psi)_0 = \neg \beta \wedge \psi; \quad (1)$$

$$\text{wp}(P, \psi)_j = \text{wp}(\text{if } \beta \text{ then } S \text{ fi}, \text{wp}(P, \psi)_{j-1}) \vee \text{wp}(P, \psi)_0.$$

(Здесь  $\text{if } \beta \text{ then } S \text{ fi}$  надо понимать в том же смысле, что и инструкцию  $\text{if } \beta \rightarrow S \text{ fi}$  Дейкстры: эта инструкция неприменима к тем состояниям, для которых не выполнено  $\beta$ .) Ясно, что для  $j = 1, 2, \dots$  имеют место импликации  $\text{wp}(P, \psi)_{j-1} \supset \text{wp}(P, \psi)_j$ .

Исходя из описанной последовательности приближений,  $\text{wp}(S, \psi)$  определяется как  $\exists j \text{ wp}(P, \psi)_j$  — это и есть формула Дейкстры. Отметим, что поскольку мы рассматриваем предикаты как частичные константы, заменяя значение  $\perp$  неопределенностью, то можно написать

$$\text{wp}(P, \psi)_{j-1} \subseteq \text{wp}(P, \psi)_j, \quad \text{wp}(P, \psi) = \bigcup_{j=0}^{\infty} \text{wp}(P, \psi)_j.$$

Соотношение (1) в этом смысле можно переписать так:

$$\text{wp}(P, \psi)_0 = \text{if } \neg \beta \text{ then } \psi \text{ fi}; \quad (2)$$

$$\text{wp}(P, \psi)_j = \text{if } \beta \text{ then } \text{wp}(S, \text{wp}(P, \psi)_{j-1}) \text{ else } \psi \text{ fi}.$$

Можно усмотреть, что если брать вместо логических частичных констант частичные функции совершенно произвольной природы, то формулы (2) распространяются без каких-либо препятствий на этот случай. Если взять

$$\text{wp}(P, g)_0 = \text{if } \neg \beta \text{ then } g \text{ fi}; \quad (3)$$

$$\text{wp}(P, g)_j = \text{if } \beta \text{ then } \text{wp}(S, \text{wp}(P, g)_{j-1}) \text{ else } g \text{ fi},$$

то, как и прежде, будет  $\text{wp}(P, g)_{j-1} \subseteq \text{wp}(P, g)_j$  и  $\bigcup_j \text{wp}(P, g)_j$  даст ту функцию, которую мы решили считать слабейшим предусловием. Смысл же  $\text{wp}(P, g)_j$  таков, что если  $\text{wp}(P, g)_j$  определена для  $v$ , то для начального состояния  $v$  выполнение циклической инструкции  $P$  завершится не позднее чем через  $j$  этапов и  $g(p(v)) = \text{wp}(P, g)(v)$ . (Если  $g$  — неоднозначная функция, то и  $\text{wp}(P, g)$  — неоднозначная; равенство должно пониматься в смысле совпадения множеств принимаемых значений при равных значениях аргументов.)

Используя обозначения предыдущего параграфа, первую строчку (3) можно записать в виде  $\text{wp}(P, g)_0 = \neg \beta \wedge g$ .

**Пример 1.** Рассмотрим случай, когда постусловием циклической инструкции служит функция с числовыми значениями: пусть циклическая инструкция  $P$  имеет

while  $n \neq 0$  do  $k := k \cdot n; \quad n := n - 1$  od,

а в качестве постусловия берется  $k$ . Нетрудно по индукции показать, что  $\text{wp}(P, k)_j = \text{if } n \leq j \text{ then } k \cdot n! \text{ fi} = (n \leq j) \wedge k \cdot n!$  и, как следствие этого, что  $\text{wp}(P, k) = kn!$  Последнее показывает, что  $P$  правильно вычисляет значение  $kn!$

Пример 2. Рассмотрим программу  $Q$ :

while  $n > 1$  do

if  $2 \mid n$  then  $n := \frac{n}{2} + 1$  else  $n := n + 1$  fi

od.

Будем считать, что  $n$  принимает положительные целые значения. Пусть  $g(n)$  — какая-нибудь функция, найдем  $\text{wp}(Q, g)$ . Обозначим для краткости  $\text{wp}(Q, g)_j$  через  $\tilde{g}_j$ :

$$\tilde{g}_0(n) = (n = 1) \wedge g(n) = (n = 1) \wedge g(1),$$

$$\tilde{g}_j(n) = \text{if } n > 1 \text{ then}$$

$$\text{if } 2 \mid n \text{ then } \tilde{g}_{j-1}\left(\frac{n}{2} + 1\right) \text{ else } \tilde{g}(n + 1) \text{ fi}$$

$$\text{else } g(n) \text{ fi.}$$

Но уравнения  $\frac{n}{2} + 1 = 1$  и  $n + 1 = 1$  не имеют решений в целых положительных числах, и, следовательно,  $\tilde{g}_2(n) = (n = 1) \wedge g(1)$ , и вообще, в силу выписанного рекуррентного соотношения, при  $k = 2, 3, \dots$  имеет место  $\tilde{g}_k = (n = 1) \wedge g(1)$ . Значит, и  $\text{wp}(Q, g)(n) = (n = 1) \wedge g(1)$ . Это, в частности, показывает, что выполнение  $Q$  завершается только для  $n = 1$ .

Из этих примеров видно, что техника работы с функциями произвольной природы та же самая, что и с предикатами. Как и правилами Дейкстры для вычисления логического  $\text{wp}(P, \psi)$ , предложенными правилами удобно пользоваться при установлении свойств циклической инструкции главным образом в тех случаях, когда имеется удачная гипотеза относительно свойства приближений  $\text{wp}(S, g)_j$ ,  $j = 0, 1, \dots$  — в этих случаях рекуррентные соотношения (2) дают возможность провести математическую индукцию для доказательства того, что выписанная гипотеза действительно имеет место для лю-

бого  $j$ . Если гипотезы нет, то и собственно правила Дейкстры могут ничего не дать. Например, до сих пор не доказана и не опровергнута завершимость для любого натурального  $n$  программы

$$\text{while } n > 1 \text{ do if } 2 \mid n \text{ then } n := \frac{n}{2}$$

$$\text{else } n := 3n + 1 \text{ fi od.} \quad (4)$$

Для доказательства завершимости этой программы было бы достаточно доказать, что  $\text{wp}(P, \text{И}) = \text{И}$ , но непосредственное применение правил Дейкстры не позволяет доказать или опровергнуть это.

В книге Дейкстры слабейшее предусловие избирается главным средством анализа программ, так как оно позволяет одновременно устанавливать и согласованность предусловия с постусловием, и завершимость выполнения программы. Однако после нескольких первых глав в этой же книге дается рекомендация в трудных случаях устанавливать завершимость циклических программ отдельно — с помощью специально подобранной целочисленной функции. О таких функциях уже шла речь в предыдущем параграфе; сейчас мы отметим, что если для исследования завершимости программы

$$\text{while } \beta \text{ do } S \text{ od}$$

целочисленная функция  $t$  выбрана, то дальше надо вычислить функцию  $\text{wp}(S, t)$  и сравнить эту функцию с  $t$ . При всем этом мы в итоге должны будем доказать больше, чем требовалось в исходной задаче: требовалось всего лишь доказать завершимость для некоторых начальных состояний, а мы должны будем доказать, что для любого такого состояния  $v$  выполнение завершится не более чем через  $t(v)$  этапов.

**Примечание 1.** Сведения исследования завершимости циклической программы рассматриваемого вида к установлению свойства

$$\{[t(v), \infty)\} S \{(t(v), \infty)\}$$

правомерно только при условии завершимости  $S$ . Вычисление же  $\text{wp}(S, t)$  приводит к функции, которая не определена для тех состояний, для которых выполнение  $S$  не завершается, поэтому нет необходимости заранее предполагать завершимость  $S$ . Вместе с этим пример программы (4) показывает, что изобретение функции  $t$  и запись этой функции в удобном виде может быть очень трудной задачей. (Однако само существование функции  $t$  в том случае, когда выполнение циклической программы завершимо, — этот факт совершенно тривиальный: можно взять, например, в качестве  $t(v)$  само число этапов выполнения цикла.)

Пример 3. С помощью предложенных правил вычисления  $\text{wr}(S, t)$  нетрудно доказать, что для любого натурального  $n$  выполнение программы

**while**  $n > 1$  **do**

**if**  $2 \mid n$  **then**  $n := \frac{n}{2}$  **else**  $n := n + 1$  **fi od**

завершается. В качестве  $t(n)$  для этого достаточно взять

**if**  $n = 1$  **then**  $0$  **else**  $n - (-1)^n$  **fi**.

Рассмотрим теперь описание слабейшего предусловия циклической инструкции методами теории неподвижной точки программ [10]. Основные факты теории неподвижной точки верны как для однозначных функций, так и для многозначных (бинарных отношений).

Дадим краткий обзор этих фактов. Предварительно введем необходимые определения и обозначения. Пусть  $K, L$  — некоторые множества,  $2^K$  и  $2^L$  — множества всех подмножеств  $K$  и  $L$ . Любое преобразование  $2^K \xrightarrow{\Xi} 2^L$  называется функционалом; если  $x \subseteq K$ , то соответствующее значение функционала обозначается через  $\Xi[x]$ . Функционал  $\Xi$  называется монотонным, если из  $x \subseteq y \subseteq K$  следует  $\Xi[x] \subseteq \Xi[y]$ . Монотонный функционал  $\Xi$  называется непрерывным, если из  $x_0 \subseteq x_1 \subseteq x_2 \subseteq \dots \subseteq K$  следует, что  $\bigcup_{i=0}^{\infty} \Xi[x_i] = \Xi\left[\bigcup_{i=0}^{\infty} x_i\right]$ ;  $C_{K,L}$  обозначает множество всех непрерывных функционалов вида  $2^K \rightarrow 2^L$ . Если  $\Xi \in C_{K,K}$ , то с  $\Xi$  связывается подмножество множества  $K$ , называемое *наименьшей неподвижной точкой*  $\Xi$  и обозначаемое через н.н.т.  $\Xi$ . Для наименьшей неподвижной точки  $\Xi$  по определению, во-первых, выполнено равенство  $\Xi[\text{н.н.т. } \Xi] = \text{н.н.т. } \Xi$  и, во-вторых, выполнено включение н.н.т.  $\Xi \subseteq x$  всякий раз, когда  $\Xi[x] = x$ . Доказывается, что для любого  $\Xi \in C_{K,K}$  существует единственная наименьшая неподвижная точка и при этом

$$\text{н. н. т. } \Xi = \bigcup_{i=0}^{\infty} \Xi^i[\emptyset];$$

последнее равенство задает каноническое представление н.н.т.  $\Xi$ . Типичные непрерывные функционалы строятся, исходя из частичных функций многих переменных. Отождествление функции с ее графиком (подмножеством некоторого множества) позволяет сформулировать и доказать, что любой функционал  $\Xi[f]$ , определенный с помощью суперпозиции известных функций и функциональной переменной  $f$ , непрерывен.

Для доказательства включений вида  $\Phi[f] \subseteq \Psi[f]$ , где  $f \subseteq K$ ,  $\Phi, \Psi \in C_{K,L}$ , может применяться предельный переход по  $f$ : пусть  $f_0 \subseteq f_1 \subseteq \dots \subseteq K$  и  $\bigcup_{i=0}^{\infty} f_i = f$ , пусть для  $i = 0, 1, \dots$  выполнено  $\Phi[f_i] \subseteq \Psi[f_i]$ , тогда  $\Phi[f] \subseteq \Psi[f]$ . Из этого общего принципа вы-

водится несколько важных принципов индукции по неподвижной точке.

*Принцип шаговой вычислительной индукции* (Де Баккер, Скотт). Пусть  $\Lambda \in C_{K, K}$ ,  $\Phi, \Psi \in C_{K, L}$  и пусть, во-первых, выполняется  $\Phi[\emptyset] \subseteq \Psi[\emptyset]$ , а, во-вторых, для любого  $x \in K$  из  $\Phi[x] \subseteq \Psi[x]$  следует  $\Phi[\Lambda[x]] \subseteq \Psi[\Lambda[x]]$ ; тогда имеет место  $\Phi[\text{н.н.т. } \Lambda] \subseteq \Psi[\text{н.н.т. } \Lambda]$ .

*Принцип полной вычислительной индукции* (Моррис). Пусть  $\Lambda \in C_{K, K}$ ,  $\Phi, \Psi \in C_{K, L}$  и пусть, во-первых, выполняется  $\Phi[\emptyset] \subseteq \Psi[\emptyset]$ , а, во-вторых, для любого натурального  $t$  из  $\Phi[\Lambda^t[\emptyset]] \subseteq \Psi[\Lambda^t[\emptyset]]$ ,  $i = 1, \dots, t-1$ , следует  $\Phi[\Lambda^t[\emptyset]] \subseteq \Psi[\Lambda^t[\emptyset]]$ ; тогда имеет место  $\Phi[\text{н.н.т. } \Lambda] \subseteq \Psi[\text{н.н.т. } \Lambda]$ . Известен еще ряд принципов индукции.

Уравнение вида  $f = \Xi[f]$  часто рассматривается как рекурсивное определение  $f$ . Ввиду возможности применения принципов предельного перехода и индукции по неподвижной точке, исследование свойств наименьшего решения такого уравнения особенно удобно тогда, когда функционал  $\Xi$  непрерывен. Принципы предельного перехода и индукции по неподвижной точке могут быть обобщены на функционалы от нескольких аргументов, что позволяет применять эти принципы для исследования систем рекурсивных определений множеств (например, для исследования рекурсивных типов данных).

**Теорема 1.** *Для циклической инструкции  $P$  вида*  

$$\text{while } \beta \text{ do } S \text{ od}$$

*и постуловия  $g$  слабейшее предусловие  $\text{wp}(P, g)$  есть наименьшая неподвижная точка непрерывного функционала*

$$\Gamma[f] = \text{if } \beta \text{ then } \text{wp}(S, f) \text{ else } g \text{ fi.}$$

**Доказательство.** Для детерминированной фиксированной инструкции  $P$  функционал  $\Delta[f] = \text{wp}(S, f)$  непрерывен (это очевидно; имеет место и более общий факт — непрерывность в случае ограниченной недетерминированности по Дейкстре, об этом еще будет идти речь в гл. III). Поэтому непрерывен и функционал  $\Gamma[f]$  \*).

---

\*) Выражение  $\text{if } \beta \text{ then } a \text{ else } b \text{ fi}$  задает функцию трех переменных  $\beta, a, b$ . Для того чтобы значение этой функции было определено, считается достаточным, чтобы были определены значение  $\beta$  и, в зависимости от истинности этого значения, значение  $a$  или  $b$ . Теорема о суперпозиции функции и функциональной переменной остается справедливой, если значения некоторых функций считаются определенными при частично заданных значениях переменных; равенство  $Z(y_1, \dots, y_n) = m$ , имеющее место при неопределенных  $y_{i_1}, y_{i_2}, \dots, y_{i_j}$  ( $1 \leq j \leq n$ ), должно оставаться в силе при любом дополнительном выборе значений всех или некоторых переменных из числа  $y_{i_1}, \dots, y_{i_j}$ .

Покажем, что  $\text{wr}(P, g)$  — наименьшая неподвижная точка  $\Gamma$ . Каноническое представление наименьшей неподвижной точки функционала  $\Gamma$  выглядит так:  $\bigcup_{j=0}^{\infty} f_j$ , где  $f_0$  — нигде не определенная функция  $\emptyset$  и  $f_j = \Gamma[f_{j-1}]$  для  $j = 1, 2, \dots$ . Но  $\text{wr}(P, g)_0 = \Gamma[\emptyset]$  (это легко проверить),  $\text{wr}(P, g)_j = \Gamma[\text{wr}(P, g)_{j-1}]$  для  $j = 1, 2, \dots$  (в соответствии с (3)) и, наконец,  $\text{wr}(P, g) = \bigcup_{j=0}^{\infty} \text{wr}(P, g)_j$ .

Как видно из приведенного доказательства теоремы, удобно считать  $\text{wr}(P, g)_{-1} = \emptyset$ .

Теорема 1 может оказаться полезной для доказательства индукцией по неподвижной точке некоторых включений, в частности включения вида  $\text{wr}(P, g) \subseteq h$ , где  $h$  — некоторая фиксированная функция. Но более характерна задача установления обратного включения  $h \subseteq \text{wr}(P, g)$  (например при доказательстве свойства  $\{h\}P\{g\}$ ). Ниже предлагается сходный аппарат для описания слабейшего свободного предусловия, который для доказательства свойства  $\{h\}P\{g\}$  оказывается более удобным.

По определению Дейкстры, слабейшее свободное предусловие  $\text{wlp}(P, \psi)$  — это слабейшее условие того, что если выполнение  $P$  завершается, то заключительное состояние удовлетворяет  $\psi$ . Мы приведем правила вычисления  $\text{wlp}$ , но ограничимся случаем, когда значения всех выражений, встречающихся в программе, определены для любых значений входящих в них переменных. Это не принципиальное ограничение, но отказ от него привел бы к усложнению формул за счет появления в них компонент, описывающих соответствующие области определения и их дополнения.

Инструкции присваивания по-прежнему сопоставляется подстановка. С пустой и составной инструкциями все обстоит совсем просто. Далее, если  $P$  имеет вид

if  $\beta$  then  $R$  else  $S$  fi, то  $\text{wlp}(P, \psi) = \text{if } \beta \text{ then } \text{wlp}(R, \psi) \text{ else } \text{wlp}(S, \psi) \text{ fi}$ .

Займемся циклической инструкцией. Сначала дадим формулу для  $\text{wlp}(P, \psi)_j$  — условия того, что если выполнение завершается не более чем через  $j$  этапов, то заключительное состояние удовлетворяет  $\psi$ . Из этого определения  $\text{wlp}(P, \psi)_j$  видно, что  $\text{wlp}(P, \psi)_0 = \beta \vee \psi$ .

Далее, для  $j > 0$  имеет место

$$\text{wlp}(P, \psi)_j = \text{if } \beta \text{ then } \text{wlp}(R, \text{wlp}(P, \psi)_{j-1}) \text{ else } \psi \text{ fi}, \quad (5)$$

это устанавливается разбором двух случаев:  $\beta$  и  $\neg\beta$ . Исходя из определения  $\text{wlp}(P, \psi)_j$ , нетрудно убедиться, что для  $j > 0$  выполнено

$$\text{wlp}(P, \psi)_j \subseteq \text{wlp}(P, \psi)_{j-1}. \quad (6)$$

При обычном подходе к предикатам этому соответствует импликация  $\text{wlp}(P, \psi)_j \supset \text{wlp}(P, \psi)_{j-1}$ . В последнем случае формула для  $\text{wlp}(P, \psi)$  имеет вид

$$\forall j \text{ wlp}(P, \psi)_j, \quad (7)$$

для дальнейших обобщений более удобной будет формула

$$\bigcap_{j=0}^{\infty} \text{wlp}(P, \psi)_j, \quad (8)$$

которая предполагает взгляд на предикаты как на частичные константы.

Рекуррентное соотношение (5) позволяет получить и значение  $\text{wlp}(P, \psi)_0$ , если в качестве  $\text{wlp}(P, \psi)_{-1}$  взять полностью определенную функцию, тождественно равную И. Интересный вывод, который можно сделать из последнего замечания и из сравнения рекуррентных соотношений для  $\text{wp}(P, \psi)_j$  и  $\text{wlp}(P, \psi)_j$ , состоит в том, что если выполнение  $S$  всегда завершается, т. е.  $\text{wp}(S, \psi) = \text{wlp}(S, \psi)$ , то  $\text{wp}(P, \psi)$  и  $\text{wlp}(P, \psi)$  являются неподвижными точками одного и того же непрерывного функционала

$$\Gamma[\varphi] = \text{if } \beta \text{ then } \text{wp}(S, \varphi) \text{ else } \psi \text{ fi},$$

но  $\text{wp}(P, \psi)$  — наименьшая неподвижная точка, а  $\text{wlp}(P, \psi)$  — наибольшая. В общем случае инструкции  $S$  имеет место следующая теорема.

**Теорема 2.** Для циклической инструкции  $P$  вида

$$\text{while } \beta \text{ do } S \text{ od}$$

и постусловия  $\psi$  слабейшее свободное предусловие  $\text{wlp}(P, \psi)$  есть наибольшая неподвижная точка непрерывного функционала

$$\Lambda[\varphi] = \text{if } \beta \text{ then } \text{wlp}(S, \varphi) \text{ else } \psi \text{ fi}.$$

**Доказательство.** Слабейшее свободное предусловие  $\text{wlp}(S, \varphi)$  есть расширение слабейшего предусло-



вия  $\text{wlp}(S, \varphi)$ , при этом дополнение  $\text{wlp}(S, \varphi)$  в  $\text{wlp}(S, \varphi)$  не зависит от  $\varphi$ . Это показывает непрерывность  $\text{wlp}(S, \varphi)$  по  $\varphi$  при фиксированной  $S$ , а следовательно, и непрерывность  $\Lambda[\varphi]$ . Это же обстоятельство позволяет доказать, что  $\text{wlp}(P, \psi)$  есть неподвижная точка  $\Lambda$ :

$$\begin{aligned}\text{wlp}(P, \psi) &= \bigcap_{j=0}^{\infty} \text{wlp}(P, \psi)_j = \bigcap_{j=1}^{\infty} \text{wlp}(P, \psi)_j = \\ &= \bigcap_{j=1}^{\infty} (\text{if } \beta \text{ then } \text{wlp}(S, \text{wlp}(P, \psi)_{j-1}) \text{ else } \psi \text{ fi}) = \\ &= \text{if } \beta \text{ then } \text{wlp}\left(S, \bigcap_{j=1}^{\infty} \text{wlp}(P, \psi)_{j-1}\right) \text{ else } \psi \text{ fi} = \\ &= \text{if } \beta \text{ then } \text{wlp}(S, \text{wlp}(P, \psi)) \text{ else } \psi \text{ fi}.\end{aligned}$$

Переход от второй строки этой выкладки к третьей корректен в силу упомянутого соотношения между  $\text{wlp}$  и  $\text{wlp}$ , а также в силу того, что для любого  $\chi$  область определения  $\text{wlp}(S, \chi)$  является полным прообразом относительно  $s$  области определения  $\chi$ .

Для того чтобы убедиться, что  $\text{wlp}(P, \psi)$  — наибольшая неподвижная точка функционала  $\Lambda$ , можно рассмотреть произвольную неподвижную точку  $\chi$  этого функционала и по индукции доказать, что  $\chi \subseteq \text{wlp}(P, \psi)_j$  для  $j = -1, 0, 1, 2, \dots$ . Для  $j = -1$  это очевидно; если  $\chi \subseteq \text{wlp}(P, \psi)_j$ , то в силу монотонности  $\Lambda$  имеет место  $\Lambda[\chi] \subseteq \Lambda[\text{wlp}(P, \psi)_j]$ , а это означает, что  $\chi \subseteq \text{wlp}(P, \psi)_{j+1}$ .

Наибольшие неподвижные точки функционалов обычно в литературе не рассматриваются, и главная причина этого в том, что такие точки в общем случае будут многозначными функциями. Но если множество значений рассматриваемых функций — одноэлементное, то многозначных функций вообще не может быть. Если мы обобщим понятие слабейшего свободного предусловия на постусловия, являющиеся произвольными функциями, то даже для однозначной функции  $g$  функция  $\text{wlp}(P, g)$  оказывается многозначной для тех  $v$ , для которых выполнение  $P$  не завершается,  $\text{wlp}(P, g)$  принимает все значения. Это находится в соответствии с понятием наибольшей неподвижной точки. Теорема 2 сохраняет силу для произвольных постусловий, равно как и формулы (5), (8), поскольку произвольную функцию  $g(v)$  со значениями в  $M$  можно разложить в объединение множеств ее уровней  $g_{(m)}(v)$ ,  $m \in M$ , где  $g_{(m)}(v) \subseteq V \times \{m\}$ , а каж-

дый уровень можно рассматривать как предикат. Для того чтобы можно было воспользоваться рекуррентным соотношением (5), надо взять в качестве приближения с номером  $-1$  полное бинарное отношение, или, другими словами, функцию, которая для каждого  $v$  принимает все значения.

Для циклической инструкции  $Q$

**while**  $a \neq 0$  **do**  $b := b + 1$ ;  $a := a - 1$  **od**,

в которой переменные  $a$ ,  $b$  считаются принимающими значения в множестве всех целых чисел, и для постусловия  $b$  приближение  $wlp(Q, b)_j$ , при  $j = -1, 0, 1, \dots$  равно  $a + b$ , если  $0 \leq a \leq j$ , и равно любому целому числу при  $a < 0$  и  $a > j$ . Само  $wlp(Q, b)$  есть многозначная функция, равная  $a + b$  при  $a \geq 0$  и равная любому целому числу при  $a < 0$ .

Слабейшее предусловие, в сравнении со слабейшим свободным предусловием, позволяет решать более сложную задачу: устанавливать свойство  $\{f\}P\{g\}$  и дополнительно устанавливать завершимость  $P$  для любого начального состояния, которое входит в область определения  $f$ . Но включение  $f \subseteq wlp(P, g)$  доказывать, вообще говоря, легче, чем включение  $f \subseteq wr(P, g)$ ; в первом случае достаточно индукцией по  $j$  доказать  $\forall j (f \subseteq wlp(P, g)_j)$ . После такого доказательства остается, конечно, неисследованной завершимость, однако приведенный выше пример (4) показывает, что формулы для  $wr$  тоже могут мало что давать для исследования завершимости; в итоге приведенные выше формулы для вычисления  $wlp$  могут оказаться более полезными, чем формулы Дейкстры для вычисления  $wr$ .

Простой пример применения  $wlp$ : докажем свойство  $\{k \cdot n!\} \text{ while } n > 0 \text{ do } k := k \cdot n; n := n - 1 \text{ od } \{k\}$ . (9)

Обозначим рассматриваемую циклическую инструкцию через  $P$ , а инструкцию, расположенную между **do** и **od**, через  $S$ . Включение  $kn! \subseteq wlp(P, k)_{-1}$  очевидно, так как  $wlp(P, k)_{-1} = V \times N$ . Включение же  $kn! \subseteq wlp(P, k)_{j+1}$  выводится из включения  $kn! \subseteq wlp(P, k)_j$ , исходя из монотонности функционала

$$\Lambda [h(n, k)] = \text{if } n > 0 \text{ then } wlp(P, h) \text{ else } k \text{ fi}$$

и из того, что  $kn!$  — неподвижная точка этого функционала (неважно, какая неподвижная точка: наибольшая, наименьшая или какая-либо еще). Однако доказатель-

ство оказывается столь простым в силу того, что предусловие (9) — неподвижная точка  $\Lambda$ ; оно останется столь же простым при любом предусловии  $f$  таком, что  $f \equiv \Lambda[f]$ , но это означает, что при доказательстве свойства методом Хоора само предусловие может быть взято инвариантом.

**Примечание 2.** Непосредственно проверяется, что для циклической инструкции  $P$  вида **while**  $\beta$  **do**  $S$  **od функция  $f$  служит инвариантом (т. е. имеет место свойство  $\{\beta \wedge f\} S \{f\}$ ) тогда и только тогда, когда  $f \equiv (\text{if } \beta \text{ then } \text{wlp}(S, f) \text{ else } f \text{ fi})$ ; аналогично  $f$  служит инвариантом, подходящим для доказательства свойства  $\{f\} S \{g\}$  (т. е. имеет место свойство  $\{\beta \wedge f\} P \{f\}$  и включение  $f \equiv g$ ), тогда и только тогда, когда  $f \equiv (\text{if } \beta \text{ then } \text{wlp}(S, f) \text{ else } g \text{ fi})$ .**

В общем случае для установления свойства  $\{f\} P \{g\}$  циклической инструкции  $P$  достаточно доказать, что для любого  $j \geq -1$  из

$$f \equiv \text{wlp}(P, g)_0, \quad f \equiv \text{wlp}(P, g)_1, \quad \dots, \quad f \equiv \text{wlp}(P, g)_j;$$

следует  $f \equiv \text{wlp}(P, g)_{j+1}$  (такая индукция может рассматриваться как индукция по неподвижной точке — полная вычислительная индукция или индукция по наибольшей неподвижной точке).

Рассмотрим несколько более сложный пример. Роль предусловий и постусловий в этом примере будут играть предикаты, и мы будем пользоваться импликациями, а не включениями и будем выписывать формулы с кванторами. Установим свойство программы  $R$

$$\{m = n\} \text{ while } k > 0 \text{ do } k := k - 1; n := n + (-1)^n \\ \text{od } \{|n - m| \leq 1\}. \quad (10)$$

Так как  $\text{wlp}(R, |n - m| \leq 1)_0 = ((k \neq 0) \vee (|n - m| \leq 1))$ , то импликация  $(m = n) \supset \text{wlp}(R, |n - m| \leq 1)_0$  очевидна. Далее нетрудно показать, что из  $(m = n) \supset \text{wlp}(R, |n - m| \leq 1)_{j-1}$  следует, что  $(m = n) \supset \text{wlp}(R, |n - m| \leq 1)_{j+1}$ : последовательность значений переменной  $n$  периодична с периодом 2, значение переменной  $m$  не меняется, и общая левая часть обеих импликаций не зависит от  $k$ . Завершимость программы  $R$  очевидна ввиду монотонного убывания значений переменной  $k$ .

Одновременное доказательство и свойства (10); и завершимости, проводимое при помощи слабейшего предусловия, состояло бы, скорее всего, из трех этапов:

1) выдвижение некоторой удачной гипотезы относительно вида  $\text{wp}(R, |n - m| \leq 1)$ ;

2) доказательство этой гипотезы с привлечением правил Дейкстры;

3) доказательство импликации  $(m = n) \supset \text{wp}(R, |n - m| \leq 1)$ .

Приведенное же выше доказательство свойства (9) было по трудности примерно эквивалентно этапу 2).

Свойство (10) можно было бы доказывать и методом Хоора, но это потребовало бы изобретения инварианта (заметим, что ни предусловие, ни постусловие свойства (10) не являются инвариантами, хотя постусловие  $|n - m| \leq 1$  и является индуктивным утверждением в смысле Флойда; такие эффекты подробно будут рассмотрены позднее, в гл. II). Метод Хоора — это индукция такого рода, когда из истинности утверждения для  $j$  выводят его истинность для  $j + 1$ , в рассматриваемой же задаче естественно сделать переход к  $j + 1$  от пары  $j - 1, j$ .

Итак, привлечение слабейшего свободного предусловия позволяет иногда доказывать свойство программы, не прибегая к усилению этого свойства (изобретение инварианта — это усиление доказываемого свойства путем ослабления предусловия и усиления постусловия). Известны и другие приемы, аналогичные в этом смысле привлечению слабейшего свободного предусловия. В работе [15] предложены записанные в виде аксиом правила вычисления сильнейшего постусловия по данному предусловию. Как и в случаях слабейшего и слабейшего свободного предусловий, здесь получается некоторое рекуррентное соотношение для последовательности предикатов  $\varphi_n$ , зависящих, однако, дополнительно от некоторой переменной  $y$ , принимающей значения в множестве состояний  $V$ . Сильнейшее постусловие задается формулой  $\exists n \exists y \varphi_n$ . Свойство  $\{\varphi\}P\{\psi\}$  циклической программы  $P$  эквивалентно импликации  $(\exists n \exists y \varphi_n) \supset \psi$ , которая в свою очередь эквивалентна  $\forall n \forall y (\varphi_n \supset \psi)$ . То, что при работе с сильнейшим постусловием возникает, в сравнении со слабейшим и слабейшим свободным предусловиями, «лишний» квантор, не выглядит неожиданным после рассмотрения сильнейшего постусловия относительно инструкции присваивания (формула (3), § 1). Исключение изобретения инварианта цикла из процесса доказательства свойства программы достигается и другими путями. Например, разрабатываются специальные методики верификации, рассчитанные на некоторые определенные виды программ. Так, в статье [11] анализируются типичные программы,

реализующие неитерационные методы линейной алгебры, и делается вывод, что эти программы могут быть написаны с использованием циклических инструкций лишь простейшего вида и таких операций над элементами матриц, что для вывода свойств многих программ можно будет применять специальные правила, не предполагающие изобретения инвариантов.

#### § 4. Не связанные с изменением состояния последствия выполнения программ

Для любой программы  $P$  и постусловия  $g$  имеет место равенство

$$\text{wlp}(P, g) = \text{wp}(P, g) \cup \text{wlp}(P, \emptyset). \quad (1)$$

Так же как и слабое предусловие, слабое свободное предусловие есть результат применения к постусловию некоторого сопоставляемого инструкции  $P$  преобразования функций. Но это преобразование уже не является сопряженным к  $p$  в выбранном для анализа программы множестве функций. Многочисленные преобразования такого рода представляют определенный интерес как средства анализа программ. Рассмотрим некоторую общую схему, в которую укладывается и слабое свободное предусловие (1) и многие другие преобразования функций. С каждой инструкцией  $S$  связывается некоторое преобразование  $H_S$  выбранного множества функций. Далее, каждой инструкции  $S$  сопоставляется преобразование  $S^H$ , определяемое формулой

$$S^H = H_S(s^*(g)). \quad (2)$$

На выбор преобразований вида  $H_S$  мы накладываем единственное ограничение, заключающееся в том, что определяемые с помощью  $H_S$  преобразования  $S^H$  должны подчиняться следующим соотношениям: для любых инструкций  $S_1, S_2$  и составной инструкции  $S_1; S_2$  выполнено

$$(S_1; S_2)^H = S_2^H S_1^H,$$

при этом композиция (произведение)  $S_2^H S_1^H$  понимается так: сначала  $S_2^H$ , потом  $S_1^H$ , и выписанное соотношение означает, что для любой функции  $g$  из соответствующего множества выполнено

$$(S_1; S_2)^H(g) = S_1^H(S_2^H(g)).$$

Если все соотношения такого вида выполнены, то каждое  $H_s$  будем называть *операционным смещением*, связанным с  $S$ , или просто *операционным смещением*.

**Примечание 1.** Рассмотрим мотивы, по которым при конструировании семантик мы из совершенно произвольных преобразований функций выделяем такие, которые являются композициями сопряженных преобразований и некоторых операционных смещений. Во многих случаях мы при анализе программы хотим исследовать накопление определенных последствий выполнения в должной очередности всех инструкций программы.

Пусть  $S$  — некоторая инструкция программы. Применение преобразования  $s^*$  к функции  $g$  не является каким-либо искажением функции  $g$ , а является всего лишь переходом к еще не измененным (после выполнения  $S$ ) значениям переменных (см. формулу (4) из § 1). В силу этого применение композиции  $s^*$  и операционного смещения, связанного с  $S$ , представляет собой добавление информации, которую несет операционное смещение о последствиях выполнения  $S$ , к неискаженной информации о последствиях дальнейшего выполнения инструкций программы.

В (2) использовано обозначение  $S^H$ , а не  $s^H$ , и, точно так же,  $H_s$ , а не  $h_s$ . Этим мы подчеркиваем, что определяемое преобразование сопоставляется самой инструкции: из того что двум текстуально различающимся инструкциям  $S_1$  и  $S_2$  соответствует одно и то же преобразование  $s$  множества состояний  $V$ , еще, вообще говоря, не следует, что  $H_{S_1} = H_{S_2}$  и  $S_1^H = S_2^H$ .

Рассмотрим подробнее важный частный случай операционного смещения. С каждой инструкцией  $S$  связывается функция  $h_s$ , принадлежащая выбранному множеству функций на  $V$  со значениями в  $M$ . В этом множестве определяется некоторая бинарная операция  $\nabla$ . Для каждой инструкции  $S$  вводится  $S^\nabla$  — преобразование выбранного множества функций: пусть  $g$  — некоторая функция, тогда, по определению,

$$S^\nabla(g) = s^*(g) \nabla h_s. \quad (3)$$

Для того чтобы преобразование  $\nabla h_s$  было операционным смещением, порождаемые преобразования  $S^\nabla$  должны удовлетворять соотношениям вида

$$(S_1; S_2)^\nabla = S_2^\nabla S_1^\nabla; \quad (4)$$

позднее будут даны некоторые достаточные условия выполнимости этих соотношений. А сейчас приведем примеры, когда по схеме (3) описываются интересные преобразования (один пример уже имеется — это слабейшее свободное предусловие, для которого  $\nabla$  — это  $\cup$ , а  $h_s$  —

это  $wlp(S, \emptyset)$ ). Пример операционного смещения, которое имеет отличный от  $\nabla h_s$  вид, разбирается в § 6.

Попятна важность задачи точного определения или получения оценки числа затраченных операций при выполнении программы, например числа умножений при вычислении  $a^n$ , числа сравнений при сортировке и т. д. Выполнение операции не влечет за собой изменения состояния, и это вносит трудности в анализ. Здесь можно определить по схеме (3) преобразование частичных функций на  $V$  с целочисленными неотрицательными значениями. Операцией  $\nabla$  будет операция сложения, функция-сумма определена на пересечении множеств определения функций-слагаемых. Если  $S$  — инструкция присваивания, то  $h_s$  — это константа, равная числу вхождений знака исследуемой операции в выражение, задающее правую часть  $S$  (мы допускаем в качестве правых частей лишь безусловные выражения). Если  $S$  — инструкция вида

$$\text{if } \beta \text{ then } P \text{ else } Q \text{ fi}, \quad (5)$$

то

$$h_s(v) = h_{t:=\beta}(v) + (\text{if } \beta \text{ then } h_p(v) \text{ else } h_q(v) \text{ fi}),$$

здесь  $h_{t:=\beta}$  учитывает те операции, которые выполняются при определении значения  $\beta$ . Соответственно можно написать и

$$S^+(g) = h_{t:=\beta}(v) + (\text{if } \beta \text{ then } P^+(g) \text{ else } Q^+(g) \text{ fi}).$$

Для пустой инструкции  $S$ , очевидно,  $h_s = 0$ ,  $S^+(g) = g$ .

Рассмотрим теперь случай циклической инструкции  $S$  вида

$$\text{while } \beta \text{ do } P \text{ od}. \quad (6)$$

Мы не будем здесь выписывать отдельно  $h_s$ , а займемся сразу  $S^+(g)$ . При этом мы опять, как и в § 3, пойдем путем описания последовательных приближений  $S^+(g)_0$ ,  $S^+(g)_1$ , ..., смысл которых таков:  $S^+(g)_j(v) = m$  означает, что для начального состояния  $v$  выполнение циклической инструкции завершается не позднее чем через  $j$  этапов в некотором заключительном состоянии  $v_1$  и при этом затрачивается  $m - g(v_1)$  (т. е.  $m - s^*(g)(v)$ ) подсчитываемых операций. Последовательность приближений подчинена рекуррентному соотношению:

$$\begin{aligned} S^+(g)_0 &= \text{if } \neg\beta \text{ then } g \text{ fi} = \neg\beta \wedge g, \\ S^+(g)_j &= \text{if } \beta \text{ then } P^+(S^+(g)_{j-1}) \text{ else } g \text{ fi}. \end{aligned} \quad (7)$$

Для определенных таким способом приближений выпол-

нено  $S^+(g)_0 \subseteq S^+(g)_1 \subseteq \dots$  и

$$S^+(g) = \bigcup_{j=0}^{\infty} S^+(g)_j.$$

С помощью предложенной семантики можно, например, показать, что для вычисления  $a^n$  по программе  $W$ :

$c := 1; b := a;$

**while**  $n > 0$  **do**

$l := \text{остаток}(n, 2); n := [n/2];$

**if**  $l = 1$  **then**  $c := c \cdot b$  **else fi**;

$b := b \cdot b$

**od**

потребуется не более  $2\Pi_2(n)$  умножений (через  $\Pi_2(n)$  обозначено число цифр в двоичной записи  $n$ ). Достаточно установить, что  $2\Pi_2(n) \geq W^+(0)(a, b, c, l, n)$ , где  $0$  — это функция, тождественно равная нулю. Привлекая приведенные выше формулы и индукцию, докажем для циклической инструкции  $\bar{W}$ , входящей в программу  $W$ , что при  $j = 0, 1, \dots$  на множестве определения  $\bar{W}^+(0)(a, b, c, l, n)$  имеет место  $2\Pi_2(n) \geq \bar{W}^+(0)_j(a, b, c, l, n)$ . С целью упрощения записи примем для  $\bar{W}^+(0)(a, b, c, l, n)$  обозначение  $f_j(n, \dots)$ .

Неравенство  $2\Pi_2(n) \geq f_0(n, \dots)$  очевидно. Предположим, что выполнено  $2\Pi_2(n) \geq f_{j-1}(n, \dots)$ . По формуле (5) получается  $f_j(n, \dots) = \text{if } n > 0 \text{ then } f_{j-1}([n/2], \dots) + d \text{ else } 0 \text{ fi}$ , где  $d$  — это 1 или 2. Случай  $n = 0$  тривиален. Пусть  $n > 0$ , тогда имеем  $f_j(n, \dots) \leq f_{j-1}([n/2], \dots) + 2$ . По предположению индукции  $f_{j-1}([n/2], \dots) \leq 2\Pi_2([n/2])$ , но  $\Pi_2([n/2]) = \Pi_2(n) - 1$ . В итоге получаем  $2\Pi_2(n) \geq f_j(n, \dots)$ , что и требовалось.

Еще один пример — определяемая по схеме (3) семантика инструкции вывода. Здесь мы будем иметь дело с частичными функциями, значения которых представляют собой последовательности (упорядоченные конечные наборы) величин, принадлежащих к рассматриваемым в программе типам. В качестве  $\nabla$  будет выступать операция  $\cup$  — конкатенация значений функций (результат определен на пересечении множеств определения исходных функций).

Если  $S$  — инструкция присваивания или пустая инструкция, то  $h_S$  есть функция, принимающая для каждого



$v \in V$  значение, равное пустой последовательности (эта функция здесь будет обозначаться через  $e(v)$  или просто через  $e$ ). Если  $S$  — инструкция вывода значения простой переменной, т. е. инструкция  $output(x)$ , то  $h_s$  есть функция, равная последовательности, состоящей из одного-единственного элемента — значения переменной  $x$ . Для инструкции вида (5) получается

$$h_s = \text{if } \beta \text{ then } h_p \text{ else } h_q \text{ fi,}$$

$$S^{\sim}(g) = \text{if } \beta \text{ then } P^{\sim}(g) \text{ else } Q^{\sim}(g) \text{ fi.}$$

Формулы для инструкции  $S$ , имеющей вид (6), получаются из формулы (7) заменой верхнего значка  $+$  на  $\sim$ . Эти новые формулы позволяют, например, доказывать утверждения такого рода: все значения, выводимые в ходе выполнения программы, обладают свойством  $\varphi$ ; выводимые в ходе выполнения программы значения образуют всю совокупность значений, обладающих свойством  $\varphi$ . Так, относительно программы

$$k := 1; \text{ while } k | n \text{ do } output(k); k := 2k \text{ od} \quad (8)$$

может быть доказано, что все выводимые значения суть делящие  $n$  степени двойки, или более сильное утверждение — что выводятся все степени двойки, делящие  $n$ , и только они. Обозначим через  $U$  циклическую инструкцию, входящую в программу (8). Достаточно доказать индукцией по  $j$ , что если функция  $U^{\sim}(e)_j$  определена для некоторых  $n$  и  $k$ , то ее значением в этом случае служит последовательность всех делителей числа  $n$ , имеющих вид  $2^m k$ ,  $0 \leq m < j$ . Индуктивный переход проводится исходя из формулы

$$U^{\sim}(e)_j = \text{if } k | n \text{ then } P^{\sim}(U^{\sim}(e)_{j-1}) \text{ else } e \text{ fi,} \quad (9)$$

где  $P$  есть составная инструкция  $output(k); k := 2k$ .

В случае, когда  $k$  делит  $n$  и значение  $U^{\sim}(e)_j(n, k)$  определено, это последнее значение, как видно из (9), получается из значения  $U^{\sim}(e)_j(n, 2k)$  присоединением  $k$ ; ввиду индуктивного предположения, отсюда следует все, что нужно.

**Примечание 2.** Обращает на себя внимание то, что, с одной стороны, при описании схемы (2) мы взяли за образец слабейшее свободное предположение, а с другой стороны, для построенных по этой схеме  $S^+$  и  $S^{\sim}$  в том случае, когда  $S$  — циклическая инструкция, нам удалось воспользоваться формулами, которые, можно

сказать, являются повторениями формул для слабейшего предусловия. При этом для слабейшего свободного предусловия такое повторение невозможно, и в предыдущем параграфе предлагались формулы совсем другого вида. Все дело в том, что между  $S^+$  и  $S^-$ , с одной стороны, и  $wlp(S, \dots)$  — с другой, есть различие: когда выполнение  $S$  не завершалось для начального состояния  $v$ , мы считали  $S^+(g)$  и  $S^-(g)$  неопределенными для  $v$ , но такой взгляд не мог быть принят при рассмотрении слабейшего предусловия.

Приведенные примеры показывают, что по схеме (2) можно, в частности, определять семантики, позволяющие обнаруживать и исследовать такие эффекты выполнения программ, которые не связаны с изменением состояния или не определяются полностью заключительным состоянием. Можно подобным образом задать преобразования функций, помогающие исследовать, например, число обращений к некоторой переменной (присваиваний ей значения и вычислений ее значения) и т. д. Для проведения такого рода исследований нет необходимости в изменении операционной семантики языка.

Можно определить и другие семантики, сходные с семантикой инструкции вывода, суть которых состоит в собирании всех значений, принимаемых некоторыми или всеми переменными в ходе выполнения программы. Одновременное рассмотрение всех таких значений оказывается полезным при анализе многих программ. Пусть  $a, b, \dots$ ,  $x$  — переменные, участвующие в некоторой программе  $S$ , тогда в качестве множества  $M$  можно взять множество списков вида  $(L_a, L_b, \dots, L_x)$ , где  $L_a, L_b, \dots, L_x$  — в свою очередь списки значений переменных  $a, b, \dots, x$ . Формулы, описывающие преобразования функций со значениями указанного вида, аналогичны формулам, определяющим семантику вида  $S^-$ . Рассмотрим программу  $Q$ :

$$\text{while } a > 0 \text{ do } b := b + 1; a := a - 1 \text{ od,}$$

где  $a, b$  — переменные, принимающие целочисленные неотрицательные значения. Применяя сопоставляемое этой программе преобразование функций к постоянной функции, значение которой есть список  $((), ())$ , мы получим функцию, значение которой на состоянии, определяемом значением переменной  $a$ , равном  $A$ , и значением переменной  $b$ , равном  $B$ , есть  $((A, A - 1, \dots, 1), (B, B + 1, \dots))$ . При этом количество элементов в списках  $L_a$  и  $L_b$ , т. е. в  $(A, A - 1, \dots, 1)$  и  $(B, B + 1, \dots)$ , будет одним и тем же.

Исходя из этого, легко доказывается, например, свойство  $\{a > c\}Q\{b > c\}$ . Мы видим, что первый элемент списка  $L_b$  не меньше последнего элемента списка  $L_a$ , второй элемент  $L_b$  не меньше предпоследнего элемента  $L_a$ , и далее, по индукции, что последний элемент  $L_b$  не меньше первого элемента  $L_a$ . Но последний элемент  $L_b$  — это итоговое значение  $b$ , а про первый элемент  $L_a$  известно, что он больше значения  $c$ , откуда следует все, что нужно. В доказательстве применена, в частности, индукция, которая более сложна, чем простейшая индукция, на которой основывается метод Хоора. Программа  $Q$  и свойство  $\{a > c\}Q\{b > c\}$  будут еще рассматриваться в гл. II.

Рассмотренную серию разнообразных семантик вида  $S^\nabla$  следует, однако, признать в известном смысле непоказательной, так как, по существу, ни разу не возникло необходимости отдельного рассмотрения функций вида  $h_s$ , а вместо этого сразу применялись несложные формулы для  $S^\nabla$ . Разберем еще пример. Он интересен не только тем, что в нем демонстрируется ситуация параллельного рассмотрения  $s^*(g)$  и  $h_s$ , но и тем, что определяемые в этом примере преобразования вида  $S^\nabla$  позволяют изучать последствия бесконечного («зацикленного») выполнения программ, что вообще невозможно сделать средствами семантик Хоора и Дейкстры.

Пусть для некоторого начального состояния выполнение циклической инструкции никогда не завершается и пусть внутри этой циклической инструкции содержится инструкция вывода. Каким образом мы можем исследовать те значения, которые будут выведены в процессе выполнения программы? Покажем, как с этой целью определяются для различных инструкций  $S$  соответствующие функции  $h_s$ , и зададим операцию  $\nabla$ . Этот пример прояснит и общий подход к исследованию различных характеристик бесконечного выполнения программ.

В этом примере будут использованы однозначные частичные функции, значения которых суть последовательности (может быть, бесконечные) величин, принадлежащих к рассматриваемым в программе типам данных, — мы для определенности ограничимся последовательностями положительных целых чисел. В качестве  $\nabla$  возьмем  $\equiv$  — несколько измененную операцию конкатенации значений функций: будем считать, что результат операции  $f \equiv g$  определен на объединении множеств определения функций  $f$  и  $g$ : если при значении аргумента  $v_0$  опреде-

лено значение только одной из функций, то это значение будет и значением функции  $f \doteq g$  для аргумента  $v_0$ . Если  $f(v_0)$  — бесконечная последовательность, то, по определению,  $(f \doteq g)(v_0) = f(v_0)$ .

Ниже мы определим функции вида  $h_s$  и, пользуясь формулой

$$S^{\sim}(g) = h_s \doteq s^*(g), \quad (10)$$

преобразования вида  $S^{\sim}$ . В использовании формулы (10) не будет принципиального отхода от схемы (3), хотя  $h_s$  и выступает в правой части этой формулы в качестве первого операнда, а  $s^*(g)$  — второго. Ничего не стоило бы формально определить для любых  $f, h$  операцию  $\doteq$ :  $f \doteq h = h \doteq f$ . После этого определения можно было бы уже буквально следовать схеме (3). Однако операция  $\doteq$  более естественна, и мы будем придерживаться формулы (10).

Пусть  $S$  — инструкция присваивания, или пустая инструкция, тогда  $h_s = \emptyset$ . Пусть  $S$  — инструкция вывода  $output(x)$ , тогда  $h_s$  — это такая функция, что  $h_s(v_0)$  будет последовательностью из одного элемента, равного значению переменной  $x$  на состоянии памяти, равном  $v_0$ . Для инструкции  $S$  вида (4) полагаем

$$h_s = \text{if } \beta \text{ then } h_p \text{ else } h_q \text{ fi,}$$

откуда

$$S^{\sim} = \text{if } \beta \text{ then } P^{\sim} \text{ else } Q^{\sim} \text{ fi.}$$

Теперь займемся циклической инструкцией вида (5). Выписанные в рассмотренных выше примерах формулы для  $S^+$ ,  $S^{\sim}$ , а также выведенная в предыдущем параграфе формула слабейшего свободного предусловия относительно циклической инструкции  $S$  вида (5) не учитывают различные особенности бесконечного выполнения: подробно описывается лишь то, что происходит, если выполнение (5) завершается после одного шага, двух шагов и т. д. В этих примерах функция  $h_s$  — это  $S^+(0)$ ,  $S^{\sim}(e)$  или  $S^U(\emptyset)$ . Вычисление же  $S^+(0)$ ,  $S^{\sim}(e)$  и  $S^U(g)$  принципиально не отличается от вычисления  $S^+(g)$ ,  $S^{\sim}(g)$ ,  $S^U(g)$  для произвольной функции  $g$  соответствующего вида, поэтому и не возникало необходимости в отдельном вычислении операционного смещения, т. е.  $h_s$ .

В рассматриваемом примере вычислять  $S^{\sim}(\emptyset)$  существенно проще, чем  $S^{\sim}(g)$  для произвольной  $g$ . Это происходит вот по какой причине. Если для некоторого состояния  $v_0$  выполнение  $S$  не завершается, то  $S^{\sim}(g)(v_0) = h_S(v_0)$  и информация, которую несет функция  $g$ , не играет никакой роли; если же выполнение  $S$  завершается, то эта информация существенна. В случае  $g = \emptyset$  отмеченная «дискретность» перестает быть принципиальной, т. к.  $s^*(\emptyset) = \emptyset$  независимо от того, для каких начальных состояний выполнение  $S$  завершается, а для каких — нет.

Функцию  $h_S(v)$  для  $S$  вида (5) мы выразим через последовательные приближения  $h_{S,0}(v)$ ,  $h_{S,1}(v)$ , ... Определим, что понимается под значением  $h_{S,k}(v_0)$ . Выполнение  $S$  для начального состояния  $v_0$  может, с одной стороны, завершиться не позднее чем через  $k$  этапов, а с другой — завершиться позднее или не завершиться вовсе. Положим  $l_{S,k}(v_0)$  в первом случае равным числу этапов выполнения  $S$ , а во втором — равным  $k$ . Равенство  $h_{S,k}(v_0) = (A, B, \dots, C)$ , где  $A, B, \dots, C$  — положительные целые числа, будет иметь место в том и только в том случае, если выполнение  $l_{S,k}(v_0)$  этапов инструкции  $S$  для начального состояния  $v_0$  приводит к выводу последовательности  $A, B, \dots, C$ . Если в течение  $l_{S,k}(v_0)$  этапов ничего выведено не было, то считаем значение  $h_{S,k}(v_0)$  неопределенным. Из этого определения значения  $h_{S,k}(v_0)$  следует рекуррентное соотношение

$$\begin{aligned} h_{S,0} &= \emptyset, \\ h_{S,k} &= \text{if } \beta \text{ then } h_P \preceq p^*(h_{S,k-1}) \text{ else } \emptyset \text{ fi} = \\ &= \beta \wedge (h_P \preceq p^*(h_{S,k-1})). \end{aligned} \quad (11)$$

Очевидно, что включение  $h_{S,k} \subseteq h_{S,k+1}$  для  $k = 0, 1, \dots$  выполнено в том смысле, что если для некоторого  $v_0$  определено значение  $h_{S,k}(v_0)$ , то определено и значение  $h_{S,k+1}(v_0)$ , и последовательность  $h_{S,k}(v_0)$  является начальным отрезком последовательности  $h_{S,k+1}(v_0)$ . Для последовательности  $h_{S,0}, h_{S,1}, \dots$  естественно определяется наименьшая верхняя грань, которую мы обозначим  $\sup_{k=0}^{\infty} h_{S,k}$ . Мы имеем для  $S$  вида (5)

$$h_S = \sup_{k=0}^{\infty} h_{S,k}.$$

Применим предложенный метод к исследованию программы  $Z$ :

$m := n^2$ ;

while  $n > 1$  do if  $n > 4$  then output ( $n$ ) else fi;

if  $2 \mid n$  then  $n := (n/2) + 1$

else  $n := n + 1$  fi

od;

while  $m > 1$  do output ( $m$ );  $m := m - 1$  od.

Докажем, что для любого  $n > 1$  в ходе выполнения  $Z$  либо ничего не выводится, либо выводится не более чем  $n + 1$  число. Входящие в  $Z$  циклические инструкции обозначим через  $Z_1$  и  $Z_2$ . Надо исследовать последовательности, являющиеся значениями функции  $Z^{\sim}(\emptyset)$ . Для этого изучим функцию  $(Z_1; Z_2)^{\sim}(\emptyset)$ . Эта функция зависит от  $n$  и  $m$ , нас интересуют ее значения при  $m = n^2$ . Воспользуемся равенствами

$$(Z_1; Z_2)^{\sim}(\emptyset) = Z_1^{\sim}(Z_2^{\sim}(\emptyset)) = Z_1^{\sim}(h_{Z_2}) = h_{Z_1} \approx z_1^*(h_{Z_2}).$$

Прежде всего вычислим  $h_{Z_2}(m)$ . Имеем последовательность приближений

$$h_{Z_2,0} = \emptyset,$$

$$h_{Z_2,k}(m) = (m > 1) \wedge ((m) \approx h_{Z_2,k-1}(m - 1)).$$

Из этого рекуррентного соотношения по индукции выводится, что для  $k = 1, 2, \dots$

$$h_{Z_2,k}(m) = \text{if } m > k \text{ then } (m, m - 1, \dots, m - k + 1) \\ \text{else } (m, m - 1, \dots, 1) \text{ fi}$$

и как следствие, что  $h_{Z_2}(m) = (m, m - 1, \dots, 1)$ ; это дает  $h_{Z_2}(n^2) = (n^2, n^2 - 1, \dots, 1)$ .

Теперь надо вычислить  $h_{Z_1} \approx z_1^*(g)$ , где  $g(n) = h_{Z_2}(n^2)$ . Вычисления, проведенные в § 3 в связи с программой  $Q$ , показывают, что выполнение  $Z_1$  завершается только для  $n = 1$ , и, таким образом,

$$z_1^*(g) = (n = 1) \wedge g(1),$$

так как  $z(1) = 1$ . Но нас интересует только случай  $n > 1$ , так что функцией  $z_1^*(g)$  можно пренебречь и исследовать

только  $h_{Z_1}(n)$ . Последовательность приближений

$$h_{Z_1,0}(n), h_{Z_1,1}(n), h_{Z_1,2}(n), \dots$$

описывается рекуррентно:

$$h_{Z_1,0}(n) = \emptyset,$$

$$h_{Z_1,k}(n) = (n > 1) \wedge ((n > 4) \wedge ((n) \subset h_{Z_1,k-1}(n'))) = \quad (12) \\ = (n > 4) \wedge ((n) \subset h_{Z_1,k-1}(n')),$$

где  $n' = \text{if } 2 \mid n \text{ then } \frac{n}{2} + 1 \text{ else } n + 1 \text{ fi}$ .

Докажем следующую лемму.

Лемма 1. Пусть  $n \geq 1$  и  $k \geq n - (-1)^n$ . Тогда

$$h_{Z_1,k}(n) = h_{Z_1,n-(-1)^n}(n). \quad (13)$$

Доказательство. Проведем индукцией по величине  $n - (-1)^n$ . Если  $n - (-1)^n \leq 4$ , то, как нетрудно проверить,  $n \leq 4$ ; но при  $n \leq 4$  равенство (13) справедливо, т. к. обе части этого равенства не определены. Это служит основанием индукции. Индуктивный переход проводится исходя из рекуррентных формул (12). Пусть  $n > 4$  и  $n' = \text{if } 2 \mid n \text{ then } \frac{n}{2} + 1 \text{ else } n + 1 \text{ fi}$ , тогда  $h_{Z_1,n-(-1)^n}(n)$  и  $h_{Z_1,k}(n)$  выражаются одним и тем же способом через  $h_{Z_1,n-(-1)^{n-1}}(n')$  и  $h_{Z_1,k-1}(n')$  соответственно. Но при  $n > 4$  выполнено неравенство  $n - (-1)^n > n' - (-1)^{n'}$  (устанавливается разбором двух случаев:  $n$  четно и  $n$  нечетно), откуда следует, что  $k - 1 \geq n - (-1)^n - 1 \geq n' + (-1)^{n'}$  и, по предположению индукции, что

$$h_{Z_1,n-(-1)^{n-1}}(n') = h_{Z_1,k-1}(n') = h_{Z_1,n'-(-1)^{n'}}(n').$$

Лемма 1 обеспечивает доказательство того, что количество элементов в последовательности, являющейся значением  $h_{Z_1}(n)$ , не превзойдет  $n - 3$ ; по формуле

$$h_{Z_1}(n) = \sup_{k=0}^{\infty} h_{Z_1,k}(n)$$

получаем, что  $h_{Z_1}(n) = h_{Z_1,n-(-1)^n}(n)$ . Далее, исходя из формул (12), простой индукцией устанавливаем, что последовательность, являющаяся значением  $h_{Z_1,k}(n)$ , содержит не более  $k$  элементов, и, наконец, замечаем, что  $n - (-1)^n \leq n + 1$ .

Проведенное изучение программы  $Z$  не включало в себя исследования самих выведенных чисел — исследовалось только их количество. Приведем пример более подробного исследования выведенных значений. Рассмотрим программу  $Z_3$ , сходную с  $Z_1$ , выполнение которой дает при  $n > 1$  бесконечную последовательность выведенных чисел:

while  $n > 1$  do output ( $n$ );  
           if  $2 | n$  then  $n := \frac{n}{2} + 1$  else  $n := n + 1$  fi  
           od.

Докажем, что при начальном значении  $n$ , равном 2, эта бесконечная последовательность состоит из одних двоек, а при всех начальных значениях  $n$ , больших 2, эта бесконечная последовательность, начиная с элемента с некоторым номером, не превосходящим  $n + 1$ , состоит только из троек и четверок. От общих формул (11) переходим к формулам, связанным с программой  $Z_3$ :

$$\begin{aligned} h_{Z_3,0}(n) &= \emptyset, \\ h_{Z_3,k}(n) &= (n > 1) \wedge ((n) \simeq h_{Z_3,k-1}(n')), \end{aligned} \tag{14}$$

где  $n' = \text{if } 2 | n \text{ then } \frac{n}{2} + 1 \text{ else } n + 1 \text{ fi}$ .

Непосредственным применением формул (14) доказывается следующая лемма.

**Лемма 2.** Для  $n = 3$ , 4 и  $k > 0$  последовательность, являющаяся значением  $h_{Z_3,k}(n)$ , состоит только из троек и четверок, а для  $n = 2$  и  $k > 0$  эта последовательность состоит только из двоек.

Ясно, что при  $n > 1$  и при любом  $k$  последовательность, являющаяся значением  $h_{Z_3,k}(n)$ , содержит ровно  $k$  элементов. Далее имеет место следующая лемма.

**Лемма 3.** Пусть  $n > 3$  и  $k \geq n - (-1)^n$ . Тогда значением  $h_{Z_3,k}$  служит последовательность, все элементы которой, начиная с элемента с некоторым номером, не превосходящим  $n - (-1)^n$ , суть тройки и четверки.

**Доказательство.** Проведем индукцию по величине  $n - (-1)^n$ . Основание индукции обеспечивается первой частью леммы 2. Индуктивный переход при  $n - (-1)^n > 4$  здесь аналогичен индуктивному переходу в доказательстве леммы 1. Корректность индукции обосновывается так; если  $n - (-1)^n > 4$ , то  $n - (-1)^n >$



$> n' - (-1)^{n'}$ . Случай  $n' - (-1)^{n'} = 1$  невозможен ни при каком  $n$ , случай  $n' - (-1)^{n'} = 2$  возможен только при  $n = 1$  или  $n = 2$ , но тогда  $n - (-1)^n < 4$ .

Теперь, поскольку  $h_{Z_3}(n) = \sup_{h=0} h_{Z_3, h}(n)$ , мы на основании второй части леммы 2, леммы 3 и неравенства  $n + 1 > n - (-1)^n$  можем утверждать, что последовательность, являющаяся значением  $h_{Z_3}(n)$ , при  $n = 2$  состоит из одних двоек, а при  $n \geq 3$ , начиная с элемента с некоторым номером, не превосходящим  $n + 1$ , состоит только из троек и четверок.

Примечание 3. Выведенная в исследованиях программ  $Z$  и  $Z_3$  оценка, имеющая вид  $n + 1$  (или  $n - (-1)^n$ ), является слишком грубой. Мы привели эту оценку лишь из-за простоты связанного с ней доказательства. Пусть  $n$  таково, что четыре последовательных преобразования  $n$  по формуле  $\text{if } 2 \mid n \text{ then } \frac{n}{2} + 1 \text{ else } n + 1 \text{ fi}$  приводят к  $n''''$  такому, что  $n'''' > 4$ . Тогда можно показать, что  $n > n''''/2$ . На основании этого с помощью рекуррентных формул (12), (14) можно провести доказательства, сходные с теми, которые были проведены выше, с заменой, однако, величины  $n - (-1)^n$  на некоторую другую величину, имеющую асимптотическое представление  $4 \log_2 n + O(1)$ .

## § 5. Структурное операционное смещение; доказательства хооровского типа

Вернемся к общим вопросам, касающимся семантик вида  $S^H$ :

$$S^H(g) = H_s(s^*(g)) \quad (1)$$

и, в частности, семантик вида  $S^\nabla$ :

$$S^\nabla(g) = \underline{s}^*(g) \nabla h_s. \quad (2)$$

Прежде всего укажем некоторые естественные условия, достаточные для того, чтобы участвующее в (2) преобразование функции  $s^*(g)$  было операционным смещением, т. е. для того, чтобы для любых инструкций  $S_1, S_2$  и составной инструкции  $S_1; S_2$  имело место

$$(S_1; S_2)^\nabla = S_2^\nabla S_1^\nabla, \quad (3)$$

где композиция (произведение) преобразований понимается, как и прежде, так: сначала  $S_2^\nabla$ , потом  $S_1^\nabla$ .

Теорема. Пусть  $\nabla$  — ассоциативная операция в множестве функций и пусть для любых функций  $f, g$  и

любой инструкции  $S$  выполнено  $s^*(f \nabla g) = s^*(f) \nabla s^*(g)$ . Пусть, наконец, для любых инструкций  $S_1, S_2$  и составной инструкции  $S_1; S_2$  выполнено  $h_{S_1; S_2} = s_1^*(h_{S_2}) \nabla h_{S_1}$ .

Тогда имеет место (3).

Доказательство. Для произвольной функции  $g$  имеем

$$\begin{aligned}(S_1; S_2)^\nabla(g) &= (s_1 s_2)^*(g) \nabla h_{S_1; S_2} = \\ &= (s_2^* s_1^*)(g) \nabla (s_1^*(h_{S_2}) \nabla h_{S_1}) = \\ &= ((s_2^* s_1^*)(g) \nabla s_1^*(h_{S_2})) \nabla h_{S_1} = \\ &= s_1^*(s_2^*(g) \nabla h_{S_2}) \nabla h_{S_1} = S_2^\nabla S_1^\nabla(g).\end{aligned}$$

Примечание 1. Условие  $s^*(f \nabla g) = s^*(f) \nabla s^*(g)$  оказывается выполненным всякий раз, когда операция  $\nabla$  первоначально определена в множестве значений функций, а затем перенесена на сами функции поточечно. Условие  $h_{S_1; S_2} = s_1^*(h_{S_2}) \nabla h_{S_1}$  в свою очередь означает, что последствия применения операционных смещений (т. е. добавки вида  $h_{S_1}, h_{S_2}$ ) накапливаются без искажений при переходе от отдельных инструкций к составной. В примерах предыдущего параграфа накапливались число операций, сделанных при выполнении отдельных инструкций, и величины, выведенные при выполнении отдельных инструкций.

Для того чтобы исследование всей программы легко сводилось к исследованию ее структурных компонент, желательно, чтобы преобразования функций, индуцируемые инструкциями языка программирования по схеме (1) или более частной схеме (2), удовлетворяли дополнительно к (3) еще ряду соотношений. Эти соотношения мы сейчас перечислим.

Пустой инструкцией индуцируется тождественное преобразование, т. е. для любой функции  $g$  выполняется

$$^H(g) = g. \quad (4)$$

Для вычисления в состоянии  $v_0$  значения функции

$$\text{if } \beta \text{ then } P \text{ else } Q \text{ fi } ^H(g)$$

берется значение в  $v_0$  одной из функций  $g_P = P^H(g)$ ,  $g_Q = Q^H(g)$ , а именно берется значение

$$\text{if } \beta(v_0) \text{ then } g_P(v_0) \text{ else } g_Q(v_0) \text{ fi}. \quad (5)$$

Для циклической инструкции и произвольной функ-

ции  $g$  должно иметь место

$\text{while } \beta \text{ do } P \text{ od } {}^H(g) =$

$$= \text{while } \beta \text{ do } P \text{ od}^H(\text{if } \beta \text{ then } P \text{ else fi}^H(g)). \quad (6)$$

Если соотношения (4), (5), (6) выполнены, то каждое операционное смещение  $H_s$ , определяющее для соответствующей ему инструкции  $S$  преобразование  $S^H$ , мы назовем *структурным операционным смещением*.

Будем рассматривать такие последствия выполнения программ, которые описываются структурными операционными смещениями. При условии завершенности выполнения программ справедливость многих утверждений о таких последствиях можно доказывать с помощью системы правил, аналогичной системе Хоора. Исходным материалом для построения такой системы правил должно служить некоторое отношение (которое нам заменит импликацию) частичного порядка  $\geq$  в том множестве  $M$ , в котором принимают значения функции, привлекаемые для анализа программ. Отношение  $\geq$  мы перенесем на однозначные частичные функции, считая, что  $f_1 \geq f_2$ , если область определения  $f_2$  включает в себя область определения  $f_1$ , и для каждого  $v$  из области определения  $f_1$  выполнено  $f_1(v) \geq f_2(v)$ . Обязательное условие того, чтобы система правил могла быть построена, состоит в монотонности  $P^H$ : для любых функций  $f_1, f_2$  таких, что  $f_1 \geq f_2$ , и любой программы  $P$  должно иметь место

$$P^H(f_1) \geq P^H(f_2). \quad (7)$$

В свойство  $\langle f \rangle P \langle g \rangle$ , где  $f$  и  $g$  — функции из выбранного множества, а  $P$  — программа, мы будем вкладывать следующий смысл:

$$f' \geq P^H(g), \quad (8)$$

при этом  $f'$  получается из  $f$  сужением области определения —  $f'$  не определена для тех состояний, для которых выполнение  $P$  не завершается. Возможность применения к свойствам вида  $\langle f \rangle P \langle g \rangle$  правил, аналогичных правилам Хоора, следует из соотношений (4) — (7).

Мы продемонстрируем применение техники работы со свойствами вида  $\langle f \rangle P \langle g \rangle$  на примере оценки числа умножений. В качестве предусловий и постусловий будут привлекаться частичные функции, принимающие значения в обычным образом упорядоченном множестве неотрицательных целых чисел. Для упрощения изложения

примем, что в логических выражениях, располагающихся после *if* и *while*, нет знаков умножения и что правые части инструкций присваивания задают всюду определенные функции (эти ограничения — не принципиальные).

Общее определение (8) свойства  $\langle f \rangle P \langle g \rangle$  мы сформируем более подробно применительно к рассматриваемой конкретной задаче.

**Определение.** Имеет место свойство  $\langle f \rangle P \langle g \rangle$ , если всякий раз, когда для некоторых состояний  $v_0, v_1$  выполнено  $p(v_0) = v_1$  и при этом определено значение  $f(v_0)$ , то определено значение  $g(v_1)$  и выполнение программы  $P$  для начального состояния  $v_0$  требует не более  $f(v_0) - g(v_1)$  умножений.

Можно дать следующие правила работы со свойствами вида  $\langle f \rangle P \langle g \rangle$  для различных инструкций.

1) *Инструкция присваивания.* Справедливо

$$\langle g(d(a, b, \dots, x), b, \dots, x) + C \rangle a := d(a, b, \dots, x) \langle g(a, b, \dots, x) \rangle,$$

где  $C$  — число умножений, требующихся для вычисления значения  $d(a, b, \dots, x)$ .

2) *Составная инструкция.* Если имеют место свойства  $\langle f \rangle P \langle g \rangle$  и  $\langle g \rangle Q \langle h \rangle$ , то имеет место и  $\langle f \rangle P; Q \langle h \rangle$ .

3) *Условная инструкция.* Если имеют место свойства  $\langle \beta \wedge f \rangle P \langle g \rangle$  и  $\langle \neg \beta \wedge f \rangle Q \langle g \rangle$ , то имеет место и  $\langle f \rangle \text{ if } \beta \text{ then } P \text{ else } Q \text{ fi } \langle g \rangle$ .

4) *Циклическая инструкция.* Если имеет место свойство  $\langle \beta \wedge f \rangle P \langle f \rangle$ , то имеет место  $\langle f \rangle \text{ while } \beta \text{ do } P \times \text{od } \langle \neg \beta \wedge f \rangle$ .

5) *Пустая инструкция.* При любой функции  $f$  имеет место свойство  $\langle f \rangle \langle f \rangle$ .

6) *Ослабление свойства.* Если имеет место свойство  $\langle f \rangle P \langle g \rangle$  и выполнено  $f_1 \geq f, g \geq g_1$ , то имеет место и  $\langle f_1 \rangle P \langle g_1 \rangle$ .

Еще раз подчеркнем, что эти правила справедливы ввиду эквивалентности свойства  $\langle f \rangle P \langle g \rangle$  отношению (8) и ввиду справедливости (4) — (7).

Вновь рассмотрим программу  $W$  вычисления  $a^n$  и с помощью правил 1) — 6) докажем свойство  $\langle 2\Pi_2(n) \rangle \times \times W \langle 0 \rangle$ . В силу правила 1) достаточно доказать свойство  $\langle (c = 1) \wedge (b = a) \wedge 2\Pi_2(n) \rangle \tilde{W} \langle 0 \rangle$  для циклической инструкции  $\tilde{W}$ , входящей в программу  $W$ . В силу правила 6) для доказательства последнего свойства достаточно доказать свойство  $\langle 2\Pi_2(n) \rangle \tilde{W} \langle 0 \rangle$ .

Будем действовать по правилу 4) и докажем свойство

$$\langle (n > 0) \wedge 2\Pi_2(n) \rangle l := \text{остаток}(n, 2); n := \lfloor n/2 \rfloor;$$

$$\text{if } l = 1 \text{ then } c := c \cdot b \text{ else fi};$$

$$b := b \cdot b \langle 2\Pi_2(n) \rangle;$$

в силу правил 1) и 6) достаточно доказать

$$\langle 2\Pi_2(n) + 2 \rangle \text{ if } l = 1 \text{ then } c := c \cdot b \text{ else fi};$$

$$b := b \cdot b \langle 2\Pi_2(n) \rangle, \quad (9)$$

но так как, очевидно, имеют место свойства

$$\langle 2\Pi_2(n) + 2 \rangle \text{ if } l = 1 \text{ then } c := c \cdot b \text{ else fi} \langle 2\Pi_2(n) + 1 \rangle,$$

$$\langle 2\Pi_2(n) + 1 \rangle b := b \cdot b \langle 2\Pi_2(n) \rangle,$$

то, применяя правило 2), мы получаем (9) и, тем самым,  $\langle 2\Pi_2(n) \rangle W \langle 0 \rangle$ .

Как обычно, возникает вопрос о «полноте системы». Пусть, например, имеет место свойство  $\langle h_1 \rangle S \langle h_2 \rangle$ , где  $S$  — это

$$\text{while } \beta \text{ do } P \text{ od}.$$

Гарантировано ли существование функции  $f$  такой, что а)  $h_1 \geq f$ , б)  $\langle \beta \wedge f \rangle P \langle f \rangle$ , в)  $(\neg \beta \wedge f) \geq h_2$ ?

Пусть множество  $M$ , в котором принимают значения привлекаемые для анализа программ функции (мы берем общий случай, а не только оценку числа умножений), содержит элемент  $O$ , который является универсальной нижней границей в смысле отношения порядка  $\geq$ . Свяжем с каждой инструкцией  $P$  такое преобразование функций  $L_P$ , что  $L_P(f)$  для любой функции  $f$  не отличается от  $f$  для тех  $v$ , для которых выполнение  $P$  завершается, и что  $L_P(f)$  принимает значение  $O$  для всех тех  $v$ , для которых выполнение  $P$  не завершается. Тогда можно показать, что если композиция  $H_P L_P$  (сначала  $H_P$ , потом  $L_P$ ) является структурным операционным смещением и если (8) эквивалентно

$$f \geq P^{LH}(g), \quad (10)$$

где  $P^{LH}(g) = L_P(H_P(p^*(g)))$ , то в качестве  $f$ , удовлетворяющей свойствам а), б), в), может быть взята  $S^{LH}(h_2)$ .

В ситуациях, удовлетворяющих указанным достаточным условиям (существование универсальной нижней границы, структурность  $H_P L_P$ , эквивалентность (8) и

(10)), мы оказываемся, когда имеем дело с обычными свойствами Хоора (в качестве  $O$  выступает  $I$ ) и когда рассматриваем свойства  $\langle f \rangle P \langle g \rangle$  с целью оценки числа умножений (в качестве  $O$  выступает  $0$ ).

Анализ таких последствий выполнения программ, которые или не связаны с изменением состояния, или не определяются по заключительному состоянию, в рамках теорий Хоора и Дейкстры проводятся обычно следующим образом. В программу включаются дополнительные переменные и некоторые инструкции присваивания, изменяющие значения этих переменных, — в первом примере можно было бы рассмотреть дополнительную переменную  $k$ , вставить в начало программы инструкцию  $k := 0$ , вставить инструкцию  $k := k + 1$  после каждой имеющейся в программе инструкции присваивания, правая часть которой содержит один знак умножения. Соответственно надо вставить в нужных местах инструкции  $k := k + 2$ ,  $k := k + 3$ , ... После такого преобразования программы следует выписать постусловие в виде предиката, связывающего значение дополнительной переменной со значениями некоторых других переменных.

В работе [14], среди прочего, предлагается включить изменяющие значения специально введенных переменных («невидимых переменных») инструкции присваивания не в программу, а в промежуточные утверждения (в частности, в предусловие и постусловие). Например, становится допустимым промежуточное утверждение  $\{k := k + 1; k \leq 2\Pi_2(m)\}$ , предусловие  $\{k := 0; m := n; m \geq 0\}$  и т. д. Таким образом, промежуточное утверждение в новом смысле слова может состоять из инструкций языка программирования и предиката (но может состоять и только из предиката или только из инструкций). Существенное ограничение заключено в том, что входящие в промежуточные утверждения инструкции не изменяют значений переменных программы. Примеры использования невидимых переменных даются также в книге [9], там эти переменные называются «переменными-призраками». Невидимые переменные привлекались и другими авторами для разработки методик исследования программ: см., например, [17]. Мы не будем здесь разбирать разнообразные вопросы применения методики Г. С. Цейтина (эта методика не сводится к одному лишь внесению инструкций в утверждения), но отметим, что, как показывают приведенные выше примеры, введение и рассмотрение невидимых переменных даже в задачах иссле-

дования «невидимых» последствий выполнения программ не является необходимостью.

В следующем параграфе будет рассмотрена существенно более сложная, в сравнении с оценкой числа умножений, задача. Для ее решения мы используем индуцированные программами преобразования функций, вписывающиеся в схему (1).

## § 6. Неустраняемая погрешность вычислений, описываемых программой

В качестве продолжения изучения не связанных с изменением состояния последствий выполнения программы мы разберем задачу анализа погрешности вычислений, описываемых программой. Речь пока будет идти о неустраняемой погрешности, т. е. погрешности, возникающей вследствие неточности начальных данных. Анализ вычислительной погрешности, т. е. погрешности, возникающей из-за округлений в вычислениях, а также совместному анализу погрешностей этих двух типов посвящен § 3 гл. III, где такое выполнение программы, в ходе которого допускаются округления при вычислении значений арифметических выражений, рассматриваются как недетерминированное. Задача исследования погрешности отличается от задач, которые обычно решаются методами Хоора и Дейкстры тем, что приходится рассматривать связанные некоторыми утверждениями пары состояний (точные и приближенные значения переменных).

Условимся обозначать через  $P$  рассматриваемую программу, считая при этом, что  $a, b, \dots, x$  — переменные, участвующие в программе  $P$ . Таким образом, состояние — это набор значений переменных  $a, b, \dots, x$ . В постановке задачи исследования погрешности у нас будут фигурировать предикаты  $\varphi(a', b', \dots, x', a'', b'', \dots, x'')$ ,  $\psi(a', b', \dots, x', a'', b'', \dots, x'')$ . Мы подразумеваем, что переменные  $a', b', \dots, x'$  принимают точные значения, а переменные  $a'', b'', \dots, x''$  — приближенные значения. Пусть значения  $A'_0, B'_0, \dots, X'_0, A''_0, B''_0, \dots, X''_0$  таковы, что имеет место  $\varphi(A'_0, B'_0, \dots, X'_0, A''_0, B''_0, \dots, X''_0)$ . Требуется выяснить, верно ли, что выполнение программы  $P$  завершается для начальных значений переменных  $a, b, \dots, x$ , равных как  $A'_0, B'_0, \dots, X'_0$ , так и  $A''_0, B''_0, \dots, X''_0$ , и что при этом соответствующие заключительные значения переменных  $a, b, \dots, x$  —  $A'_1, B'_1, \dots, X'_1$  и

$A''_1, B''_1, \dots, X''_1$  — удовлетворяют условию  $\psi(A'_1, B'_1, \dots, X'_1, A''_1, B''_1, \dots, X''_1)$ . Об этой задаче мы будем говорить как о задаче, поставленной в терминах предусловия  $\varphi$  и постусловия  $\psi$ , или, проще, в терминах  $\varphi$  и  $\psi$ . Мы будем говорить также об утверждении о погрешности, сформулированном в терминах  $\varphi$  и  $\psi$ .

Вначале мы покажем, как, используя конструкции, которые определяются сходно со слабым предусловием Дейкстры, поставленную в терминах  $\varphi$  и  $\psi$  задачу исследования погрешностей можно свести к задаче доказательства такой импликации, левая и правая части которой описаны функционально, т. е. без привлечения инструкций языка программирования. В частности, при исследовании погрешностей мы будем оперировать понятием двойного слабого предусловия для постусловия  $\psi(a', b', \dots, x', a'', b'', \dots, x'')$  относительно программы  $P$ , которое представляет собой самый слабый предикат  $\chi(a', b', \dots, x', a'', b'', \dots, x'')$ , обладающий тем свойством, что справедливо утверждение о погрешностях, сформулированное в терминах  $\chi$  и  $\psi$ .

Сопоставим программе  $P$  две программы —  $P'$  и  $P''$ , первая из которых получается заменой переменных  $a, b, \dots, x$  на  $a', b', \dots, x'$ , а вторая — заменой тех же переменных на  $a'', b'', \dots, x''$ . Рассмотрим программу  $P'; P''$ . То, что требуется выяснить в задаче, поставленной в терминах  $\varphi$  и  $\psi$ , эквивалентно импликации

$$\varphi \supset \text{wp}(P'; P'', \psi)$$

или

$$\varphi \supset \text{wp}(P', \text{wp}(P'', \psi)). \quad (1)$$

В обоих случаях правая часть импликации — это двойное слабое предусловие для  $\psi$  относительно  $P$ .

Для применения формулы (1), разумеется, нет необходимости переписывать программу  $P$ , заменяя в ней переменные. Например, при нахождении  $\text{wp}(P'', \psi)$  мы просто можем отождествить переменные  $a, b, \dots, x$  с переменными  $a'', b'', \dots, x''$ .

Пример. Пусть программа  $P$  — это  $a := 3a$ . Утверждению, что относительной погрешности исходного значения, не превосходящей 0.5, отвечает относительная погрешность результата, также не превосходящая 0.5, соответствует пара предикатов

$$\varphi(a', a'') = \psi(a', a'') = (|(a' - a'')/a''| < 0.5).$$



Используя формулу (1) доказательство сводится к установлению импликации

$$(|a' - a''|/a'' < 0.5) \Rightarrow (|3a' - 3a''|/3a'' < 0.5).$$

Известные формулы для слабейшего предусловия относительно присваивания, а также условной, циклической и составной инструкций позволяют выписывать импликацию для произвольных  $\varphi$  и  $\psi$  и любой программы, составленной из инструкций этих основных видов. Несколько позже мы покажем, что утверждение о погрешностях, сформулированное в терминах  $\varphi$  и  $\psi$ , оказывается в некоторых случаях эквивалентным более простому и зависящему от меньшего числа переменных (эти переменные, как и переменные  $a', b', \dots, x', a'', b'', \dots, x''$ , неявно предполагаются связанными в импликациях типа (1) кванторами всеобщности).

Примечание 1. Понятно, что если  $P$  — составная инструкция вида  $R; S$ , то в формуле (1) можно разобрать  $P'$  и  $P''$  на составные части  $R', S'$  и  $R'', S''$ ; правая часть импликации (1) может быть переписана в виде

$$\text{wp}(R', \text{wp}(R'', \text{wp}(S', \text{wp}(S'', \psi)))).$$

Это значит, что если мы определяем семантику инструкций как специального вида преобразование предикатов — двойное слабейшее предусловие, то составной инструкции сопоставляется преобразование, которое разлагается в композицию преобразований, сопоставляемых ее инструкциям. Подчеркнем, однако, что у нас нет столь простой возможности разбирать на составные части те преобразования, которые соответствуют условной инструкции.

Если исследуемая программа такова, что ее выполнение завершается для любого начального состояния, то слабейшее предусловие в (1) можно заменить слабейшим свободным предусловием; эту же замену можно сделать и тогда, когда вопрос о завершимости нас почему-либо не интересует и для нас достаточна истинность  $\psi$ , когда выполнение программы завершается для начальных состояний, определяемых значениями переменных  $a', b', \dots, x', a'', b'', \dots, x''$ , удовлетворяющими  $\varphi$ . В этом случае мы также будем говорить о поставленной в терминах  $\varphi$  и  $\psi$  задаче исследования неустранимой погрешности.

После перехода к  $\text{wlp}$  мы получаем формулу

$$\varphi \supset \text{wlp}(P', \text{wlp}(P'', \psi)). \quad (2)$$

Предикат, эквивалентный правой части последней импликации, будем называть двойным слабейшим свобод-

ным предусловием для  $\psi$  относительно  $P$ . Для вычисления этого предиката можно воспользоваться формулами для  $wlp$ , выведенными в § 3. Остановимся на подробностях, относящихся к циклической инструкции  $P$  вида

$$\text{while } \beta \text{ do } S \text{ od.} \quad (3)$$

Формулы § 3 применительно к постусловию-предикату  $\eta$  запишем при помощи логических связок и квантора всеобщности:

$$wlp(P, \eta) = \forall i wlp(P, \eta)_i,$$

где последовательность предикатов  $wlp(P, \eta)_0, wlp(P, \eta)_1, \dots$  определяется рекуррентно:

$$\begin{aligned} wlp(P, \eta)_0 &= \beta \vee \eta, \\ wlp(P, \eta)_i &= \beta \wedge wlp(S, wlp(P, \eta)_{i-1}) \vee \neg \beta \wedge \eta. \end{aligned} \quad (4)$$

При этом  $wlp(P, \eta)_i, i = 0, 1, \dots$ , имеет следующий смысл: если начальные значения переменных, участвующих в программе  $P$ , удовлетворяют  $wlp(P, \eta)_i$  и если выполнение  $P$  завершается не позднее чем через  $i$  этапов, то заключительные значения переменных удовлетворяют  $\eta$ . Отметим, что

$$wlp(P, \eta)_{j+1} \supset wlp(P, \eta)_j, \quad j = 0, 1, \dots \quad (5)$$

Двойное слабейшее свободное предусловие для постусловия  $\psi(a', b', \dots, x', a'', b'', \dots, x'')$  относительно программы вида (3) представляется в виде:

$$wlp(P', wlp(P'', \psi)) = \forall i \forall j wlp(P', wlp(P'', \psi))_i.$$

Обозначим  $wlp(P', wlp(P'', \psi))_i$  через  $\kappa_{ij}$ . Тогда утверждение о неустранимой погрешности, сформулированное в терминах  $\phi$  и  $\psi$ , эквивалентно импликации  $\phi \supset \forall i \forall j \kappa_{ij}$ , которая эквивалентна в свою очередь  $\forall i \forall j (\phi \supset \kappa_{ij})$ . Это дает возможность применять двойную индукцию. Однако, с учетом (5), нетрудно вывести, что для любых  $i, j$  и  $m = \max(i, j)$  имеет место импликация  $\kappa_{mm} \supset \kappa_{ij}$ . Приняв обозначение  $\lambda_m = \kappa_{mm}$ , мы можем написать  $\lambda_{m+1} \supset \lambda_m$  для  $m = 0, 1, \dots$  и представить двойное слабейшее свободное предусловие для  $\psi$  относительно  $P$  в виде  $\forall m \lambda_m$ , а утверждение о неустранимой погрешности — в виде  $\phi \supset \forall m \lambda_m$  или  $\forall m (\phi \supset \lambda_m)$ . Мы получили, таким образом, возможность доказывать утверждение о неустранимой погрешности одинарной индукцией, но зависимость  $\lambda_{m+1}$  от  $\lambda_m$ , вытекающая из двух рекуррентных

соотношений вида (4), довольно сложна, и проведение такой индукции по  $m$  может оказаться весьма затруднительным. Мы рассмотрим распространенный тип циклической инструкции, для которого все существенно упрощается.

Пусть в программе  $P$ , заданной с помощью (3),  $\beta$  зависит только от точно заданных переменных  $a, \dots, k$  (т. е. имеет место  $\varphi \supset ((a' = a'') \wedge \dots \wedge (k' = k''))$ ), изменение значений которых происходит лишь в результате выполнения таких инструкций присваивания, в правые части которых не входят никакие другие переменные, кроме  $a, \dots, k$ . Сразу отметим, что число этапов выполнения  $P'$  и  $P''$  будет одним и тем же. В постановке задачи исследования неустранимой погрешности мы можем уменьшить число переменных в предусловии и постусловии — эти предикаты можно считать зависящими от  $a, \dots, k, l', \dots, x', l'', \dots, x''$ ; здесь  $l, \dots, x$  — отличные от  $a, \dots, k$  переменные, участвующие в программе  $P$ ; будем говорить, что переменные  $a, \dots, k$  входят в предикаты нераздвоенно. Привлечение предикатов с нераздвоенным вхождением некоторых (но не всех) переменных представляет собой значительный отход от традиционной техники верификации.

Аналог того предусловия, которое при рассмотрении зависящих от  $a', \dots, x', a'', \dots, x''$  предикатов мы называли двойным слабейшим свободным предусловием для постусловия  $\psi(a, \dots, k, l', \dots, x', l'', \dots, x'')$  относительно программы  $P$  будем обозначать через  $w(P, \psi)$ . Его точный смысл таков: это предикат, характеризующий множество всех наборов  $(A_0, \dots, K_0, L'_0, \dots, X'_0, L''_0, \dots, X''_0)$  таких значений, что если выполнение  $P$  завершается как для начальных значений переменных  $a, \dots, k, l, \dots, x$ , равных соответственно  $A_0, \dots, K_0, L'_0, \dots, X'_0$ , так и для начальных значений этих переменных, равных соответственно  $A_0, \dots, K_0, L''_0, \dots, X''_0$ , то соответствующие заключительные значения переменных  $a, \dots, k, l, \dots, x$  —  $A_1, \dots, K_1, L'_1, \dots, X'_1$  и  $A_1, \dots, K_1, L''_1, \dots, X''_1$  — удовлетворяют условию  $\psi(A_1, \dots, K_1, L'_1, \dots, X'_1, L''_1, \dots, X''_1)$ .

Постусловия и предусловия указанного типа можно рассматривать не только в связи с циклическими инструкциями. Не представляет труда вычисление  $w(P, \psi)$  в тех случаях, когда  $P$  — это инструкция присваивания,

например:

$$w(a := f(a, \dots, k), \psi(a, \dots, k, l', \dots, x', l'', \dots, x'')) = \\ = \psi(f(a, \dots, k), \dots, k, l', \dots, x', l'', \dots, x'')$$

и

$$w(x := g(a, \dots, k, l, \dots, x), \psi(a, \dots, k, l', \dots, x', \\ l'', \dots, x'')) = \psi(a, \dots, k, l', \dots, g(a, \dots, k, l', \dots, x'), \\ l'', \dots, g(a, \dots, k, l'', \dots, x''));$$

для составной инструкции имеем  $w(S_1; S_2, \psi) = w(S_1, w(S_2, \psi))$ , для пустой —  $w(, \psi) = \psi$ .

Для условной инструкции  $P$  вида

**if  $\beta$  then  $S_1$  else  $S_2$  fi**

и циклической инструкции (3) следует различать два случая. Если значение  $\beta$  зависит от значения хотя бы одной из переменных  $l, \dots, x$ , то для  $w(P, \psi)$ , видимо, нельзя получить существенно более простые формулы, чем те, которые выводятся исходя из определения двойного слабейшего предусловия;  $w(P, \psi(a, \dots, k, l', \dots, x', l'', \dots, x''))$  может быть вычислено в этом случае, например, так: найдем двойное слабейшее свободное предусловие для

$$\psi(a', \dots, k', l', \dots, x', l'', \dots, x'') \wedge \\ \wedge \psi(a'', \dots, k'', l', \dots, x', l'', \dots, x'')$$

относительно  $P$ , а затем заменим в нем как переменные  $a', \dots, k'$ , так и переменные  $a'', \dots, k''$  на  $a, \dots, k$  соответственно (этот прием годится для любой программы  $P$ ). Но если значение  $\beta$  зависит только от значений переменных  $a, \dots, k$ , то формулы получаются не более сложными, чем формулы для обычного слабейшего свободного предусловия.

Непосредственной проверкой устанавливается справедливость следующей теоремы.

**Теорема 1.** Пусть значение  $\beta$  зависит только от переменных  $a, \dots, k$ , тогда

$$w(\text{if } \beta \text{ then } S_1 \text{ else } S_2 \text{ fi}, \psi) = \\ = \text{if } \beta \text{ then } w(S_1, \psi) \text{ else } w(S_2, \psi) \text{ fi} = \\ = \beta \wedge w(S_1, \psi) \vee \neg \beta \wedge w(S_2, \psi).$$

При этом же предположении относительно  $\beta$  для цикли-

ческой инструкции (3) имеет место

$$w(P, \psi) = \forall i w(P, \psi)_i,$$

где

$$\begin{aligned} w(P, \psi)_0 &= \beta \vee \psi, \\ w(P, \psi)_i &= \beta \wedge w(S, w(P, \psi)_{i-1}) \vee \neg \beta \wedge \psi. \end{aligned} \quad (6)$$

В дальнейшем будет удобным обозначение  $K[\mu] = \beta \wedge w(S, \mu) \vee \neg \beta \wedge \psi$ , где  $\mu$  — предикат, зависящий от  $a, \dots, k, l', \dots, x', l'', \dots, x''$ . Положив, по определению,  $w(P, \psi)_{-1} = \text{И}$ , от формул (6) можно перейти к формулам

$$\begin{aligned} w(P, \psi)_{-1} &= \text{И}, \\ w(P, \psi)_i &= K[w(S, \psi)_{i-1}]. \end{aligned} \quad (7)$$

Отметим монотонность функционала  $K$ : из  $\xi \supset \eta$  следует  $K[\xi] \supset K[\eta]$ .

**Пример.** Рассмотрим программу  $P$

$$\begin{aligned} \text{while } i < N \text{ do } x &:= x + h \cdot \left(\frac{1}{x} + t\right); \\ t &:= t + h; i := i + 1 \text{ od.} \end{aligned}$$

Пусть приближенно задается только  $x$ . Покажем справедливость утверждения о неустранимой погрешности, сформулированное в терминах следующих  $\varphi$  и  $\psi$ :

$$\begin{aligned} \varphi(i, N, h, t, x', x'') &= \\ &= (t, h > 0) \wedge (N > i = 0) \wedge (x', x'' > c) \wedge (|x' - x''| < \varepsilon), \\ \varphi(i, N, h, t, x', x'') &= \left(|x' - x''| < \varepsilon \left(1 + \frac{h}{c^2}\right)^N\right), \end{aligned}$$

где  $c, \varepsilon$  — некоторые конкретные положительные числа. Нам требуется доказать, что  $\varphi \supset w(P, \psi)$ . Для того чтобы сделать это путем применения формул (6) или (7), усилим доказываемое утверждение: будем доказывать, что  $\varphi_1 \supset w(P, \psi)$ , где

$$\begin{aligned} \varphi_1(i, N, h, t, x', x'') &= \\ &= (i \leq N) \wedge \left(|x' - x''| < \varepsilon \left(1 + \frac{h}{c^2}\right)^i\right) \wedge \\ &\quad \wedge (x', x'' > c) \wedge (t, h > 0). \end{aligned}$$

Доказательство можно провести так: установить вначале, что  $\varphi_1 \supset w(P, \psi)_{-1}$  (это тривиально, так как

$w(P, \psi)_{-1} = \Pi$ ), затем из предположения о справедливости для некоторого  $\xi$  импликации  $\varphi_1 \supset \xi$  доказать импликацию  $\varphi_1 \supset K[\xi]$ . В силу монотонности  $K$  из  $\varphi_1 \supset \xi$  следует  $K[\varphi_1] \supset K[\xi]$ , поэтому достаточно доказать, что

$$\varphi_1 \supset K[\varphi_1]. \quad (8)$$

Имеем

$$\begin{aligned} K[\varphi_1] &= (i < N) \wedge w(P, \varphi_1) \vee (i \geq N) \wedge \psi = \\ &= (i < N) \wedge \left( \left| x' + h \left( \frac{1}{x'} + t \right) - x'' - h \cdot \left( \frac{1}{x''} + t \right) \right| < \right. \\ &< \varepsilon \left( 1 + \frac{h}{c^2} \right)^{i+1} \Big) \wedge \left( x' + h \left( \frac{1}{x'} + t \right), x'' + h \left( \frac{1}{x''} + t \right) > \dot{c} \right) \wedge \\ &\wedge (t + h, h > 0) \vee (i = N) \wedge \left( |x' - x''| < \varepsilon \left( 1 + \frac{h}{c^2} \right)^N \right). \end{aligned}$$

Случаи  $i < N$  и  $i = N$  можно разобрать отдельно. Проверка (8) для случая  $i < N$  проводится так, как это делается обычно в теории численных методов при анализе погрешности метода Эйлера (программа  $P$  представляет собой реализацию метода Эйлера применительно к уравнению  $\dot{x} = \frac{1}{x} + t$ ).

**Примечание 2.** Можно было бы определить преобразование предикатов, зависящих от  $a, \dots, k, l', \dots, x', l'', \dots, x''$ , основываясь не на  $wlp$ , а на  $wp$ ; это привело бы к аналогичным формулам, в которых, однако, вместо квантора всеобщности появился бы квантор существования. Это внесло бы неудобство в доказательство импликаций, эквивалентных утверждениям о погрешности, но дало бы возможность решать вопрос о завершимости.

В § 3 отмечена связь преобразований  $wp$  и  $wlp$  с теорией неподвижной точки непрерывных по Скотту — Манне функционалов; эта связь открывает возможности доказательства утверждений о погрешностях методами этой теории (различными видами индукции по неподвижной точке). Мы на этом останавливаться не будем.

Задачу исследования неустранимой погрешности рассмотрим теперь с точки зрения теории Хоора, которая, как известно, позволяет не проводить вычислений предусловий по сложным формулам, но предполагает изобретение промежуточных утверждений (в частности, инвариантов цикла). Вопрос о том, для каких именно начальных состояний выполнение  $P$  завершается, в рамках теории Хоора не рассматривается. Решение поставленной в терминах  $\varphi$  и  $\psi$  задачи о неустранимой погрешности —

это доказательство свойства

$$\{\varphi(a', b', \dots, x', a'', b'', \dots, x'')\}P'; P' \\ \{\psi(a', b', \dots, x', a'', b'', \dots, x'')\}. \quad (9)$$

Для доказательства же этого свойства надо, в соответствии с семантикой Хоора составной инструкции, подобрать предикат  $\xi(a', b', \dots, x', a'', b'', \dots, x'')$  такой, что имеют место свойства

$$\{\varphi(a', b', \dots, x', a'', b'', \dots, x'')\} \times \\ \times P'\{\xi(a', b', \dots, x', a'', b'', \dots, x'')\}, \quad (10)$$

$$\{\xi(a', b', \dots, x', a'', b'', \dots, x'')\} \times \\ \times P''\{\psi(a', b', \dots, x', a'', b'', \dots, x'')\}.$$

Программы  $P'$  и  $P''$  отличаются только именами переменных. Мы сделаем замену переменных и перейдем в обоих свойствах к программе  $P$ ; при этом мы откажемся от штрихованных переменных и будем иметь дело с входящими в программу  $P$  переменными  $a, b, \dots, x$  и с переменными  $\bar{a}, \bar{b}, \dots, \bar{x}$ , которые не входят в  $P$ :

$$\{\varphi(a, b, \dots, x, \bar{a}, \bar{b}, \dots, \bar{x})\}P\{\xi(a, b, \dots, x, \bar{a}, \bar{b}, \dots, \bar{x})\}, \quad (11)$$

$$\{\xi(\bar{a}, \bar{b}, \dots, \bar{x}, a, b, \dots, x)\}P\{\psi(\bar{a}, \bar{b}, \dots, \bar{x}, a, b, \dots, x)\}.$$

**Примечание 3.** Свойства (11) относятся к одной и той же программе  $P$ , и их можно рассматривать, как показано в § 2, в виде одного обобщенного хооровского свойства с бинарными отношениями в качестве предусловия и постусловия.

**Примечание 4.** Совокупность свойств (11) можно рассматривать как систему соотношений для  $\xi$ , разрешимость которой эквивалентна утверждению о неустранимой погрешности, сформулированному в терминах  $\varphi$  и  $\psi$ .

**Пример.** Пусть программа  $P$  — это

$$s := 1; i := 0; \text{ while } i \neq n \text{ do } s := s \cdot a; i := i + 1 \text{ od.} \quad (12)$$

Будем считать, что  $a$  задано приближенно с относительной погрешностью, превосходящей  $1 + \frac{1}{n}$ , т. е.  $\frac{a}{a} > \frac{n+1}{n}$ , и, кроме этого, что  $a, \bar{a} \neq 0$ . Докажем, что после окончания выполнения программы имеет место  $s/\bar{s} > 2$ . Таким образом,

$$\varphi(a_1 s_1 i_1 n_1 \bar{a}_1 \bar{s}_1 \bar{i}_1 \bar{n}) = \\ = ((a_1 \bar{a} \neq 0) \wedge (a/\bar{a} > (n+1)/n) \wedge (n = \bar{n})),$$

$$\psi(a, s, i, n, \bar{a}, \bar{s}, \bar{i}, \bar{n}) = (s/\bar{s} > 2).$$

В качестве  $\xi$  здесь можно взять  $(a, \bar{a} \neq 0) \wedge (n = \bar{n}) \wedge (s/\bar{a}^n > 2)$ ; легко доказываются свойства

$$\{(a, \bar{a} \neq 0) \wedge (a/\bar{a} > (n+1)/n) \wedge (n = \bar{n})\} s := 1; i := 0;$$

$$\text{while } i \neq n \text{ do } s := s \cdot a; i := i + 1$$

$$\text{od } \{(a, \bar{a} \neq 0) \wedge (n = \bar{n}) \wedge (s/\bar{a}^n > 2)\},$$

$$\{(a, \bar{a} \neq 0) \wedge (n = \bar{n}) \wedge (\bar{s}/\bar{a}^n > 2)\} s := 1; i := 0;$$

$$\text{while } i \neq n \text{ do } s := s \cdot a; i := i + 1 \text{ od } \{\bar{s}/s > 2\};$$

для доказательства первого из них подходит инвариант цикла

$$(a, \bar{a} \neq 0) \wedge (n = \bar{n}) \wedge (s/\bar{a}^i > 1 + i/n),$$

для доказательства второго —

$$(a, \bar{a} \neq 0) \wedge (n = \bar{n}) \wedge (\bar{s}/(sa^{\bar{n}-i}) > 2).$$

Рассматривая методы анализа неустранимой погрешности, основанные на применении специальных предусловий, мы нашли для некоторых случаев возможность перехода в предикатах  $\phi$ ,  $\psi$  и т. д. к меньшему числу переменных. В описанном же методе, основанном на применении техники Хоора, каждая переменная, которая может входить в предикаты нераздвоенно, дополнительно к прежним условиям должна подчиняться одному из двух (очень жестких) дополнительных условий:

1) эта переменная не входит в левые части инструкций присваивания;

2) входное значение этой переменной не влияет на результат выполнения программы (например, этой переменной в ходе выполнения программы присваивается некоторое значение прежде, чем в первый раз используется ее значение).

В приведенном исследовании программы (12) предикаты  $\phi$ ,  $\psi$ ,  $\xi$  и т. д. можно задать так, что они будут зависеть только от переменных  $i, n, a, s, \bar{a}, \bar{s}$ .

Но независимо от того, какие переменные участвуют в предикатах, наиболее существенная сложность, связанная с применением этого метода, заключена в необходимости подбора  $\xi$ , исходя не только из  $\phi$  и  $\psi$ , но и из программы  $P$ . В связи с этим возникает следующий вопрос. Нельзя ли проверку утверждения о неустранимой



погрешности, сформулированного в терминах  $\varphi$  и  $\psi$ , свести к проверке ряда хооровских свойств исследуемой программы  $P$ , предусловие и постусловие каждого из которых получается исходя только из  $\varphi$  и  $\psi$ . Оказывается, что ответ на этот вопрос отрицателен. Это можно показать, рассматривая программы, содержащие только одну целочисленную переменную  $a$ . Среди этих программ выделим класс таких, для которых справедливо утверждение о погрешностях при следующем выборе предикатов  $\varphi, \psi$ :

$$\varphi(a, \bar{a}) = \psi(a, \bar{a}) = (a - \bar{a} = 1).$$

К этому классу относятся, в частности, все программы вида  $a := a + n$ , где  $n$  — некоторое конкретное целое число. Но программа  $a := 0$  не принадлежит рассматриваемому классу.

*Лемма.* Пусть предикаты  $\alpha(a), \beta(a)$ , определенные на множестве целых чисел, таковы, что свойством

$$\{\alpha(a)\}P\{\beta(a)\} \quad (13)$$

обладает любая программа рассматриваемого класса. Тогда этим свойством обладает вообще любая программа, содержащая целочисленную переменную  $a$ .

Доказательство. Так как при любом целом  $n$  программа  $a := a + n$  принадлежит рассматриваемому классу, то для любого  $n$  имеет место  $\{\alpha(a)\}a := a + n\{\beta(a)\}$ . Последнее означает, что для любого  $n$  имеет место импликация  $\alpha(a) \supset \beta(a + n)$ , откуда следует, что если  $\alpha(a)$  принимает значение И хотя бы для одного целого значения переменной  $a$ , то  $\beta(a)$  тождественно равно И. Итак, либо  $\alpha$  тождественно равно Л, либо  $\beta$  тождественно равно И. В обоих случаях свойством (13) обладает любая программа (в частности, программа  $a := 0$ ).

Из доказанной леммы следует, что не существует предикатов  $\varphi_1(a), \varphi_2(a), \dots; \psi_1(a), \psi_2(a), \dots$  таких, что относительно программы  $P$ , содержащей целочисленную переменную  $a$ , тогда и только тогда справедливо сформулированное выше утверждение о погрешностях, когда программа  $P$  обладает свойствами

$$\{\varphi_1(a)\}P\{\psi_1(a)\}, \{\varphi_2(a)\}P\{\psi_2(a)\}, \dots$$

Это положение нельзя исправить введением в  $\varphi_1, \varphi_2, \dots, \psi_1, \psi_2, \dots$  параметра  $\bar{a}$  (и каких-либо еще параметров), так как свойство

$$\{\varphi'_1(a, \bar{a})\}P\{\psi_1(a, \bar{a})\} \quad (14)$$

есть множество свойств вида (13), элементы которого получаются подстановкой в (14) вместо  $\bar{a}$  конкретных неотрицательных целых чисел.

Итак, доказана следующая теорема.

**Теорема 2.** *Исходя только из предикатов  $\varphi(a, \bar{a})$ ,  $\psi(a, \bar{a})$ , вообще говоря, невозможно определить два множества предикатов  $\varphi_1(a, \bar{a})$ ,  $\varphi_2(a, \bar{a})$ , ...;  $\psi_1(a, \bar{a})$ ,  $\psi_2(a, \bar{a})$ , ... таких, что относительно произвольной программы  $P$ , содержащей целочисленную переменную  $a$ , утверждение о неустранимых погрешностях, сформулированное в терминах  $\varphi$ ,  $\psi$ , справедливо в том и только в том случае, когда  $P$  обладает всеми свойствами*

$$\{\varphi_1(a, \bar{a})\}P\{\psi_1(a, \bar{a})\}, \{\varphi_2(a, \bar{a})\}P\{\psi_2(a, \bar{a})\}, \dots$$

Выясним отношение описанных методов к схеме анализа программ, которая была предложена в § 4 и исходя из которой получают разнообразные методы исследования последствий выполнения программ, не зависящих от заключительного состояния. Те предикаты, в терминах которых ставилась задача исследования погрешностей, можно рассматривать как функции на множестве состояний  $V$ , каждая из которых сопоставляет состоянию  $v$  предикат на  $V$ : так, предикат  $\varphi(a', \dots, x', a'', \dots, x'')$  рассматривается как функция, сопоставляющая состоянию  $A_0, \dots, X_0$  предикат  $\varphi(A_0, \dots, X_0, a, \dots, x)$ . Такого вида функции сопоставляют точно заданному состоянию предикат, определяющий множество его допустимых приближений. Преобразования типа двойного слабейшего предусловия и двойного слабейшего свободного предусловия становятся преобразованиями функций указанного вида, устроенными сходно с теми преобразованиями, которые задаются описанной в § 4 схемой. Если постусловие  $\psi(a', \dots, x', a'', \dots, x'')$  рассмотреть как функцию  $g(a, \dots, x)$  указанного вида, то двойное слабейшее предусловие относительно программы  $P$  можно, в духе § 4, записать как  $H_P(p^*(g))$ , где  $p^*$  — преобразование функций указанного вида, сопряженное индуцированному программой  $P$  преобразованию множества  $V$  в себя;  $H_P$  — это преобразование предикатов на  $V$ , сопряженное индуцированному программой  $P$  преобразованию множества  $V$  в себя, а применение  $H_P$  к  $p^*(g)$  состоит в применении  $H_P$  к каждому значению  $p^*(g)$ .

Преобразование  $w(P, \psi)$  при фиксированном аргументе  $P$  тоже может рассматриваться как преобразование вида  $P^H$  — нетрудно описать операционное смещение для

каждого вида  $P$ . Но операционное смещение не будет структурным (как и в случае двойного слабейшего пред- условия) — вообще говоря,

$w(\text{if } \beta \text{ then } P \text{ else } Q \text{ fi}, \psi) \neq \text{if } \beta \text{ then } w(P, \psi) \text{ else } w(Q, \psi) \text{ fi}.$

Это приводит, как было показано выше, к усложнению формул, затрагивающих циклические инструкции.

Результаты этого параграфа и § 3 гл. III, где разбираются задачи, связанные с вычислительной погрешностью, получены Е. А. Казьминой и В. А. Кукляевой совместно с автором.

## § 7. Семантика Хоора и число этапов выполнения цикла

Для исследования трудоемкости некоторой программы вида

$\text{while } \beta \text{ do } P \text{ od}, \quad (1)$

как, например, для исследования числа выполнения  $P$  и зависимости этого числа от начального состояния  $v$ , часто рассматривают характер изменения размера данных в процессе выполнения программы (1). При этом размер данных — это некоторая естественно определяемая величина: например, порядок квадратной матрицы, наибольшее или наименьшее число из набора данных положительных чисел и т. д.

Один из наиболее простых методов оценки числа этапов по размеру исходных данных (мы будем обозначать этот размер для состояния  $v$  через  $|v|$ ) основан на выборе бесконечной возрастающей числовой последовательности  $s_1, s_2, \dots$ , обладающей свойствами:

$$1) (|v| < s_1) \supset \neg \beta,$$

$$2) ((|v| < s_k) \wedge (k > 1)) \supset (|p(v)| < s_{k-1}).$$

Тогда при условии  $s_m \leq |v| < s_{m+1}$  число выполнений  $P$  не превзойдет  $m$ .

С помощью последовательности  $s_k = 2^{k-1}$ ,  $k = 1, 2, \dots$ , устанавливается, например, трудоемкость простейших программ, построенных на идее деления пополам; это дает оценку  $\lceil \log_2 |v| \rceil + 1$  или  $\lfloor \log_2 |v| \rfloor$ .

Примечание 1. Применение методов, описанных в § 4, дает для числа  $f(n)$  делений пополам соотношение типа

$$f(n) = \text{if } n > 0 \text{ then } f(\lfloor n/2 \rfloor) + 1 \text{ else } 0 \text{ fi},$$

Рассмотрим другой метод оценки числа выполнений  $P$  по размеру исходных данных  $|v|$ . Пусть имеется неограниченно возрастающая последовательность чисел  $r_1, r_2, \dots$  такая, что перед последним выполнением  $P$  в ходе выполнения (1) данные имеют размер  $\geq r_1$ , перед предпоследним —  $\geq r_2$  и т. д. Пусть  $v$  — начальное состояние. Пусть число выполнений  $P$  равно  $m$ , тогда по условию  $|v| \geq r_m$ . Если найдено  $k$  такое, что  $|v| < r_{k+1}$ , то, поскольку последовательность  $r_1, r_2, \dots$  возрастающая,  $m \leq k$ . Таким образом, наличие последовательности  $r_1, r_2, \dots$  дает способ оценки числа выполнений  $P$ ; при этом даже не предполагается, что размер данных убывает в результате одного выполнения  $P$ . Последний метод является более универсальным, чем предыдущий, им можно, в частности, вывести оценку и для делений пополам. Этот метод является обобщением метода, который применялся Г. Ламе для оценки числа делений с остатком, требуемых применением алгоритма Евклида к натуральным  $a_0$  и  $a_1$  ( $a_0 > a_1$ ):

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, \\ a_1 &= q_2 a_2 + a_3, \\ &\dots \dots \dots \\ a_{m-2} &= q_{m-1} a_{m-1} + a_m, \\ a_{m-1} &= q_m a_m \end{aligned}$$

(далее будем считать, что  $a_{m+1} = 0$ ).

Для пары  $a_i, a_{i+1}$  размер определяется как величина  $a_{i+1}$ . В качестве последовательности  $r_1, r_2, \dots$  берутся числа Фибоначчи, начиная со второго:

$$r_1 = u_2 = 1, \quad r_2 = u_3 = 2, \quad r_3 = u_4 = 3, \dots$$

Доказывается, что  $a_m \geq u_2, a_{m-1} \geq u_3, \dots, a_1 \geq u_{m+1}$ , а последнее неравенство

$$a_1 \geq u_{m+1} \tag{2}$$

и составляет, как известно, содержание теоремы Ламе.

Оценку (2) нельзя при столь простом определении размера пары вывести предыдущим методом, так как один этап выполнения алгоритма Евклида — переход от пары  $a_i, a_{i+1}$  (состояния) к паре  $a_{i+1}, a_{i+2}$  — может изменить величину второго числа пары всего на 1, и глобальная оценка, которую можно отсюда получить — это

$a_1 \geq m + 1$ , в то время как оценка (2) — логарифмическая.

Этот пример показывает, что для установления некоторого свойства циклической программы (1) иногда оказывается существенным взгляд на эту программу как на единое целое и, напротив, что вычленение из (1) инструкции  $P$  и сведение исследования программы (1) к исследованию  $P$  может не дать решения задачи. Последнее является причиной возникновения препятствий, на которые наталкивается применение метода Хоора: то свойство инструкции  $P$ , которое эквивалентно рассматриваемому свойству программы (1), фактически состоит в том, что если из  $P$  и  $\beta$  изготовить программу (1), то она будет обладать нужным свойством.

Примечание 2. Свойства 1), 2) последовательности  $s_1, s_2, \dots$ , на которых основывается первый из рассмотренных методов анализа трудоемкости, эквивалентны следующему обобщенному хооровскому свойству программы  $P$ :

$$\{\{k \mid (|v| < s_k) \wedge (k > 1)\}\} P \{\{l \mid |v| < s_{l-1}\}\}.$$

В следующей главе мы изучим некоторые эффекты, связанные с применением метода Хоора. В частности, мы подробно обсудим характер неполноты логической системы Хоора. Сейчас будут предложены еще примеры, которые, с одной стороны, связаны с алгоритмом Евклида и которые, с другой стороны, дают дополнительные возможности сопоставить оценки, получающиеся при рассмотрении программы (1) в целом, с оценками, получающимися при переходе к изолированному рассмотрению программы  $P$ .

Как следствие неравенства (2), в результате логарифмирования по основанию 10, получается, что число делений с остатком в процессе применения алгоритма Евклида не превосходит увеличенного в 5 раз количества цифр числа  $a_1$ , записанного в десятичной системе (иногда теоремой Ламе называется именно это утверждение). Можно получить аналогичные следствия для других систем счисления. Из (2) легко выводится

$$\begin{aligned} m + 1 &< \log_{(1+\sqrt{5})/2} (1 + a_1) + \log_{(1+\sqrt{5})/2} \sqrt{5} = \\ &= \log_{(1+\sqrt{5})/2} (1 + a_1) + 1.672275 \dots \quad (3) \end{aligned}$$

Если целое  $q$ , большее единицы, — основание интересую-

щей пас системы счисления, то можно написать

$$m \leq ([\log_{(1+\sqrt{5})/2} q] + 1)([\log_q a_1] + 1) = \\ = ([\log_{(1+\sqrt{5})/2} q] + 1) \Pi_q(a_1), \quad (4)$$

где  $\Pi_q(a_1)$  — количество цифр числа  $a_1$  в  $q$ -ичной системе. Для двоичной системы имеем

$$\log_{(1+\sqrt{5})/2} 2 = 1.440 \dots, \quad m \leq 2\Pi_2(a_1);$$

уже говорилось о десятичной оценке:

$$\log_{(1+\sqrt{5})/2} 10 = 4.784 \dots, \quad m \leq 5\Pi_{10}(a_1).$$

Для упрощения записи примем обозначение  $c_q = [\log_{(1+\sqrt{5})/2} q] + 1$ , оценка (4) запишется в виде  $m \leq c_q \Pi_q(a_1)$ .

Некоторые из этих оценок могут быть получены совершенно элементарно, не прибегая к неравенству (2). Например, для  $i = 1, 2, \dots, m - 2$  имеет место неравенство  $a_i > 2a_{i+2}$  — в самом деле,  $a_i = q_{i+1}a_{i+1} + a_{i+2} \geq a_{i+1} + a_{i+2} > 2a_{i+2}$ . Отсюда легко получается  $m \leq 2([\log_2 a_1] + 1)$ ; например, пусть  $m$  — нечетное, т. е.  $m = 2l + 1$ , тогда  $a_1 > 2^l a_m \geq 2^l$ ,  $l < \log_2 a_1$ , откуда  $l < [\log_2 a_1] + 1$  и  $2l + 1 < 2([\log_2 a_1] + 1)$ . Но десятичная оценка таким способом получена быть не может, так как, вообще говоря, неверно, что  $a_i > 10a_{i+2}$  (пример:  $a_0 = 49$ ,  $a_1 = 30$ ,  $i = 1$ ).

Двоичная и ряд других оценок получаются исходя из некоторого подобия хооровской семантики — если вернуться к обозначениям (1), то можно сказать, что фиксируется некоторое свойство инструкции  $P$ ;  $P$ , например

$$\{[|v|, \infty)\}P; P\{(2|v|, \infty)\}$$

( $P$  описывает переход от пары чисел  $x, y$  к паре  $y$ , остаток  $(x, y)$ ). Но оказывается, что оценки, выводимые таким способом, т. е. с использованием простой семантики хооровского типа инструкций вида  $P; P; \dots; P$ , хуже тех, которые требуют для своего вывода рассмотрения всего алгоритма в целом, т. е. для вывода которых нужно привлечение неравенства (2). Эти элементарно выводимые оценки, и только они, оказываются улучшаемыми — знак  $\leq$  в них можно заменить на знак  $<$ . В частности, двоичную оценку можно переписать в виде  $m < 2\Pi_2(a_1)$ , а десятичная оценка не допускает такой замены. Перед

тем как привести необходимые доказательства, условимся о терминологии.

Определение 1. Оценка  $m \leq c_q \Pi_q(a_i)$  при фиксированном  $q$  называется *улучшаемой* (или *допускающей улучшение*), если на самом деле имеет место  $m < c_q \Pi_q(a_i)$ .

Определение 2. Оценка  $m \leq c_q \Pi_q(a_i)$  при фиксированном  $q$  называется *локализуемой* (или *допускающей локализацию*), если для  $i = 1, 2, \dots, m - c_q$  выполнено  $a_i > qa_{i+c_q}$ .

По индукции с использованием равенства  $a_j = q_{j+1}a_{j+1} + a_{j+2}$  для  $i = 1, 2, \dots, m$  может быть доказано

$$a_i = Q_{i-1}(q_2, \dots, q_i)a_i + Q_{i-2}(q_2, \dots, q_{i-1})a_{i+1}, \quad (5)$$

где  $Q_0, Q_1, \dots$  — многочлены Эйлера:

$$Q_0 = 1, \quad Q_1(x_1) = x_1,$$

$$Q_n(x_1, \dots, x_n) = x_n Q_{n-1}(x_1, \dots, x_{n-1}) + Q_{n-2}(x_1, \dots, x_{n-2}).$$

Индукцией нетрудно доказать, что

$$a_i > u_i a_i, \quad i = 1, 2, \dots, m. \quad (6)$$

Соотношения (5), (6) позволяют описать все  $q$ , для которых оценка (4) допускает локализацию.

Теорема 1. Пусть  $i \geq 2$ . Для каждого действительного  $\varepsilon > 0$  можно подобрать такие натуральные  $a_0$  и  $a_1$ , что алгоритм Евклида в применении к ним потребует не менее  $i-1$  делений и при этом будет выполнено  $a_i/a_i < u_i + \varepsilon$ .

Доказательство. Выберем  $N$  настолько большим, чтобы выполнялось  $u_{i-1}/N < \varepsilon$ , и рассмотрим последовательность чисел, заданную рекуррентно:

$$v_1 = 1, \quad v_2 = N, \quad v_n = v_{n-1} + v_{n-2}.$$

Положим теперь  $a_0 = v_{i+2}$ ,  $a_1 = v_{i+1}$ . Имеем  $q_1 = \dots = q_{i-1} = 1$ ; принимая во внимание, что  $Q_i(1, 1, \dots, 1) = u_i$ , в силу (5) получаем:

$$a_1 = u_i a_i + u_{i-1}, \quad \frac{a_1}{a_i} = u_i + \frac{u_{i-1}}{N} < u_i + \varepsilon.$$

Сопоставление утверждения теоремы 1 с неравенством (6) позволяет получить два следствия.

Следствие 1. Для  $i \geq 2$  и для любых  $a_0, a_1$  таких, что применение к ним алгоритма Евклида не заканчивается после  $i-1$  делений, имеет место неравенство

$a_1/a_i > u_i$ . В то же время можно подобрать такие числа  $a_0$  и  $a_1$ , что применение к ним алгоритма Евклида не заканчивается после  $i - 1$  делений и  $a_1/a_i < u_i + 1$ .

**Следствие 2.** Оценка для числа делений  $t \leq c_q \Pi_q(a_1)$  допускает локализацию, т. е.  $a_1/a_{c_q+1} > q$  тогда и только тогда, когда

$$u_{c_q+1} \geq q. \quad (7)$$

Возвращаясь к оценке  $t \leq 5\Pi_{10}(a_1)$ , видим, что она не допускает локализации:  $u_8 = 8 < 10$ . Оценка  $t \leq 2\Pi_2(a_1)$  допускает локализацию, так как  $u_3 = 2$ . Приведем список всех значений  $q$ , не превосходящих 10, для которых оценка (8) допускает локализацию: 2, 3, 5, 7, 8.

**Теорема 2.** Оценка  $t \leq c_q \Pi_q(a_1)$  допускает локализацию тогда и только тогда, когда для некоторого натурального  $t$  выполнено

$$\left(\frac{1 + \sqrt{5}}{2}\right)^{t-1} < q \leq u_{t+1}.$$

В этом случае  $c_q = t$ , т. е.  $t \leq t\Pi_q(a_1)$ .

Доказательство выводится из следствия 2 и из неравенства

$$\left(\frac{1 + \sqrt{5}}{2}\right)^{t-1} < u_{t+1} < \left(\frac{1 + \sqrt{5}}{2}\right)^t,$$

справедливого для всех натуральных  $t$ .

**Теорема 3.** Пусть оценка  $t \leq c_q \Pi_q(a_1)$  не допускает локализации. Тогда она неулучшаема в следующем смысле: можно подобрать такие  $a_0$ ,  $a_1$ , что применение к ним алгоритма Евклида потребует точно  $c_q \Pi_q(a_1)$  делений.

**Доказательство.** По теореме 2 найдется натуральное  $t$  такое, что

$$u_t < q < \left(\frac{1 + \sqrt{5}}{2}\right)^{t-1}.$$

Берем  $a_0 = u_{t+1}$ ,  $a_1 = u_t$ . Тогда получим  $\Pi_q(a_1) = 1$ ,  $c_q = \lceil \log_{(1+\sqrt{5})/2} q \rceil + 1 = t - 1$ ; число делений равно  $t - 1$ .

**Теорема 4.** Пусть оценка  $t \leq c_q \Pi_q(a_1)$  допускает локализацию. Тогда она улучшаема: справедливо неравенство  $t < c_q \Pi_q(a_1)$ .

**Доказательство.** Пусть  $q^{s-1} \leq a_1 < q^s$  для некоторого натурального  $s$ . Проведем индукцию по  $s$ :



1)  $s = 1$ ; используя (7) и (6), получаем

$$u_{c_q+1} \geq q > a_1 \geq u_{m+1},$$

откуда  $c_q > m$ , что и требуется;

2)  $s > 1$ ; если  $c_q > m$ , то доказывать нечего; в противном случае в силу того, что оценка допускает локализацию, имеем  $a_{c_q+1} < q^{s-1}$ .

По предположению индукции,

$$m - c_q < c_q \Pi_q(a_{c_q+1}),$$

отсюда получаем

$$m < c_q (\Pi_q(a_{c_q+1}) + 1) \leq c_q \Pi_q(a_1).$$

Доказательство завершено.

Рассмотрим теперь дополнительно некоторые вопросы, связанные с освобождением памяти в процессе применения алгоритма Евклида. Из неравенства  $a_i > 2a_{i+2}$  следует, что переход от пары чисел  $a_i, a_{i+1}$  к паре  $a_{i+1}, a_{i+2}$  понижает суммарное количество цифр в двоичной записи чисел по крайней мере на 1, т. е. с каждым шагом алгоритма Евклида заведомо освобождается 1 бит памяти. Освобождающаяся память может быть занята под другие вычисления. Известно несколько алгоритмов, которые выполняют шаг за шагом параллельно с алгоритмом Евклида, например построение для двух целых чисел  $a_0$  и  $a_1$  таких целых чисел  $s$  и  $t$ , что  $a_0s + a_1t = \text{н. о. д. } (a_0, a_1)$ . Для получения  $s$  и  $t$  последовательно строятся  $s_0, t_0; s_1, t_1; \dots$  такие, что  $a_0s_i + a_1t_i = a_i$ . Ясно, что  $s_0 = t_1 = 1, s_1 = t_0 = 0$  и для  $i = 2, 3, \dots, m$  выполнено

$$s_i = (-1)^i Q_{i-2}(q_2, \dots, q_{i-1}), \quad t_i = (-1)^{i+1} Q_{i-1}(q_1, \dots, q_{i-1}).$$

Частный случай этого алгоритма — алгоритм обращения натурального  $a$  в поле вычетов по простому модулю  $p$  при  $a < p$ :  $ps + at = 1$  влечет за собой  $at \equiv 1 \pmod{p}$ , значение  $s$  здесь интереса не представляет.

Выведем оценку объема памяти, необходимого для применения последнего алгоритма. Пусть для записи чисел выбрана двоичная система. Будет показано, что  $2\Pi_2(p) + C$  разрядов, где  $C$  — небольшая константа, достаточно для решения задачи. Сохраняя введенную нами выше систему обозначений, считаем, что  $a_0 = p, a_1 = a$ . При выводе оценки основную роль будет играть следующая теорема.

Теорема 5. Для  $i = 0, 1, \dots, m$  имеют место неравенства

$$\Pi_2(|t_i|) + \Pi_2(a_i) \leq \Pi_2(a_0) + 1,$$

$$\Pi_2(|t_i|) + \Pi_2(a_{i+1}) + \Pi_2(q_i) \leq \Pi_2(a_0) + 2.$$

Доказательство. Используя тот факт, что  $a_0 = Q_m(q_1, \dots, q_m), |t_i| = Q_{i-1}(q_1, \dots, q_{i-1}), a_{i+1} = Q_{m-i-1}(q_{i+2}, \dots, q_m)$ , имеем  $\Pi_2(|t_i|) + \Pi_2(a_i) = ([\log_2 |t_i|] + 1) + ([\log_2 a_i] + 1) \leq \leq \log_2 |t_i| a_i + 2 = \log_2 Q_{i-1}(q_1, \dots, q_{i-1}) Q_{m-i}(q_{i+1}, \dots, q_m) + 2 < < \log_2 a_0 + 2.$

Поскольку  $\Pi_2(|t_i|) + \Pi_2(a_i) - \text{целое число}$ , то выполнено  $\Pi_2(|t_i|) + \Pi_2(a_i) \leq [\log_2 a_0] + 2 = \Pi_2(a_0) + 1$ . Далее

$$\begin{aligned} \Pi_2(|t_i|) + \Pi_2(q_i) + \Pi_2(a_{i+1}) &\leq \log_2 |t_i| q_i a_{i+1} + 3 = \\ &= \log_2 Q_{i-1}(q_1, \dots, q_{i-1}) q_i Q_{m-i-1}(q_{i+2}, \dots, q_m) \leq \log_2 a_0 + 3, \end{aligned}$$

и аналогично предыдущему получаем, что

$$\Pi_2(|t_i|) + \Pi_2(q_i) + \Pi_2(a_{i+1}) \leq [\log_2 a_0] + 3 = \Pi_2(a_0) + 2.$$

Теорема доказана.

Деля «уголком»  $a_i$  на  $a_{i+1}$ , можно неполное частное  $q_{i+1}$  и остаток  $a_{i+2}$  получить на том месте, где располагалось  $a_i$ : разрядов, занятых  $a_i$ , заведомо хватит на это. Для вычисления  $|t_{i+2}| = |q_{i+1}| |t_{i+1}| + |t_i|$  используем место, занятое под  $|t_i|$  и  $q_{i+1}$ : цифры числа  $q_{i+1}$ , начиная с младшей, при вычислении произведения «столбиком» становятся одна за другой ненужными. Если память — это упорядоченная последовательность двоичных разрядов, то желательно, чтобы каждое из рассматриваемых чисел занимало несколько последовательных разрядов. Можно при четном  $i$  рассматриваемые числа расположить так:

$$\begin{array}{cc} \overleftarrow{|t_i|} \overrightarrow{\alpha a_i} & \overleftarrow{a_{i+1} \alpha} \overrightarrow{|t_{i+1}|} \\ \hline \Pi_2(a_0) + 2 & \Pi_2(a_0) + 2 \\ \text{разрядов} & \text{разрядов} \end{array} \quad (8).$$

Надписанная стрелка  $\rightarrow$  означает, что цифры числа идут поряд (слева направо), начиная со старшей цифры, стрелка  $\leftarrow$  означает обратный порядок; буквой  $\alpha$  всюду обозначены группы не представляющих интереса цифр. При нечетном  $i$  величины с индексами  $i$  и  $i+1$  меняются местами.

Преобразование содержимого первых  $\Pi_2(a_0) + 2$  двоичных разрядов в (8) для перехода от величин с индексом  $i$  к величинам с индексом  $i+2$  происходит так:

$$\overleftarrow{|t_i|} \overrightarrow{\alpha q_{i+1} \alpha a_{i+2}}, \quad \overleftarrow{|t_i|} \overrightarrow{\alpha q_{i+1} a_{i+2}}, \quad \overleftarrow{|t_{i+2}|} \overrightarrow{\alpha a_{i+2}}.$$

Теорема 5 гарантирует, что места для этого преобразования хватит.

**Примечание 3.** Выше говорилось о  $\Pi_2(p) + C$  разрядах, а не о  $2\Pi_2(p) + 4$ , так как еще небольшое число разрядов требуется для переносов единиц при сложении и вычитании «столбиком» и т. д.

Если самое последнее значение  $|t_m|$  окажется справа (т. е.  $m$  нечетно), то  $t_m$  — отрицательное число и для нахождения обратного элемента к  $a$  надо еще вычесть  $|t_m|$  из  $p$ . Это обстоятельство, однако, вовсе не вынуждает помнить значение  $p$  до конца вычислений; легко усматривается, что  $p = t_{m+1}$ .

## ГЛАВА II

### О ВОЗМОЖНОСТЯХ МЕТОДА ХООРА

Рассматриваются примеры циклических программ, некоторые свойства которых являются «труднодоказуемыми» с позиций метода Хоора. Большинство этих программ укладывается в схему

$$\text{while } \xi(a) \text{ do } b := g(b); \quad a := f(a) \text{ od},$$

где  $f$  — монотонно убывающая, а  $g$  — монотонно возрастающая, в смысле некоторого отношения порядка, функции. Эта схема подсказана доказательством теоремы Ламе: там последовательность значений переменной  $a$  — это последовательность остатков, а последовательность значений переменной  $b$  — последовательность чисел Фибоначчи, начиная со второго.

Подробно обсуждается характер трудностей, возникающих при применении метода Хоора к такого рода программам.

Этими исследованиями завершается рассмотрение вопросов анализа детерминированных программ.

#### § 1. Импликации, связанные с циклами

Ниже предлагаются две циклические программы  $P$  и  $P'$ , главный итог выполнения каждой из которых — получение логического значения. Ввиду этого соответствующие программам  $P$  и  $P'$  функции могут рассматриваться как предикаты — мы их обозначим  $\pi$  и  $\pi'$ . Предикаты  $\pi$  и  $\pi'$  нельзя представить конечными композициями элементарных (допустимых) операций и для описания  $P$ ,  $P'$  необходимы циклические инструкции. Предикаты  $\pi$ ,  $\pi'$  оказываются более слабыми, чем некоторые элементарные отношения; соответствующие импликации записываются в виде свойств программ  $P$  и  $P'$ . Обнаруживается, что для доказательства каждого из этих свойств методом Хоора требуется инвариант, эквивалентный исходному предикату —  $\pi$  или  $\pi'$ . Из этого следует, что если доказательство импликации, записанной в виде свойства программы  $P$  или  $P'$ , было выполнено методом Хоо-

ра, то правая часть импликации была заменена на эквивалентную и затем способом, не оговоренным методом Хоора, была доказана получившаяся импликация. Оказывается также, что для программы  $P'$  в выбранном языке не существует такой эквивалентной ей программы, для которой применение метода Хоора не приводило бы к этому же эффекту.

При построении программы  $P$  будем исходить из двух типов значений: арифметического — неотрицательных целых чисел — и логического — символов И, Л\*). Для значений арифметического типа допустимыми объявляются операции следования  $x + 1$ , предшествования  $x - 1$  (считается, что  $0 - 1 = 0$ ) и операции  $>$ , дающая значение И или Л. Для значений логического типа допустимыми объявляются операции  $\wedge, \vee, \neg$ .

Предварительно рассмотрим программу  $Q$ :

**while**  $a > 0$  **do**  $b := b + 1; a := a - 1$  **od**.

Очевидно, что после выполнения  $Q$  переменная  $b$  примет значение суммы исходных значений  $a$  и  $b$  и что сумма не выражается в виде конечной композиции допустимых операций. Для доказательства свойства  $\{a > c\}Q\{b > c\}$  методом Хоора надо подобрать такой предикат  $\xi(a, b, c)$ , что:

а)  $(a > c) \supset \xi(a, b, c)$ ;

б)  $((a > 0) \wedge \xi(a, b, c)) \supset \xi(a - 1, b + 1, c)$ ;

в)  $\xi(a, b, c) \supset (b > c)$ .

Лемма 1.  $\xi(a, b, c) = (a + b > c)$ .

Если  $a + b = v$ , то, подставляя в а) число  $v$  вместо переменной  $a$  и 0 вместо переменной  $b$ , получаем  $(v > c) \supset \xi(v, 0, c)$ . В силу б) имеем  $\xi(v, 0, c) \supset \xi(v - 1, 1, c) \supset \dots \supset \xi(0, v, c)$ . В силу в) имеем  $\xi(0, v, c) \supset (v > c)$ . Таким образом,  $(v > c) \supset \xi(v - b, b, c) \supset (v > c)$ , или  $(a + b > c) \supset \xi(a, b, c) \supset (a + b > c)$  — это и есть утверждение леммы.

Лемму можно усилить: для доказательства любого свойства вида  $\{p\}Q\{\sigma\}$ , где  $(a > c) \supset p$ ,  $\sigma \supset (b > c)$ , требуется взять инвариант, эквивалентный  $a + b > c$ , так как а), б), в) остаются выполненными.

---

\*) В этой главе рассматривается традиционный вариант метода промежуточных утверждений. Соответственно принята традиционная точка зрения и на предикаты.

Напишем программу  $P$  вычисления логического значения  $t$ :

**while**  $a > 0$  **do**  $b := b + 1$ ;  $a := a - 1$  **od**;  $t := (b > c)$ .

Очевидно, что  $(a > c) \supset \pi(a, b, c)$  — здесь  $\pi(a, b, c)$  означает то логическое значение, которое дает выполнение программы  $P$ ; это естественно записывается в виде свойства  $\{a > c\}P\{t\}$ . Для доказательства последнего свойства методом Хоора надо подобрать утверждение  $\sigma$  такое, что имеют место свойства  $\{a > c\}Q\{\sigma\}$ ,  $\{\sigma\} t := (b > c) \{t\}$ . Рассмотрение последнего свойства дает  $\sigma \supset (b > c)$ . После всего сказанного становится очевидной следующая теорема.

*Теорема 1. Для доказательства методом Хоора свойства  $\{a > c\}P\{t\}$  в качестве инварианта  $\xi$  циклической инструкции  $Q$ , входящей в программу  $P$ , необходимо привлечь эквивалентную  $\pi$  логическую функцию.*

После выбора утверждения  $\xi$  надо, в частности, доказать (способ доказательства, естественно, в методе Хоора не указывается), что  $(a > c) \supset \xi(a, b, c)$ .

Метод Хоора для цикла — это, по сути, индуктивное доказательство подходящего утверждения по числу этапов цикла. Исследование свойства  $\{a > c\}P\{t\}$  показывает, что основание индукции может оказаться эквивалентным доказываемому утверждению.

Приведенные рассуждения указывают на связанные с применением метода Хоора сложности, которые возникают отнюдь не из-за того, что выбран тот, а не иной язык записи промежуточных утверждений («язык спецификаций»), и эти сложности нельзя обойти выбором «хорошего» языка. Но следствием этих сложностей являются и некоторые сложности языкового характера; один их аспект будет рассмотрен сейчас, другому аспекту посвящен следующий параграф.

Полученные результаты мы обсудим с распространенной в программировании точки зрения, согласно которой программы, предлагаемые для вычисления каких-то величин (т. е. функций от исходных значений переменных), могут рассматриваться как определения этих величин (функций). Эта точка зрения правомерна, так как, например, в тех случаях, когда конечных композиций допустимых операций (функций) не хватает для описания итоговых величин, циклическая инструкция описывает интересные для нас величины точнее, чем, скажем, многоточие или слова «и так далее». Определяя функции с

помощью программ, мы затем сталкиваемся с необходимостью как-то работать с этими определениями и прежде всего доказывать свойства определенных этим способом функций. Для этого часто выбирается метод Хоора.

Рассмотренная программа  $P$ , можно считать, представляет собой определение логической функции  $a + b > c$ . В программе использован цикл, что неизбежно при данном наборе допустимых операций. Метод Хоора в отношении циклов представляет собой определенный тип математической индукции. Но для данного конкретного случая этот тип индукции оказывается неудачным, и применение метода Хоора приводит фактически к получению рекомендации воспользоваться другим определением функции  $\pi(a, b, c)$  и, быть может, другим методом доказательства. Для  $\pi(a, b, c)$  можно дать другой вариант определяющей его программы:

$\text{while } b > 0 \text{ do } b := b - 1; a := a + 1 \text{ od}; t := (a > c). \quad (1)$

Тогда в качестве инварианта, обеспечивающего доказательство импликации  $(a > c) \supset \pi(a, b, c)$ , можно взять  $a > c$ . Если определением  $\pi$  была программа  $P$ , то программу (1) можно считать определением инварианта  $\xi$ . Простой инвариант  $a > c$  входящей в (1) циклической инструкции делает, таким образом, успешным второй этап применения метода Хоора. Существование удобного эквивалентного определения объясняется в данном случае коммутативностью операции сложения чисел.

Однако в некоторых случаях, в удобной для проведения доказательства редакции программы, заведомо должны использоваться операции, которые заранее не объявлялись допустимыми. Пусть  $A^*$  — множество слов в алфавите  $A$ , пустое слово обозначим  $\emptyset$ . Допустимыми объявляются следующие операции:  $\text{out}(x)$  — выбрасывание первой буквы из непустого слова  $x$ ;  $\text{in}(x, y)$  — приписывание первой буквы непустого слова  $x$  в конец произвольного слова  $y$  (все слова, таким образом, рассматриваются как очереди); операция отношения вхождения слова  $x$  в слово  $y$ :  $x \equiv y$ .

Рассмотрим программу  $P'$ :

$\text{while } \neg(a \equiv \emptyset) \text{ do } b := \text{in}(a, b); a := \text{out}(a) \text{ od}; t := (c \equiv b).$

Свойство  $\{c \equiv a\}P'\{t\}$  соответствует импликации  $(c \equiv a) \supset P'(a, b, c)$ . Если алфавит  $A$  — однобуквенный, то мы, по существу, имеем дело с прежней программой; если букв больше одной, то возникают новые осложнения.

Аналогично предыдущему доказываем, что нужным инвариантом может быть только  $c \subseteq b \ a$ , где  $\subseteq$  — знак конкатенации. Циклическая инструкция, входящая в  $P'$ , и определяет операцию конкатенации; более того, если для построения слов  $b \ a$  пользоваться только операциями *in*, *out*, то построение  $b \ a$  потребует именно того порядка выполнения операций, который задается этой циклической инструкцией (в частности, операция  $\subseteq$  не коммутативна). Поэтому для применения метода Хоора придется переписать  $P'$ , используя какие-то другие операции.

Если мы хотим стоять на упоминавшейся удобной программистской позиции и пользоваться программами для определения функций, то при доказательстве импликации  $(c \subseteq a) \supset \pi'(a, b, c)$ , где  $\pi'(a, b, c)$  определяется посредством  $P'$ , метод Хоора будет вынуждать нас либо определять  $\pi'(a, b, c)$  посредством другой программы, написанной с привлечением не тех операций, исходя из которых мы с самого начала хотели определить  $\pi'$ , либо доказывать импликацию другим методом.

## § 2. Характер неполноты системы Хоора

Известно, что система Хоора для программ, состоящих из условных, циклических, составных инструкций и из инструкций присваивания, может оказаться неполной, коль скоро языком для записи предусловий и постусловий (языком спецификаций) выбран язык первого порядка, т. е. язык, устроенный по образцу языка исчисления предикатов первого порядка. Такой язык использует, в частности, обозначения некоторых констант (индивидуальных предметов), индивидуальных предикатов и предметных функций. К числу этих обозначений принадлежат обозначения всех констант, предикатов и функций, допустимых в самих программах.

Для доказательства возможности неполноты М. Вэнд рассмотрел циклическую программу и такое ее хооровское свойство, для вывода которого необходимо привлечение непредставимого в выбранном языке спецификаций инварианта [17]. Однако предусловие рассмотренного свойства описано при помощи предиката  $\rho$ , который не участвует в записи программы; сама программа написана исходя из очень простых функций и предикатов, предикат же  $\rho$  по отношению к этим необходимым функ-

циям и предикатам настолько сложен, что не может быть выражен через них формулой первого порядка.

Для того чтобы обстоятельно обсудить вопрос о неполноте, необходимо внести некоторые уточнения. Любая программа может быть выполнена только тогда, когда в некотором множестве задана интерпретация констант, знаков операций и предикатов, участвующих в программе. Множество с набором операций, позволяющих получать одни элементы, исходя из других, набор предикатов, аргументы которых принимают значения в этом множестве, а также система обозначений для этих операций, предикатов и конкретных элементов множества будут в дальнейшем называться, вместе взятые, *базисом*. О программе, написанной в обозначениях базиса  $B$ , выполнение которой понимается в смысле связанной с  $B$  интерпретации, будем говорить как о *программе в базисе  $B$* .

Отметим, что логические операции  $\vee, \wedge, \neg, \supset$  явно не включаются в базис — предполагается, что они могут использоваться в программах в любом базисе. С каждым базисом  $B$  свяжем два языка:  $L_0(B)$  и  $L_1(B)$  — язык бескванторных предикатных формул и язык предикатных формул первого порядка; в обоих этих языках константы суть обозначения элементов входящего в базис  $B$  множества, обозначения предметных функций и индивидуальных предикатов представляют собой обозначения операций и предикатов, входящих в базис  $B$ . Таким образом, формулы языков  $L_0(B)$  и  $L_1(B)$  представляют предикаты, аргументы которых принимают значения во входящем в  $B$  множестве.

**Определение 1.** Свойство  $\{\varphi\}Q\{\psi\}$  программы  $Q$  в базисе  $B$  называется *характерным*, если, во-первых,  $\varphi$  и  $\psi$  представимы формулами языка  $L_0(B)$  и, во-вторых, какие бы формулы  $F_\varphi, F_\psi \in L_0(B)$ , представляющие  $\varphi$  и  $\psi$ , не взять, удаление из базиса  $B$  хотя бы одной функции или одного предиката, обозначения которых входят в  $F_\varphi$  или  $F_\psi$ , приводит к такому базису, в котором уже не найдется эквивалентной  $Q$  программы.

Пусть  $\{\varphi\}Q\{\psi\}$  — характерное свойство программы  $Q$  в базисе  $B$ . Пусть  $\varphi$  и  $\psi$  представлены формулами  $F_\varphi, F_\psi \in L_0(B)$ . Поставим вопрос, ответ на который никак не следует из результата Вэнда: всегда ли возможно доказать свойство  $\{\varphi\}Q\{\psi\}$ , прибегая лишь к промежуточным утверждениям, представимым в языке  $L_0(B)$  или хотя бы в языке  $L_1(B)$ ? Соответственно в терминологии,



связанной с формальной системой Хоора, вопрос ставится так: всегда ли возможно вывести формулу  $\{F_\psi\}Q\{F_\psi\}$ , прибегая лишь к таким формулам этой системы, у которых предусловия и постусловия представляют собой формулы языка  $L_0(B)$  или хотя бы формулы языка  $L_1(B)$ ? Главное отличие поставленных вопросов от того вопроса, на который дал ответ Вэнд, состоит в том, что в качестве компонент предусловий и постусловий как исходного, так и всех промежуточных свойств, разрешено использовать только такие предикаты, которые необходимы для написания рассматриваемой программы  $Q$ . При этом сказано, что предусловие и постусловие исходного свойства описаны без привлечения кванторов.

Ответ на оба поставленных вопроса оказывается отрицательным. Это показывает, что неполнота системы Хоора имеет более серьезный характер, чем тот, который выявлен в работе Вэнда.

В предыдущем параграфе рассмотрена программа  $Q$ :

$\text{while } a > 0 \text{ do } b := b + 1; a := a - 1 \text{ od.}$

Показано, что для доказательства свойства  $\{a > c\}Q\{b > c\}$  необходим инвариант, эквивалентный в функциональном смысле предикату  $a + b > c$ . Предполагалось, что все вычисления ведутся в множестве неотрицательных целых чисел, над которыми можно проводить операции прибавления единицы и вычитания единицы ( $0 - 1 = 0$ ); допустимый предикат  $- >$ . Будем в дальнейшем обозначать этот базис через  $T$ . Ясно, что  $\{a > c\}Q\{b > c\}$  — это характерное свойство  $Q$  в  $T$ . Исследуем предствавимость  $a + b > c$  в языках  $L_0(T)$ ,  $L_1(T)$ .

*Лемма 1. Пусть  $\alpha(a, b, \dots, r)$  — предикат, аргументы которого принимают значения в множестве неотрицательных целых чисел. Пусть  $\alpha$  представляется формулой языка  $L_0(T)$ . Тогда найдется  $N \geq 0$  такое, что если для неотрицательных целых чисел  $A, B, \dots, R$  имеет место  $A > N, B > N, \dots, R > N$ , то  $\alpha(A, B, \dots, R) = \alpha(A - 1, B - 1, \dots, R - 1)$ .*

*Доказательство.* Можно считать, что отношения, из которых с помощью знаков логических операций и скобок строятся формулы языка  $L_0(T)$ , таковы, что их левые и правые части — это либо числа, либо выражения вида

$$x - \underbrace{1 - 1 - \dots - 1}_u + \underbrace{1 + 1 + \dots + 1}_v, \quad (1)$$

где  $u \geq 0$ ,  $v \geq 0$ ,  $x$  — некоторая переменная из набора  $a, b, \dots, r$ ; левые и правые части соединяются знаком  $>$ . Ясно, что утверждение леммы справедливо для отношений: если обе части отношения — числа, то  $N = 0$ ; если обе части отношения имеют вид (1), то в качестве  $N$  можно взять максимум (для левой и правой частей) чисел, обозначенных в (1) через  $u$ ; если, наконец, одна из частей отношения — число  $n$ , а другая имеет вид (1), то можно положить  $N = u + n + 1$ .

Предположим, что утверждение леммы справедливо для предикатов  $\alpha_1$  и  $\alpha_2$ . Тогда оно справедливо и для  $\alpha_1 \vee \alpha_2$ : в качестве  $N$  надо взять максимум соответствующих чисел для  $\alpha_1$  и  $\alpha_2$ .

Легко выводится из предположения о справедливости утверждения леммы для  $\beta$  справедливость этого утверждения для  $\neg \beta$ :  $N$  для  $\neg \beta$  — то же самое, что и для  $\beta$ .

Остальные логические операции выражаются через  $\vee$ ,  $\neg$ .

Из доказанной леммы немедленно следует непредставимость  $a + b > c$  в  $L_0(T)$ : для любого  $N$  имеет место  $(N + 1) + (N + 1) > (2N + 1)$ , но не имеет места  $N + N > 2N$ .

В сущности, леммой 1 устанавливается, что те свойства точек на прямой, которые описываются формулами языка  $L_0(T)$ , суть проективные, а не метрические свойства. Свойство же  $a + b > c$  — метрическое. Докажем теперь, что привлечение кванторов не дает возможности описывать исходя из базиса  $T$  метрические свойства. Более того, это привлечение кванторов вообще не дает ничего нового.

**Лемма 2.** Если предикат представим формулой языка  $L_1(T)$ , то он представим и формулой языка  $L_0(T)$ .

**Доказательство.** Покажем, что привлечение квантора существования не расширяет множества представимых предикатов (квантор всеобщности можно отдельно не рассматривать, так как  $(\forall x \alpha) = (\neg \exists x \neg \alpha)$ ). Достаточно разобрать случай, когда некоторая переменная, например  $r$ , связывается при помощи квантора существования в бескванторной формуле  $\alpha(a, b, \dots, r)$  — далее можно применить индукцию.

Можно вновь считать, что формула  $\alpha(a, b, \dots, r)$  получается с помощью знаков логических операций и скобок из отношений, левая и правая части каждого из которых есть либо число, либо выражение вида (1). Каждое отношение можно заменить эквивалентной (в смысле

функционального равенства предикатов) формулой языка  $L_0(T)$ , в которой каждое отношение таково, что либо левая и правая его части не содержат переменной  $r$ , либо сама переменная  $r$  образует одну из частей отношения, а в другую часть переменная  $r$  не входит (левая и правая части, как и прежде, соединяются знаком  $>$ ). Например, отношение  $r - 1 - 1 - 1 + 1 + 1 + 1 + 1 > y$  можно заменить на

$$(4 > y) \vee ((r > 3) \wedge (r > y - 1)).$$

В результате подобных замен все отношения, входящие в формулу  $\alpha$ , разрешаются относительно  $r$ . Получающаяся таким образом формулу можно привести к дизъюнктивной форме

$$\beta_1 \vee \beta_2 \vee \dots \vee \beta_n, \quad (2)$$

где каждое  $\beta_i$  есть некоторая конъюнкция

$$\gamma_1 \wedge \gamma_2 \wedge \dots \wedge \gamma_m \quad (3)$$

отношений указанного вида, при этом некоторые отношения могут входить в эту конъюнкцию со знаком отрицания.

Бескванторную формулу (конъюнкцию), которая представляет предикат

$$\exists r (\gamma_1 \wedge \gamma_2 \wedge \dots \wedge \gamma_m),$$

можно построить исходя из следующего соображения. Каждая из содержащих переменную  $r$  формул  $\gamma_1, \dots, \gamma_m$  дает верхнюю или нижнюю оценку для значения  $r$ . Необходимым и достаточным условием существования значения  $r$ , удовлетворяющего всем этим оценкам, является то, что любая строгая верхняя оценка должна быть больше любой строгой нижней оценки по крайней мере на 2. Будем действовать так. Те члены конъюнкции (3), которые не содержат  $r$ , без изменения включаются в новую конъюнкцию. Остальные члены сами не будут входить в новую конъюнкцию, но каждая пара  $r > s$ ,  $\neg(r > t)$ , где  $s$  и  $t$  — числа или выражения вида (1), даст член конъюнкции  $s > t$ , каждая пара  $r > t$ ,  $s > r$  даст член  $s + 1 > t$  и т. д. Этим способом мы получим формулу языка  $L_0(T)$ , эквивалентную в функциональном смысле формуле  $\exists r (\gamma_1 \wedge \gamma_2 \wedge \dots \wedge \gamma_m)$ . Преобразовав так каждый член конъюнкции (2), мы получим формулу, функционально эквивалентную формуле  $\alpha(a, b, \dots, r)$ ,

так как формула  $\exists r(\beta_1 \vee \beta_2 \vee \dots \vee \beta_n)$  функционально эквивалентна  $\exists r\beta_1 \vee \exists r\beta_2 \vee \dots \vee \exists r\beta_n$ .

Из всего сказанного следует, что известная теорема о неполноте логической системы Хоора над языком первого порядка может быть усилена. Справедлива следующая теорема.

**Теорема 1.** *В рассмотренном базисе  $T$  существует циклическая программа  $Q$ , которая обладает характерным свойством  $\{\phi\}Q\{\psi\}$ , и при этом доказательство последнего требует привлечения инварианта, не представимого никакой формулой языка  $L_1(T)$ .*

### § 3. Флойдовские индуктивные утверждения и хооровские инварианты

Понятие инварианта, определенное Хоором для циклической инструкции вида

$$\text{while } \beta \text{ do } S \text{ od} \quad (1)$$

как предикат  $\xi$ , для которого выполнено

$$\{\beta \wedge \xi\} S \{\xi\},$$

представляет собой формализацию понятия индуктивного утверждения, введенного ранее Флойдом. Индуктивное утверждение, согласованное с предусловием  $\phi$ , — это такое приписанное некоторой точке программы утверждение о значениях переменных, которое, при условии, что до начала выполнения программы относительно переменных было справедливо  $\phi$ , будет справедливым при каждом проходе через упомянутую точку. Рассмотрим некоторое свойство программы  $P$ , описанной формулой (1):

$$\{\phi\} \text{ while } \beta \text{ do } S \text{ od } \{\psi\}. \quad (2)$$

Хооровское доказательство такого свойства — это указание предиката  $\xi$ , для которого устанавливается, что:

- а)  $\phi \supset \xi$ ;
  - б)  $\{\beta \wedge \xi\} S \{\xi\}$ ;
  - в)  $(\xi \wedge \neg\beta) \supset \psi$ .
- (3)

Для доказательства свойства (2) по Флойду можно, например, точке программы, непосредственно предшествующей символу **od**, приписать некоторый предикат  $\eta$  такой, что, во-первых, из предположения об истинности  $\phi$  до выполнения программы следует его истинность при каждом проходе через точку, непосредственно предшеств-

вующую символу  $\text{od}$  (т. е. его истинность после каждого выполнения  $S$ ), а во-вторых, из  $\eta \wedge \neg \beta$  следует  $\psi$ .

Понятно, что любой хооровский инвариант задает индуктивное утверждение, пригодное для доказательства того же самого свойства. Цель этого параграфа — показать, что некоторые естественные индуктивные утверждения не являются хооровскими инвариантами. Первый пример получается по той схеме, которая была описана во введении в настоящую главу. Рассмотрим программу  $P$

while  $a > 1$  do  $b := 2b; a := \left\lfloor \frac{a}{2} \right\rfloor$  od.

В качестве значений переменных будем допускать натуральные числа. Нас будет интересовать следующее свойство рассматриваемой программы:

$$\{(a = A) \wedge (b = 1)\} P \{A < 2b\}.$$

Докажем, что если до начала выполнения  $P$  имело место  $(a = A) \wedge (b = 1)$ , то во время выполнения  $P$  при каждом проходе через точку, непосредственно следующую за **do**, и через точку, непосредственно предшествующую **od**, будет иметь место неравенство  $A < 2ab$ . Из этого, конечно, сразу будет следовать постусловие. Доказательство проведем в два этапа. Во-первых, при всех прохождениях через упомянутые точки имеет место равенство  $\lceil \log_2 A \rceil = \lceil \log_2 a \rceil + \lceil \log_2 b \rceil$  (индукция). Во-вторых,

$$(\lceil \log_2 A \rceil = \lceil \log_2 a \rceil + \lceil \log_2 b \rceil) \supset (A < 2ab);$$

в самом деле, если  $A \geq 2ab$ , то

$$\lceil \log_2 A \rceil \geq \lceil 1 + \log_2 a + \log_2 b \rceil \geq 1 + \lceil \log_2 a \rceil + \lceil \log_2 b \rceil$$

(противоречие).

Теперь покажем, что предикат  $A < 2ab$  не является хооровским инвариантом программы  $P$ . В самом деле, предикат  $\xi(a, b, A)$ , являющийся инвариантом программы  $P$ , должен удовлетворять пункту б) формулы (3), который в данном случае может быть переписан (согласно семантике Хоора для инструкции присваивания) так:

$$((a > 1) \wedge \xi(a, b, A)) \supset \xi([a/2], 2b, A).$$

Но для  $A < 2ab$  это не выполнено: неверно, что

$$((a > 1) \wedge (A < 2ab)) \supset (A < 2 \cdot [a/2] \cdot 2 \cdot b).$$

Пусть, например,  $A = 19$ ,  $a = 5$ ,  $b = 2$ . Тогда  $19 < 2 \cdot 5 \times 2$ , но  $19 > 2 \cdot 2 \cdot 2 \cdot 2$ .

Взаимоотношение понятий индуктивного утверждения и хооровского инварианта можно охарактеризовать так: хооровский инвариант — это такое индуктивное утверждение, истинность которого при каждом проходе через точку, непосредственно следующую за **do**, и через точку, непосредственно предшествующую **od**, устанавливается простейшей индукцией: индуктивный переход таков, что из предположения об истинности утверждения до выполнения  $i$ -го этапа следует его истинность после  $i$ -го этапа. В том примере, который был разобран выше, истинность предиката  $A < 2ab$  после  $i$ -го этапа не может быть выведена из столь слабого предположения — надо использовать предположение об истинности этого предиката после этапов с номерами  $1, 2, \dots, i - 1$ .

Приведенный пример дает возможность еще раз убедиться, что указанная во введении к настоящей главе схема действительно описывает программы, которые преподносят различные трудности для применения к исследованию этих программ метода Хоора. Но если отвлечься от указанной схемы, то можно привести еще более простой пример: имеет место свойство

$$\{x = 0\} \text{ while } y < 10 \text{ do } x := 2x; y := y + 1 \text{ od } \{x < 1\}, \quad (4)$$

при этом очевидно, что индуктивным утверждением, согласованным с предусловием  $x = 0$  и обеспечивающим доказательство свойства (4), является, в частности,  $x < 1$ . Однако  $x < 1$  не является хооровским инвариантом цикла, так как из  $(y < 10) \wedge (x < 1)$  не следует, что  $2x < 1$ .

Приведем теперь пример, когда истинность предиката после  $i$ -го этапа при  $i \geq 3$  следует из предположения об истинности этого предиката после этапов с номерами  $i - 1$  и  $i - 2$ . Этот пример уже появлялся в гл. 1, § 3 (программа  $R$ ). Рассмотрим свойство

$$\{m = n\} \text{ while } k \neq 0 \text{ do } k := k - 1; n := n + (-1)^n \\ \text{od} \{|n - m| \leq 1\}.$$

При каждом проходе через точку, непосредственно следующую за **do**, и через точку, непосредственно предшествующую **od**, справедливо  $|n - m| \leq 1$  — дело в том, что, как говорилось, последовательность значений, принимаемых переменной  $n$ , периодична с периодом 2; отсюда индуктивный переход: если  $|n - m| \leq 1$  выполнялось после  $(i - 2)$ -го этапа, то, ввиду неизменяемости значения переменной  $m$ , неравенство  $|n - m| \leq 1$  будет

выполняться после  $i$ -го этапа. Основание индукции — несложная проверка утверждения для первого и второго этапов. Но  $|n - m| \leq 1$  не является хооровским инвариантом: неверно, что

$$((k \neq 0) \wedge (|n - m| \leq 1)) \supset (|n + (-1)^n - m| \leq 1),$$

это опровергается примером  $k = 1$ ,  $m = 4$ ,  $n = 3$ .

Т. Б. Ельцина ввела и изучила понятие индуктивного утверждения, согласованного с постусловием, понимая под этим такое приписанное некоторой точке программы утверждение о значениях переменных, которое справедливо при каждом проходе через рассматриваемую точку, коль скоро выполнение программы завершается и при этом для заключительных значений переменных истинно постусловие; Т. Б. Ельциной установлено, что индуктивное утверждение, согласованное с постусловием и приписанное внутренней точке цикла, может не быть хооровским инвариантом этого цикла.

Результаты этого параграфа можно кратко сформулировать так: фактическая инвариантность предиката (т. е. неизменность значения И этого предиката) не всегда может быть доказана непосредственным применением метода Хоора. Множество фактических инвариантов, пригодных для доказательства некоторого свойства циклической программы, шире аналогичного множества хооровских инвариантов. Получается, что при доказательстве свойства циклической программы методом Хоора приходится не просто придумывать инварианты в естественном смысле этого слова, т. е. неизменные логические величины, а находить именно такие величины, неизменность которых может быть показана определенным образом — аналогом простейшей индукции.

## ГЛАВА III

### ПРОГРАММЫ И ИНДУЦИРУЕМЫЕ ИМИ ПРЕОБРАЗОВАНИЯ МНОЖЕСТВ ФУНКЦИЙ (ОБЩИЙ СЛУЧАЙ)

Недетерминированность программ дает значительную свободу для определения понятия предусловия; можно, например, следовать соображениям сопряженности и брать композицию сопоставленного должным образом программе бинарного отношения и постусловия. Однако есть и другие способы: уже в простейшем случае, когда функции на  $V$  — это предикаты, конструкция Дейкстры для недетерминированной программы не тождественна сопряженному преобразованию, действующему на предикаты. Ниже рассматриваются главным образом различные варианты понятия предусловия и смежные вопросы.

#### § 1. Обобщение понятий; формулы алгебры бинарных отношений и их следствия

Исследуя недетерминированные программы, мы станем связывать с ними, как связывали и с детерминированными программами, предусловия и постусловия, заданные бинарными отношениями-подмножествами  $V \times M$ , где  $V$  — это множество состояний, а  $M$  — зафиксированное произвольное множество. Детерминированной программе  $P$  сопоставлялась функция  $p$ , определенная и принимающая значения в  $V$ . Если же  $P$  — недетерминированная программа, то ей мы будем сопоставлять два (в общем случае несовпадающих) бинарных отношения  $p^b, p^\#$  таких, что  $p^b \subseteq p^\# \subseteq V \times V$ . Считаем, что  $v_1 p^\# v_1$  тогда и только тогда, когда для начального состояния  $v_0$  выполнение программы  $P$  может завершиться и дать заключительное состояние  $v_1$ . Считаем также, что  $v_0 p^b v_1$  тогда и только тогда, когда имеет место  $v_0 p^\# v_1$  и для начального состояния  $v_0$  выполнение  $P$  обязательно завер-



шается\*). Для любой детерминированной программы справедливо  $p^b = p^u = p$ .

Далее без оговорок принимается, что  $f, g, h \subseteq V \times M$ . Как и в гл. I, бинарные отношения-подмножества произведений  $V \times V$  и  $V \times M$  мы преимущественно будем рассматривать как многозначные частичные функции на множестве со значениями в  $V$  или  $M$ .

Недетерминированные программы задаются, в частности, предложенными Дейкстрой инструкциями *if* и *do* («выбора» и «повторения»):

$$\text{if } \beta_1 \rightarrow P_1 \square \beta_2 \rightarrow P_2 \square \dots \square \beta_n \rightarrow P_n \text{ fi}, \quad (1)$$

$$\text{do } \beta_1 \rightarrow P_1 \square \beta_2 \rightarrow P_2 \square \dots \square \beta_n \rightarrow P_n \text{ od}, \quad (2)$$

где  $\beta_1, \dots, \beta_n$  — предикаты,  $P_1, \dots, P_n$  — инструкции. Будем использовать *if* для записи не только инструкций, но и условных выражений, определяющих частичные многозначные функции на  $V$  со значениями некоторого типа. В таких случаях вместо всех  $P_i$  должны быть записаны некоторые выражения. Предикаты  $\beta_1, \dots, \beta_n$  должны рассматриваться как функции с двухэлементным множеством значений  $\{И, Л\}$ . Пример многозначной функции (бинарного отношения)  $g(n)$ :

$$\text{if } 3 | n \rightarrow И \square 5 | n \rightarrow Л \text{ fi}. \quad (3)$$

Среди значений  $g(n)$ , в зависимости от  $n$ , могут быть одновременно И и Л, только И, только Л, или же  $g(n)$  может быть не определена.

**Определение 1.** Программа  $P$  (вообще говоря, недетерминированная) обладает свойством  $\{f\}P\{g\}$ , если всякий раз, когда для некоторых  $v_0, v_1 \in V$  и  $m \in M$  имеет место  $p^*(v_0) = v_1$  и  $f(v_0) = m$ , то имеет место и  $g(v_1) = m$ . В выражении  $\{f\}P\{g\}$  бинарное отношение  $f$  называется *предусловием* (для  $g$  относительно  $P$ ), а  $g$  — *постусловием* (для  $f$  относительно  $P$ ).

Совершенно ясно, что свойство детерминированной программы, определенное в § 2 гл. I, есть частный случай свойства, определенного только что, так как детер-

---

\*) В литературе недетерминированной программе  $P$  обычно сопоставляется одно бинарное отношение  $p$ , при этом из контекста иногда можно определить, что именно имеется в виду. На самом деле некоторые задачи анализа программ требуют рассмотрения  $p^*$ , а некоторые —  $p^b$ ; поэтому мы и вводим два бинарных отношения,

минированная программа есть частный случай недетерминированной.

Для инструкций **if** и **do**, определенных при помощи (1), (2), можно предположить следующие правила, аналогичные правилам Хоора.

*Инструкция if*: если имеют место свойства

$$\{\beta_i \wedge f\} P_i \{g\}, \quad i = 1, \dots, n,$$

то имеет место и

$$\{f\} \text{if } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{fi } \{g\}$$

( $\beta_i \wedge f$  понимается так, как объяснялось в § 2 гл. I).

*Инструкция do*: если имеют место свойства

$$\{\beta_i \wedge h\} P_i \{h\}, \quad i = 1, \dots, n,$$

то имеет место и

$$\{h\} \text{do } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{od } \{\neg \beta \wedge h\},$$

где  $\beta \equiv \beta_1 \vee \dots \vee \beta_n$ ;  $h$  называется *инвариантом*.

Отметим, что правила 2), 3), 4), 6), данные в § 2 гл. I, сохраняют силу для недетерминированных инструкций  $P, Q$ .

Рассмотрим программу

$$\text{do } 6 \mid n \rightarrow n := n/6 \square 20 \mid n \rightarrow n := 27n/20 \text{od}. \quad (4)$$

Здесь множество состояний — это множество натуральных чисел, значений переменной  $n$ . Пусть бинарное отношение  $f$  определено с помощью

$$\text{if } (3^{100} \mid n) \wedge \neg (2^{100} \mid n) \rightarrow \text{И} \square (50^{50} \mid n) \wedge (2^{60} \mid n) \rightarrow \text{Л fi}. \quad (5)$$

Можно показать, что имеет место свойство  $\{f\}P\{g\}$ , где  $g$  определено с помощью (3). Для описания инварианта удобно воспользоваться функциями  $c_2(n)$ ,  $c_3(n)$ , ...:  $c_k(n) = t$  означает, что  $(k^t \mid n) \wedge \neg (k^{t+1} \mid n)$ . Подходящим инвариантом может служить

$$h(n) = \text{if } (c_3(n) > c_2(n)) \rightarrow \text{И} \square (c_5(n) > c_4(n)) \rightarrow \text{Л fi}, \quad (6)$$

Как и в детерминированном случае, свойство  $\{f\}P\{g\}$  представляет собой совокупность хооровских свойств  $\{f_{(m)}\}P\{g_{(m)}\}$ ,  $m \in M$ , где  $f_{(m)}$ ,  $g_{(m)}$  — предикаты (см. § 2 гл. I).

Рассмотренное выше свойство программы  $P$ , заданной с помощью (4), — это совокупность двух хооровских

$$\{(3^{100} | n) \wedge \neg (2^{100} | n)\} P \{3 | n\},$$

$$\{(5^{50} | n) \wedge \neg (2^{60} | n)\} P \{5 | n\}.$$

Теперь нашей целью будет определение и исследование слабейшего свободного предусловия, слабейшего предусловия и сильнейшего постусловия. Мы будем действовать по аналогии с уже введенными понятиями для детерминированных программ и с понятиями, введенными Дейкстрой для недетерминированных программ и предикатов. Однако новые понятия не описываются столь простыми формулами алгебры бинарных отношений, как, например,

$$\text{wpr}(P, g) = p^b g, \quad (7)$$

где правая часть понимается как композиция (произведение) бинарных отношений. Таким образом, недетерминированность вносит существенные осложнения. Позднее, в § 3, будут разобраны различные другие подходы к определению понятия слабейшего предусловия, в частности будет рассмотрен и подход, основанный на обычных соображениях сопряженности, т. е. на формуле (7).

Определение 2. Пусть для бинарных отношений  $f, g$  и программы  $P$  имеет место  $\{f\}P\{g\}$ , тогда предусловие  $f$  называется *слабейшим свободным для  $g$  относительно  $P$*  (обозначение:  $f = \text{wlp}(P, g)$ ), если для всякого  $f$ , являющегося предусловием для  $g$  относительно  $P$ , имеет место  $\{f'\}\{f\}$ , т. е.  $f' \subseteq f$ .

Лемма 1. Для любых  $g, P$  существует  $\text{wlp}(P, g)$ .

Доказательство. Во-первых, имеет место  $\{\emptyset\}P\{g\}$ , во-вторых, если  $\{f_1\}P\{g\}$  и  $\{f_2\}P\{g\}$ , то  $\{f_1 \cup f_2\}P\{g\}$ ; можно взять объединение всех предусловий для  $g$ , и это объединение, очевидно, будет слабейшим свободным для  $g$  относительно  $P$ .

Свойство  $\{f\}P\{g\}$  эквивалентно свойству  $\{f\}\{\text{wlp}(P, g)\}$  или включению  $f \subseteq \text{wlp}(P, g)$ . Если  $P$  имеет вид (4), а  $g$  — вид (3), то  $\text{wlp}(P, g)$  определяется выражением (6) (это бинарное отношение мы обозначали через  $h$ ): нетрудно видеть, что если  $c_3(n) < c_2(n)$ , то значение  $n$  может потерять в результате выполнения программы все делители, равные 3; аналогично, если  $c_3(n) < c_4(n)$ , то  $n$  может потерять все пятерки. Итак,  $h = \text{wlp}(P, g)$ .

Примечание 1. Существование  $\text{wlp}(P, g)$  для любых  $P$  и  $g$  — это важное обстоятельство: из него следует, что для любого свойства циклической инструкции, или инструкции **do**, найдется

пвариант, подходящий для доказательства этого свойства, так как таким инвариантом может служить  $wlp(do, g)$ , где  $g$  — предположение рассматриваемого свойства. Разумеется, в некоторых случаях могут существовать и более простые инварианты.

**Теорема 1.** *Имеет место равенство  $wlp(P, g) = \overline{p^* \bar{g}}$ .*

(Надписанная черта означает дополнение к бинарному отношению.)

**Доказательство.** В самом деле,  $(\overline{p^* \bar{g}})(v) = t$  означает, что  $(p^* \bar{g})(v) \neq t$  и, по определению произведения двух бинарных отношений, что в результате выполнения  $P$  состояние  $v$  может перейти только в такое, для которого  $\bar{g}(v) \neq t$ , но это равносильно тому, что  $g(v) = t$ . Аналогично, если  $(\overline{p^* \bar{g}})(v) \neq t$ , то  $g(v) \neq t$ .

Кроме слабейшего свободного предусловия обычно рассматривается еще просто слабейшее предусловие, которое дополнительно гарантирует завершимость программы.

Подобно лемме 1, доказываемое существование слабейшего предусловия для  $g$  относительно  $P$  (обозначение:  $wp(P, g)$ ).

Дадим точную формулу для  $wp(P, g)$ .

**Теорема 2.** *Имеют место равенства  $wp(P, g) = \overline{p^b \bar{g}} \cap p^b g = \overline{p^b \bar{g}} \cap p^b u$ , где  $u = V \times M$  — полное бинарное отношение.*

**Доказательство.** Равенство  $(\overline{p^b \bar{g}})(v) = t$  означает, что если  $p^b(v) = v_1$ , то  $g(v_1) = t$ ;  $(p^b u)(v) = t$  при любом  $t$  означает, что для некоторого  $v_1$  выполнено  $p^b(v) = v_1$ . В свою очередь  $(p^b g)(v) = t$  означает, что для некоторого  $v_1$  выполнено  $p^b(v) = v_1$  и  $g(v_1) = t$ ; последнее равенство ничему не противоречит, так как выполнение его следует из  $(\overline{p^b \bar{g}})(v) = t$  и  $p^b(v) = v_1$ .

Из теоремы 2 можно чисто формально вывести, в частности, аналоги свойств 1—4 слабейшего предусловия, сформулированных в [4, с. 33—39]. Выведем, например, что  $wp(P, g_1 \cap g_2) = wp(P, g_1) \cap wp(P, g_2)$ . В самом деле,

$$\begin{aligned} \overline{p^b (g_1 \cap g_2)} \cap p^b u &= \overline{p^b (\bar{g}_1 \cup \bar{g}_2)} \cap p^b u = \\ &= \overline{p^b \bar{g}_1 \cup p^b \bar{g}_2} \cap p^b u = \overline{p^b \bar{g}_1} \cap \overline{p^b \bar{g}_2} \cap p^b u = \\ &= (\overline{p^b \bar{g}_1} \cap p^b u) \cap (\overline{p^b \bar{g}_2} \cap p^b u). \end{aligned}$$

Более того, это доказательство пройдет и для случая объединения  $\bigcup_{i \in I} g_i$ , где  $I$  — произвольное множество индексов. Для предикатов этому соответствует равенство

$$\text{wp}(P, \forall i g_i) = \forall i \text{wp}(P, g_i).$$

Главный момент доказательства — использование дистрибутивности умножения бинарных отношений относительно объединения. Равенство  $\text{wp}(P, g_1 \cup g_2) = \text{wp}(P, g_1) \cup \text{wp}(P, g_2)$  не имеет места (как не имеет места и его аналог для предикатов) — умножение бинарных отношений не дистрибутивно относительно пересечения; можно лишь гарантировать, исходя из законов алгебры бинарных отношений, выполнение включения

$$\bigcup_{i \in I} \text{wp}(P, g_i) \subseteq \text{wp}\left(P, \bigcup_{i \in I} g_i\right).$$

В случае, когда  $P$  и  $g$  определены с помощью (4), (3), имеем  $\text{wp}(P, g) = \text{wlp}(P, g) = h$ , где  $h$  определено с помощью (6): выполнение программы (4) завершается для любого натурального значения  $n$ , так как при ее выполнении убывает  $c_2(n)$ .

Формулы

$$\text{wlp}(P, g) = \overline{p^* g}, \quad (8)$$

$$\text{wp}(P, g) = \overline{p^b g} \cap p^b g \quad (9)$$

сохраняют силу и для предикатов, если последние рассматривать как бинарные отношения, содержащиеся в  $V \times L$ , где  $L$  — какое-нибудь одноэлементное множество, например  $\{I\}$ . Каждое  $g \subseteq V \times M$  представляется в виде объединения уровней:  $g = \bigcup_{m \in M} g_{(m)}$ , где  $g_{(m)} \subseteq V \times \{m\}$ .

Нетрудно проверить, что верна следующая теорема.

**Теорема 3.** *Справедливо  $\text{wp}(P, g) = \bigcup_{m \in M} \text{wp}(P, g_{(m)})$ , т. е.  $\text{wp}(P, g)_{(m)} = \text{wp}(P, g_{(m)})$ . Имеют место соответствующие равенства и для  $\text{wlp}$ .*

Исходя из (8), нетрудно проверить, что для любого множества индексов  $I$  и любого семейства  $g_i$ ,  $i \in I$ , бинарных отношений-подмножеств множества  $V \times M$  имеют место

$$\text{wlp}\left(P, \bigcap_{i \in I} g_i\right) = \bigcap_{i \in I} \text{wlp}(P, g_i),$$

$$\bigcup_{i \in I} \text{wlp}(P, g_i) \subseteq \text{wlp}\left(P, \bigcup_{i \in I} g_i\right).$$

Определение 3. Постусловие  $g$  для  $f$  относительно  $P$  называется *сильнейшим* (обозначение:  $g = \text{sp}(P, f)$ ), если для всякого  $g'$ , являющегося постусловием для  $f$  относительно  $P$ , имеет место  $\{g\}\{g'\}$  или  $g \subseteq g'$ .

Аналогично предыдущему доказывается следующая теорема.

Теорема 4. Для любых  $P, f$  существует  $\text{sp}(P, f)$ ; имеют место равенства  $\text{sp}(P, f) = (p^\#)^{-1} f$ ,  $\text{sp}(P, \bigcup_{m \in M} f_{(m)}) = \bigcup_{m \in M} \text{sp}(P, f_{(m)})$ ,  $\text{sp}(P, f_{(m)}) = \text{sp}(P, f)_{(m)}$ .

Выражение  $(p^\#)^{-1} f$  для  $\text{sp}(P, f)$  дает возможность вывести все свойства сильнейшего постусловия. Сразу, например, видно, что для любого множества индексов  $I$  и любого семейства  $f_i, i \in I$ , бинарных отношений-подмножеств множества  $V \times M$  имеет место

$$\text{sp}\left(P, \bigcup_{i \in I} f_i\right) = \bigcup_{i \in I} \text{sp}(P, f_i),$$

$$\text{sp}\left(P, \bigcap_{i \in I} f_i\right) \subseteq \bigcap_{i \in I} \text{sp}(P, f_i).$$

Свойство  $\{f\}P\{g\}$  эквивалентно  $\{\text{sp}(P, f)\}\{g\}$  или  $\text{sp}(P, f) \subseteq g$ , или  $(p^\#)^{-1} f \subseteq g$ , или  $\forall m \text{ sp}(P, f_{(m)}) \subseteq g_{(m)}$ . Следует отметить, что  $\text{sp}(P, f)$  оказывается сложно устроенным даже для элементарных детерминированных инструкций, например для инструкции присваивания (приведенная выше «простая» формула

$$(p^\#)^{-1} f \tag{10}$$

использует операцию обращения). Однако преобразования бинарного отношения  $f$  по формуле (10) обладает одним важным свойством — непрерывностью; об этом свойстве речь пойдет в следующем параграфе.

Примечание 2. Возможно еще по крайней мере одно толкование свойства  $\{f\}P\{g\}$ : если для некоторых  $v_0 \in V, m \in M$  имеет место  $f(v_0) = m$ , то существует  $v_1 \in V$  такое, что  $p^\#(v_0) = v_1$ ,  $g(v_1) = m$ . При таком толковании, как нетрудно видеть, слабейшее предусловие для  $g$  относительно  $P$  равно  $p^\flat g$ , а слабейшее свободное предусловие —  $p^\# g$ . Определить понятие сильнейшего постусловия оказывается невозможным. Пусть программа  $P$  определена так:

$$\text{if } v = 0 \rightarrow v := 1 \square v = 0 \rightarrow v := 2 \text{ fi.}$$

Тогда  $v = 1$  и  $v = 2$  являются постусловиями для предусловия  $v = 0$ , но нельзя подобрать третье постусловие, из которого бы следовало каждое из двух указанных.

## § 2. Непрерывные функционалы; инструкция ввода

Установление включений в множестве частичных функций или бинарных отношений характерно для решения задач анализа программ. Средство доказательства включений дает, например, теория неподвижной точки программ Скотта — Манны. Хотя мы не собираемся в оставшихся параграфах заниматься применением этой теории, все-таки, ввиду ее достаточной распространенности, остановимся кратко на некоторых ее вопросах, имеющих отношение к рассматриваемым в этой главе задачам. Применение предельного перехода по  $h$ , шаговой или полной вычислительной индукции заведомо корректно, когда доказываемое утверждение эквивалентно конечной конъюнкции

$$\bigwedge_{i=1}^n (\Phi_i[h] \subseteq \Psi_i[h]),$$

где  $\Phi_i, \Psi_i$  ( $i = 1, \dots, n$ ) — непрерывные функционалы. Включения, связанные с задачами анализа программ, часто содержат  $\text{wr}(P, g)$ ,  $\text{wlp}(P, g)$ ,  $\text{sp}(P, f)$ . Последние, очевидно, можно рассматривать при фиксированной программе  $P$  как функционалы по второму аргументу, а если обратиться к формулам

$$\begin{aligned} \text{wr}(P, g) &= \overline{p^b g} \cap p^b g, \text{wlp}(P, g) = \\ &= \overline{p^* g}, \text{sp}(P, f) = (p^*)^{-1} f, \end{aligned}$$

то, при фиксированном втором аргументе, как функционалы по  $p^b$  или  $p^*$ . В § 3 гл. I рассматривался детерминированный случай и исследовалась возможность доказательства включений, связанных с циклическими программами индукцией по  $p$  (детерминированность  $P$  обеспечивает выполнение равенства  $p = p^b = p^*$  и непрерывность перечисленных функционалов по  $p$ ). Для  $\text{wr}(P, g)$ , например, была введена последовательность приближений  $\text{wr}(P, g)_i$  ( $i = 0, 1, \dots$ ). Фактически, при переходе от приближения  $\text{wr}(P, g)_i$  к  $\text{wr}(P, g)_{i+1}$  использовался переход от  $p_i$  к  $p_{i+1}$ , где  $p_i$  ( $j = 0, 1, \dots$ ) — приближение  $p$  такое, что функция  $p_j$  определена и равна  $p$ , если выполнение циклической программы  $P$  для данного начального состояния завершается не позднее чем через  $j$  этапов, и функция  $p_j$  не определена в противном случае. Формулы типа (2) § 3 гл. I определяют изменение  $\text{wr}$  при перехо-

де от  $p_i$  к  $p_{i+1}$ . Подобные формулы могут быть даны и для недетерминированного случая, далее, в § 4, при рассмотрении разных вариантов предусловий они будут написаны. Здесь же мы исследуем общие вопросы, связанные с непрерывностью  $wр$ ,  $wlp$ ,  $sp$  в недетерминированном случае, и более детально обсудим возможность применения индукции по постусловию  $g$  и предусловию  $f$ .

Легко проверяется, что когда для любого  $v$  многозначная функция  $p^b$  либо не определена, либо принимает лишь конечное число значений (в терминологии Дейкстры — когда недетерминированность ограничена), функционал  $wр(P, g)$  будет непрерывным по  $g$ . Недетерминированность, вносимая инструкциями **if** и **do** языка Дейкстры, является ограниченной, и при любой фиксированной программе  $P$ , написанной на этом языке, функционал  $wр(P, g)$  непрерывен по  $g^*$ ). В случае ограниченной недетерминированности будет непрерывным по  $g$  и  $wlp(P, g)$ , так как  $wlp(P, g) = wр(P, g) \cup wlp(P, \emptyset)$ , а  $wlp(P, \emptyset)$  не зависит от  $g$ .

**Теорема.** Пусть  $P$  — некоторая программа и функционал  $wр(P, g)$  непрерывен по  $g$ . Тогда недетерминированность  $P$  ограничена.

**Доказательство.** Предположим противное. Пусть для некоторого состояния  $v_0$  имеется бесконечное множество значений  $p^b(v_0)$ . Выделим из этого множества счетное подмножество  $v_1, v_2, \dots$ . Выберем из множества  $M$ , в котором может принимать значения постусловие  $g$ , некоторый элемент  $m$ . Построим последовательность функций  $g_0(v) \subseteq g_1(v) \subseteq \dots$  следующим образом: для  $i = 0, 1, \dots$  функция  $g_i(v)$  не определена для  $v = v_{i+1}, v_{i+2}, \dots$ , а для всех остальных значений аргумента принимает значение, равное  $m$ . Тогда  $g = \bigcup_{i=0}^{\infty} g_i$  — это функция, тождественно равная  $m$ , и, следовательно,  $wр(P, g)(v_0) = m$ . В это же время для всех  $i$  значение  $wр(P, g_i)(v_0)$  не определено, и, следовательно, не определено значение  $\left( \bigcup_{i=0}^{\infty} wр(P, g_i) \right)(v_0)$ . Противоречие.

Теорема остается в силе, если в ее условии  $wр$  заменить на  $wlp$ .

---

\*) Этот факт зафиксирован в [4].



Непрерывность  $\text{wr}(P, g)$  и  $\text{wlp}(P, g)$  по  $p^b$  или  $p^*$  может не иметь места даже при ограниченной недетерминированности. Более того, эти функционалы могут быть даже не монотонными по  $p^b, p^*$ . Пусть, например,  $P_1, P_2$  суть следующие программы (множество состояний  $V$  — это множество неотрицательных целых чисел):

$$\text{if } v = 0 \rightarrow v := 1 \text{ fi,}$$

$$\text{if } v = 0 \rightarrow v := 1 \square v = 0 \rightarrow v := 2 \text{ fi,}$$

а  $g(v)$  есть предикат  $v = 1$ ; тогда  $p_1^b \subseteq p_2^b$ ,  $\text{wr}(P_1, g) = (v = 0)$ , а  $\text{wr}(P_2, g)$  нигде не определено (или, при традиционном взгляде на предикаты, тождественно равно  $\perp$ ).

Что касается  $\text{sp}(P, f)$ , то этот функционал, описываемый формулой  $(p^*)^{-1}f$ , непрерывен и по  $p^*$ , и по  $f$ , так как  $(p^*)^{-1}$  — непрерывный функционал. Свойство  $\{f\}P\{g\}$  эквивалентно включению  $(p^*)^{-1}f \subseteq g$ , и, следовательно, при доказательстве этого свойства может применяться индукция по  $p^*$  и по  $f$ .

**Примечание 1.** Если принять толкование свойства  $\{f\}P\{g\}$ , предложенное в примечании 1 предыдущего параграфа, то слабейшее предусловие и слабейшее свободное предусловие будут функционалами, непрерывными по своим аргументам.

Использование индукции по предусловию и постусловию, рассматриваемым как неподвижные точки некоторых непрерывных функционалов, совершенно естественно в случаях, когда множество  $M$ , в котором принимают значения предусловия и постусловия, является бесконечным множеством, и, в частности, таким бесконечным множеством, которое соответствует рекурсивному типу данных. Такой тип данных сам является неподвижной точкой некоторого непрерывного функционала, и при рассмотрении чисто синтаксических языковых аспектов это использовалось еще до возникновения денотационной семантики, хотя терминология была несколько иной: (см. [3]).

Возьмем, например, множество неотрицательных целых чисел — множество слов  $0, 01, 011, 0111, \dots$  в алфавите, содержащем знаки  $0, 1$ . Можно определить это понятие формулой Бэкуса:

$$\langle \text{неотрицательное целое} \rangle ::= 0 \mid \langle \text{неотрицательное целое} \rangle 1.$$

Множество всех неотрицательных целых есть наименьшая неподвижная точка функционала, результатом применения которого к произвольному множеству слов  $B$  будет  $\{0\} \cup B1$ , элементы  $B1$  получаются из элементов  $B$  приписыванием справа знака 1.

Если множество  $M$  есть наименьшая неподвижная точка некоторого непрерывного функционала, то возникает возможность применения предельного перехода и различных принципов индукции для установления включений вида

$$\{l | l \in M, \varphi(v, l)\} \subseteq \{m | m \in M, \psi(v, m)\}, \quad (1)$$

которые часто служат эквивалентами свойств программы. Если включение (1) описано с привлечением синтаксических определений типа данных, то шаговая индукция и полная вычислительная индукция будут представлять собой индукцию по структуре элемента множества («структурную индукцию»)\*.

Удобнее, видимо, иметь дело с явным рекурсивным описанием многозначной функции, чем с описанием  $\{l | l \in M, \varphi(v, l)\}$ : имея явное описание, легче указать естественную форму функционала, для которого эта функция будет неподвижной точкой. Перейти к явному описанию можно, например, следующим образом. Используя формулы Бэкуса, определяющие  $M$ , описать  $V \times M$  в виде явной многозначной функции; например, для типа «неотрицательное целое» это описание выглядит так:

$$M(v) = \text{if } I \rightarrow 0 \square I \rightarrow M(v) 1 \text{ fi},$$

или, если определена операция прибавления 1,

$$M(v) = \text{if } I \rightarrow 0 \square I \rightarrow M(v) + 1 \text{ fi}.$$

Определить  $\varphi'(v, l)$  следующим образом:

$$\varphi'(v, l) = \text{if } \varphi(v, l) \rightarrow l \text{ fi}.$$

После этого написать искомую явную многозначную функцию

$$f(v) = \varphi'(v, M(v)).$$

Помимо инструкций Дейкстры **if** и **do**, недетерминированность возникает при использовании инструкции

---

\*) Если при определении синтаксиса типа данных приходится, используя рекурсию, вводить некоторые дополнительные синтаксические понятия, то при доказательстве включений должны рассматриваться системы непрерывных функционалов нескольких аргументов.

ввода, и эта недетерминированность уже не является ограниченной. Инструкция ввода тесно связана с типами данных. Обсудим детали, относящиеся к инструкции  $input(a)$  ввода значения переменной  $a$ .

Будем пока рассматривать только такие предусловия и постусловия относительно  $P$ , которые являются предикатами. Непосредственно проверяется, что

$$\begin{aligned}wp(input(a), \psi) &= wlp(input(a), \psi) = \forall a \psi, \\sp(input(a), \varphi) &= \exists a \varphi;\end{aligned}$$

как следствие этих равенств получаются формулы

$$\{\forall a \psi\} input(a) \{\psi\}, \quad \{\varphi\} input(a) \{\exists a \varphi\}.$$

Переменная  $a$  считается принимающей значения в множестве  $A$ , которое соответствует типу этой переменной; именно с этой точки зрения должны рассматриваться кванторы  $\forall a$ ,  $\exists a$ .

Примечание 2. Функционал  $\forall a \varphi$  двух аргументов  $A$  и  $\varphi$  не является непрерывным ни по  $A$ , ни по  $\varphi$ , что согласуется с теоремой 1. В то же время функционал  $\exists a \varphi$  непрерывен по  $A$  и по  $\varphi$ .

Считая, что  $V = A \times V'$  и  $v = (a, v')$ , приведенные формулы можно записать более подробно:

$$\{\forall a \psi(a, v')\} input(a) \{\psi(a, v')\}, \quad (2)$$

$$\{\varphi(a, v')\} input(a) \{\exists a \varphi(a, v')\} \quad (3)$$

для любых  $\varphi(a, v')$ ,  $\psi(a, v')$ .

Пусть для некоторых конкретных  $\varphi(a, v')$  и  $\psi(a, v')$  имеет место свойство

$$\{\varphi(a, v')\} input(a) \{\psi(a, v')\}. \quad (4)$$

Это свойство, в силу (2), эквивалентно

$$\varphi(a, v') \supset \forall b \psi(b, v'), \quad (5)$$

а в силу (3) —

$$(\exists c \varphi(c, v')) \supset \psi(a, v'); \quad (6)$$

далее (5) переписывается в виде

$$\forall b (\varphi(a, v') \supset \psi(b, v')), \quad (7)$$

а (6) — в виде

$$\forall c (\varphi(c, v') \supset \psi(a, v')). \quad (8)$$

Вспоминая, что все переменные, которые в формулах такого рода выглядят как свободные переменные, на самом деле неявно предполагаются связанными квантором всеобщности, получаем, что (7) и (8) допускают общую форму записи

$$\forall v' \forall b \forall c (\varphi(c, v') \supset \psi(b, v')). \quad (9)$$

Таким образом, (4) эквивалентно (9). Заметим, что в формуле (9) переменные  $c$  и  $b$  принимают значения в множестве  $A$ , а  $v'$  — в множестве  $V'$ ; именно с этой точки зрения должны рассматриваться кванторы всеобщности.

Пример. Свойство

$$\{I\} \text{ input}(n); m := 4^n - 1 \{3 \mid m\} \quad (10)$$

эквивалентно свойству  $\{I\} \text{ input}(n) \{3 \mid (4^n - 1)\}$ , последнее же свойство, в силу (9), эквивалентно

$$\forall b 3 \mid (4^b - 1). \quad (11)$$

Если рассматриваемый тип данных — это неотрицательные целые числа и этот тип определен как наименьшая неподвижная точка описанного выше функционала, то доказательство свойства (10) полной вычислительной индукцией сводится к доказательству обычной математической индукцией утверждения (11).

Разобранный метод работы с инструкциями ввода без труда обобщается на случай, когда в качестве предусловий и постусловий используются многозначные функции со значениями в некотором множестве  $M$ . Для этого обобщения, как обычно, квантор всеобщности заменяется в формулах для предусловий и постусловия знаком пересечения  $\cap$ , а квантор существования — знаком объединения  $\cup$ . При этом, например, функцию  $\bigcup_{a \in A} g(a, v')$  надо понимать следующим образом: эта функция зависит только от значения  $v'$ ; множество значений этой функции для значения аргумента  $v'_0$  получается как пересечение всех множеств значений  $g(a, v'_0)$ , когда  $a$  пробегает  $A$ . Простой пример: для того, чтобы убедиться в свойстве  $\{x + y\} \text{ input}(x) \{[y, \infty)\}$ , где  $x, y$  имеют неотрицательный целый тип, мы строим функцию  $\bigcup_x [y, \infty) = [y, \infty)$  и удостоверяемся, что при всех неотрицательных целых  $x$  значение  $x + y$  принадлежит множеству  $[y, \infty)$ .

**Примечание 3.** Употребление многозначных функций в качестве предусловий и постусловий позволяет, например, записать такое свойство индукцией ввода: для любого предиката  $\varphi(y)$  имеет место свойство  $\{\varphi(y)\}input(x)\{\varphi(y)\}$ . Это свойство запишем так:  $\{f\}input(x)\{f\}$ , где значениями  $f$  являются предикаты на множестве  $y$ , и  $f(x, y) = \varphi$  тогда и только тогда, когда истинно  $\varphi(y)$ .

Привлечение семантики инструкции ввода совершенно необходимо при исследовании программ с многократным вводом значений некоторой переменной, например при исследовании циклической программы, каждый этап которой включает в себя выполнение инструкции *input*.

### § 3. Погрешность округления в вычислениях, описываемых программой

Этот параграф служит продолжением § 6 гл. I.

Обычно некоторые арифметические операции выполняются приближенно — возникает так называемая *погрешность округления* или *вычислительная погрешность*. Для того чтобы исследовать эту погрешность, мы примем, как это часто делается в подобных исследованиях, что вместо значения  $a * b$  где  $*$  — знак операции из набора  $+$ ,  $-$ ,  $\cdot$ ,  $/$ , получается значение  $(a * b)(1 + \varepsilon)$  и при этом для действительного числа  $\varepsilon$  выполнено  $|\varepsilon| < \delta$ , а величина  $\delta$  определяется разрядностью машинной ячейки (точнее — разрядностью операндов) и способом округления.

Мы наложим следующие ограничения на программы. Во-первых, каждая инструкция присваивания должна содержать в правой части выражение, в которое либо не входят знаки операций, либо входит только один знак, который должен принадлежать набору  $+$ ,  $-$ ,  $\cdot$ ,  $/$ . Во-вторых, в условных и циклических инструкциях в условия не входят знаки арифметических операций. Благодаря этим ограничениям изменяется семантика (правила вычисления  $wr$ ,  $wlr$  или правила Хоора) лишь инструкции присваивания  $x := a * b$  в этом отношении нуждается в специальном рассмотрении. Эта инструкция оказывается фактически недетерминированной. Ее «приближенное» выполнение эквивалентно «точному» выполнению составной инструкции  $input(\varepsilon); x := (a * b)(1 + \varepsilon)$  при ограничении  $|\varepsilon| < \delta$ . Семантика инструкции ввода приводилась в предыдущем

параграфе:

$$\begin{aligned} \text{wp}(\text{input}(x), \chi(a, b, \dots, x)) &= \text{wlp}(\text{input}(x), \\ \chi(a, b, \dots, x)) &= \forall x \chi(a, b, \dots, x). \end{aligned}$$

Исходя из этой семантики инструкции ввода определенной семантику  $x := a * b$ :

$$\{\forall \epsilon \psi(a, b, \dots, (a * b)(1 + \epsilon))\} x := a * b \{\psi(a, b, \dots, x)\} \quad (1)$$

или, аналогично,

$$\begin{aligned} \text{wp}(x := a * b, \psi(a, b, \dots, x)) &= \\ &= \text{wlp}(x := a * b, \psi(a, b, \dots, x)) = \\ &= \forall \epsilon \psi(a, b, \dots, (a * b)(1 + \epsilon)); \quad (2) \end{aligned}$$

отметим, что если  $*$  — это знак операции деления, то (1), (2) требуют некоторых вполне очевидных уточнений, касающихся случая  $b = 0$ .

Мы можем привести теперь примеры исследования вычислительной погрешности, накапливающейся по ходу выполнения программ. Будем без оговорок считать, что число  $\delta$ , ограничивающее сверху  $|\epsilon|$ , само не превосходит 1 (это условие, очевидно, выполняется при вычислениях на ЭВМ); подчеркнем, что связывание переменной квантором всеобщности в (1), (2) надо понимать в следующем смысле: для каждого  $\epsilon$  такого, что  $|\epsilon| < \delta \dots$

Пусть программа  $P$  имеет вид

$$\text{while } n > 0 \text{ do } s := s \cdot a; n := n - 1 \text{ od.}$$

Будем считать, что вычитание единицы из целого числа производится точно. Докажем, что если начальное значение  $s$  задано с не превосходящей  $\delta$  абсолютной погрешностью, приближенное и точное значения  $s$  по модулю не превосходят 1, значение  $a$  задано точно и при этом  $|a| < 0.5$ , то абсолютная погрешность заключительного значения  $s$  не превзойдет  $\delta$ .

Итак, мы выбираем следующие  $\varphi$  и  $\psi$ :

$$\begin{aligned} \varphi(a, n, s', s'') &= \\ &= (|s' - s''| < \delta) \wedge (|s_1|, |s_2| < 1) \wedge (|a| < 0.5), \\ \psi(a, n, s', s'') &= (|s' - s''| < \delta) \end{aligned}$$

(переменные  $a, n$  удовлетворяют всем выведенным в § 5 гл. I условиям, достаточным для нераздвоенного вхожде-

ния переменных в предикаты при точном выполнении операций, но вычисления, результаты которых присваиваются переменным  $a$ ,  $n$ , производятся точно).

Для доказательства рассмотрим предикат  $wr(P, \psi)$ , который вычисляется с помощью выписанных в § 5 гл. I рекуррентных соотношений (6), (7). Для проверки импликации  $\varphi \supset wr(P, \psi)$  достаточно доказать, что  $\varphi \supset K[\varphi]$ , где  $K$  — функционал, фигурирующий во втором рекуррентном соотношении;  $K[\varphi]$  имеет вид

$$(n > 0) \wedge \forall \varepsilon (|s'a - s''a(1 + \varepsilon)| < \delta) \wedge (|sz| < 1) \wedge \\ \wedge (\forall \varepsilon |s''a(1 + \varepsilon)| < 1) \wedge (|a| \leq 0.5) \vee (n \leq 0) \wedge \\ \wedge (|s' - s''| < \varepsilon).$$

Для установления  $\varphi \supset K[\varphi]$  можно рассмотреть два случая:  $n > 0$  и  $n \leq 0$ . Пусть выполнено  $\varphi$  и  $n > 0$ , тогда

$$|s'a - s''a(1 + \varepsilon)| \leq |a| \cdot |s' - s''| + |a| \cdot |s''\varepsilon| < \delta.$$

Пусть выполнено  $\varphi$  и  $n \leq 0$ , тогда очевидно, что  $|s' - s''| < \delta$ .

Основываясь на выписанных в § 5 гл. I формулах (9), (10), для исследования вычислительной погрешности можно применять и хооровскую теорию, но при этом программы  $P'$  и  $P''$  должны пониматься так, что  $P'$  имеет дело с точными начальными данными и выполняется точно, в то время как  $P''$  имеет дело с неточными начальными данными и выполняется приближенно. Свойства

$$\{\varphi(a, b, \dots, x, \bar{a}, \bar{b}, \dots, \bar{x})\}P\{\xi(a, b, \dots, x, \bar{a}, \bar{b}, \dots, \bar{x})\}, \\ (3) \\ \{\xi(\bar{a}, \bar{b}, \dots, \bar{x}, a, b, \dots, x)\}P\{\psi(\bar{a}, \bar{b}, \dots, \bar{x}, a, b, \dots, x)\}$$

отнесены к одной и той же программе  $P$ , но здесь мы должны прибегать к разным семантикам программы  $P$ : в первом свойстве программа  $P$  понимается в смысле точного выполнения, во втором — приближенного.

Пусть программа  $P$  — это

$$s := 1; i := 0; \text{ while } i \neq n \text{ do } s := s \cdot a; i := i + 1 \text{ od}.$$

Докажем, что если  $a$  задано точно,  $|a| < 0.5$ , умножение производится приближенно, а прибавление единицы — точно, то абсолютная погрешность  $s$  будет меньше  $\delta$ . Для простоты будем считать, что при приближенном умножении выполняется  $a \cdot 1 = 1 \cdot a = a$ .

Таким образом, мы выбираем следующие предусловие и постусловие:

$$\varphi(i, n, a, s, \bar{s}) = (|a| < 0.5),$$

$$\psi(i, n, a, s, \bar{s}) = (|s - \bar{s}| < \delta).$$

Для описания предиката  $\xi$ , участвующего в свойствах (3), предварительно определим величину  $a \uparrow m$ , где  $b$  — любое число, а  $m$  — неотрицательное целое, как результат приближенного вычисления произведения

$$\underbrace{a \cdot \dots \cdot a}_m \quad (4)$$

( $b \uparrow 0$  считаем равным 1). В качестве  $\xi(i, n, a, s, \bar{s})$  выберем предикат  $|a \uparrow n - s| < \delta$ .

Мы будем далее пользоваться тем, что если  $|a| < 0.5$  и  $m \geq 1$ , то  $|a \uparrow m| < 0.5$  при любом допустимом выборе величин  $\varepsilon$  в ходе вычисления произведения (4): для  $m = 1$  это очевидно, далее  $a \uparrow m = (a \uparrow (m-1))a(1 + \varepsilon)$  и  $a(1 + \varepsilon) < 1$ .

Для доказательства свойства

$$\{\varphi(i, n, a, s, \bar{s})\}P\{\xi(i, n, a, s, \bar{s})\}$$

(имеется в виду точное выполнение программы) переходим к свойству

$$\{(|a| < 0.5) \wedge (s = 1) \wedge (i = 0)\} \text{ while } i \neq n \text{ do} \\ s := s \cdot a; i := i + 1 \text{ od } \{|a \uparrow n - s| < \delta\}$$

и используем инвариант цикла, равный

$$(|a \uparrow i - s| < \delta) \wedge (|a| < 0.5) \wedge (i \geq 1),$$

тогда главной частью доказательства выписанного свойства циклической программы является установление истинности импликации

$$((i \neq n) \wedge (|a \uparrow i - s| < \delta) \wedge (|a| < 0.5) \wedge (i \geq 1)) \supset \\ \supset ((|a \uparrow (i+1) - sa| < \delta) \wedge (|a| < 0.5) \wedge (i+1 \geq 1)).$$

Но  $|a \uparrow (i+1) - sa| = |(a \uparrow i)a(1 + \varepsilon) - sa| \leq a|a \uparrow i - s| + |a \uparrow i| \cdot |a| \cdot \varepsilon \leq a\delta + 0.5(a\delta) < \delta$ .

Для доказательства свойства

$$\{\xi(i, n, a, \bar{s}, s)\}P\{\psi(i, n, a, \bar{s}, s)\}$$

(имеется в виду приближенное выполнение программы)



докажем свойство

$$\{(|a \uparrow n - \bar{s}| < \delta) \wedge (s = 1) \wedge (i = 0)\} \text{ while } i \neq n \text{ do} \\ s := s \cdot a; i := i + 1 \text{ od } \{|\bar{s} - s| < \delta\};$$

здесь в качестве инварианта подходит  $(|a \uparrow n - \bar{s}| < \delta) \wedge \wedge (s = a \uparrow i)$ .

**Примечание.** Величина  $b \uparrow m$  определяется по  $b$  и  $m$  неоднозначно. Несколько усложнив формулы, можно было бы обойтись в записи предикатов без этой многозначной функции. Например, последний инвариант  $(|a \uparrow n - \bar{s}| < \delta) \wedge (s = a \uparrow i)$  допускает следующую запись:  $(\forall \dots \forall \varepsilon_1 \dots \varepsilon_{n-1} | a^n(1+\varepsilon_1) \dots (1+\varepsilon_{n-1}) - \bar{s}| < \delta) \wedge (\exists \dots \exists \varepsilon'_1 \dots \varepsilon'_{n-1} s = a^i(1+\varepsilon'_1) \dots (1+\varepsilon'_{n-1}))$ .

#### § 4. Вычисление предусловий относительно if и do

Для слабейшего предусловия  $\text{wp}(\text{if}, \psi)$ , где  $\text{if}$  — это

$$\text{if } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{ fi}, \quad (1)$$

Дейкстра дает определение

$$\text{wp}(\text{if}, \psi) =$$

$$= \beta\beta \wedge (\beta_1 \supset \text{wp}(P_1, \psi)) \wedge \dots \wedge (\beta_n \supset \text{wp}(P_n, \psi)), \quad (2)$$

где  $\beta\beta = \beta_1 \vee \dots \vee \beta_n$ . Напомним, что инструкция (1) может быть применена к состоянию  $v$  (т. е. «завершается» для начального состояния  $v$ ) лишь тогда, когда для  $v$  выполнено хотя бы одно из  $\beta_1, \dots, \beta_n$ , т. е. выполнено  $\beta\beta$ .

Для слабейшего предусловия  $\text{wp}(\text{do}, \psi)$ , где  $\text{do}$  — это

$$\text{do } \beta_1 \rightarrow P_1 \square \dots \square \beta_n \rightarrow P_n \text{ od}, \quad (3)$$

дается определение, которое уже было рассмотрено нами применительно к детерминированной циклической инструкции. Определяется  $\text{wp}(\text{do}, \psi)_t$ ,  $t = 0, 1, \dots$ , следующим образом:

$$\text{wp}(\text{do}, \psi)_0 = \psi \wedge \neg \beta\beta; \\ \text{wp}(\text{do}, \psi)_t = \text{wp}(\text{if}, \text{wp}(\text{do}, \psi)_{t-1}) \vee \text{wp}(\text{do}, \psi)_0. \quad (4)$$

Далее описывается переход от  $\text{wp}(\text{do}, \psi)_t$ ,  $t = 0, 1, \dots$ , к  $\text{wp}(\text{do}, \psi)$ :

$$\text{wp}(\text{do}, \psi) = \exists t \text{ wp}(\text{do}, \psi)_t. \quad (5)$$

Прежде чем рассмотреть обобщение понятия слабейшего предусловия относительно инструкций (1), (3), от-

метим, что определяющее равенство (5) требует обоснования: непонятно, почему если для некоторого начального состояния  $v$  выполнение инструкции **do** обязательно завершается и дает заключительное состояние, которое удовлетворяет  $\psi$ , то для данного начального состояния  $v$  оказывается ограниченным сверху число этапов выполнения инструкции **do** (если точно следовать терминологии Дейкстры — число «выборки охраняемых команд»)? На этот счет в книге Дейкстры нет объяснений; вместе с этим интуитивно-операционные соображения о том, как выполняется инструкция **do**, используются в его книге (особенно при выводе программ) наравне с формальным определением семантики через слабое предусловие. Поэтому нужно было бы установить соответствие между теми и другими семантическими представлениями, но, однако, этого не сделано.

Наряду с тем что факт ограниченности числа этапов признается как бы не заслуживающим обсуждения, совершенно аналогичный и родственный ему факт, состоящий в том, что если для фиксированного начального состояния выполнение инструкции **do** обязательно завершается, то множество достижимых заключительных состояний будет конечным, разбирается в книге Дейкстры очень подробно — ему посвящена вся девятая глава «Когда недетерминированность ограничена».

На ограниченности (в случае завершенности) числа этапов выполнения **do** основывается и техника доказательства завершенности, заключающаяся в подборе целочисленной неотрицательной функции, определенной на множестве состояний и заведомо убывающей при выполнении одного этапа — ясно, что если такая функция найдена, то число этапов для данного начального состояния не превзойдет значения найденной функции, принимаемого на данном состоянии. Поэтому факт ограниченности числа этапов является очень важным, и мы его докажем.

Рассмотрим некоторую программу  $P$ , записанную на языке Дейкстры. Пронумеруем как-нибудь все стрелки, участвующие в записи  $P$ . Программа, содержащая стрелки, вообще говоря, может допускать разные пути выполнения. Будем говорить, что путь выполнения программы проходит через стрелку с номером  $m$ , если в некоторый момент выполняется инструкция, непосредственно следующая в тексте программы за стрелкой с номером  $m$ . Каждому пути выполнения  $P$  с начальным состоянием  $v$  можно сопоставить последовательность номеров стрелок, через

которые шаг за шагом проходит этот путь. Такая последовательность полностью характеризует путь выполнения программы.

**Теорема 1.** Пусть выполнение программы  $P$  для начального состояния  $v_0$  обязательно завершается. Тогда множество всех последовательностей номеров стрелок, характеризующих пути выполнения программы  $P$  для начального состояния  $v_0$ , конечно.

**Доказательство.** (Аналогично доказательству теоремы о бесконечных цепях в бесконечном, но локально конечном связном графе [12].)

Предположим противное. Пусть имеется бесконечное множество  $N$  неравных между собой последовательностей номеров стрелок

$$a_{11}, a_{12}, \dots, a_{1t_1},$$

$$a_{21}, a_{22}, \dots, a_{2t_2},$$

$$\dots \dots \dots$$

характеризующих пути выполнения недетерминированной программы  $P$  с начальным состоянием  $v_0$ . Очевидно, что ни одна из последовательностей множества  $N$  не является начальным отрезком другой. Поскольку в программе участвует лишь конечное число стрелок, то и возможностей для значения любого  $a_{pq}$ ,  $p = 1, 2, \dots$ ,  $q = 1, 2, \dots, t_q$ , имеется лишь конечное число. Поэтому среди первых элементов последовательностей множества  $N$  имеется бесконечное число равных некоторому числу  $b_i$ , и из  $N$  мы можем выбрать бесконечное подмножество последовательностей  $N_1$ , начинающихся с  $b_i$ . При этом все последовательности из  $N_1$  содержат более одного элемента, так как  $N$  состоит из неравных последовательностей, ни одна из которых не является начальным отрезком другой.

Рассмотрим вторые элементы последовательностей из  $N_1$ . Мы получим бесконечное множество  $N_2$  последовательностей, у которых первый элемент равен  $b_i$ , а второй — некоторому  $b_2$  и т. д. Так мы получим бесконечную последовательность  $b_1, b_2, \dots$  номеров стрелок. Эта бесконечная последовательность характеризует путь выполнения рассматриваемой программы  $P$  с начальным состоянием  $v_0$ : в самом деле, возьмем любой начальный отрезок  $b_1, b_2, \dots, b_i$  этой последовательности — он совпадает с начальным отрезком каждой из последовательностей, принадлежащих  $N_1$ , и, следовательно, характеризует начальные этапы некоторого пути выполнения программы  $P$ .

Получается, что программа  $P$  допускает такое выполнение с начальным состоянием  $v_0$ , которое не завершается. Противоречие.

Из теоремы 1 следует, что при фиксированном начальном состоянии, для которого выполнение обязательно завершается, число этапов выполнения **do** ограничено, а множество заключительных состояний конечно.

Теперь укажем некоторые способы построения аналогов слабейшего предусловия для постусловий, являющихся частичными функциями на  $V$  со значениями в некотором множестве  $M$ . Для некоторых из предлагаемых вариантов теорема 1 будет иметь важное значение; для того чтобы можно было, по-прежнему, определив слабейшее предусловие относительно инструкции **if**, автоматически получать определение слабейшего предусловия относительно инструкции **do**, в некоторых случаях придется опираться на эту теорему.

Будем по-прежнему рассматривать предикаты как частичные функции на  $V$ , принимающие значения в одноэлементном множестве  $M$ . Исключим из определений (2), (4), (5) логические операции; для (2) получаем

$$\text{if } \beta_1 \rightarrow \text{wp}(P_1, \psi) \square \dots \square \beta_n \rightarrow \text{wp}(P_n, \psi) \text{ fi.} \quad (6)$$

Последнюю запись следует считать условным выражением, значение которого не определено для состояния  $v$ , если не определено некоторое  $\text{wp}(P_i, \psi)$  и выполнено  $\beta_i$ ,  $1 \leq i \leq n$ .

Определение (4) относительно  $P$ , заданной с помощью (3), переписывается так:

$$\text{wp}(P, \psi)_0 = \text{if } \bigvee \beta \beta \rightarrow \psi \text{ fi,}$$

$$\begin{aligned} \text{wp}(P, \psi)_i &= \text{if } \beta_1 \rightarrow \text{wp}(P_1, \text{wp}(P, \psi)_{i-1}) \square \dots \\ &\dots \square \beta_n \rightarrow \text{wp}(P_n, \text{wp}(P, \psi)_{i-1}) \square \bigvee \beta \beta \rightarrow \psi \text{ fi.} \end{aligned} \quad (7)$$

Имеют место включения

$$\text{wp}(P, \psi)_0 \subseteq \text{wp}(P, \psi)_1 \subseteq \dots \quad (8)$$

Для определения  $\text{wp}(P, \psi)$  квантор существования заменяется знаком объединения:

$$\text{wp}(P, \psi) = \bigcup_{i=0}^{\infty} \text{wp}(P, \psi)_i.$$

Как и в детерминированном случае,  $\text{wp}(P, \psi)$  можно рассматривать как наименьшую неподвижную точку функционала, связанного с (7).

Переписанные определения можно использовать для постусловий, принимающих значения в произвольном множестве  $M$ . Если  $g(v)$  — некоторое постусловие такого рода, то ясно, что  $\text{wp}(P, g)(v) = m$  для произвольной программы  $P$  тогда и только тогда, когда выполнение  $P$  для начального состояния  $v$  обязательно завершается, для каждого возможного заключительного состояния  $v'$  определено  $g(v')$  и хотя бы для одного заключительного состояния  $v''$  имеет место  $g(v'') = m$  (мы, как и прежде, вкладываем в запись  $g(v'') = m$  следующий смысл: среди значений  $g(v'')$  имеется  $m$ ).

Пусть программа  $R$  содержит целочисленную переменную  $x$ . Если рассмотреть постусловие  $x$ , то  $\text{wp}(R, x)$  — это многозначная функция, сопоставляющая в случае заведомой завершенности начальному набору значений переменных все значения, которые может получить  $x$  в результате выполнения программы для данных начальных значений.

Выведенная в § 1 формула

$$\text{wp}(P, g) = p^b g \cap p^{\overline{b}} \overline{g}$$

показывает, что  $\text{wp}(P, g)$  есть, в терминологии § 4 гл. I, преобразование вида  $P^H$ : роль  $p^*(g)$  играет  $p^b g$ , а роль операционного смещения — построение пересечения с  $\overline{p^b g}$ . Возможны и другие варианты операционных смещений. Кроме этого, в качестве  $p^*(g)$  можно брать не только  $p^b g$ , но и  $p^{\#} g$ . Иными словами, когда мы имеем дело с недетерминированной программой и, возможно, с многозначным постусловием, есть много способов для определения некоторой функции, которую можно считать в некотором смысле слабейшим предусловием. Перечислим некоторые варианты, отличающиеся от только что рассмотренного (этот вариант будем называть *первым вариантом* и обозначать  $\text{wp}_1(P, g)$ ).

Второй вариант:  $\text{wp}_2(P, g)(v) = m$  тогда и только тогда, когда для начального состояния  $v$  выполнение  $P$  может завершаться и дать такое состояние  $v'$ , для которого  $g(v') = m$ .

Для получения этого варианта надо в формулах (6), (7) каждое выражение *if* понимать так, что его значение определено для состояния  $v$ , если для  $v$  выполнено условие, стоящее слева от некоторой стрелки, и одновременно определено значение выражения, стоящего справа от этой стрелки; вся совокупность значений  $\text{wp}_2(P, g)(v)$  пони-

мается как соответствующее объединение. Поскольку равенство  $\text{wr}_2(P, g)(v) = m$  не гарантирует завершенности  $P$  для  $v$ , то ссылка на теорему 1 здесь не нужна. Напомним, что для этой задачи невозможно дать никаких формул для сильнейшего постусловия, так как и само понятие сильнейшего постусловия определить невозможно.

Для упомянутого выше примера программы  $R$  и постусловия  $x$  многозначная функция  $\text{wr}_2(R, x)$  сопоставляет начальному набору значений переменных все целые числа, которые может получить в качестве заключительного значения переменная  $x$ .

Третий вариант:  $\text{wr}_3(P, g)(v) = m$  тогда и только тогда, когда выполнение  $P$  для начального состояния  $v$  обязательно завершается и для каждого возможного заключительного состояния  $v'$  имеет место  $g(v') = m$ . Для построения этого варианта выражение **if** в (6), (7) надо понимать так, что в совокупность его значений для данного  $v$  включаются те значения, которые получаются при любом выборе альтернатив (предикатов, истинных для данного состояния). Здесь ссылка на теорему 1 существенна.

В частности, для разбираемого примера  $\text{wr}_3(R, x) \times \times (v) = m$  означает, что для данных начальных значений переменных выполнение  $R$  обязательно завершается и для любого заключительного состояния значение переменной  $x$  есть число  $m$ .

Следующий — четвертый — вариант может быть применен, если множество  $M$  — это некоторая решетка с универсальными нижней и верхней границами. Можно рассмотреть, например, множество всех целых чисел, добавив к нему  $-\infty$  и  $\infty$ .

Пусть  $g$  — частичная однозначная функция со значениями в  $M$ ;  $\text{wr}_4(R, g)(v) = m$  тогда и только тогда, когда для начального состояния  $v$  выполнение  $R$  обязательно завершается, для каждого заключительного состояния  $v'$  определено значение  $g(v')$  и  $m$  есть максимум всех таких  $g(v')$ . Этот вариант удобен, в частности, для исследования завершенности циклических инструкций. Когда  $P$  определено с помощью (1),  $\text{wr}_4(P, g)$  определяется выражением

$$\text{if } \sigma_1 \rightarrow g_1 \square \sigma_2 \rightarrow g_2 \square \dots \square \sigma_{2^n-1} \rightarrow g_{2^n-1} \text{ fi},$$

где  $\sigma_1, \sigma_2, \dots, \sigma_{2^n-1}$  — конъюнкции

$$\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n, \beta_1 \wedge \beta_2 \wedge \dots \wedge \neg \beta_n, \dots$$

$$\dots, \neg \beta_1 \wedge \neg \beta_2 \wedge \dots \wedge \beta_n,$$

в каждую из которых любое  $\beta_i$  входит со знаком отрицания или без него и заведомо хотя бы одно  $\beta_i$  входит без знака отрицания;  $g_i$  определяется так, что, например,  $g_i = \max_{j=1}^k (\text{if } \sigma_i \rightarrow \text{wp}_4(P_j, g) \text{ fi})$ , если  $\sigma_i = \beta_1 \wedge \dots \wedge \beta_k \wedge \bigwedge \bigwedge \beta_{k+1} \wedge \dots \wedge \bigwedge \beta_n$ . При этом максимум нескольких функций, зависящих от  $v$  считается не определенным для  $v = v_0$ , если для  $v = v_0$  не определена хотя бы одна функция.

Для инструкции  $P$  вида (3) можно, как и прежде, описать приближения  $\text{wp}_4(P, g)_i$ :

$$\text{wp}_4(P, g)_0 = \text{if } \bigwedge \beta \rightarrow g \text{ fi},$$

$$\text{wp}_4(P, g)_i = \text{if } \sigma_1 \rightarrow g'_1 \square \dots \square \sigma_{2^n-1} \rightarrow g'_{2^n-1} \square \bigwedge \beta \rightarrow g \text{ fi},$$

где  $g'_i$  отличается от  $g_i$  тем, что рассматривается

$$\max_{j=1}^k (\text{if } \sigma_i \rightarrow \text{wp}_4(P_j, \text{wp}_4(P, g)_{i-1}) \text{ fi}).$$

Для приведенного выше примера программы  $R$  и постусловия  $x$  предусловие  $\text{wp}_4(R, x)$  представляет собой частичную однозначную функцию, которая определена тогда и только тогда, когда  $R$  обязательно завершается для данного набора начальных значений переменных; значение ее есть максимум достижимых значений переменной  $x$ .

Если работать со всюду определенными функциями, доопределяя частичные функции значениями, равными универсальной нижней границе решетки  $M$ , то прибегать к выражению  $\text{if}$  с увеличенным числом элементов не придется — в формулах (6), (7) выражения вида

$$\text{if } \beta_1 \rightarrow E_1 \square \dots \square \beta_m \rightarrow E_m \text{ fi},$$

где  $E_1, \dots, E_m$  — однозначные, всюду определенные функции, надо заменить на  $\max_{j=1}^m E'_j$ , где функция  $E'_j$  совпадает с  $E_j$ , когда выполнено  $\beta_j$ , а в противном случае — равна универсальной нижней границе.

Для чисел в качестве универсальных границ вводятся  $\infty$  и  $-\infty$ . Для булевой решетки  $\{И, Л\}$  границами будут И, Л. Все, что было сказано выше, применимо, конечно, после соответствующих изменений, и к поиску минимума. Слабейшее предусловие Дейкстры представляет собой именно минимум в решетке  $\{И, Л\}$ .

Пятый вариант:  $\text{wr}_5(P, g)(v) = m$  тогда и только тогда, когда при условии, что для начального состояния  $v$  выполнение  $P$  может завершиться, хотя бы для одного заключительного состояния  $v'$  имеет место  $g(v') = m$ . Для вычисления  $\text{wr}_5(P, g)$  в формулах (6), (7) надо понимать  $\text{if}$  в том же смысле, что и во втором варианте, с той, однако, разницей, что если для  $v$  не выполняется  $\beta_1 \vee \dots \vee \beta_n$ , то  $\text{if}$  принимает все значения из множества  $M$ . Здесь  $\text{wr}_5(P, g)_0$  отдельно рассматривать не надо, следует взять  $\text{wr}_5(P, g)_{-1} = V \times M$ . Далее, для  $P$ , заданного с помощью (3),

$$\text{wr}_5(P, g) = \bigcap_{i=-1}^{\infty} \text{wr}_5(P, g)_i. \quad (9)$$

Для нашего примера  $\text{wr}_5(R, x)$  получается из  $\text{wr}_2(R, x)$  доопределением (там, где  $\text{wr}_2(R, x)$  не определено) всеми элементами  $M$ .

Отметим, что  $\text{wr}_5(P, g)$  — это слабейшее свободное предусловие для  $g$  относительно  $P$ , т. е.  $\text{wr}_5(P, g) = \text{wlp}(P, g)$ . Если постусловие — это предикат, то во многих случаях удобно было бы пользоваться такими формулами для вычисления  $\text{wlp}$ , которые построены с помощью логических связок и кванторов, т. е. формулами в духе предложенных Дейкстрой для  $\text{wr}$ . В. В. Чесноков показал, что для постусловия-предиката  $\psi$  и программы  $P$  вида (3) предикат  $\text{wlp}(P, \psi)$  определяется формулами

$$\text{wlp}(P, \psi)_0 = \beta\beta \vee \psi,$$

$$\text{wlp}(P, \psi)_i = \text{wlp}(\text{if}, \text{wlp}(P, \psi)_{i-1}) \vee \neg \beta\beta \wedge \psi, \quad (10)$$

$$\text{wlp}(P, \psi) = \forall t \text{ wlp}(P, \psi)_t.$$

Формулами (9), (10) удобно пользоваться для доказательства свойств вида  $\{f\}P\{g\}$ , применяя индукцию по  $t$  (аналогично детерминированному случаю, обсуждавшемуся в § 3 гл. I; на эту индукцию можно смотреть как на индукцию по неподвижной точке). В. В. Чесноков также показал, что даже в детерминированном случае формулам для  $\text{wr}(P, \psi)$  нельзя придать вид, аналогичный виду формул (10); если бы  $\text{wr}(P, \psi)$  можно было определить подобными формулами, то это дало бы способ доказательства по индукции свойств, включающих завершимость. Доказательство В. В. Чеснокова основано на существовании программы, которая задает функцию с рекурсивно неразрешимой областью определения.



Рассмотрим еще некоторые примеры. Пусть  $a_1, \dots, a_n$  — числовой массив. Будем исследовать программу  $G$ :

$$i := 1; j := 0; \text{ do } i \leq \frac{n}{2} \wedge a_i < 0 \rightarrow j := j + 1;$$

$$i := i + 1 \square$$

$$i \leq \frac{n}{2} \wedge a_{n-i+1} < 0 \rightarrow j := j + 1;$$

$$i := i + 1 \square$$

$$i \leq \frac{n}{2} \wedge a_i \geq 0 \rightarrow i := i + 1 \square$$

$$i \leq \frac{n}{2} \wedge a_{n-i+1} \geq 0 \rightarrow i := i + 1$$

od.

Существенный результат выполнения  $G$  — заключительное значение переменной  $j$ , поэтому естественно рассмотреть  $j$  в качестве постусловия. Если мы хотим описать для произвольного начального состояния все те значения  $j$ , которые могут получиться в результате выполнения входящей в программу  $G$  инструкции **do**, то можно рассмотреть  $\text{wp}_2(\text{do}, j)$ . Обозначим  $f_t = \text{wp}_2(\text{do}, j)_t$  для  $t = 0, 1, \dots$  и  $f = \text{wp}_2(\text{do}, j)$ ; все  $f_t$  и  $f$  являются функциями  $i, j, a_1, \dots, a_n$ , и мы будем иногда писать соответственно  $f_t(i, j, n, a)$ ,  $f(i, j, n, a)$ .

В рассматриваемом случае  $\neg \beta \beta = (i > \frac{n}{2})$ ,  $f_0 = \text{if } i > \frac{n}{2} \rightarrow j \text{ fi}$ . Далее, при  $t > 0$  имеет место

$$f_t(i, j, n, a) = \text{if } i \leq \frac{n}{2} \wedge a_i < 0 \rightarrow f_{t-1}(i + 1, j + 1, n, a) \square$$

$$i \leq \frac{n}{2} \wedge a_{n-i+1} < 0 \rightarrow$$

$$f_{t-1}(i + 1, j + 1, n, a) \square$$

$$i \leq \frac{n}{2} \wedge a_i \geq 0 \rightarrow f_{t-1}(i + 1, j, n, a) \square$$

$$i \leq \frac{n}{2} \wedge a_{n-i+1} \geq 0 \rightarrow$$

$$f_{t-1}(i + 1, j, n, a) \square$$

$$i > \frac{n}{2} \rightarrow j$$

fi.

Очевидно, что  $f(i, j, n, a) = f_{\lfloor n/2 \rfloor + 1}(i, j, n, a) = f_{\lfloor n/2 \rfloor + 2}(i, j, n, a) = \dots$ . С помощью выписанных формул можно доказать, что если  $n/2 - i < s$ , то  $f_s(i, j, n, a)$  определена. Можно еще доказать, что если  $f_s(i, j, n, a)$  определена, то любое ее значение  $\geq g$ . Эти свойства полезны для любых специальных рассмотрений  $f_0, f_1, \dots$  и  $f$ . Используя эти свойства и индукцию, можно установить, что если существует  $k$  такое, что  $0 < k \leq n/2$ ,  $a_k < 0$  и  $a_{n-k+1} < 0$ , то при  $i \leq k$  каждое значение  $f$  больше  $j$ ; так же можно доказать, что  $f(i, j, n, a)$  имеет по крайней мере два значения, коль скоро существует  $k$  такое, что  $i \leq k \leq n/2$  и  $a_k a_{n-k+1} < 0$ .

Можно было бы проводить аналогичные исследования программы  $G$ , привлекая для этого  $\text{wr}_4$ . Можно, например, определить условия, при которых  $f(i, j, n, a) > j$  или  $f(i, j, n, a) = j$ . При этом мы будем получать разную информацию, понимая слабое условие как минимум достижимых значений и как максимум.

Другой пример — рассмотрим программу  $H$

$$\text{do } 6 \mid n \rightarrow n := \frac{n}{6} \square 20 \mid n \rightarrow \frac{27n}{20} \text{ od.} \quad (11)$$

Для доказательства завершимости  $H$  при любом натуральном  $n$  рассмотрим функцию  $c_2(n)$  — степень вхождения простого множителя 2 в число  $n$ . Рассмотрим  $H$

$$\text{if } 6 \mid n \rightarrow n := \frac{n}{6} \square 20 \mid n \rightarrow n := \frac{27n}{20} \text{ fi,}$$

Вычислим  $\text{wr}_4(H', c_2(n))$ , имея в виду максимум достижимых значений:

$$\begin{aligned} \text{wr}_4(H', c_2(n)) &= \\ &= \text{if } 6 \mid n \wedge 20 \mid n \rightarrow \max(c_2(n/6), c_2(27n/20)) \square \\ &\quad \neg(6 \mid n) \wedge 20 \mid n \rightarrow c_2(27n/20) \square \\ &\quad (6 \mid n) \wedge \neg(20 \mid n) \rightarrow c_2(n/6) \\ &\text{fi} = \\ &= \text{if } 6 \mid n \wedge 20 \mid n \rightarrow c_2(n) - 1 \square \\ &\quad \neg(6 \mid n) \wedge 20 \mid n \rightarrow c_2(n) - 2 \square \\ &\quad (6 \mid n) \wedge \neg(20 \mid n) \rightarrow c_2(n) - 1 \\ &\text{fi.} \end{aligned}$$

Теперь видно, что в результате выполнения одного этапа значение  $n$  изменяется таким образом, что  $c_2(n)$  убывает.

В § 4 гл. I были рассмотрены способы задания таких индуцированных детерминированными программами преобразований, которые позволяют исследовать, в частности, число требуемых выполнением программы операций, а также многие другие не связанные с изменением состояния последствия выполнения программы. Построение подобных преобразований возможно и для недетерминированных программ. Различные пути выполнения недетерминированной программы могут требовать различного числа операций, поэтому целесообразно вести речь, например, о наибольшем или наименьшем их числе. Пусть нас интересует наименьшее число. Считая, что на множестве состояний  $V$  рассматриваются частичные однозначные функции, принимающие значения в множестве неотрицательных целых чисел, легко модифицировать определенное выше преобразование  $wp_4$  (понимаемое в смысле максимума достижимых значений) и определить преобразование  $wp'_4$  такое, что  $wp'_4(P, g)(v) = m$  тогда и только тогда, когда для начального состояния  $v$  выполнение  $P$  обязательно завершается. Для каждого соответствующего заключительного состояния  $v'$  определено значение  $g(v')$ , и при этом  $m = \max(n' + g(v'))$ , где  $n'$  — это наибольшее число операций, которое может потребоваться для достижения заключительного состояния  $v'$ , а  $v'$  пробегает все заключительные состояния, соответствующие начальному состоянию  $v$ .

Если  $P$  — это инструкция присваивания, то  $wp'_4(P, g) = p * (g) + C$ , где  $C$  — число операций, необходимое для вычисления значения правой части инструкции присваивания. Если  $P$  — пустая инструкция, то  $wp'_4(P, g) = g$ ; если  $P$  — составная инструкция, то

$$wp'_4(P, g) = wp'_4(P_1, wp'_4(P_2, g)).$$

Если  $P$  — инструкция **if** или **do**, то можно пользоваться приведенными выше формулами для  $wp_4$ , заменив в них  $wp_4$  на  $wp'_4$  (при этом мы будем игнорировать случаи, когда вычисление значений предикатов  $\beta_1, \dots, \beta_n$  требует выполнения некоторого числа исследуемых операций).

Например, для программы  $H$  вида (11) можно следующим образом описать  $f_i = wp'_4(H, 0)_i$ , где  $0$  — тождест-

венно равная нулю функция:

$$f_i(n) = \text{if } 6 \mid n \wedge 20 \nmid n \rightarrow 1 + \max(f_{i-1}(n/6), f_{i-1}(27n/6)) \square$$

$$\neg(6 \mid n) \wedge 20 \mid n \rightarrow 1 + f_{i-1}(27n/20) \square$$

$$\neg(6 \mid n) \wedge \neg(20 \mid n) \rightarrow 1 + f_{i-1}(n/6) \square$$

$$\neg(6 \mid n) \wedge \neg(20 \mid n) \rightarrow 0 \text{ fi,}$$

$$\text{wp}'_4(H, 0)(n) = \max_{i=0}^{\infty} f_i(n).$$

**Примечание.** Можно привести примеры, сходные с последними примерами § 4 гл. I, когда требуется раздельное вычисление операционного смещения и сопряженного преобразования. Возникающее в связи с недетерминированностью неудобство состоит в том, что, задавая отдельно операционное смещение и сопряженное преобразование, приходится описывать не только интересующее нас последствие выполнения программы, но и изменение самого состояния. Мы, как обычно, выбираем для анализа программ некоторое множество  $M$ , но должны, фактически, рассматривать функции со значениями в множестве  $M \times V$ .

Предложенные в этом параграфе варианты предусловия дают возможность анализировать различные пути выполнения программы, позволяют выявлять как ситуации, которые обязательно возникают при выполнении программы, так и ситуации, которые могут возникнуть (при выборе определенных путей выполнения), а могут и нет. В то же время слабейшее предусловие Дейкстры предназначено для исследования таких ситуаций, которые обязательно возникают при любом выборе пути выполнения программы; кроме того, если привлекается слабейшее предусловие Дейкстры, то нет способа различать те начальные состояния, для которых выполнение программы не завершается ни на одном пути выполнения, от тех, для которых выполнение программы может как завершиться, так и не завершиться.

## § 5. Анализ трудоемкости в среднем

Если выполнение недетерминированной инструкции носит вероятностный характер, т. е. если для каждого фиксированного начального состояния  $v$  определены вероятности перехода к различным заключительным состояниям, то можно рассмотреть еще один вариант слабейшего предусловия:  $\text{wr}_s(P, g)$ , где  $g$  — однозначная числовая частичная функция. Функция  $\text{wr}_s(P, g)$  — тоже однозначная числовая частичная функция, при этом  $\text{wr}_s(P, g)(v) = m$

тогда и только тогда, когда выполнение  $P$  для начального состояния  $v$  обязательно завершается, для каждого заключительного состояния определено значение функции  $g$  и математическое ожидание значения равно  $m$ . Использование предусловия  $wр_*$  при удачном выборе функции  $g(v)$  позволяет проводить анализ в среднем некоторых алгоритмов.

Многие задачи поиска могут быть сформулированы так: в множестве  $V$  выделено подмножество  $W$ ; имеется некоторый набор преобразований  $P_1, \dots, P_L$  множества  $V$  в себя (для простоты далее в этом параграфе не различаются инструкции и соответствующие им преобразования), требуется указать алгоритм перевода произвольного состояния  $v_0$  с помощью преобразований данного набора в какое-нибудь состояние  $v_1 \in W$ . Подмножество  $W$  может быть выделено, например, с помощью некоторой всюду определенной на  $V$  числовой функции  $g(v)$ :  $W = \{v | v \in V, g(v) \leq 0\}$ . Особый интерес представляют такие алгоритмы, которые требуют минимального числа преобразований, т. е. оптимальные алгоритмы. Наиболее трудной для анализа является ситуация, когда упомянутые преобразования носят недетерминированный характер, т. е. являются не однозначными функциями, а бинарными отношениями. Совершенно ясно, что если для  $i = 1, 2, \dots, L$  и для любого  $v$  выполнено  $g(v) - wр_i(P_i, g)(v) \leq 1$ , то для построения  $v_1$  такого, что  $g(v_1) \leq 0$ , потребуется не менее  $g(v_0)$  преобразований. Пусть выполнение каждого преобразования носит вероятностный характер, тогда  $wр_*$  может сослужить такую же службу для получения нижних оценок трудоемкости алгоритма, оптимального в среднем. Идея состоит в том, что среднее число необходимых преобразований будет  $\geq g(v_0)$ , коль скоро для любого  $v$  неравенства

$$M(-\Delta g) = g(v) - wр_*(P_i, g)(v) \leq 1, \quad i = 1, \dots, L, \quad (1)$$

выполняются независимо от того, в результате каких преобразований (и с какими исходами), отправляясь от состояния  $v_0$ , получено  $v$ . Заметим, что таким образом мы допускаем зависимость вероятностей не только от  $v$ , но и от процесса получения  $v$ ; при вычислении  $wр_*(P_i, g)(v)$  — математического ожидания достигаемого значения функции  $g$  — должны в таком случае брать условные вероятности. Дадим обоснование этого метода.

**Теорема 1.** Пусть  $\xi_1, \xi_2, \dots$  — неотрицательные случайные величины на некотором вероятностном простран-

стве. Пусть  $h_1, h_2, \dots$  — такие константы, что для  $k = 1, 2, \dots$  независимо от значений, принятых  $\xi_1, \dots, \xi_{k-1}$ , выполнено  $M\xi_k < h_k$  (т.е.  $M(\xi_k | \xi_1, \dots, \xi_{k-1}) \leq h_k$ ). Зафиксируем неотрицательное число  $q$  и рассмотрим целочисленную случайную величину  $\tau = \min \left\{ n \mid \sum_{k=1}^n \xi_k \geq q \right\}$ . Тогда

$$M \left( \sum_{k=1}^{\tau} h_k \right) \geq q.$$

Доказательство. Рассмотрим случайные величины  $\chi_1, \chi_2, \dots$ : для  $k = 1, 2, \dots$

$$\chi_k = \begin{cases} 1, & \text{если } \xi_1 < q, \xi_1 + \xi_2 < q, \dots, \xi_1 + \dots + \xi_{k-1} < q, \\ 0 & \text{в противном случае.} \end{cases} \quad (2)$$

Заметим, что

$$\chi_k = \begin{cases} 1, & \text{если } \tau \geq k, \\ 0 & \text{в противном случае} \end{cases}$$

$$\text{и } 1 = \chi_1 \geq \chi_2 \geq \dots$$

Обозначим  $p_k = P\{\tau = k\} = P\{\chi_k = 1\} - P\{\chi_{k+1} = 1\}$ ,  $k = 1, 2, \dots$ ; ясно, что  $P\{\chi_k = 1\} = 1 - p_1 - p_2 - \dots - p_{k-1}$ . Имеем

$$\begin{aligned} M \left( \sum_{k=1}^{\tau} h_k \right) &= \sum_{k=1}^{\infty} \sum_{j=1}^k h_j P\{\tau = k\} = \\ &= \sum_{k=1}^{\infty} \left( h_k \sum_{j=k}^{\infty} p_j \right) = \sum_{k=1}^{\infty} h_k P\{\chi_k = 1\}. \end{aligned} \quad (3)$$

Введем случайные величины  $\eta_k = \chi_k \xi_k$ ,  $k = 1, 2, \dots$ ; из определения (2) случайных величин  $\chi_k$  ( $k = 1, 2, \dots$ ) следует, что

$$\sum_{k=1}^{\infty} \eta_k \geq q. \quad (4)$$

Докажем, что для  $k = 1, 2, \dots$

$$M\eta_k \leq h_k P\{\chi_k = 1\}. \quad (5)$$

Действительно,

$$M\eta_k = M(\chi_k \xi_k) = M(M(\chi_k \xi_k | \xi_1, \dots, \xi_{k-1})),$$

но из определения случайных величин  $\chi_k$  выводится, что  $M(\chi_k | \xi_1, \dots, \xi_{k-1}) = \chi_k$ , поэтому

$$M(\chi_k \xi_k | \xi_1, \dots, \xi_{k-1}) = \chi_k M(\xi_k | \xi_1, \dots, \xi_{k-1}).$$

Отсюда

$$M\eta_k = M(\chi_k M(\xi_k | \xi_1, \dots, \xi_{k-1})) \leq M(\chi_k h_k) = h_k P\{\chi_k = 1\},$$

и неравенство (5) доказано.

Из (3), (4), (5) получаем

$$\begin{aligned} q = Mq &\leq M\left(\sum_{k=1}^{\infty} \eta_k\right) = \sum_{k=1}^{\infty} M\eta_k \leq \sum_{k=1}^{\infty} h_k P\{\chi_k = 1\} = \\ &= M\left(\sum_{k=1}^{\tau} h_k\right), \end{aligned}$$

что и требовалось \*).

Следствие. Пусть  $h_k = 1$  для  $k = 1, 2, \dots$ ; тогда  $M\tau \geq q$ .

Для обоснования описанного выше метода полагаем, что  $h_k = 1$  для  $k = 1, 2, \dots$  и что последовательность случайных величин  $\xi_1, \xi_2, \dots$  — это последовательность величин  $-\Delta g$ , которые возникают в результате применения преобразования данного набора.

Видно, что теорема 1 позволяет сформулировать метод в более общей форме: последовательность величин  $M(-\Delta g)$  должна быть ограничена сверху некоторой последовательностью  $h_1, h_2, \dots$ ; случайная величина  $\tau$ , равная числу необходимых преобразований, такова, что  $M\left(\sum_{k=1}^{\tau} h_k\right) \geq g(v_0)$ .

Описанный метод анализа может быть применен к алгоритмам, связанным с сортировкой. Рассмотрим задачу одновременного нахождения наибольшего и наименьшего элементов массива  $x_1, \dots, x_n$  с помощью операций сравнения; будет описан алгоритм, минимизирующий среднее число операций (оптимальный в среднем алгоритм). Решение задачи было получено благодаря применению этого метода. Алгоритм, минимизирующий максимальное число сравнений, был предложен Полом [17].

Алгоритм Пола требует не более  $\lfloor 3n/2 \rfloor - 2$  сравнений \*\*), в ходе его применения просматриваются пары  $x_{2i-1}, x_{2i}$ ,  $i = 1, 2, \dots, \lfloor n/2 \rfloor$ , в каждой паре выбирается больший и меньший элементы, значение большего обозначается  $b_i$ , значение меньшего —  $c_i$  (если  $n$  нечетно, то

---

\*) Приведенное доказательство сообщено автору П. И. Черновым.

\*\*)  $\lfloor y \rfloor$  — наименьшее целое число, большее или равное  $y$ .

предлагается положить  $b_{\lfloor n/2 \rfloor + 1} = x_n$  в случае  $x_n > c_1$  и  $c_{\lfloor n/2 \rfloor + 1} = x_n$  в противном случае); после этого из  $b_1, b_2, \dots$  выбирается наибольший элемент  $M$ , из  $c_1, c_2, \dots$  — наименьший  $m$ , и задача решена.

**Примечание 1.** Массивы  $b_1, b_2, \dots$  и  $c_1, c_2, \dots$  можно не рассматривать. Наибольший и наименьший элементы пары  $x_{2i-1}, x_{2i}$ ,  $i = 2, 3, \dots, \lfloor n/2 \rfloor$ , можно сразу сравнивать с наибольшим и наименьшим элементами из  $x_1, \dots, x_{2i-2}$  (если  $n$  нечетно, то на последнем шаге  $x_n$  сравнивается с наибольшим и наименьшим элементами из  $x_1, \dots, x_{n-1}$ ).

Алгоритм Пола будем называть *алгоритмом Р*. Очевидно, что, когда применяется алгоритм  $P$ , на каждую пару элементов  $x_{2i-1}, x_{2i}$ ,  $i = 2, \dots$  уходит в итоге три сравнения, на пару  $x_1, x_2$  — одно сравнение. Если  $n$  нечетно, то на  $x_n$  уйдет два сравнения. Всего получится  $\lfloor 3n/2 \rfloor - 2$  сравнений. Тривиальный алгоритм (алгоритм  $T$ ) предписывает поиск по отдельности наибольшего и наименьшего элементов и требует  $2n - 3$  сравнений.

Доказательство того, что алгоритм  $P$  действительно минимизирует максимальное число сравнений, основано в статье Пола на наблюдении, что каждая стадия применения произвольного алгоритма  $S$  одновременного нахождения наибольшего и наименьшего элементов среди  $x_1, \dots, x_n$  характеризуется четверкой множеств  $(A, B, C, D)$ :  $A$  состоит из тех элементов  $x_1, \dots, x_n$ , которые вообще не сравнивались;  $B$  — из элементов, во всех сравнениях оказавшихся большими;  $C$  — из элементов, во всех сравнениях оказавшихся меньшими;  $D$  — из элементов, в некоторых сравнениях оказавшихся большими, а в некоторых — меньшими. Пусть  $a, b, c, d$  — количество элементов множеств  $A, B, C, D$ . Сначала имеем  $a = n, b = c = d = 0$ , в конце должны иметь  $a = 0, b = c = 1, d = n - 2$ . После первого же сравнения постоянно будет выполнено

$$b \geq 1, \quad c \geq 1. \quad (6)$$

Все сравнения, совершаемые при выполнении произвольного алгоритма  $S$ , можно разбить на типы, обозначаемые  $AA, AB, AC, AD, BB, BC, BD, CC, CD, DD$ , например: сравнение принадлежит типу (или является сравнением типа)  $AB$ , если один из сравниваемых элементов берется из  $A$ , другой — из  $B$ .

Исходя из этого, Пол выписывает все возможные изменения четверки чисел  $(a, b, c, d)$  под действием сравнений разных типов. Так, например, в его записи

$$AB: (a, b, c, d) \rightarrow (a - 1, b, c, d + 1) \mid (a - 1, b, c + 1, d)$$



(вертикальная черта означает здесь «или»). Приведенная запись отражает две возможности: элемент, взятый из  $A$ , в сравнении с элементом, взятым из  $B$ , оказался большим и, соответственно, оказался меньшим. На основании подобного анализа всех типов сравнений и доказательства некоторых сопутствующих утверждений выводится, что не существует алгоритма  $S$ , который бы во всех случаях требовал менее  $\lfloor 3n/2 \rfloor - 2$  сравнений.

Множество четверок неотрицательных целых чисел  $(a, b, c, d)$ , подчиненных условию  $a + b + c + d = n$ , можно рассматривать как множество состояний, а  $AA, AB, \dots$  — как недетерминированные инструкции. Если принять эту точку зрения, то можно сказать, что Пол в своем исследовании прибегает к операционным соображениям — рассматривает возможные переходы из состояния в состояние. Применим метод, изложенный в начале настоящего параграфа. Будем считать, что  $x_1, \dots, x_n$  — перестановка чисел  $1, \dots, n$ . При подсчете вероятностей будем предполагать, что все перестановки равновероятны.

**Теорема 2.** *Среднее число сравнений для произвольного алгоритма  $S$  не меньше  $\frac{3}{2}n - 2$  при четном  $n$  и  $\frac{3}{2}n - 2 + \frac{1}{2n}$  при нечетном  $n$ .*

**Доказательство.** Рассмотрим  $g(a, b, c, d) = \frac{3}{2}a + b + c - 2$ . Сначала  $g = \frac{3}{2}n - 2$ , в конце должно быть  $g = 0$ . Принимая во внимание (6), получаем, что  $g = 0$  — это необходимое и достаточное условие того, что сделано все, что требуется.

Каждое сравнение типов  $AA, BB, CC$  понижает значение  $g$  на 1, т. е. дает  $\Delta g = -1$ . Сравнения типов  $BD, CD$  дают  $\Delta g = 0$  или  $\Delta g = -1$ . Сравнения типов  $AB, AC$  дают  $\Delta g = -3/2$  или  $\Delta g = -1/2$ . Сравнение типа  $BC$  дает  $\Delta g = -2$  или  $\Delta g = 0$ . Сравнения типов  $AD, DD$  всегда дают  $\Delta g = 0$ . Но каждое сравнение кроме понижения значения  $g$  еще и предоставляет некоторую информацию об элементах  $x_1, \dots, x_n$ , которая, как можно ожидать, в определенные моменты создает возможность такого специального выбора элементов для сравнения, что (по крайней мере в среднем) окажется  $\Delta g < -1$ . Однако эти ожидания не оправдываются. Рассмотрим поочередно типы сравнений, дающих в некоторых случаях  $\Delta g < -1$ .

**Тип  $BC$ .** Покажем невозможность основанного на результатах предыдущих сравнений специального выбора

индексов  $i$  и  $j$  таких, что  $x_i \in B$ ,  $x_j \in C$ , и что вероятностью, не меньшей половины, выполняется  $x_i < x_j$ .

Пусть выбрана некоторая пара индексов  $i, j$ . Рассмотрим множество всех перестановок  $y_1, \dots, y_n$ , применение к которым алгоритма  $S$  потребует перед сравнением  $y_i$  с  $y_j$  произвести сравнения элементов в том же порядке и с теми же индексами, что и применение  $S$  к перестановке  $x_1, \dots, x_n$ , при этом результаты сравнений должны быть теми же самыми. Если в любой перестановке этого множества, для которой  $y_i < y_j$ , поменять местами  $y_i$  и  $y_j$ , то, в силу определения множеств  $B$  и  $C$ , получится перестановка из этого же множества. При этом результат сравнения  $y_i$  с  $y_j$  изменится на противоположный. Каждой удачной перестановке из этого множества (т. е. такой, что  $y_i < y_j$ ) отвечает неудачная. Однако не каждой неудачной перестановке при транспозиции  $y_i$  с  $y_j$  отвечает удачная. Можно построить неудачную перестановку, в которой  $y_j = 1$ : транспозиция  $y_i$  с  $y_j$  в такой перестановке выведет последнюю из рассматриваемого множества. Для построения возьмем какую-нибудь удачную перестановку  $y_1, \dots, y_n$  и преобразуем ее: каждый элемент  $y_k, k = 1, 2, \dots, n$ , для которого выполнено  $y_k < y_j$ , заменяем на  $y_k - 1$ ,  $y_j = 1$ .

Таким образом, вероятность того, что  $i$ -й элемент меньше  $j$ -го, есть  $\frac{1}{2} - \epsilon$ ,  $\epsilon > 0$ . Получаем

$$M\Delta g = -2\left(\frac{1}{2} - \epsilon\right) = -1 + 2\epsilon > -1.$$

*Тип АВ.* Аналогично предыдущему можно показать, что на любой стадии применения алгоритма, при любом способе выбора индексов  $i, j$  таких, что  $x_i \in A$ ,  $x_j \in B$ , вероятность выполнения неравенства  $x_i \geq x_j$  есть  $\frac{1}{2} - \epsilon$ ,  $\epsilon > 0$ . Получаем

$$M\Delta g = -\frac{3}{2}\left(\frac{1}{2} - \epsilon\right) - \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) = -1 + \epsilon > -1.$$

Для дальнейшего будет полезна оценка  $\epsilon$ . Очевидно, что чем больше сравнений к рассматриваемому моменту прошел элемент  $x_j \in B$ , тем меньше вероятность того, что элемент  $x_i \in A$  окажется больше него. Наибольшая вероятность получается в случае, когда  $x_j$  сравнивался только с одним элементом, например с  $x_m$  (таким образом,  $x_j > x_m$ ), а  $x_m$  сравнивался со всеми элементами, кроме  $x_i$ , и всегда оказывался меньшим. Без труда устанавливается,

что в этом крайнем случае неравенство  $x_i > x_j$  выполняется с вероятностью  $\frac{1}{2} - \frac{1}{2n}$ . Таким образом,

$$M\Delta g \geq -1 + \frac{1}{2n}. \quad (7)$$

*Тип AC.* Рассматривается аналогично предыдущему. Имеет место (7).

Отметим, что для сравнений типов  $BD$ ,  $CD$  тоже будет выполняться  $M\Delta g > -1$ .

При четном  $n$  оптимальный в среднем алгоритм должен использовать только сравнения типов  $AA$ ,  $BB$ ,  $CC$  (сравнений потребуется в точности  $\frac{3}{2}n - 2$ ), так же как и алгоритм  $P$ . Если  $n$  нечетно, то алгоритм обязательно предписывает хотя бы одно сравнение типа  $AB$ ,  $AC$  или  $AD$ . Сравнение типа  $AD$  дает  $\Delta g = -1/2$ , что для нас менее привлекательно, чем (7). Поэтому среднее число сравнений не может быть меньше, чем

$$\frac{3}{2}n - 2 - \left(1 - \frac{1}{2n}\right) + 1 = \frac{3}{2}n - 2 + \frac{1}{2n}.$$

**Примечание 2.** В доказательстве Пола, также как и в нашем доказательстве теоремы 2, для каждого алгоритма решения поставленной задачи удастся предложить упрощенную модель этого алгоритма. Модельный алгоритм является недетерминированным, но поддается полному исследованию. Таким образом, изучение недетерминированных алгоритмов интересно не только в связи с языком Дейкстры.

Опишем конструкцию оптимального в среднем алгоритма. Это будет некоторая переделка алгоритма  $P$ .

При четном  $n$  оставляем алгоритм  $P$  без изменений. Рассмотрим случай нечетного  $n$ . Пусть вновь имеем

$$\begin{aligned} b_i &= \max(x_{2i-1}, x_{2i}), \\ c_i &= \min(x_{2i-1}, x_{2i}), \quad i = 1, 2, \dots, [n/2]. \end{aligned}$$

Пусть  $M = b_s = \max_{i=1}^{[n/2]} b_i$ . Теперь сравниваем  $x_n$  с  $c_s$ ; если  $x_n < c_s$ , то  $c_s$  полагаем равным  $x_n$ , и остается вычислить  $m = \min_{i=1}^{[n/2]} c_i$ , в противном случае вычисляем  $m$  по этой же формуле без предварительной замены  $c_s$ , а затем заменяем  $M$  значением  $\max(M, x_n)$ . Согласно предыдущему вероятность того, что  $x_n < c_s$ , равна  $\frac{1}{2} - \frac{1}{2n}$ . Сред-

нее число сравнений здесь равно  $\frac{3}{2}n + \frac{1}{2n} - 2$ . Этот оптимальный в среднем (по теореме 2) алгоритм обозначим  $P'$ .

Алгоритм  $P'$  можно описать без привлечения дополнительных массивов  $b_1, b_2, \dots$  и  $c_1, c_2, \dots$ , но описание вышло бы несколько более громоздким, чем описание всех других вариантов алгоритма  $P$  в этом параграфе.

Если отвлечься от способа соединения в пары элементов  $x_1, \dots, x_n$ , т. е. от выбора элементов для сравнений типов  $AA, BB, CC$ , то имеется ровно два оптимальных в среднем алгоритма:  $P'$  и  $P''$ ; последний предписывает при нечетном  $n$  сравнение  $x_n$  с  $b_s$ , где  $s$  — индекс наименьшего элемента в массиве  $c_1, c_2, \dots$ .

В худшем случае алгоритмы  $P'$  и  $P''$  требуют  $\lfloor 3n/2 \rfloor - 2$  сравнений и, таким образом, минимизируют не только среднее, но и максимальное число сравнений.

Итак, видно, что предложенный метод действительно может существенно помочь в доказательстве оптимальности в среднем алгоритма сортировки или выбора; по сравнению с задачей доказательства оптимальности для худшего случая, эта задача, как известно, является значительно более трудной.

**Примечание 3.** Собственно результат Пола, касающийся невозможности такого алгоритма одновременного нахождения наибольшего и наименьшего элементов массива, который требует во всех случаях менее  $\lfloor 3n/2 \rfloor - 2$  сравнений, можно было бы получить, рассматривая ту же самую функцию  $g(a, b, c, d) = \frac{3}{2}a - b - c$  и  $w_4$  для  $g$  относительно  $AA, BB, \dots$ . Те рассуждения, которые были детально проведены в доказательстве теоремы 2 относительно невозможности некоторого специального выбора индексов  $i, j$  для последующего удачного сравнения элементов с помощью  $BC$ , показывают, что  $w_4$  надо понимать в смысле максимума достигаемых значений.

В заключение мы дополнительно рассмотрим класс алгоритмов, близких по конструкции тривиальному алгоритму  $T$  одновременного нахождения наибольшего и наименьшего элементов массива.

Пусть уже найдены наибольший и наименьший элементы ( $M$  и  $m$ ) среди  $x_1, \dots, x_k, k < n$ . Если пренебречь главной идеей Пола и рассмотреть вместе с  $M$  и  $m$  только один элемент  $x_{k+1}$ , то после сравнения  $x_{k+1}$  с  $M$  может оказаться, что сравнивать  $x_{k+1}$  с  $m$  не нужно, так как  $x_{k+1} > M$ . Итак, тривиальный алгоритм, уточненный этим соображением (алгоритм  $T'$ ), в худшем случае потребует

$2n - 3$  сравнения, в лучшем же случае (когда  $\max(x_1, x_2) < x_3 < \dots < x_n$ ) сравнений будет  $n - 1$ .

В алгоритме  $T'$  каждый элемент  $x_{k+1}$  сравнивается сначала с наибольшим  $M$  из  $x_1, \dots, x_k$ , а потом, в случае надобности, — с наименьшим  $m$ . Можно было бы, наоборот, сравнивать  $x_{k+1}$  сначала с  $m$ , а потом с  $M$ . Можно чередовать в некотором порядке сравнения типа «сначала с  $M$ , а потом, в случае надобности, с  $m$ » со сравнениями типа «сначала с  $m$ , а потом, в случае надобности, с  $M$ ». Можно представить себе различные стратегии чередования: например, типы сравнений применяются по очереди или же тип изменяется лишь в тех случаях, когда очередной элемент оказалось достаточным сравнить с одним из значений — с  $M$  или  $m$ , в зависимости от того, какой тип сравнений применялся при его рассмотрении. Возможны и более сложные стратегии чередования, которые при выборе типа сравнений для очередного элемента  $x_{k+1}$  учитывают все результаты сравнений, произведенных при выборе наибольшего и наименьшего значений из  $x_1, \dots, x_k$ . Оказывается, однако, что ни одна из стратегий не дает алгоритма, близкого в среднем к  $P$ . Доказательство мы проведем, пользуясь традиционной техникой рекуррентных соотношений и производящих функций, достигая этим еще одной цели — демонстрации различия предложенного здесь метода и традиционного.

Предварительно разберемся с алгоритмом  $T'$ . Хотя на первый взгляд кажется, что среднее число сравнений будет полусуммой  $2n - 3$  и  $n - 1$  и, как следствие этого, что алгоритм  $T'$  в среднем не хуже алгоритма  $P$ , в действительности дело обстоит иначе. Для правильного подсчета среднего числа сравнений можно воспользоваться результатом, полученным Фостором и Стюартом при анализе алгоритма нахождения наибольшего элемента в массиве  $x_1, \dots, x_n$ . Показано (см. [6]), что если каждой перестановке  $x_1, \dots, x_n$  чисел  $1, 2, \dots, n$  сопоставить количество таких индексов  $t$ , что

$$2 \leq t \leq n, \quad x_t > \max_{i=1}^{t-1} x_i,$$

то среднее этой величины по всем перестановкам окажется равным  $\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = H_n - 1$ . Теперь и без рассмотрения деталей ясно, что для алгоритма  $T'$  среднее число сравнений есть  $2n - \ln n + O(1)$ , т. е. при больших  $n$  алгоритм  $T'$  даже в среднем значительно хуже  $P$ .

Перейдем к рассмотрению стратегий чередования.

**Теорема 3.** *Среднее число сравнений для любого алгоритма указанного вида совпадает со средним числом сравнений для  $T'$ .*

**Доказательство.** Применим технику, близкую технике Фостера и Стюарта. Обозначим через  $p_{nm}$  вероятность того, что, применяя выбранную стратегию чередования типов сравнений, при рассмотрении произвольной перестановки  $x_1, \dots, x_n$  придется выполнить ровно  $m$  сравнений. Тогда при  $m \geq n > 2$  должно иметь место соотношение

$$p_{nm} = fp_{(n-1)(m-1)} + (1-f)p_{(n-1)(m-2)},$$

где  $f$  — вероятность того, что последний элемент в перестановке больше максимального из всех предыдущих элементов (т. е. равен  $n$ ), если при рассмотрении последнего элемента стратегия предписывает сравнения типа «сначала с  $M$ , а потом, в случае надобности, с  $m$ », или что последний элемент меньше минимального из всех предыдущих элементов (т. е. равен  $1$ ), если стратегия предписывает сравнения другого типа. Начальные условия задаются в виде  $p_{2m} = \delta_{1m}$ ,  $\delta_{ij}$  — символ Кронекера.

**Лемма.** *Независимо от стратегии чередования типов сравнений  $f = 1/n$ .*

**Доказательство.** Надо показать, что результаты сравнений, произведенных при последовательном рассмотрении элементов  $x_1, \dots, x_{n-1}$ , не дают никакой информации о значении  $x_n$ .

Покажем, например, что  $1$  и  $n$  могут встретиться в качестве значения  $x_n$  с одинаковой вероятностью. Пусть в перестановке  $x_1, \dots, x_n$  последний элемент равен  $1$ . Рассмотрим перестановку  $x_1 - 1, \dots, x_{n-1} - 1, n$ . Результаты сравнений при рассмотрении ее первых  $n - 1$  элементов будут точно такими же, как для первых  $n - 1$  элементов перестановки  $x_1, \dots, x_n$ , но на последнем месте стоит уже не  $1$ , а  $n$ . Если  $x_n = n$ , то следует рассмотреть перестановку  $x_1 + 1, \dots, x_{n-1} + 1, 1$ . Это рассуждение нетрудно обобщить со случая чисел  $1, n$  на случай произвольных чисел  $v, w$ . Итак,  $f = 1/n$ .

Для завершения доказательства теоремы запишем имеющееся ракуррентное соотношение для вероятностей в виде

$$p_{nm} = \frac{1}{n} p_{(n-1)(m-1)} + \frac{n-1}{n} p_{(n-1)(m-2)}, \quad p_{2m} = \delta_{1m}. \quad (8)$$

Этим соотношением однозначно определяется среднее

число сравнений. Поскольку это соотношение имеет место для любого алгоритма обсуждаемого вида, все средние равны между собой. Алгоритм  $T'$  принадлежит к обсуждаемым алгоритмам, для него, как уже нам известно, среднее задается асимптотической формулой  $2n - \ln n + O(1)$ . Эта формула сохраняет силу для любого алгоритма обсуждаемого вида.

Нетрудно получить точное значение среднего. Для производящей функции (многочлена)  $G_n(z) = p_{n0} + p_{n1}z + \dots$  имеем соотношение, которое является следствием (8):

$$\begin{aligned} G_n(z) &= \frac{z}{n} G_{n-1}(z) + z^2 \frac{n-1}{n} G_{n-1}(z) = \\ &= \left( \frac{z}{n} + z^2 \frac{n-1}{n} \right) G_{n-1}(z). \end{aligned}$$

Среднее будет равно  $G'(1)$ ; имеем

$$G'_n(z) = \left( \frac{1}{n} + 2z \frac{n-1}{n} \right) G_{n-1}(z) + \left( \frac{z}{n} + z^2 \frac{n-1}{n} \right) G_{n-1}(z).$$

Учитывая, что  $G_{n-1}(1) = 1$ , получаем  $G'_n(1) = 2 - \frac{1}{n} + G'_{n-1}(1)$ ,  $G'_2(1) = p_{21} = 1$ . Следовательно,  $G'_n(1) = 2n - 2.5 - H$ .

## СПИСОК ЛИТЕРАТУРЫ

- 1 Андерсон Р. Доказательство правильности программ.— М.: Мир, 1982.
- 2 Вирт Н. Систематическое программирование: Введение.— М.: Мир, 1977.
- 3 Гинзбург С. Математическая теория контекстно-свободных языков.— М.: Мир, 1975.
- 4 Дейкстра Э. Дисциплина программирования.— М.: Мир, 1978.
- 5 Донаху Дж. Взаимодополняющие определения семантики языков программирования.— В кн.: Семантика языков программирования. М.: Мир, 1980, с. 222—234.
- 6 Кнут Д. Искусство программирования для ЭВМ. Т. 1.— М.: Мир, 1976.
- 7 Косовский Н. К. Частичная корректность почти в целом программ, специфицированных утверждениями.— В кн.: Применение методов математической логики. Таллин: Ин-т кибернетики АН ЭССР, 1983.
- 8 Лавров С. С. Методы задания семантики языков программирования.— Программирование, 1978, № 6, с. 3—10.
- 9 Лавров С. С. Основные понятия и конструкции языков программирования.— М.: Финансы и статистика, 1982.
- 10 Манна З. Теория неподвижной точки программы.— В кн.: Кибернетический сборник. Вып. 15. М.: Мир, 1978, с. 38—100.
- 11 Непомнящий В. А. Доказательство правильности программ линейной алгебры.— Программирование, 1982, № 4, с. 63—72.
- 12 Оре О. Теория графов.— М.: Наука, 1980.
- 13 Редько В. Н. Основания композиционного программирования.— Программирование, 1979, № 3, с. 3—13.
- 14 Цейтин Г. С. Некоторые черты языка для системы программирования, проверяющей доказательства.— В кн.: Теория программирования. Ч. II. Новосибирск: ВЦ СО АН СССР, 1972, с. 234—249.
- 15 Черпоброд Л. В. Верификация класса циклических программ без использования инвариантов циклов.— Программирование, 1984, № 2, с. 3—14.
- 16 Pohl I. A sorting problem and its complexity.— Comm. ACM, 1972, 15, № 6, p. 462—466.
- 17 Wand M. A new incompleteness result for Hoare's system.— Journal of the ACM, 1978, 25, № 1, p. 168—175.



## ОГЛАВЛЕНИЕ

Предисловие . . . . .	3
<b>Глава I. Программы и индуцируемые ими преобразования множеств функций (детерминированный случай)</b>	8
§ 1. Преобразования предикатов; сопряженные задачи	8
§ 2. Обобщение хооровских свойств и правил Хоора . .	15
§ 3. Обобщение слабейшего предусловия и слабейшего свободного предусловия . . . . .	23
§ 4. Не связанные с изменением состояния последствия выполнения программ . . . . .	36
§ 5. Структурное операционное смещение; доказательства хооровского типа . . . . .	48
§ 6. Неустраняемая погрешность вычислений, описываемых программой . . . . .	54
§ 7. Семантика Хоора и число этапов выполнения цикла . . . . .	66
<b>Глава II. О возможностях метода Хоора . . . . .</b>	74
§ 1. Импликации, связанные с циклами . . . . .	74
§ 2. Характер неполноты системы Хоора . . . . .	78
§ 3. Флойдовские индуктивные утверждения и хооровские инварианты . . . . .	83
<b>Глава III. Программы и индуцируемые ими преобразования множеств функций (общий случай)</b>	87
§ 1. Обобщение понятий; формулы алгебры бинарных отношений и их следствия . . . . .	87
§ 2. Непрерывные функционалы; инструкция ввода . . . . .	94
§ 3. Погрешность округления в вычислениях, описываемых программой . . . . .	100
§ 4. Вычисление предусловий относительно if и do . . . . .	104
§ 5. Анализ трудоемкости в среднем . . . . .	115
<b>Список литературы . . . . .</b>	127

45 коп.

5 BT 66  
A 160

