1958 г. июль—август *т. XIII, вып. 4 (82)*

УСПЕХИ МАТЕМАТИЧЕСКИХ НАУК

К ОПРЕДЕЛЕНИЮ АЛГОРИТМА[1])

А. Н. Колмогоров и В. А. Успенский

Алгоритмом принято называть систему вычислений, которая для некоторого класса математических задач из записи

A

«условий» задачи позволяет при помощи однозначно определенной последовательности операций, совершаемых «механически», без вмешательства творческих способностей человека, получить запись

B

«решения» задачи. Вопрос о путях уточнений этого традиционного приблизительного представления об алгоритмах с наибольшей широтой поставлен во вводных параграфах статьи А. А. Маркова [1]. Давая ясное и развернутое представление о существе вопроса, статья А. А. Маркова (и последовавшая за ней монография [2]) ограничивается алгоритмами, перерабатывающими «слова», а среди такого рода алгоритмов — теми, которые в статье названы «нормальными». Нашей целью является наметить возможность более широкого подхода к делу и вместе с тем более отчетливо показать, что самое общее доступное при современном состоянии науки понятие алгоритма вполне естественным образом связывается с понятием частично-рекурсивной функции.

---

[1]) Статья, кроме незначительных редакционных изменений, была написана, когда авторы, стремясь для самих себя осмыслить понятия «вычислимая функция» и «алгоритм», пытались пересмотреть найденные в литературе вопросы определения и окончательно убедиться в том, что за ними не скрывается каких-либо возможностей расширения самого объема понятия «вычислимая функция». Результат получился, естественно, отрицательным. Не без колебаний авторы публикуют отчет о своих поисках, так как предметно он не дает ничего нового по сравнению с более простыми определениями. По-видимому, однако, усвоенный нами подход к делу соответствует умонастроению многих математиков, которых публикация нашей статьи может избавить от повторного прохождения тех же путей сомнений и размышлений. Отметим еще, что обсуждался и такой вариант определений, в которых вместо пассивной, неограниченного объема памяти в машине могли возникать в неограниченном количестве параллельно работающие «активные области», каждая из которых сохраняет ограниченную сложность строения. Естественно, что и на этом пути не было получено ничего, кроме частично-рекурсивных функций.

Во всех интересующих математиков случаях доступные переработке данным алгоритмом записи условий A легко включаются в занумерованную неотрицательными целыми числами последовательность

$$A_0, A_1, A_2, \ldots, A_n, \ldots,$$

а записи могущих получиться решений B — в последовательность

$$B_0, B_1, B_2, \ldots, B_n, \ldots,$$

тоже занумерованную неотрицательными целыми числами[1]). Если обозначить через G множество номеров n тех условий A_n, которые алгоритм способен переработать в решения, то результат работы алгоритма, осуществляющего переработку

$$A_n \to B_m,$$

однозначно определяется заданной на G числовой функцией

$$m = \varphi(n).$$

Таким образом, произвольный алгоритм сводится к алгоритму вычисления значений некоторой числовой функции (*числа* всюду далее имеются целые неотрицательные).

Обратно, если для функции φ существует алгоритм, который, будучи применен к стандартной записи[2]) значения аргумента n из области определения функции φ, приводит к стандартной записи значения функции $m = \varphi(n)$, то функцию φ естественно называть алгоритмически вычислимой, или для краткости просто вычислимой функцией. Поэтому вопрос об определении алгоритма, по существу, равносилен вопросу об определении вычислимой функции. В § 1 дается обзор существующих определений вычислимой функции и алгоритма и обсуждается степень их общности и логической завершенности. В § 2 излагается новое определение алгоритма[3]). В § 3 показывается, что любой алгоритм, подпадающий под это новое, весьма общее по форме определение, все же сводится к алгоритму вычисления значений частично-рекурсивной функции. В приложении I дается сводка определений и фактов, относящихся к рекурсивным функциям, а в приложении II рассматривается пример алгоритма.

§ 1

Мы остановимся на следующих вариантах математического определения вычислимой функции или алгоритма:

---

[1]) Способ, позволяющий по виду записи находить ее номер, а также по номеру восстанавливать самую запись, является обычно весьма простым (так что существование алгоритма, «перерабатывающего» запись в номер, и алгоритма, «перерабатывающего» номер в запись, не вызывает сомнений).

[2]) Для чисел, больших нуля, стандартной записью можно считать запись вида $1+1+\ldots+1$ или запись по десятичной системе счисления и т. п. Без фиксирования стандартного способа записи чисел говорить об алгоритме вычисления $m = \varphi(n)$ по n не имеет смысла.

[3]) Основные черты этого определения, найденного еще в 1952 г., были впервые опубликованы в [3].

А) Определение вычислимой функции как функции, значения которой выводимы в некотором логическом исчислении (Гёдель [4], Чёрч [5] [1])).

Б) Определение вычислимой функции как функции, значения которой получаются при помощи исчисления λ-конверсии Чёрча [5], [7].

В) Определение вычислимой функции как функции частично-рекурсивной (см. работу Клини [8]) [2]) или — для случая всюду определенной функции — как общерекурсивной (Клини [10]). (Термины «частично-рекурсивная» и «общерекурсивная» понимаются здесь в смысле приложения I).

Г) Вычислительная машина Тьюринга [11] [3]).

Д) Финитный комбинаторный процесс Поста [13].

Е) Нормальный алгорифм А. А. Маркова [1], [2].

Рассмотрим прежде всего перечисленные определения с точки зрения естественно входящего в понятие алгоритма требования наличия однозначно определенной последовательности операций, «перерабатывающих» условия A в решение B, или значение n в значение $m = \varphi(n)$. Сразу ясно, что определения А), Б) и В) являются с этой точки зрения незавершенными, так как такой однозначно определенной последовательности операций не указывают. Правда, например, определение частично-рекурсивной функции, по существу, содержит в себе все предпосылки для построения однозначно определенного алгоритма вычисления ее значений по заданному значению ее аргумента, но в явном виде такой алгоритм не указывается.

Вычислительная машина Тьюринга производит вполне определенную последовательность операций, приводящую к последовательному развертыванию записей значений

$$\varphi(0),\ \varphi(1),\ \varphi(2),\ \ldots$$

Достаточно снабдить ее простым приспособлением, которое остановило бы ее работу при получении $\varphi(n)$ для заданного n, чтоби она могла рассматриваться как способ реализации подлинного алгоритма нахождения $\varphi(n)$ по заданному n [4]).

Определения Д) и Е) полностью отвечают требованию однозначной определенности алгоритмического вычислительного процесса заданием условий A.

Перейдем теперь к обсуждению рассматриваемых определений с точки зрения расчленения алгоритмического процесса на элементарные, могущие выполняться «механически» отдельные шаги. В этом требовании

---

[1]) По поводу определения А) см. также [6].

[2]) По поводу определения В) см. также [6].

[3]) Здесь имеется в виду первоначальное определение вычислимой функции посредством машины Тьюринга, предложенное самим Тьюрингом [11]. Близкое к этому изложение имеется в книге Петер [12].

[4]) В этом духе изложено вычисление функции на машине Тьюринга в монографии Клини [6]. Отправляясь от стандартной записи числа $\varphi(n)$, машина получает стандартную запись числа n и останавливается.

расчленения вычислительного процесса на элементарные шаги *ограниченной* (в пределах данного алгоритма) *сложности* мы усматриваем вторую, не менее существенную сторону понятия алгоритма. Чтобы сделать понятными основные трудности, связанные с уточнением понятия алгоритма в смысле выполнения требования ограниченной сложности его отдельных шагов, заметим, что интересующие математиков алгоритмы применимы обычно к *бесконечному* классу задач. Таковы, например, уже алгоритмы сложения и умножения записанных по десятичной системе натуральных чисел. Поэтому уподобление процесса предписываемых алгоритмом вычислений работе некоторой «машины» не может пониматься слишком буквально. Реальная машина допускает только конечное число четко разграниченных «состояний», и в соответствии с этим число различных «условий» A, которые реальная машина способна перерабатывать в «решения» B, неизбежным образом тоже конечно. Поэтому построение теории алгоритмов по образцу теории вычислительных машин требует во всяком случае некоторой идеализации понятия «машины».

Тем не менее наглядные образы и понятия, выработанные в связи с развитием теории вычислительных машин дискретного действия, могут быть полезны для общей ориентировки в проблеме рационального определения понятия алгоритма. Вся идеализация, необходимая для перехода от реальных вычислительных машин к математическим алгоритмам, заключается в допущении неограниченного объема «памяти» машины. Можно иллюстрировать изложение теории алгоритмов и более традиционными образами, связанными с вычислениями, записываемыми на бумаге. По сравнению с реальным процессом таких вычислений идеализация будет заключаться в допущении неограниченного объема записей, постепенно накапливаемых и вновь используемых по определенной системе в дальнейших выкладках.

Допуская неограниченный объем «памяти», понятие математического алгоритма должно сохранить из свойств реальных вычислительных машин дискретного действия и реальных письменных вычислений следующие два их свойства:

1. Сами вычислительные операции производятся дискретными шагами, причем на каждом шагу используется лишь *ограниченный* запас накопленных на предшествующих шагах данных. Если запас данных, имеющихся к началу шага, превосходит максимальную емкость «активной области», то они вовлекаются в эту активную область постепенно, по мере освобождения в ней места (и уже на следующих шагах). Можно, например, представлять себе дело так, что листки с записями, употребляемыми на каждом шагу вычислений, должны умещаться на столе и вмещают строго ограниченное число знаков, а папки, в которые складываются исписанные листки и из которых они извлекаются по мере надобности, имеют неограниченную толщину.

2. Неограниченность «объема памяти», или сохраняемых в запас записей, понимается чисто *количественным* образом: допускается лишь неограниченное накопление элементов (знаков, элементарных частей

машины) ограниченного числа типов, связанных между собой связями ограниченной сложности и тоже принадлежащих к ограниченному числу различных типов. Извлечение сведений, накопленных в «памяти» машины или в отложенных в сторону записях, производится последовательно: от попавшего в активную область элемента переходят к связанным с ним элементам и вовлекают их в активную область.

Высказанным сейчас требованиям в полной мере удовлетворяет вычислительная машина Тьюринга и финитный комбинаторный процесс Поста. В определениях А), Б) и В) остается нерасшифрованной действительная элементарность и «механическая» осуществимость таких операций, как подстановка вместо переменных тех или иных выражений, сложность которых (в смысле числа составляющих их элементарных знаков) правилами исчисления никак не ограничена. Аналогичное возражение формально может быть отнесено и к определению нормального алгорифма А. А. Маркова, хотя здесь возможность его снятия при помощи легкого дополнительного построения особенно ясна. Недостаточно расчлененной с точки зрения выдвинутого нами требования в нормальном алгорифме А. А. Маркова является только операция разыскания в перерабатываемом слове P «первого вхождения» слова X. Так как перерабатываемые слова P могут быть сколько угодно длинными, то нахождение первого вхождения в P данного слова X должно рассматриваться как процесс, требующий, вообще говоря, ряда последовательных операций. Можно сказать, что при определении нормальных алгорифмов А. А. Марков предполагает уже построенным некий специальный «алгорифм нахождений первых вхождений». Читатель может без большого труда при желании построить такой алгоритм в терминах нашего § 2.

Можно, однако, предполагать, что как А. А. Марков, так и авторы различных обсуждаемых нами определений понятия вычислимой функции оставляют некоторые операции неограниченного объема нерасчлененными лишь в силу очевидной возможности их расчленения на действительно элементарные шаги.

Последним и самым трудным вопросом является вопрос о достаточной *общности* предложенных определений алгоритма и вычислимой функции. На первый взгляд весьма общим является определение А). Однако эта общность обманчива. В самом деле, чтобы сформулировать определение вычислимой в смысле А) функции, мы неизбежно ограничиваемся некоторым фиксированным исчислением с фиксированными правилами вывода. Если же поставить вопрос о выводе в произвольном исчислении, с произвольными правилами вывода, то следует учесть, что понятие вывода в исчислении в самом общем виде может быть уточнено лишь на базе уже уточненного понятия алгоритма.

Что касается вполне четких с формальной стороны определений Б), В), Г), Д), Е), то можно сказать, что в известном смысле слова они все эквивалентны друг другу, т. е., по существу, определяют классы вычислительных процессов (алгоритмов) одинаковой мощности. Эквивалентность определений Б), В) и Е) установлена в работах [5], [14],

[15], [16][1]). Определения Д) и Е) являются специальными случаями алгоритмов в смысле нашего § 2, а эквивалентность нашего определения определению В) выясняется в § 3 нашей работы. Заметим, впрочем, что самая точная формулировка предложений об эквивалентности определений, построенных с формальной стороны столь различным образом и определяющих формально разные объекты (вычислимые функции; в случае Тьюринга — лишь вычислимые функции, принимающие только два значения, 0 и 1; алгоритмы, перерабатывающие слова, и т. д.), представляет некоторые затруднения.

Как бы то ни было, именно ряд установленных, или подразумеваемых, предложений об эквивалентности всех предлагавшихся, начиная с 1936 г., определений алгоритма или вычислимой функции является эмпирической основой сложившегося к настоящему времени общего убеждения в окончательности полученных определений. В применении к вычислимым функциям (для определенности от натурального аргумента с натуральными значениями) сложившиеся к настоящему времени общепринятые представления могут быть изложены так:

Класс алгоритмически вычислимых числовых функций, определенных для всех натуральных n, совпадает с классом *общерекурсивных функций* (определение этих последних см. приложение I). Что касается функций, определенных лишь на некотором подмножестве G полного множества натуральных чисел, то их алгоритмическое задание можно понимать в двух различных смыслах. При первом, более узком понимании предписанная алгоритмом процедура должна для любого натурального n после конечного числа шагов привести либо к нахождению $m = \varphi(n)$, либо к установлению того, что n не входит в G. При втором, более широком понимании дела требуется только, чтобы предписанная алгоритмом процедура для любого n из G приводила к верному результату $m = \varphi(n)$, а при попытке применить ее к натуральному n, не входящему в G, не могла кончиться установлением ошибочного (так как φ вне G не определена) результата о равенстве $\varphi(n)$ какому-либо определенному натуральному числу m. В остальном при втором (широком) понимании дела от характера работы алгоритма, примененного к невходящему в G числу n, ничего не требуется: допускаются в этом случае как безрезультатная «остановка» алгоритмической процедуры после конечного числа шагов, так и случай, когда алгоритмическая процедура продолжается неограниченно, не приводя ни к какому результату.

Целесообразно (что и делается в дальнейшем изложении) считать основным второе (широкое) понимание алгоритмической вычислимости числовой функции, определенной на множестве натуральных чисел. В настоящее время это широкое понятие алгоритмически вычислимой функции идентифицируется с понятием *частично-рекурсивной функции* (см. приложение I). Возможными областями определения G алгоритмически вычислимых функций $\varphi(n)$ оказываются тогда *рекурсивно-перечислимые* мно-

[1]) См. также примечание переводчика на стр. 341 книги [6].

жества натуральных чисел (определение последних см. приложение 1)[1]). Соответствующим образом строится и общее понятие алгоритма, предназначенного для переработки «условий» A того или иного более общего вида в «решения» B. Для любого алгоритма Г класс $\mathfrak{A}(\Gamma)$ формально допустимых записей «условий» A и класс $\mathfrak{B}(\Gamma)$ возможных записей «решений» B определяется таким образом, чтобы принадлежность или непринадлежность к ним любой могущей встретиться записи могла без затруднений распознаваться. Но при этом не требуется, чтобы алгоритм Г, примененный к любой записи A класса $\mathfrak{A}(\Gamma)$, приводил после конечного числа шагов к записи B из класса $\mathfrak{B}(\Gamma)$. Вместо этого допускают как возможность остановки алгоритмической процедуры без получения решения, так и возможность бесконечного ее продолжения. Множество $\mathfrak{G}(\Gamma)$ тех записей A из $\mathfrak{A}(\Gamma)$, которые алгоритм Г перерабатывает в «решения», т. е. в записи B из $\mathfrak{B}(\Gamma)$, будем называть *областью применимости алгоритма*.

Наиболее интересные результаты теории алгоритмов относятся как раз к алгоритмам с нетривиальной областью применимости; основные получ нные до настоящего времени результаты о *невозможности* алгоритмов могут быть приведены к такому виду: для некоторого алгоритма Г доказывается невозможность «разрешающего» алгоритма Ψ, который «распознавал» бы принадлежность или непринадлежность любой записи A из $\mathfrak{A}(\Gamma)$ к $\mathfrak{G}(\Gamma)$[2]).

Классы $\mathfrak{A}(\Gamma)$ и $\mathfrak{B}(\Gamma)$ для всех имеющихся определений алгоритма легко нумеруются, как указано во введении к этой статье. Это позволяет каждому алгоритму Г поставить в соответствие числовую функцию φ_Γ, которая определена для номеров n условий A, принадлежащих $\mathfrak{G}(\Gamma)$, и в качестве значения $m = \varphi(n)$ имеет номер решения, получаемый при переработке условия с номером n. Сложившееся к настоящему времени общее убеждение степени общности понятия алгоритма можно резюмировать, высказав, что для любого алгоритма Г функция φ_Γ должна быть частично-рекурсивной. Так и обстоит дело со всеми предложенными до настоящего времени определениями. Нашей задачей является возможно более полное разъяснение причин такого положения вещей. С этой целью мы и предприняли попытку построения возможно более общего *по форме* определения алгоритма, удовлетворяющего выдвинутым выше требованиям *ограниченной сложности* каждого шага алгоритма. В результате, как и следовало ожидать, получилось, что, несмотря на все меры, принятые для сохранения возможной общности построения, мы не выходим за пределы алгоритмов, описываемых указанным выше способом частично-рекурсивными функциями.

---

[1]) Изложение некоторых основных фактов теории вычислимых функций предполагающее лишь интуитивное представление об алгоритмах, дано в статье [17].

[2]) То есть невозможность алгоритма Ψ, который перерабатывал бы любую запись A из $\mathfrak{A}(\Gamma)$ в 1, когда A входит в $\mathfrak{G}(\Gamma)$, и в 0, когда A не входит в $\mathfrak{G}(\Gamma)$.

§ 2

Алгоритм Г задается при помощи предписания, указывающего способ перехода от «состояния» S процесса вичислений к «следующему» состоянию S^*, т. е. при помощи оператора $\Omega_\Gamma(S) = S^*$, который мы назовем оператором «непосредственной переработки». В классе $\mathfrak{S}(\Gamma) = \{S\}$ возможных состояний выделяются класс $\mathfrak{A}(\Gamma)$ «исходных» состояний, представляющих собой записи «условий» задач, и класс $\mathfrak{C}(\Gamma)$ «заключительных» состояний. При получении состояния из класса $\mathfrak{C}(\Gamma)$ алгоритмический процесс заканчивается, и из полученного «заключительного» состояния извлекается «решение». Область $\mathfrak{D}(\Gamma) \subseteq \mathfrak{S}(\Gamma)$, на которой оператор Ω определен, естественно считать не пересекающейся с $\mathfrak{C}(\Gamma)$. Класс состояний, не входящих ни в $\mathfrak{C}(\Gamma)$, ни в $\mathfrak{D}(\Gamma)$, обозначим через $\mathfrak{N}(\Gamma)$.

Алгоритмический процесс

$$\begin{aligned} S^0 &= A \in \mathfrak{A}(\Gamma), \\ S^1 &= \Omega_\Gamma(S^0), \\ S^2 &= \Omega_\Gamma(S^1), \\ &\cdots\cdots\cdots \\ S^{t+1} &= \Omega_\Gamma(S^t) \end{aligned}$$

может развиваться одним из следующих трех способов:

1. При каком-либо $t = \bar{t}$ процесс приводит к заключительному состоянию

$$S^{\bar{t}} \in \mathfrak{C}(\Gamma),$$

после чего из этого заключительного состояния извлекается решение B.

2. При каком-либо $t = \bar{t}$ процесс заканчивается безрезультатно:

$$S^{\bar{t}} \in \mathfrak{N}(\Gamma).$$

3. Процесс продолжается неограниченно, не приводя к результату.

Класс $\mathfrak{G}(\Gamma)$ тех $A \in \mathfrak{A}(\Gamma)$, которые приводят к первому типу течения процессов, называется «областью применимости» алгоритма.

Перейдем теперь к уточнению понятий «состояния» и «активной части» состояния. Задачей тех рассмотрений, которые сейчас будут изложены, является лишь наглядное оправдание достаточной общности того формального определения этих понятий, которое будет дано несколькими страницами дальше.

Естественно считать, что состояние S определяется наличием некоторого конечного числа элементов

$$\bigcirc_1, \bigcirc_2, \ldots, \bigcirc_v,$$

принадлежащих каждый к одному из типов

$$T_0, T_1, T_2, \ldots, T_n,$$

и наличием между некоторыми группами элементов связей, принадлежащих к одному из типов

$$R_1, R_2, \ldots, R_m.$$

Для каждого типа связи R_i будем считать фиксированным число k_i связываемых элементов. Запас типов элементов и запас типов связей (а следовательно, и числа n, m, k_1, ..., k_m) выбираются для данного алгоритма раз навсегда.

Элементы будем обозначать кружками с проставляемыми *внутри* номерами типов. Что касается индексов *около* кружков, то они будут относиться лишь к нашим рассуждениям относительно алгоритма и к составу состояния не относятся. Так как число элементов, образующих вместе с заданными между ними связями состояние, неограниченно, то и разнообразие этих внешних индексов заранее никак не ограничивается: можно, например, в качестве этих индексов употреблять сколь угодно большие натуральные числа.

Наконец, в соответствии со сказанным ранее будем считать, что в пределах одного алгоритма число связей, в которые может одновременно (т. е. при конструировании одного определенного состояния) входить один и тот же элемент, ограничено раз навсегда выбранным числом.

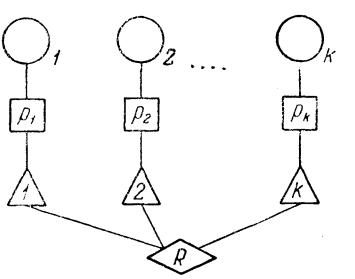
Вообще говоря, порядок связываемых какой-либо связью элементов может быть существенным или несущественным. Далее нам удастся ограничиться рассмотрением только симметрических связей между парами элементов. Однако сначала, для того чтобы не упустить каких-либо возможностей построения мощных алгоритмов, мы предположим, что в понятие состояния существенно входит порядок, в котором связанные какой-либо связью элементы в эту связь включаются, т. е., например, наличие парной связи

$$R(\bigcirc_1, \bigcirc_2)$$

будем отличать от наличия парной связи

$$R(\bigcirc_2, \bigcirc_1).$$

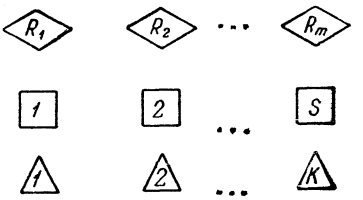
Будем также считать существенно входящим в понятие состояния и задание определенного упорядочения всех связей, в которые входит данный элемент. Таким образом, связывание элементов $\bigcirc_1$, $\bigcirc_2$, ..., $\bigcirc_k$ связью типа R будем мыслить себе осуществляющимся по схеме



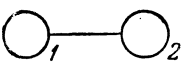
В этой схеме индексы p_j указывают, какое место занимает рассматриваемая связь типа R в упорядоченной последовательности связей, в которые входит элемент $\bigcirc_j$. Индексы p_j при этом можно считать принимающими только значения 1, 2, ..., s. Что касается индексов 1, 2, ..., k,

то они принимают, вообще говоря, значения из последовательности 1, 2, ..., K, где K равно максимуму чисел k_i ($i = 1, 2, \ldots, m$).

Вместо рассмотрения состояний, составляемых из элементов, связанных по указанной сейчас общей схеме разнообразными связями $R_1, R_2, \ldots, R_m$, целесообразно ввести дополнительные типы элементов



Очевидно, что это позволит заменить рассматриваемый алгоритм новым, отличающимся лишь способом записи состояний, в котором будет применяться только один тип симметричных связей между парами элементов. Такую связь между элементами $\bigcirc_1$ и $\bigcirc_2$ можно изображать просто чертой

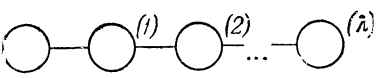


Для преобразованного алгоритма автоматически будет выполняться

Условие α). *Все элементы, связанные с каким-либо одним элементом, принадлежат различным типам.*

Естественно, что при условии α) специальная нумерация порядка вхождения элемента в различные связи излишня: связи, в которые входит данный элемент, автоматически нумеруются номерами типов тех элементов, с которыми он связывается.

Можно без потери общности аналогичным образом стандартизировать и способ выделения «активной части» состояния, строение которой однозначно определяет те преобразования, которые переводят S в $S^* = \Omega(S)$. Именно, будем считать, что в активной части состояния имеется отмеченный «начальный» элемент. Требование ограниченной сложности активной части состояния естественно понять так, что все входящие в нее элементы должны быть связаны с начальным элементом $\bigcirc$ цепями



ограниченной длины $\lambda \leqslant N$, где N — характеристическое число, постоянное для данного алгоритма.

Оператор

$$S^* = \Omega_\Gamma(S),$$

дающий переход от состояния S к «следующему» состоянию S^*, должен быть таков, что S^* строится исключительно на основании информации о виде активной части S. Точно так же на основании информации о виде активной части S должно распознаваться, достигнуто заключительное

состояние или нет. Проще всего, что очевидным образом не нарушит общности, использовать для этого начальный элемент: мы примем, что к типам T_0 и T_1 может относиться только начальный элемент, причем переход начального элемента от типа T_0 к типу T_1 означает получение заключительного состояния.

Вовсе не все заключительное состояние целесообразно считать решением. Так, при прекращении вычислений на бумаге не все написанное является решением: подавляющую часть обычно составляют промежуточные выкладки. Точно так же при остановке вычислительной машины только сравнительно небольшая часть полного «состояния» машины является решением решаемой на машине задачи. Естественно поэтому считать «решением» лишь некоторую часть «заключительного состояния», связанную с начальным элементом. Мы примем в качестве «решения» совокупность всех тех элементов, которые можно связать с начальным элементом цепями произвольной длины.

Переходим теперь к формулировке определения алгоритма.

1. Алгоритм Г предполагает наличие упорядоченного множества $\mathfrak{T} = \mathfrak{T}(\Gamma)$ неограниченных по объему классов

$$T_0,\ T_1,\ \ldots,\ T_\Gamma$$

«элементов», из которых будут строиться «состояния». Классы эти не пересекаются между собой. Соединение их обозначается T. Элементы T обозначаются кружками ○ с различными индексами около кружков. Для обозначения принадлежности элемента ○ к классу T_i внутри кружка ставится номер i. Элемент, принадлежащий классу T_i, называется также элементом типа T_i.

2. Комплексом над множеством $\mathfrak{T}$ называется обычный одномерный комплекс с вершинами из T, т. е. соединение $K = K_0 \cup K_1$ конечного множества

$$K_0 = \{○_\alpha\}$$

некоторых элементов из T, называемых «вершинами» комплекса, и множества K_1 (очевидно, тоже конечного)

$$K_1 = \left\{ ○_{\alpha'} \text{——} ○_{\alpha''} \right\}$$

некоторых пар элементов из K_0. Эти пары называются «отрезками» комплекса K.

3. Выделяется множество $\mathfrak{S}_\mathfrak{T}$ таких комплексов K над $\mathfrak{T}$, которые обладают следующими свойствами:

а) вершины, соединенные отрезками с фиксированной вершиной, принадлежат разным типам T_i;

б) в K существует одна и только одна вершина типа T_0 или T_1, называемая «начальной»; остальные вершины принадлежат типам T_i с $i \geqslant 2$.

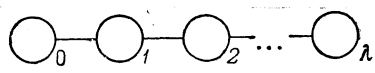
4. В множестве $\mathfrak{S}_\Sigma$ выделяются подмножество $\mathfrak{A}_\Sigma$ комплексов, начальная вершина которых принадлежит типу T_0, и подмножество $\mathfrak{C}_\Sigma$ комплексов, начальная вершина которых принадлежит типу T_1.

5. Комплексы из $\mathfrak{S}_\Sigma$ считаются «состояниями» алгоритма Г, причем комплексы из $\mathfrak{A}_\Sigma$ — «исходными» состояниями, или «условиями», а комплексы из $\mathfrak{C}_\Sigma$ — «заключительными» состояниями:

$$\mathfrak{S}(\Gamma) = \mathfrak{S}_\Sigma, \quad \mathfrak{A}(\Gamma) = \mathfrak{A}_\Sigma, \quad \mathfrak{C}(\Gamma) = \mathfrak{C}_\Sigma.$$

6. Активной частью $U(S)$ состояния S называется подкомплекс комплекса S, состоящий из вершин и отрезков, принадлежащих цепям длины $\lambda \leqslant N$, содержащим начальную вершину. Здесь N — произвольное, но для данного алгоритма Г фиксированное число.

Термин «подкомплекс» поднимается в обычном теоретико-множественном смысле: комплекс K' есть подкомплекс комплекса K, если $K'_0 \subseteq K_1$, $K'_0 \subseteq K_1$. Цепь есть комплекс вида



Длиной цепи называется число λ входящих в нее отрезков. Комплекс называется связным, если любые две его вершины можно соединить цепью.

7. Внешней частью $V(S)$ состояния S называется подкомплекс, состоящий из вершин, не соединимых с начальной вершиной цепями длины $\lambda < N$, и отрезков, входящих в те цепи, содержащие начальную вершину, которые имеют длину $\lambda \leqslant N$. Пересечение

$$L(S) = U(S) \cap V(S)$$

называется границей активной части состояния S. Легко видеть, что $L(s)$ есть нульмерный комплекс (т. е. не содержит отрезков) и состоит из тех вершин, которые соединимы с начальной цепями длины $\lambda = N$, но не соединимы более короткими цепями (длины $\lambda < N$).

Прежде чем закончить формулировку определения алгоритма, сделаем несколько замечаний.

Сформулированное в § 1 требование 1 мы будем понимать так: оператор

$$\Omega_\Gamma(S) = S^*$$

должен быть таков, чтобы перестройка S в S^* требовала лишь перестройки в какой-либо новый комплекс активной части $U(S)$ и не меняла строения внешней части $V(S)$; при этом сам характер перестройки $U(S)$ должен определиться исключительно строением самого комплекса $U(S)$.

Два комплекса K' и K'' будем считать *изоморфными*, если их (как множества $K' = K'_0 \cup K'_1$ и $K'' = K''_0 \cup K''_1$) можно поставить во взаимно однозначное соответствие так, что

1) вершины соответствуют вершинам, а отрезки — отрезкам,

2) отрезку, соединяющему вершины $\bigcirc'_\alpha$ и $\bigcirc'_\beta$, соответствует отрезок, соединяющий соответствующие вершины $\bigcirc''_\alpha$ и $\bigcirc''_\beta$,

3) соответствующие вершины имеют одинаковый тип.

Очевидно, что число неизоморфных активных частей $U(S)$, возможных в данном алгоритме, ограничено. Правила их переработки поэтому легко формулируются в конечном виде. Возвращаемся к изложению определения алгоритма.

8. Правила непосредственной переработки алгоритма задаются при помощи указания пар состояний

$$U_1 \to W_1,$$
$$U_2 \to W_2.$$
$$\ldots\ldots$$
$$U_r \to W_r$$

с установленным для каждого i изоморфным отображением φ_i подкомплекса $L(U_i)$ на некоторый подкомплекс $\tilde{L}(W_i)$ состояния W_i. Если вершины $\bigcirc_\mu \in L(U_i)$ и $\bigcirc_\nu \in \tilde{L}(W_i)$ соответствуют друг другу при изоморфизме φ_i, то произвольная не начальная вершина комплекса U_i, связанная с $\bigcirc_\nu$, имеет тип одной из вершин подкомплекса $L(W_i)$, связанных с $\bigcirc_\mu$ [1]). Каждый комплекс U_i является состоянием класса $\mathfrak{A}(\Gamma)$ (условием), удовлетворяющим требованию $U(U_i) = U_i$. Все $U_1, U_2, \ldots, U_r$ предполагаются не изоморфными между собой. Что же касается W_i, то это — произвольные состояния, которые могут быть как исходными, так и заключительными.

9. Область $\mathfrak{D}(\Gamma)$ определения оператора $\Omega_\Gamma(S) = S^*$ состоит из тех состояний S, для которых активная часть $U(S)$ изоморфна одному из комплексов $U_1, U_2, \ldots, U_r$.

10. Пусть подкомплекс $U(S)$ изоморфен U_i. Так как $U(S)$ и U_i суть связные комплексы, принадлежащие к $\mathfrak{S}_\Sigma$, то, если они изоморфны, между ними может существовать лишь одно изоморфное соответствие. Это соответствие однозначно определяет изоморфизм между $L(S)$ и $L(U_i)$, а так как задано изоморфное отображение φ_i комплекса $L(U_i)$ на $\tilde{L}(W_i)$, то тем самым индуцируется однозначно определенный изоморфизм θ_i^S между $L(S)$ и $\tilde{L}(W_i)$.

11. Пусть $S \in \mathfrak{D}(\Gamma)$, причем подкомплекс $U(S)$ изоморфен комплексу U_i. Очевидно, можно образовать комплекс $\tilde{W}$, изоморфный комплексу W_i и удовлетворяющий следующим условиям:

1) $\tilde{W} \cap V(S) = L(S)$;

2) изоморфизм θ_i^S между подкомплексами $L(S) \subseteq \tilde{W}$ и $\tilde{L}(W_i) \subseteq W_i$ продолжается до изоморфизма между комплексами $\tilde{W}$ и W_i.

Результат применения оператора Ω_Γ к комплексу S определяется (однозначно с точностью до изоморфизма) как комплекс вида

$$S^* = \tilde{W} \cup V(S).$$

12. Если S — заключительное состояние, то связная компонента начальной вершины считается «решением» (под связной компонентой какой-либо вершины понимается максимальный связный подкомплекс, содержащий эту вершину); совокупность связных компонент начальных

[1]) Вследствие этого требования свойство а) из п. 3 определения не нарушится в процессе работы алгоритма.

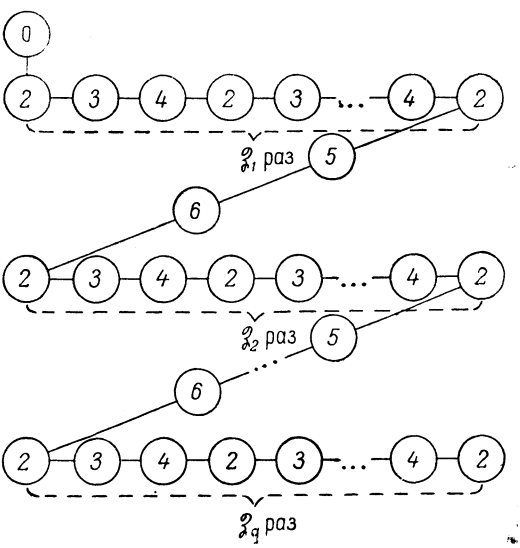
вершин заключительных состояний может быть, в соответствии с ранее сказанным, обозначена через $\mathfrak{B}(\Gamma)$.

Теперь определение алгоритма заканчивается точно так, как указано в начале этого параграфа (в отношении трех типов течения алгоритмического процесса и определения и «области применимости» $\mathfrak{C}(\Gamma)$.

Заданный указанным способом алгоритм, характеризующийся упорядоченным множеством $\mathfrak{T}$ и числом N, мы отнесем к классу $\mathbf{A}_N(\mathfrak{T})$. Всякий алгоритм класса $\mathbf{A}_N(\mathfrak{T})$ мы отнесем к классу $\mathbf{A}(\mathfrak{T})$.

Для того чтобы сделать описание алгоритма более обозримым, мы ввели некоторые условности, которые не связаны неразрывно с общим замыслом, но нам кажется, что достаточная общность предложенного определения остается убедительной, несмотря на эти условности. Нам представляется убедительным, что произвольный алгоритмический процесс удовлетворяет нашему определению алгоритма. Хочется при этом подчеркнуть, что речь идет не о сводимости любого алгоритма к алгоритму в смысле нашего определения (так, в следующем параграфе будет устанавливаться сводимость любого алгоритма к алгоритму вычислений частично-рекурсивной функции), а о том, что любой алгоритм по существу подходит под предложенное определение.

Трудности, могущие возникнуть у читателя, если он попытается представить, скажем, вычисление значения частично-рекурсивной функции в виде определенного выше алгоритма над комплексами, обусловлены лишь тем, что, как мы уже отмечали в § 1, схема вычисления значений частично-рекурсивной функции не задана непосредственно в виде алгоритма. Если же развернуть эти вычисления в виде алгоритмического процесса (что, конечно, нетрудно сделать), то тем самым автоматически получится некоторый алгоритм в смысле предложенного определения. Сформулируем высказанное утверждение более точно. Рассмотрим множество типов $\mathfrak{T}_6 = \{T_0, T_1, T_2, T_3, T_4, T_5, T_6\}$. Назовем «$A$-изображением» системы q чисел $(\delta_1, \delta_2, \ldots, \delta_q)$ комплекс $A_{\delta_1, \delta_2, \ldots, \delta_q}$:



а «B-изображением» той же системы — комплекс $B_{\delta_1, \delta_2, \ldots, \delta_q}$, получающийся из $A_{\delta_1, \delta_2, \ldots, \delta_q}$ заменой вершины ⓪ на вершину ①.

Для каждой частично-рекурсивной функции $f(x_1, \ldots, x_q)$ существует такое множество типов $\mathfrak{T} \supseteq \mathfrak{T}_6$ и такой алгоритм Γ типа $\mathbf{A}(\mathfrak{T})$, который применим к комплексу $A_{\delta_1, \delta_2, \ldots, \delta_q}$ тогда и только тогда, когда f определена для системы чисел $\delta_1, \delta_2, \ldots, \delta_q$, и в этом последнем случае перерабатывает этот комплекс в комплекс B_t, где

$$t = f(\delta_1, \delta_2, \ldots, \delta_q).$$

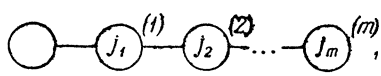
Все сказанное о рекурсивных функциях полностью относится и ко всем определениям А) — Е). Как только из любого из этих определений удается извлечь алгоритмический процесс, он, этот процесс, оказывается алгоритмом в указанном выше смысле.

§ 3

Всякий алгоритм сводится к вычислению значений некоторой частично-рекурсивной функции. Это доказывается так называемым методом арифметизации.

Прежде всего, заметим, что каждый комплекс $S_{\ldots}$ из $\mathfrak{S}_{\mathfrak{T}}$ можно задавать в виде таблицы. Для этого упорядочим множество вершин комплекса S следующим образом:

Каждой цепи



соединяющей начальную вершину ○ с произвольной вершиной $\bigcirc^{(m)}$, поставим в соответствие строчку

$$j_1, j_2, \ldots, j_m.$$

В силу условия а), установленное соответствие между исходящими от начальной вершины ○ цепями вершины комплекса S и некоторым множеством строчек натуральных чисел будет взаимно однозначным. Если упорядочить строчки чисел в словарном порядке, то автоматически возникает порядок среди цепей, исходящих от начальной вершины. Каждой вершине из S поставим теперь в соответствие минимальную в смысле установленного порядка цепь, соединяющую начальную вершину с выбранной нами; это соответствие индуцирует порядок в множестве вершин (начальная вершина будет при этом первой). Таким образом, множество вершин каждого комплекса из S оказывается упорядоченным.

Естественно отнести каждому комплексу из $\mathfrak{S}_{\mathfrak{T}}$ симметричную таблицу,

	i_1	i_2	...	i_v
i_1	α_{11}	α_{12}	...	α_{1v}
i_2	α_{21}	α_{22}	...	α_{2v}
⋮	⋮	⋮	⋱	⋮
i_v	α_{v1}	α_{v2}	...	α_{vv}

в которой i_k есть номер типа k-й (в смысле установленного порядка) вершины из S, а α_{kl} есть 1 или 0 в зависимости от того, соединены или нет k-я и l-я вершины. По таблице комплекс восстанавливается однозначно с точностью до изоморфизма.

Проведенное построение переводит всякий алгоритм класса $\mathbf{A}(\mathfrak{T})$ на язык таблиц.

Каждой таблице поставим в соответствие ее «гёделевское число», для чего запишем таблицу в строчку

$$\alpha_{11}\alpha_{12}\ldots\alpha_{1v}\alpha_{21}\alpha_{22}\ldots\alpha_{2v}\ldots\alpha_{v1}\alpha_{v2}\ldots\alpha_{vv},$$

а строчке отнесем число

$$p_1^{i_1}p_2^{i_2}\ldots p_v^{i_v}p_{v+1}^{\alpha_{11}}p_{v+2}^{\alpha_{12}}\ldots p_{v^2+v}^{\alpha_{vv}},$$

где p_r есть r-е простое число.

В натуральном ряду возникает бесконечное подмножество Q тех чисел, которые суть «гёделевские числа» таблиц, отвечающих комплексам из $\mathfrak{S}_{\mathfrak{T}}$. Принадлежность или непринадлежность числа к Q распознается без труда. Перенумеруем элементы Q в естественном порядке натуральными числами. *Номером* таблицы назовем номер соответствующего ей «гёделевского числа». *Номером* комплекса из $\mathfrak{S}_{\mathfrak{T}}$ назовем номер представляющей этот комплекс таблицы.

По комплексу S из $\mathfrak{S}_{\mathfrak{T}}$ эффективно и однозначно находится его номер s, а по номеру s эффективно и однозначно (с точностью до изоморфизма) восстанавливается самый комплекс S.

Мы имеем здесь дело с той ситуацией, о которой говорилось в § 1. Множество $\mathfrak{S}_{\mathfrak{T}}$ нумеруется натуральными числами

$$S_0,\ S_1,\ S_2,\ \ldots,\ S_p,\ \ldots,$$

в эту последовательность комплексов включаются как множество $\mathfrak{A}_{\mathfrak{T}}$ «условий», так и множество $\mathfrak{B}_{\mathfrak{T}}$ «решений». Каждый алгоритм Γ типа $\mathbf{A}(\mathfrak{T})$ задает отображение своей области применимости $\mathfrak{S}(\Gamma)$ в множество $\mathfrak{B}_{\mathfrak{T}}$; это отображение индуцирует числовую функцию

$$m = \varphi_\Gamma(n),$$

заданную на множестве $G(\Gamma)$ тех номеров n, для которых $S_n \in \mathfrak{S}$. Мы покажем, что функция φ_Γ частично-рекурсивна.

В самом деле, нетрудно показать, что функция $s^* = \sigma(s)$, индуцированная на множестве номеров оператором непосредственной переработки $S^* = \Omega_\Gamma(S)$, является примитивно-рекурсивной (определение примитивно-рекурсивных функций см. приложение I). Отсюда сразу следует примитивная рекурсивность функции $\rho(n, t)$ дающей номер комплекса S^t, получающегося на t-м шагу переработки комплекса с номером n. Процесс продолжается до тех пор, пока комплекс $S^{\bar{t}}$, возникший на шаге $\bar{t}$, не окажется «заключительным», т. е. комплексом из $\mathfrak{E}(\Gamma)$. Номера комплексов из $\mathfrak{E}(\Gamma) = \mathfrak{E}_{\mathfrak{T}}$, т. е. номера комплексов, начальная вершина которых принадлежит классу T_1, удовлетворяют уравнению

$$\gamma(x) = 0,$$

где функцию γ, как легко показать, всегда можно выбрать из числа примитивно-рекурсивных. Процесс, таким образом, продолжается до того первого t, которое будет удовлетворять условию

$$\gamma(\rho(n, t)) = 0,$$

т. е. до числа

$$\bar{t} = \mu t\,[\gamma(\rho(n, t)) = 0]$$

(об операторе μ см. приложение I). Комплекс $S^{\bar{t}}$ принадлежит $\mathfrak{E}(\Gamma)$, его номер $\rho(n, \bar{t})$ находится из равенства

$$\rho(n, \bar{t}) = \rho(n, \mu t\,[\gamma(\rho(n, t)) = 0]).$$

Номер $m = \varphi_\Gamma(n)$ решения получается из номера комплекса $S^{\bar{t}}$ при помощи функции β (тоже примитивно-рекурсивной), дающей по номеру комплекса из $\mathfrak{S}_{\mathfrak{T}}$ номер связной компоненты его начальной вершины. Именно

$$\varphi_\Gamma(n) = \beta(\rho(n, \mu t\,[\gamma(\rho(n, t)) = 0])), \tag{1}$$

откуда следует (см. приложение I), что функция $\varphi_\Gamma(n)$ частично-рекурсивная.

Отсюда вытекает, в частности, что функция $f(x_1, \ldots, x_q)$, задаваемая алгоритмом, преобразующим A-изображения наборов из q чисел в B-изображения чисел, частично-рекурсивна.

Действительно, можно построить примитивно-рекурсивные функции ξ и η, удовлетворяющие следующим требованиям:

1) для всякого набора чисел $\mathfrak{z}_1, \ldots, \mathfrak{z}_q$ число $\xi(\mathfrak{z}_1, \ldots, \mathfrak{z}_q)$ есть номер A-изображения набора $\mathfrak{z}_1, \ldots, \mathfrak{z}_q$;

2) для всякого числа n, являющегося номером B-изображения числа t, значение $\eta(n)$ равно t.

Если теперь обозначить через Г алгоритм, осуществляющий вычисление функции f, то, очевидно,

$$f(x_1, \ldots, x_q) = \eta(\varphi_\Gamma(\xi(x_1, \ldots, x_q))), \tag{2}$$

откуда и получается, что f — частично-рекурсивная функция.

Поскольку для каждой частично-рекурсивной функции f существует вычисляющий ее (в смысле, указанном в конце § 2) алгоритм Г, то каждая частично-рекур-

сивная функция будет допускать представление (2). Используя формулу (1), получаем:

$$f(x_1, \ldots, x_q) = \eta(\beta(\rho(\xi(x_1, \ldots, x_q), \mu t[\gamma(\rho(\xi(x_1, \ldots, x_q), t)) = 0]))).$$

Полагая для сокращения

$$\eta(\beta(\rho(\xi(x_1, \ldots, x_q), v))) = \pi(x_1, \ldots, x_q, v),$$
$$\gamma(\rho(\xi(x_1, \ldots, x_q), t)) = \tau(x_1, \ldots, x_q, t),$$

получаем окончательно, что каждая частично-рекурсивная функция f представима в виде

$$f(x_1, \ldots, x_q) = \pi(x_1, \ldots, x_q, \mu t[\tau(x_1, \ldots, x_q) = 0]), \tag{3}$$

где π и τ суть примитивно-рекурсивные функции.

ПРИЛОЖЕНИЕ I

СВЕДЕНИЯ ИЗ ТЕОРИИ РЕКУРСИВНЫХ ФУНКЦИЙ

Мы рассматриваем функции от одного, двух и более переменных, причем каждое переменное и сама функция принимают значения из натурального ряда. Каждое натуральное число условимся считать функцией от нулевого числа переменных. Мы не требуем, чтобы функция от n переменных была определена для всех систем из n натуральных чисел. Равенство между двумя функциями

$$f(x_1, x_2, \ldots, x_n) = g(x_1, x_2, \ldots, x_n)$$

означает, что для всякой системы из n натуральных чисел $(\mathfrak{z}_1, \mathfrak{z}_2, \ldots, \mathfrak{z}_n)$, для которой определена одна из этих функций, определена и другая и

$$f(\mathfrak{z}_1, \mathfrak{z}_2, \ldots, \mathfrak{z}_n) = g(\mathfrak{z}_1, \mathfrak{z}_2, \ldots, \mathfrak{z}_n).$$

Введем ряд операторов на множестве функций. Аргументом каждого оператора является функция или система функций.

Оператор суперпозиции. Область определения оператора суперпозиции — системы функций вида

$$\psi(x_1, x_2, \ldots, x_n);\ \varphi_1(x_1, \ldots, x_m),\ \varphi_2(x_1, \ldots, x_m), \ldots, \varphi_n(x_1, \ldots, x_m)$$
$$(n = 1, 2, \ldots;\ m = 1, 2, \ldots).$$

Будучи применен к такой системе, оператор суперпозиции дает функцию $\theta(x_1, \ldots, x_m)$, являющуюся результатом подстановки функций $\varphi_1, \varphi_2, \ldots, \varphi_n$ в функцию ψ на места ее переменных:
$\theta(x_1, \ldots, x_m) = \psi(\varphi_1(x_1, \ldots, x_m), \varphi_2(x_1, \ldots, x_m), \ldots, \varphi_n(x_1, \ldots, x_m))$.
Функция $\theta(x_1, \ldots, x_m)$ определена для всех наборов натуральных чисел $(\mathfrak{z}_1, \ldots, \mathfrak{z}_m)$, обладающих следующими свойствами:

1) для $(\mathfrak{z}_1, \ldots, \mathfrak{z}_m)$ определены все функции $\varphi_i(i = 1, 2, \ldots, n)$,

2) если $\varphi_i(\mathfrak{z}_1, \ldots, \mathfrak{z}_m) = t_i\ (i = 1, 2, \ldots, n)$, то функция ψ определена для системы $(t_1, t_2, \ldots, t_n)$.

Оператор примитивной рекурсии. Область определения — пары функций вида

$$h(x_1, \ldots, x_{n-1}),\quad g(x_1, \ldots, x_{n-1}, x_n, x_{n+1})$$
$$(n = 1, 2, \ldots).$$

Примененный к такой паре, оператор рекурсии дает функцию $f(x_1, \ldots, x_{n-1}, x_n)$, связанную с h и g следующими равенствами:

$$f(x_1, \ldots, x_{n-1}, 0) = h(x_1, \ldots, x_{n-1}),$$
$$f(x_1, \ldots, x_{n-1}, x_n + 1) = g(x_1, \ldots, x_{n-1}, x_n, f(x_1, \ldots, x_{n-1}, x_n)).$$

Оператор наименьшего числа. Область определения — совокупность всех функций от одного или более переменных. Будучи применен к функции $\theta(x_1, \ldots, x_{n-1}, x_n)$, оператор наименьшего числа дает функцию $\psi(x_1, \ldots, x_{n-1})$, задаваемую следующим законом: для всякого набора натуральных чисел $\mathfrak{z}_1, \ldots, \mathfrak{z}_{n-1}$ значение функции

$$f = \psi(\mathfrak{z}_1, \ldots, \mathfrak{z}_{n-1})$$

есть такое число t, что

1) $\theta(\mathfrak{z}_1, \ldots, \mathfrak{z}_{n-1}, t) = 0$;

2) для всякого числа $\mathfrak{z}$, меньшего, чем t, значение $\theta(\mathfrak{z}_1, \ldots, \mathfrak{z}_n, \mathfrak{z})$ определено и не равно нулю.

Результат применения оператора наименьшего числа к функции $\theta(x_1, \ldots, x_{n-1}, x_n)$ записывают обычно в виде

$$\mu x_n [\theta(x_1, \ldots, x_{n-1}, x_n) = 0],$$

помня при этом, что запись

$$\mu x_n [\theta(x_1, \ldots, x_{n-1}, x_n) = 0]$$

обозначает функцию лишь от переменных $x_1, \ldots, x_{n-1}$.

Оператор наименьшего числа в отличие от операторов суперпозиции и рекурсии может, будучи применен ко всюду определенной функции, дать функцию, не всюду определенную (и даже нигде не определенную).

Введем следующие три группы функций:

1) Функции $O_n(x_1, \ldots, x_n)$ при $n = 0, 1, 2, \ldots$; для каждого n функция $O_n(x_1, \ldots, x_n)$ тождественно равна нулю.

2) Функции $I_{nk}(x_1, \ldots, x_n)$ при $n = 1, 2, \ldots$; для любого набора $(\mathfrak{z}_1, \ldots, \mathfrak{z}_k, \ldots, \mathfrak{z}_n)$ имеем $I_{nk}(\mathfrak{z}_1, \ldots, \mathfrak{z}_k, \ldots, \mathfrak{z}_n) = \mathfrak{z}_k$.

3) Функция $\lambda(x)$, равная $x + 1$.

Все эти функции назовем *базисными*.

Класс *примитивно-рекурсивных функций* определяется как минимальный класс, содержащий базисные функции и замкнутый относительно применения оператора суперпозиции и оператора примитивной рекурсии [18], [6]. Каждая примитивно-рекурсивная функция, как легко видеть, всюду определена. Все простейшие арифметические функции, как-то:

$$x + y, \quad xy, \quad x^y, \quad |x - y|, \quad \left[\frac{x}{y}\right], \quad [\sqrt[x]{y}], \quad [\log_x y], \quad x!$$

и многие другие (например, показатель, с которым простое число p_x входит в разложение на простые множители числа y), являются примитивно-рекурсивными.

Класс *частично-рекурсивных функций* определяется как минимальный класс, содержащий базисные функции и замкнутый относительно

применения оператора суперпозиции, оператора примитивной рекурсии и оператора наименьшего числа [19], [6].

Частично-рекурсивная функция от n переменных, определенная для всех систем из n натуральных чисел, называется *общерекурсивной* [19], [6]. В частности, всякая примитивно-рекурсивная функция есть функция общерекурсивная. Существуют общерекурсивные функции, не являющиеся примитивно-рекурсивными.

Каждая частично-рекурсивная функция конструируется из примитивно-рекурсивных функций посредством применения операторов суперпозиции, примитивной рекурсии и наименьшего числа. Представляет интерес наименьшее число операторов, требующееся для образования частично-рекурсивной функции из примитивно-рекурсивных. A priori это число может быть сколь угодно велико. Однако существует важная теорема Клини [19], утверждающая, что всякая частично-рекурсивная функция $f(x_1, \ldots, x_n)$ может быть получена применением оператора наименьшего числа и оператора суперпозиции из двух примитивно-рекурсивных функций [из которых одна является раз навсегда фиксированной, а другая зависит от функции $f(x_1, \ldots, x_n)$] [1]. Именно, теорема Клини гласит, что существует такая примитивно-рекурсивная функция $U(x)$, что для всякой частично-рекурсивной функции $f(x_1, \ldots, x_n)$ существует примитивно-рекурсивная функция $\theta(x_1, \ldots, x_n, y)$ со следующим свойством:

$$f(x_1, \ldots, x_n) = U(\mu y [\theta(x_1, \ldots, x_n, y) = 0]).$$

Формула Клини важна еще и тем, что она сводит к минимуму употребление оператора наименьшего числа; применение этого оператора всегда неприятно, ибо он, как отмечалось, может из всюду определенных функций конструировать не всюду определенные.

Мы скажем, что множество R порождается функцией $f(x)$, если множество значений $f(x)$ есть R. Множество называется *рекурсивно-перечислимым*, если оно либо пусто, либо есть множество значений некоторой общерекурсивной функции [20]. Класс рекурсивно-перечислимых множеств совпадает с классом множеств, порождаемых частично-рекурсивными функциями. Класс непустых рекурсивно-перечислимых множеств совпадает с классом множеств, порождаемых примитивно-рекурсивными функциями [21]. Следующие множества являются рекурсивно-перечислимыми: сумма и пересечение двух рекурсивно-перечислимых множеств; образ и полный прообраз рекурсивно-перечислимого множества при отображении, задаваемом частично-рекурсивной функцией; область определения частично-рекурсивной функции одного переменного.

Множество называется *рекурсивным*, если и оно и его дополнение рекурсивно-перечислимы [20]. Существуют рекурсивно-перечислимые, но не рекурсивные множества [20].

---

[1]) Уже формула (3) из § 3 показывает, что любую частично-рекурсивную функцию можно образовать из двух примитивно-рекурсивных применением операторов наименьшего числа и суперпозиции.

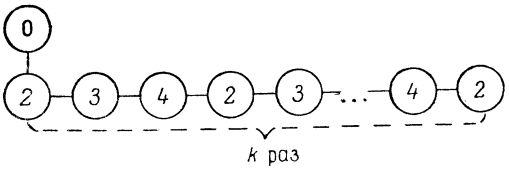
Характеристической функцией множества называется функция, принимающая значение 1 на множестве и 0 вне его. Характеристическая функция рекурсивного множества есть функция общерекурсивная [20]. Обратно, если характеристическая функция множества общерекурсивная, то само множество рекурсивно [20].

Если идентифицировать понятие всюду определенной алгоритмически вычислимой функции с понятием общерекурсивной функции, то рекурсивно-перечислимые множества суть те множества, которые можно развернуть в алгоритмически вычислимую последовательность (быть может, с повторениями), а рекурсивные множества суть такие множества, для каждого из которых существует алгоритм, позволяющий узнавать, принадлежит данное натуральное число n этому множеству или нет.

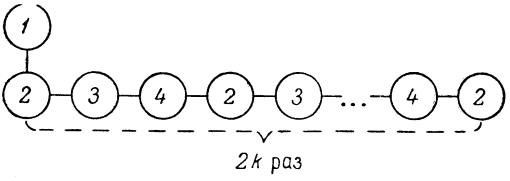
ПРИЛОЖЕНИЕ II

ПРИМЕР АЛГОРИТМА

Приведем пример алгоритма «удвоения», перерабатывающего всякий комплекс A_k



в комплекс B_{2k}



Заметим, что наиболее простым алгоритмом (в интуитивном смысле), удваивающим линейно упорядоченный ряд точек, является следующая процедура: надо просматривать по очереди все точки и вместо каждой точки ставить две. Если эту процедуру формализовать, то получится как раз построенный ниже алгоритм.

Алгоритм, который мы будем строить, принадлежит типу $\mathbf{A}(\mathfrak{T}_5)$, а точнее, $\mathbf{A}_4(\mathfrak{T}_5)$, где $\mathfrak{T}_5 = \{T_0, T_1, T_2, T_3, T_4, T_5\}$.

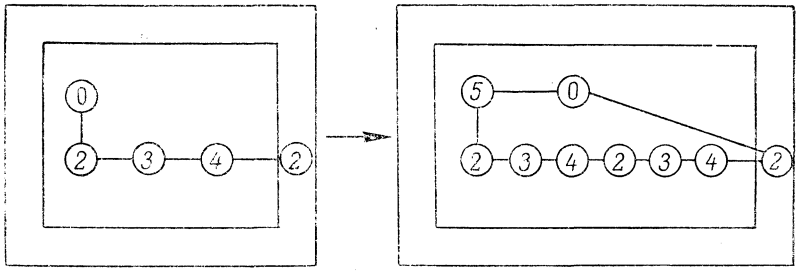
Алгоритм задается правилами непосредственной переработки, состоящими из конечного набора пар $U_i \rightarrow W_i$.

Каждую такую пару условимся графически изображать следующим образом[1]). Комплексы U_i и W_i будем чертить непересекающимися. Комплекс U_i будем помещать слева, а комплекс W_i — справа. В комплексе U_i мы выделим границу $L(U_i)$, заключив ее в рамку, состоящую из двух концентрических прямоугольников. Двумя другими (правыми),

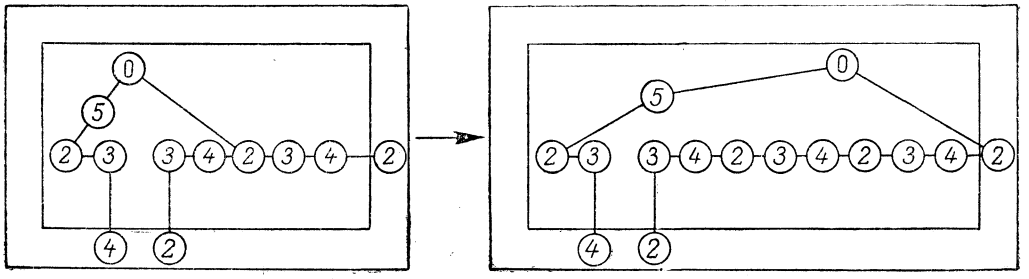
[1]) Всякий комплекс изображается на чертеже с точностью до изоморфизма.

концентрическими прямоугольниками, конгруэнтными первым (левым), соответствующий подкомплекс $\tilde{L}(U_i)$ выделяется и в W_i. (Чтобы получить изоморфное соответствие φ_i между $L(U_i)$ и $\tilde{L}(W_i)$, надо совместить правую систему прямоугольников с левой и отождествить совпавшие вершины, заключенные между прямоугольниками.) После этих соглашений выпишем следующие пять правил, задающие алгоритм удвоения.

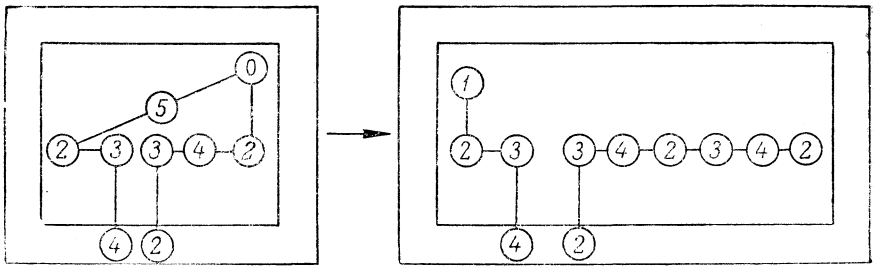
Правило 1. $U_1 \to W_1$.



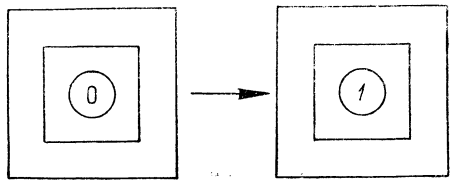
Правило 2. $U_2 \to W_2$.



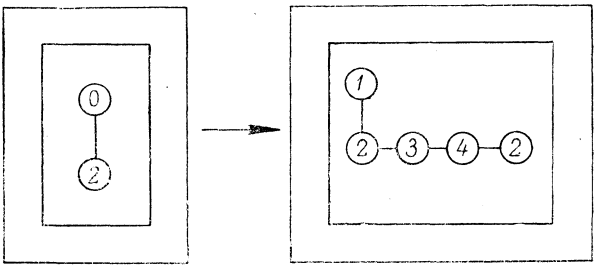
Правило 3. $U_3 \to W_3$.



Правило 4. $U_4 \to W_4$.

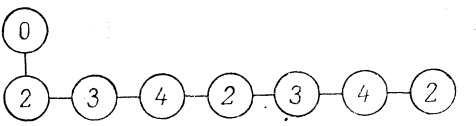


Правило 5. $U_5 \rightarrow W_5$.

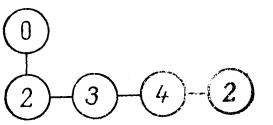


Потребность в применении правила 4 возникает тогда, когда $k=0$, в применении правила 5, — когда $k=1$.

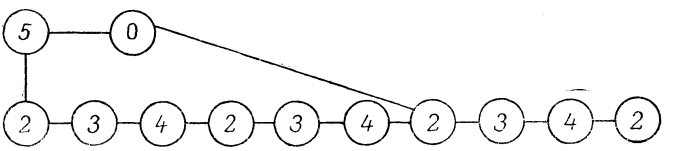
Рассмотрим в качестве примера процесс переработки построенным алгоритмом комплекса $S^0=A_3$:



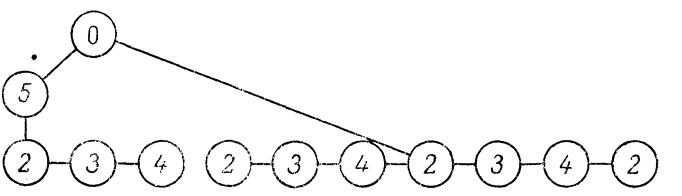
По определению, к S^0 следует применить оператор непосредственной переработки Ω, заданный правилами 1 — 5. Для этого в S^0 выделяется комплекс $U(S^0)$. Так как $N=4$, то комплекс $U(S^0)$ имеет вид



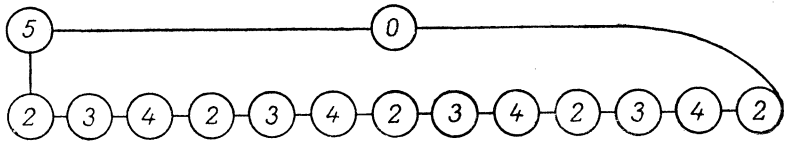
Затем среди правил 1 — 5 ищется та пара, у которой первый член изоморфен $U(S^0)$. Такой парой является правило 1. Производя непосредственную переработку согласно этому правилу, получаем комплекс $S^1=\Omega(S^0)$ вида



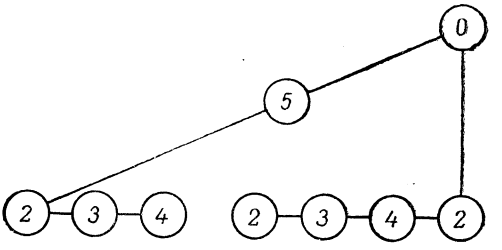
Комплекс $U(S^1)$ имеет строение



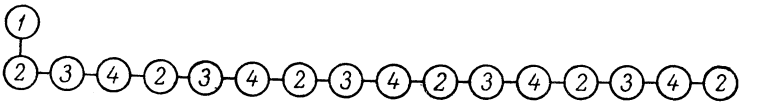
Он изоморфен первому члену U_2 правила 2, поэтому комплекс $S^2 = \Omega(S^1)$ есть следующий:



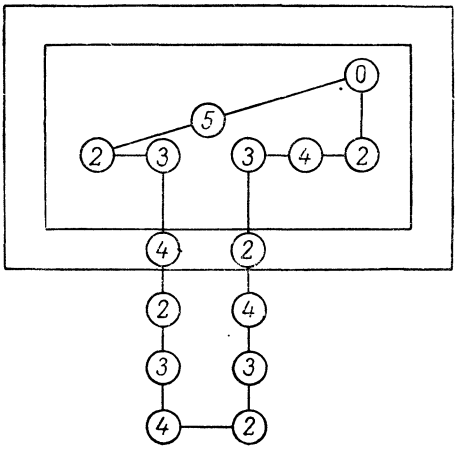
В комплексе S^2 снова выделяется комплекс $U(S^2)$; этот последний имеет вид



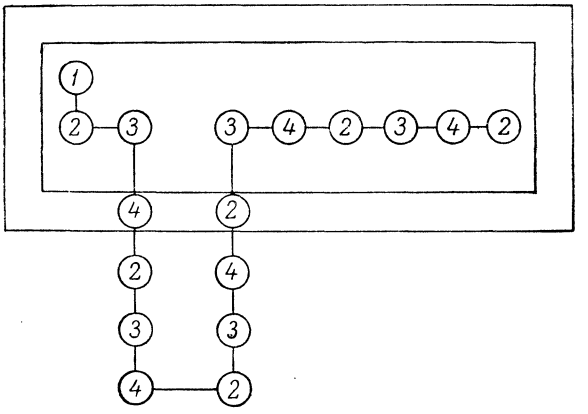
Он изоморфен первому члену U_3 правила 3, применяя которое получаем «решение» в виде комплекса $B_6 = S^3 = \Omega(S^2)$, имеющего строение



Заметим, что при сделанных соглашениях относительно графического изображения правил непосредственной переработки сама непосредственная переработка получает наглядное геометрическое истолкование. Проследим это на нашем примере. Мы видели, что $U(S^2)$ изоморфен U_3. Начертим комплекс S^2 так, чтобы подкомплекс $U(S^2)$ оказался конгруэнтен изображению U_3 (на чертеже, соответствующем правилу 3), и выделим, так же как это сделано для U_3, подкомплексы $U(S^2)$ и $L(S^2)$ прямоугольными рамками.



Тогда комплекс $\Omega(S^2)$ получается следующим простым способом: система прямоугольников, обрамляющая W_3, налагается на конгруэнтную ей систему прямоугольников, построенную для S^2; совпавшие вершины, расположенные между прямоугольниками, отождествляются; все то, что лежит внутри самого внутреннего из прямоугольников, построенных для S^2, заменяется на то, что лежит внутри самого внутреннего из прямоугольников, обрамляющих W_3. Действительно, если произвести все эти операции, получится комплекс



Это и есть (с точностью изоморфизма) $S^3 = \Omega(S^2)$.

ЦИТИРОВАННАЯ ЛИТЕРАТУРА

[1] А. А. Марков, Теория алгорифмов, Труды Матем. ин-та им. Стеклова 38 (1951), 176—189.
[2] А. А. Марков, Теория алгорифмов, Труды Матем. ин-та им. Стеклова 42 (1954), 375.
[3] А. Н. Колмогоров, О понятии алгоритма, УМН VIII, вып. 4 (56) (1953), 175—176.
[4] K. Gödel, On undecidable propositions of formal mathematical systems, Princeton, 1934.
[5] A. Church, An unsolvable problem of elementary number theory, Amer. Journ. Math. 58 (1936), 345—363.
[6] С. К. Клини, Введение в метаматематику, М., ИЛ, 1957.
[7] A. Church, The calculi of lambda-conversion, Princeton, 1941.
[8] S. C. Kleene, On notation for ordinal numbers, Journ Symb. Logic 3 (1938), 150—155.
[9] K. Gödel, Uber die Länge von Beweisen, Ergebnise eines mathematischen Kolloquiums, Heft 7 (за 1934—1935 гг., вышла из печати в 1936 г.), стр. 34—38.
[10] S. C. Kleene, General recursive functions of natural numbers, Math. Ann 112 (1936), 727—742.
[11] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc. 42 (1936—1937), 230—265.
[12] Р. Петер, Рекурсивные функции, М., ИЛ, 1954.
[13] E. L. Post, Finite combinatory processes — formulation I, Journ. Symb. Logic 1 (1936), 103—105.
[14] S. C. Kleene, λ-definitability and recursiveness, Duke Math. Journ. 2 (1936), 340—353.

[15] A. M. Turing, Computability and λ-definitability, Journ. Symb. Logic 2 (1937), 153—163.
[16] В. К. Детловс, Нормальные алгорифмы и рекурсивные функции, ДАН 90 (1953), 723—725.
[17] В. А. Успенский, К теореме о равномерной непрерывности, УМН XII, вып. 1 (73) (1957), 99—142.
[18] R. M. Robinson, Primitive recursive functions, Bull. Amer. Math. Soc. 53 (1947), 925—942.
[19] S. C. Kleene, Recursive predicates and quantifiers, Trans. Amer. Math. Soc. 53 (1943), 41—73.
[20] E. L. Post, Recursively enumerable sets of positive integers and their decision problems, Bull. Amer. Math. Soc. 5 (1944), 284—316.
[21] J. B. Rosser, Extensions of some theorems of Gödel and Church, Journ. Symb. Logic 1 (1936), 87—91.
[22] E. L. Post, Formal reduction of the general combinatorial decision problem, Amer. Journ. Math. 65 (1943), 197—215.