

МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО
ОБРАЗОВАНИЯ РСФСР

ТАГАНРОГСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

А.Н. МЕЛИХОВ, В.И. КОДАЧИГОВ

ТЕОРИЯ АЛГОРИТМОВ И ФОРМАЛЬНЫХ ЯЗЫКОВ

(Учебное пособие)

Таганрог 2006 г.

УДК 681.3.062(075.8)+518.5(075.8)

А.Н. Мелихов, В.И. Кодачигов. Теория алгоритмов и формальных языков:

Учебное пособие. Таганрог: ТРТУ, 2006г. - с.

В пособии рассматриваются основные вопросы, включенные в программу одноименного курса, читаемого студентам специальности 0647. Изучаются такие понятия, как формальные системы, комбинаторные системы, алгоритмы, машины Тьюринга, формальные грамматики и языки. Рассматриваются их основные свойства и взаимосвязь. Обсуждаются алгоритмические разрешаемые и неразрешаемые проблемы, связанные с машинами Тьюринга, автоматами с магазинной памятью и конечными автоматами. Пособие знакомит студентов с математическим аппаратом и основными проблемами теории алгоритмов и формальных языков.

Ил. 22, библиогр. 12.

Рецензенты:

кафедра алгебры и геометрии Таганрогского педагогического института:

д-р техн. наук, профессор, заведующий кафедрой Московского института инженеров транспорта А.В. Шилейко.

© Таганрогский государственный радиотехнический университет, 2006г.

ОТ АВТОРОВ

Одна из главных трудностей, с которыми сталкиваются студенты при освоении основополагающих курсов, связанных с математическим обеспечением и применением ЭВМ, заключается в том, что многие понятия этих курсов еще «не устоялись», а материал их разбросан по разным источникам, написанным, как правило, языком труднодоступным для первого знакомства.

Данное пособие ориентировано в первую очередь на студентов младших курсов. Поэтому при его написании мы постарались совместить строгость изложения основных понятий и доказательство фундаментальных положений одноименного лекционного курса с доходчивостью восприятия. Особое внимание при этом мы уделили показу взаимосвязи таких понятий, как формальные системы, алгоритмы, машины Тьюринга, грамматики, языки.

Материал, включенный в пособие, охватывает основные положения министерской программы. Однако пособие не является конспектом лекций и не подменяет, а лишь дополняет его.

Курс «Теория алгоритмов и формальных языков» сопровождается практическими занятиями, организация которых отражена в специальной методической разработке. Поэтому в пособии использован лишь необходимый минимум примеров.

При написании учебного пособия мы широко использовали книги и монографии, указанные в списке литературы. Поскольку специальных ссылок на них в тексте пособия не дается, мы заранее приносим их авторам свои извинения.

Пособие будет полезно не только при изучении курса «Теория алгоритмов и формальных языков», но и курсов, связанных с разработкой и применением систем математического обеспечения ЭВМ, системного программирования и других курсов специализации, читаемых кафедрой математического обеспечения и применения ЭВМ студентам специальности 0647.

0.3. Интуитивное понятие алгоритмов

Слово алгоритм связывают с именем арабского математика IX века Мухаммеда ибн Аль-Хорезми, впервые выдвинувшего идею о том, что решение любой поставленной математической и философской задачи может быть оформлено в виде последовательности число механически (без всякой изобретательности) выполняемых правил, т.е. может быть алгоритмизировано. Этому же мнению придерживались Декарт, Лейбниц, Гильберт. Веру в возможность алгоритмизации любой задачи подорвал Гедель, в 1931 году опубликовавший работу «О формально неразрешимых предложениях оснований математики и родственных систем», в которой он доказал, что существуют математические проблемы, для решения которых не может быть найден алгоритм.

В настоящее время более полному понятию алгоритма по сравнению с приведенным выше соответствует определение, включающее следующие категории.

1. Алгоритм есть конечная совокупность инструкций.
2. Есть некоторый механизм, воспринимающий и исполняющий инструкцию.
3. Имеются средства, позволяющие хранить и фиксировать сведения о всех этапах работы алгоритма, хранить промежуточные результаты и выдавать их по мере необходимости.
4. Все выполняемые алгоритмом действия являются дискретными.
5. Последовательность операций, из которых складывается алгоритм, детерминирована и в каждом шаге выполняется единственным образом.

Из сказанного напрашивается следующая аналогия с парой программа-ЭВМ, именно: первому пункту соответствует понятие программы; второму - ЭВМ; третьему - память ЭВМ; четвертому - дискретный характер работы ЭВМ; пятому - жесткий порядок действий ЭВМ.

Если алгоритмы предназначены для реализации на ЭВМ, то в это определение необходимо внести ограничения, накладываемые особенностями использования ЭВМ. Представим один из вариантов как совокупность следующих требований.

1. Алгоритм **A** оперирует с конкретными объектами. Их можно перенумеровать.

2. **A** задается конечным предписанием **B**, исходным, выходным и рабочим алфавитами.

3. **B** должно быть составлено таким образом, чтобы определяемые им операции выполнялись поэтапно.

4. **B** должно быть составлено так, чтобы оно исполнялось однозначно.

5. **B** должно быть исполнимо, т.е. применение **A** к одному и тому же входному слову должно приводить к одним и тем же последовательностям действий и результату.

6. **B** должно быть составлено так, чтобы его исполнение не требовало информации, отличной от имеющейся во входном слове.

7. Нет никаких ограничений на длину **B**, длину слов и число шагов. Важно только, чтобы они были конечны.

8. Для исполнения **B** требуется сколько угодно большая, но конечная память.

Алгоритм можно описать множеством других определений, например через конструктивно задаваемое соответствие в некотором алфавите, через понятие частичного алгоритма и так далее.

Будем называть алфавитным отображением или оператором всякое соответствие, сопоставляющее словам некоторого алфавита слова в том же (или другом) алфавите. При наличии двух алфавитов первый называется входным, а второй выходным.

Алфавитный оператор называют однозначным, если он ставит в соответствие одному слову входного алфавита не более одного слова выходного алфавита.

Совокупность всех входных слов, с которыми работает оператор, называется областью его определения. Область определения может быть конечной и бесконечной. В первом случае оператор задается в виде таблицы, во втором - системой правил. Эта система должна быть конечной и задавать соответствие за конечное число шагов.

В алфавитном отображении важным является понятие соответствия, а не способа, которым это соответствие задано. В понятии же алгоритма основным является способ задания соответствия. Таким образом, алгоритм - это алфавитный оператор вместе с правилами, определяющими способ его выполнения.

Частичным алгоритмом называется совокупность конечного числа команд, выполняемых механически за конечное время и с оцениваемыми конечными затратами. Примером частичного алгоритма является программа. Алгоритмом называется такой частичный алгоритм, который после некоторого конечного числа шагов либо исчерпывает все команды и выдает результат, либо выдает информацию о невозможности получения результата.

Всевозможные уточнения приведенных и других определений, как оказалось, адекватны. В соответствии с так называемым тезисом Черче:

- все уточнения интуитивного понятия алгоритма эквиваленты;
- все алгоритмы в точном смысле являются одновременно и алгоритмами в интуитивном смысле.

Алгоритмы, ставящие в соответствие одному входному слову одно выходное, называются детерминированными.

Алгоритмы обладают следующими свойствами.

- *Массовость*. Это способность быть применимым для решения множества задач.
- *Результативность*. Это свойство обеспечивает получение результата за конечное число шагов.

- *Определенность.* Это свойство обеспечивает при повторном поступлении на вход алгоритма одного и того же слова получение одного и того же выходного слова.

Помимо детерминированных существуют алгоритмы самоизменяющиеся и случайные. Самоизменяющиеся алгоритмы при обработке различных строк меняются. В случайных алгоритмах предусматривается возможность случайного выбора правил.

Все приведенные выше определения задают понятие алгоритма неформально. Одним из наиболее распространенных уточненных и формальных определений является определение, основанное на понятии машины Тьюринга.

0.4. Понятие языка

Понятие языка базируется на следующих четырех категориях:

1. Множество информации или смыслов, которые можно получить в данном языке (план содержания).
2. Множество текстов, которые характеризуются с помощью средств языка (план выражения).
3. Множество отображений, определенных на множестве текстов и ставящих в соответствие любому тексту конкретный смысл.
4. Обратное отображение.

В естественных языках множество смыслов определено неформально. Смысл текста часто меняется в зависимости от ударения, интонации и так далее. План выражения также определен неформально. Здесь существенно знание языка, эмоциональная окраска, наличие иносказаний и так далее.

В формальных языках, предназначенных для описания реальных задач, неоднозначность исключается априорно.

Все искусственные языки (названные так в отличие от естественных разговорных языков, живых и мертвых) можно подразделить на языки формальные и не вполне формализованные.

Формальные языки делятся на алгоритмические и неалгоритмические.

Алгоритмические языки предназначены для описания вычислительных процессов, являющихся алгоритмами. Они входят в предмет нашего рассмотрения, а неалгоритмические - нет.

Алгоритмическим языком называется совокупность трех множеств:

1. Множество символов (алфавит языка).
2. Множество правил, позволяющих конструировать правильные сообщения в символах алфавита (синтаксис языка).
3. Множество правил, позволяющих однозначно толковать смысл правильных записей в символах алфавита (семантика языка).

Иногда в синтаксисе выделяют подмножество - так называемую лексику языка - совокупность слов, допустимых в языке вместе с описанием способа их представления. Лексика - это набор правильных слов языка.

Для описания формальных языков вообще и алгоритмических в частности используются метаязыки - расширение описываемых языков. С помощью метаязыков удобно описывается синтаксис языка. Семантика описывается либо на естественном языке, либо путем задания алгоритмов распознавания текстов языка, либо путем задания текстов на другом языке, имеющем средства формального описания.

Все алгоритмические языки можно классифицировать по различным признакам. Наиболее общая классификация делит языки по кругу решаемых задач, например, языки для описания статистических задач, задач вычислительного характера и так далее. Другая классификация делит языки на классы по степени их зависимости от ЭВМ. В соответствии с ней различают машинно-зависимые и машинно-независимые языки. Первые делятся на машинно-ориентированные и машинные. Вторые - на процедурно-ориентированные и проблемно-ориентированные. Процедурно-

ориентированные языки предназначены для описания алгоритмов решения задач. При переводе программы с них в машинную программу каждое слово заменяется эквивалентным ему машинным. Проблемно-ориентированные языки почти не зависят от конкретной машины, перевод с них в конкретный машинный язык осуществляется по принципу «несколько слов в несколько». Перевод типа «слово - слово» невозможен.

Однако приведенные классификации не отражают синтаксических особенностей самих языков. Таковые учитываются в классификации, основанной на понятии грамматики языка.

0.5. Понятие программы

В определениях понятия алгоритма мы уже использовали интуитивно ясные понятия: команда, программа, ЭВМ и даже указали на взаимосвязь категорий понятия алгоритма с этими понятиями. Уточним в интересующем нас аспекте эти понятия. В самом общем смысле ЭВМ представляет собой автомат, состоящий из памяти и исполнительного устройства. Последнее может выполнять ограниченный набор достаточно простых команд типа: «найти в памяти нужное слово (слова)», «выполнить над ними операцию сложения (вычитания, деления, умножения)», «направить результат выполнения операции в память». Ясно, что для выполнения сколь-нибудь серьезных вычислений необходимо составить достаточно длинную последовательность команд, записать их в память ЭВМ и указать последовательность их выполнения исполнительным устройством. При этом надо позаботиться о том, чтобы в нужный момент из памяти извлекались все промежуточные значения переменных, а также необходимые исходные данные. Последовательность команд, составленных с учетом перечисленных обстоятельств, называется машинной программой, или программой, составленной на языке машины. Ясно, что вести программирование на языке машины для достаточно сложных или громоздких задач нереально. Поэтому

программирование реально выполняется на языке машинных команд, т.е. на некотором (алгоритмическом) языке, удобном для программиста, и программы, составленные на этом языке, затем переводятся с помощью трансляторов в машинные программы, которые и используются. Таким образом, программа, записанная на алгоритмическом языке в форме, удобной для программиста, является для него окончательной. Перевод же ее в машинную программу ЭВМ берет на себя.

Несмотря на различия в длине (машинная программа, естественно, гораздо длиннее программы, записанной на алгоритмическом языке), обе они имеют одинаковую структуру.

Различают программы линейной структуры, программы с условиями, программы с циклами и та далее. Структура программы зависит также от способа представления информации, с которой она работает.

Одна и та же задача может быть, вообще говоря, описана несколькими программами. Эти программы являются эквивалентными, если они записаны на одном и том же языке. Среди эквивалентных программ наиболее экономная, а значит более лучшая та, что имеет меньшее число команд и использует меньший объем памяти.

Проблема экономии памяти требует строить программы специальной структуры, проводить преобразование программы, организовывать распределение памяти и оценивать баланс между расходом памяти и расходом времени на распределение памяти. Все это реализуется специальными алгоритмами.

Таким образом, в общем случае программа - это конечное упорядоченное множество предписаний, записанное в символах некоторого алгоритмического языка и представляющее собой такую форму записи алгоритма, понятную ЭВМ (транслятору), при которой обеспечивается его наибольшее эффективное исполнение.

0.6. Понятие транслятора

Транслятор - это программа, переводящая запись произвольного текста на одном (входном) языке в эквивалентный ему текст на другом (выходном) языке.

Если входным языком является процедурно-ориентированный язык, то транслятор называют компилятором. В компиляторах вся программа сначала целиком транслируется, а затем уже принимается к исполнению ЭВМ. Если трансляция и исполнение программы совмещены во времени, то транслятор называют интерпретатором. Наиболее распространены трансляторы-компиляторы.

Любая программа состоит из предложений, записанных в символах алфавита языка. Из этих предложений и слов (лексики) языка по правилам синтаксиса строятся более сложные предложения. Описание смысла предложений производится с помощью семантики языка. Семантика нужна для того, чтобы ЭВМ могла определить, как выполнять каждое предложение.

Трансляция - это перевод программы с одного языка на другой, связанный в общем случае с изменением алфавита, лексики и синтаксиса языка программы с сохранением ее семантики.

Процесс трансляции укрупнено можно представить как последовательность действий по опознаванию конструкций (предложений программы), определению их смысла и формированию эквивалентных им записей на выходном языке. Как правило, на транслятор возлагают ряд дополнительных функций, таких, как контроль входной программы, распределение памяти и так далее. Набор этих функций влияет на структуру транслятора. Однако структура транслятора определяется в основном не этим, а методом трансляции.

Различают синтаксические и прямые методы трансляции. Прямые методы ориентированы на конкретные входные языки и конкретные ЭВМ. Синтаксические методы используют для анализа входных программ свойства

формальных грамматик и ориентированы поэтому на некоторые классы входных языков. Реальные трансляторы строятся, как правило, либо по блочному принципу, либо из подпрограмм. Существуют трансляторы, в которых трансляция выполняется за несколько просмотров. В каждом из них осуществляется перевод в некоторую промежуточную форму. Организация трансляторов требует принятия специальных мер, облегчающих работу с табличной информацией.

1. АЛГОРИТМИЧЕСКИЕ СИСТЕМЫ

1.1. Основные понятия и определения

Пусть имеем непустое множество A , которое назовем алфавитом или словарем, а его элементы - символами или буквами. Например алфавит $A = \{+, ', \S, a, e, \gamma\}$. Он содержит шесть букв.

Произвольную конечную последовательность букв алфавита A будем называть словом или цепочкой в этом алфавите. Слово считается ориентированным слева направо. Например, последовательность $A = a + e?$ является словом, а последовательность $B = 7 + a - e?$ не является словом в алфавите A . Слово, не содержащее никаких букв называется пустым и обозначается E .

Число букв, входящих в слово, называется его длиной. Длина слова A обозначается через $|A|$. Ясно, что $|A| = 5$, $|E| = 0$.

В дальнейшем, если это не оговорено специально, в качестве букв алфавита используются малые латинские и русские буквы. Слова обозначаются большими латинскими буквами.

Пусть A^* - множество всех слов в алфавите A и пусть $A, B \in A^*$. Упорядоченной паре $\langle A, B \rangle$ поставим в соответствие слово C , полученное приписыванием справа к слову A слова B . Говоря, что слово C получено конкатенацией (умножением) A и B и записывают $C = AB$. Таким образом, конкатенацией называется, с одной стороны, операция приписывания, а с

другой стороны, результат ее выполнения. Пусть $A = cасв$ и $B = авв$. Тогда $AB = сасваав$, $BA = аввсасв$. Длина конкатенации равна сумме длин образующих ее слов.

Ясно, что конкатенация является всюду определенной и ассоциативной, но не коммутативной операцией, т.е. для любых $A, B, C \in A^*$ имеет место: $A(BC) = (AB)C = ABC$ и $AB \neq BA$. Кроме того, $AE = EA = A$, где E играет роль единичного элемента. Поэтому множество A^* по операции конкатенации является свободной подгруппой над \bar{A} . Причем элементы \bar{A} являются образующими этой подгруппы.

Пусть P, Q, A, X, Y - различные слова в алфавите A и пусть $A = XPY$. Тогда слова X, P, Y называются вхождениями в слово A . Слова X, X, P, XPY называются подсловами A .

Пусть имеется подстановка вхождения слова Q вместо слова P , и наоборот, т.е. $P \sim Q$. Тогда слову A можно поставить в соответствие слово B и, наоборот, из слова B получить слово A .

Пример. Если $P = ара$, а $Q = ло$, то слову $A = n \underline{арад}$ соответствует слово $B = n \underline{лод}$, и наоборот.

Подобные соотношения называются соотношениями Туэ (по имени норвежского математика). Они приводят к одному из вариантов ассоциативного исчисления.

Два слова A и B называются смежными, если одно из них получено из другого однократным применением соотношений Туэ. В общем случае можно допустить несколько последовательных применений соотношений Туэ. Пусть A - некоторый алфавит и $R: P_1 \sim Q_1, \dots, P_n \sim Q_n$ - систем соотношений Туэ. Говорят, что слово B_o соотносимо со словом B_p , если существует последовательность слов B_o, \dots, B_p , таких, что B_i смежно с B_{i-1} для $i = 1, \dots, p$. Будем считать, что любое слово соотносимо с самим собой. Тогда соотносимость будет рефлексивным, симметричным и транзитивным отношением, т.е. отношением эквивалентности. Это позволяет соотносимые

слова называть эквивалентными и обозначать знаком \approx . Легко видеть, что для любых слов X и Y из $A \approx B$ следует $XAY \approx XBY$.

Туэ сформулировал следующую проблему, известную под названием «проблемы эквивалентности»: для любой пары слов в некотором алфавите установить, являются они эквивалентными или нет.

Большой вклад в исследование этой проблемы внесли советские математики А.А. Марков и Г.С. Цейтин.

Остановимся более подробно на этих вопросах. Введем сначала следующие определения.

Слово A^{-1} называется обращением (инверсией) слова A , если оно образовано в точности из тех же вхождений, что и A , не взятых в обратном порядке. Слово A называется симметричным, если оно совпадает со своей инверсией.

Рассмотрим ассоциативное исчисление $A = \{a, v\}$, $aa \sim E$, $vv \sim E$.

Сократить слово A в этом исчислении - значит образовать новое слово B , смежное с A , которое короче A . Взяв некоторое слово за исходное, мы можем путем последовательных сокращений образовать ряд слов, который приведет к несократимому слову (возможно пустому).

Пример. Пусть $A = aavvaaaavvvavvv$, $B = avav$, $C = aaavvavvvvaav$. Покажем один из вариантов сокращения слова A :

$$\underline{aavvaaaavvvavvv} \approx \underline{vvaavvvavvv} \approx \underline{aaavvvavvv} \approx \underline{avvvavvv} \approx \underline{avvvv} \approx \underline{avvv} \approx \underline{av}.$$

В результате получаем, что $A \approx B$ является несократимым в этом исчислении. Существует несколько вариантов сокращения слова C , один из которых имеет вид: $\underline{aaavvavvvvaav} \approx \underline{avvavvvaaav} \approx \underline{avvavvvv} \approx \underline{aaavvvv} \approx \underline{vvvv} \approx \underline{vv} \approx E$. Поэтому $C \approx E$.

Легко проверить, что результат не зависит от того, в каком порядке выполняются сокращения. В каждом классе эквивалентности существует ровно одно несократимое слово, которое называется каноническим представителем этого класса.

Для данного ассоциативного исчисления проблема эквивалентности разрешима, так как любые два слова эквивалентны тогда и только тогда, когда они обладают одним и тем же каноническим представителем, который легко найти путем механически выполняемого процесса сокращения.

Однако проблема эквивалентности разрешима далеко не всегда. Рассмотрим, например, тот же алфавит $A = \{a, v\}$ и соотношения $aaa \sim aa$, $ava \sim aa$, $vav \sim vv$, $vvv \sim vv$. Из слова ava можно получить $\underline{ava} \approx \underline{ava} \approx \underline{aa} \approx aa$ или $\underline{ava} \approx \underline{ava}$. Слова aa и $avva$ эквивалентны, несократимы, но различны.

Естественный метод решения проблемы эквивалентности слов, сводящийся к перебору, состоит в следующем. Для произвольной пары слов S и T последовательно образуются все слова, смежные с S , потом все слова, смежные с каждым из слов, полученных на первом шаге, и так далее. Короче говоря, необходимо найти все слова, эквивалентные слову S , применяя сначала одно, потом два, потом три преобразования и так далее, пока не получится слово T . Однако сколько бы преобразований не было выполнено, тот факт, что T не найдено, еще не означает, что оно не может быть получено последующими преобразованиями. То есть указанный метод не гарантирует нахождения решения. Возникает вопрос: существует ли универсальный метод, позволяющий решать проблему эквивалентности слова. В этом случае эта проблема неразрешима. Мы вернемся к ней при рассмотрении машин Тьюринга.

1.2. Комбинаторные системы

Комбинаторные системы - это частный случай формальных систем. Они используются для исследования задач комбинаторного характера, связанных с преобразованием слов.

Для определения комбинаторной системы необходимо задать:

1) конечный алфавит A , называемый основным, и, возможно, вспомогательный алфавит B . Из $A \cup B$ образуется множество слов (формул);

2) выделенное непустое слово S - аксиому ($S \in A \cup B$);

3) конечное множество правил вывода, задаваемых ориентированными подстановками.

В простейшем случае ориентированная подстановка имеет вид: $A \rightarrow B$, где A и B - слова в алфавите $A \cup B$. Подстановка такого вида является бесконтекстной, носит название полутуэвской и означает, что слово A заменяется словом B . Ясно, что обратная подстановка $B \rightarrow A$ также является полутуэвской. Если имеются одновременно прямая и обратная подстановки, то их можно заменить одной неориентированной (туэвской) подстановкой $A \sim B$. При наличии контекста полутуэвская подстановка имеет вид $PAQ \rightarrow PBQ$, где P и Q - некоторые (возможно пустые) слова в том же алфавите.

Ориентированная подстановка вида $AP \rightarrow PB$ называется нормальной. Обратная подстановка $PB \rightarrow AP$ называется антинормальной. Совокупность нормальной и антинормальной подстановок, т.е. неориентированная подстановка $AP \sim PB$, называется постовской.

Таким образом, в зависимости от типа используемых подстановок можно рассматривать следующие частные виды комбинаторных систем: полутуэвская, туэвская, нормальная и постовская.

Пример 1.1. Пусть задана полутуэвская система, в которой $A = \{a, v\}$ - основной алфавит, $B = \{s\}$ - вспомогательный алфавит, состоящий из одной буквы - аксиомы S , и три подстановки: 1) $s \rightarrow av$; 2) $B \rightarrow asv$; 3) $s \rightarrow vsa$. Множество всех формул, выводимых в этой системе можно представить в виде бесконечного ориентированного графа, каждая вершина которого помечена некоторым словом из алфавита $A \cup B$, а дуги помечаются номером используемой подстановки. Фрагмент этого графа показан на рис. 1.1.

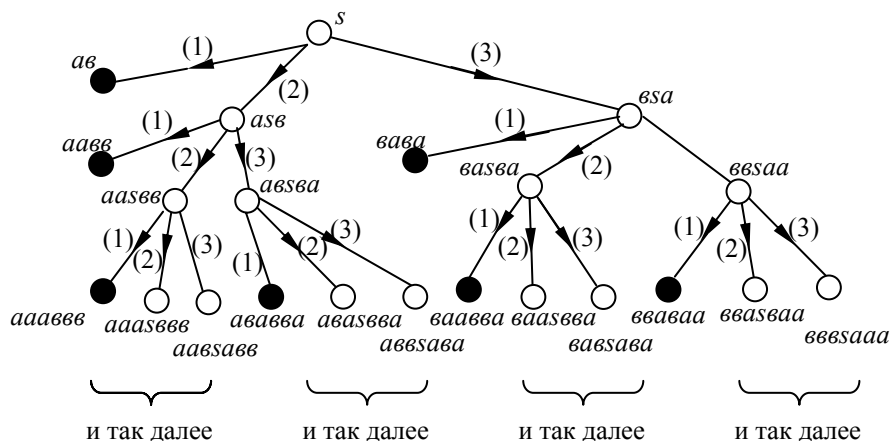


Рис. 1.1

В любую вершину графа, помеченную некоторой формулой, из вершины s ведет один путь. Он как раз и соответствует выводу этой формулы. Например, выводы формул $вава$ и $вvasваа$ имеют вид:

$$s \xrightarrow{3} vsa \xrightarrow{1} vava; \quad s \xrightarrow{3} vsa \xrightarrow{3} vvasaa \xrightarrow{2} vvasvaaa.$$

***, помеченных только буквами алфавита A , вывод формул в этой комбинаторной системе продолжен быть не может.

Комбинаторная система называется моногенной (однозначной), если каждой ее формуле может быть применено не более одной подстановки. Комбинаторная система называется неоднозначной, если в ней имеется формула, для которой существует по крайней мере два различных вывода.

Пример 1.2. Пусть имеется нормальная система с элементом $A = \{a, v\}$, аксиомой vva и подстановками 1) $aP \rightarrow Pvvv$, 2) $vP \rightarrow Pava$, где P - некоторое слово в алфавите A .

Ориентированный граф, задающий множество всех выводимых формул, показан на рис. 1.2

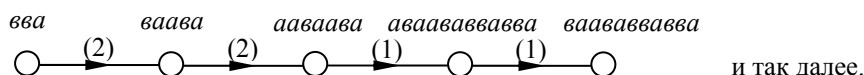


Рис. 1.2

Это нормальная система является моногенной.

Пример 1.3. Пусть имеется полутуэвская система с алфавитами $A = \{a, v, c\}$, $B = \{s\}$, аксиомой s и подстановками: 1) $s \rightarrow as$, 2) $s \rightarrow sv$, 3) $s \rightarrow asv$, 4) $s \rightarrow c$.

Если изобразить граф выводимых в данной системе формул, то можно убедиться, что формула $асв$ является неоднозначной, так как для нее существует три вывода: 1) $s \xrightarrow{1} as \xrightarrow{2} asv \xrightarrow{4} acv$, 2) $s \xrightarrow{2} sv \xrightarrow{1} asv \xrightarrow{4} acv$, 3) $s \xrightarrow{3} asv \xrightarrow{4} acv$. Поэтому эта полутуэвская система является неоднозначной.

1.3. Соответствие между полутуэвской и нормальной системами

Пусть задана полутуэвская система Π , состоящая из алфавита $A = \{a, v, c\}$, слова $C \in A$ - аксиомы и подстановок $PA_iQ \rightarrow PB_iQ$, $1 \leq i \leq m$. Добавим в алфавит A буквы со штрихами и получим алфавит $B = \{a, v, c, a', v', c'\}$. Построим нормальную систему N , входящую в алфавит B , ту же аксиому C , в следующие подстановки:

- 1) $aP \rightarrow Pa'$, $vP \rightarrow Pv'$, $cP \rightarrow Pc'$;
- 2) $a'P \rightarrow Pa$, $v'P \rightarrow Pv$, $c'P \rightarrow Pc$;
- 3) $A_iP \rightarrow PB_i$, $1 \leq i \leq m$.

Заметим, что подстановки типов 1) и 2) позволяют переставлять некоторую букву из начала слова в его конец, меняя при этом ее «штриховость» на обратную. Например, применяя к слову $ваав$ соответствующие подстановки, можно получить циклический вывод: $ваав \xrightarrow{1} аавв' \xrightarrow{1} авв'а' \xrightarrow{1} вв'а'а' \xrightarrow{1} в'а'а'в' \xrightarrow{2} \rightarrow а'а'в'в \xrightarrow{2} а'в'ва \xrightarrow{2} в'ваа \xrightarrow{2} ваав$. Все слова в этой цепочке называются сопряженными. Каждая совокупность сопряженных слов образует класс эквивалентности. Заметим, что среди всех слов, сопряженных со словом A , есть и A' . Поэтому подстановки 1) и 2) позволяют за несколько шагов переставлять слова из начала в конец.

Все слова, имеющие форму PQ' или $P'Q$, где $P, Q \in A^*$, сопряженные со словами из A^* и только с ними, называются регулярными. В классе эквивалентности, рассмотренном выше, не существует регулярных слов. А в классе эквивалентности $авва \rightarrow вваа' \rightarrow ваа'в \rightarrow аа'в'в \rightarrow а'в'в'а' \rightarrow в'в'а'а \rightarrow в'а'ав \rightarrow$

$\rightarrow a'avv \rightarrow avva$ все слова, кроме слов $avva$ и $a'v'v'a'$, являются регулярными.

Теорема 1.1. Всякая формула системы Π является формулой системы N .

Доказательство. Очевидно, что теорема справедлива для аксиомы C , так как она сопряжена с \bar{C} . Поскольку все формулы выводятся из аксиомы C , достаточно показать, что свойство «быть формулой» системы N сохраняется при применении подстановок системы Π .

Предположим, что формула X системы Π является формулой в системе N . Для того, чтобы X имела следствие в системе Π , необходимо, чтобы X могла быть представлена в виде $X = PA_iQ$. Тогда в соответствии со схемой подстановок в системе Π из X непосредственно следует $Y = PB_iQ$. На основе подстановок типа 1) системы N из X может быть получено слово A_iQP' , которое является формулой в системе N , так как оно сопряжено с X . Применяя к A_iQP_i подходящую подстановку типа 3) системы N , мы можем получить слово $QP'B'_i$, которое также является формулой в системе N . Слово $QP'B'_i$ сопряжено со словом Y . Действительно, $QP'B'_i \rightarrow P'B'_iQ' \rightarrow B'_iQ'P \rightarrow \rightarrow Q'PB_i \rightarrow PB_iQ$. Поэтому слово Y также является формулой системы N , что доказывает теорему 1.1.

Теорема 1.2. Всякая формула в системе N является регулярным словом, сопряженным с некоторой формулой системы Π .

Доказательство. Теорема справедлива для аксиомы \bar{C} , поскольку является регулярным словом в силу способа построения слова, сопряженного с аксиомой C . Поэтому достаточно доказать, что свойства регулярности и сопряженности с некоторой формулой системы Π сохраняются при использовании системы N . Сохранение указанных свойств очевидно для подстановок 1) и 2). Рассмотрим подстановки типа 3). Предположим, что формула A_iP регулярна и сопряжена с некоторой формулой системы Π . Из регулярности A_iP следует, что она может быть записана в виде $A_iP_1P'_2$, где P_1 и P_2 - некоторые слова в алфавите A . ***** сопряженное с A_iP слово в

системе Π - это слово $P_2A_iP_1$. Из сделанного предположения следует, что слово $P_2A_iP_1$ является формулой в системе Π . Однако $P_2A_iP_1$ сопряжено со словом $P_1P'_2B'_i$, которое может быть получено с помощью подстановки 3) из слова $A_iP_1P'_2$. Очевидно, что слово $P_1P'_2B'_i$ является регулярным. Тем самым теорема доказана.

Пример 1.4. Пусть дана полутуэвская система с алфавитом $A = \{a, v, c\}$, аксиомой которой служит слово $C = acv$, и подстановкой $PcQ \rightarrow PacvQ$, где, очевидно, $P = a$, $Q = v$. Эта система является моногенной и порождает формулы типа $a^n c v^n$, где $n = 1, 2, \dots$. Построим соответствующую ей нормальную систему. Ясно, что эта система имеет алфавит $B = \{a, v, c, a', v', c'\}$, аксиому $\bar{C} = cva'$ и подстановки

- 1) $aP \rightarrow Pa'$, $vP \rightarrow Pv'$, $cP \rightarrow Pc'$;
- 2) $a'P \rightarrow Pa$, $v'P \rightarrow Pv$, $c'P \rightarrow Pc$;
- 3) $cQP' \rightarrow QP'a'c'v'$.

Очевидно, что аксиома cva' принадлежит классу эквивалентности

$cva' \xrightarrow{1} va'c' \xrightarrow{2} a'c'v' \xrightarrow{3} c'v'a \xrightarrow{4} v'sa \xrightarrow{5} acv \xrightarrow{6}$. Применяя к аксиоме подстановку 3),

получим формулу $va'a'c'v'$, порождающую следующий класс эквивалентности:

$va'a'c'v' \xrightarrow{1} a'a'a'v'v' \xrightarrow{2} a'c'v'v'a \xrightarrow{3} c'v'v'aa \xrightarrow{4} v'v'aac \xrightarrow{5} v'aacv \xrightarrow{6} aacvv \xrightarrow{7} acvva' \xrightarrow{8} cvva'a \xrightarrow{9} vva'a'c' \xrightarrow{10}$.

Подстановка типа 3) может быть применена в этом классе только к слову $cva'a'$, в результате которой порождается слово $vva'a'c'v'$ и новый класс эквивалентности и так далее.

Граф, представляющий множество всех выводимых формул в обеих системах, показан на рис. 1.3.

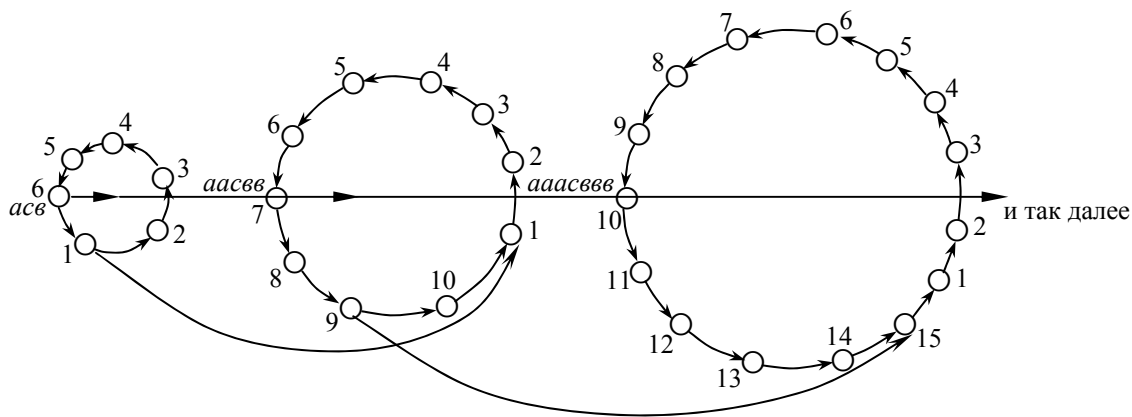


Рис. 1.3.

Таким образом, мы показали, что для любой полутуэвской системы можно построить нормальную систему, в классах эквивалентности которой содержатся все формулы исходной полутуэвской системы. Легко показать, что можно установить соответствие между полутуэвской и туэвской, нормальной и постовской, а следовательно, между туэвской и постовской системами. Поэтому результаты, полученные в какой-либо одной системе, можно распространять на любую другую комбинаторную систему.

Легко понять, что множество всех ***** слов в комбинаторной системе порождает ассоциативное исчисление в некотором алфавите A , а ее подстановки задают алфавитный оператор γ , отображающий слова из множества A^* в слова из A^* . Алфавитный оператор еще не является алгоритмом, поскольку в ассоциативном исчислении не определен порядок применения подстановок. Существует несколько способов задания однозначного порядка их применения: нормальные алгоритмы Маркова, вычислимые функции и машины Тьюринга.

1.4. Нормальные алгоритмы Маркова

Пусть задана полутуэвская система с алфавитом $A = \{a, v, c\}$ и системой подстановок: $cv \rightarrow a$; $av \rightarrow v$; $cva \rightarrow csa$; $ssa \rightarrow vaa$; $vaa \rightarrow csa$. Порядок их применения следующий. Исходя из произвольного слова P , система подстановок, просматривается в естественном порядке. Отыскивается первая подстановка, левая часть которой входит в P . Если таковой не известен, то

процесс обрывается; в противном случае (найдена подстановка), выполняется подстановка первого вхождения. В результате получаем ***** слово P_1 . Для P_1 выполняется аналогичная процедура и так далее до тех пор, пока на n -м шаге для слова P_n процесс не оборвется. Таким образом, имеем некоторый алгоритм переработки слов в алфавите A .

Этот алгоритм перерабатывает слова $P = свавсв$ в слово $вва$:

$$\underline{с}вавсв \rightarrow \underline{а}вавсв \rightarrow \underline{в}авсв \rightarrow \underline{вс}св \rightarrow \underline{вва}.$$

Дальнейшие подстановки невозможны. Однако слово $Q = свсвсавс$ не может быть переработано, так как получается бесконечная последовательность $\underline{с}всвсавс \rightarrow \underline{а}всва \rightarrow \underline{вс}сва \rightarrow \underline{всс}а \rightarrow \underline{вва}а \rightarrow \underline{вс}сва \dots$. Поэтому к слову Q алгоритм не применим.

В нормальном алгоритме Маркова порядок применения подстановок следующий. Исходя из произвольного слова P , все подстановки просматриваются в естественном порядке. Находится подстановка с левой частью, входящей в P . Если таковой нет, то процесс обрывается. В противном случае берется первая из таких подстановок и выполняется замена ее правой части вместо первого вхождения ее левой части в P , что дает новое слово P_1 . Для P_1 выполняется аналогичная процедура и так далее. Процесс оканчивается, когда получим слово P_n такое, что к нему не применима ни одна подстановка или когда применяется подстановка, объявленная последней.

Как видно в отличие от предыдущего алгоритма в нормальном алгоритме Маркова остановка может наступать в двух случаях.

Если в приведенной выше полутуэвской системе объявим подстановку $ваа \rightarrow сва$ последней, то ясно, что слово Q будет переработано в слово $всва$.

Пример 1.3. Пусть задан алфавит $A = \{1, +\}$, подстановка 1) $/+ \rightarrow +/$, 2) $+1 \rightarrow 1$; 3) $1 \rightarrow 1$ и исходное слово $1111+11+111$.

Применяя подстановки, начиная с первой, получаем последовательно слова:

$$1) 1111+11+111;$$

$$8) +111+111111;$$

- | | |
|-----------------|------------------|
| 2) 111+111+111; | 9) +11+1111111; |
| 3) 11+1111+111; | 10) +1+11111111; |
| 4) 1+11111+111; | 11) ++111111111; |
| 5) +111111+111; | 12) +111111111; |
| 6) +11111+1111; | 13) 111111111; |
| 7) +1111+11111; | 14) 111111111. |

Процедура окончена. Применена заключительная подстановка $1 \rightarrow 1$.

Гипотеза Маркова. Всякий алгоритм в алфавите A эквивалентен некоторому нормальному алгоритму в том же алфавите.

Эта гипотеза позволяет строго проводить доказательство алгоритмической неразрешимости того или иного круга проблем. В качестве примера приведем доказательство алгоритмической неразрешимости проблемы самоприменимости алгоритма.

Пусть в некотором алфавите $A = \{a_1, a_2, \dots, a_n\}$ задан нормальный алгоритм Γ . В записи подстановок, кроме букв алфавита A , содержатся символы \rightarrow и $,$ (запятая). Обозначив эти символы буквами a_{n+1} и a_{n+2} , получим возможность изображать алгоритм Γ словом в расширенном алфавите $A = \{a_1, a_2, \dots, a_n, a_{n+1}, a_{n+2}\}$. Применим теперь алгоритм Γ к слову, которое его изображает.

Если алгоритм Γ перерабатывает это слово в некоторое иное слово, после чего наступает остановка, то это означает, что алгоритм Γ применим к собственной записи. Такой алгоритм назовем самоприменимым. В противном случае алгоритм будем называть несамоприменимым. Естественно, возникает задача распознавания самоприменимости: по записи данного алгоритма определить, самоприменим этот алгоритм или нет.

Решение этой задачи можно представить в виде построения некоторого нормального алгоритма Δ , который, будучи применим ко всякой записи самоприменимого алгоритма Γ , перерабатывает эту запись в некоторое слово M , а применяемый ко всякой записи несамоприменимого алгоритма Γ , перерабатывает эту запись в некоторое иное слово L . В этом случае по

результату применения алгоритма Δ можно узнать, является ли заданный алгоритм Γ самоприменимым или нет.

Доказательство проведем от противного. Допустим, что алгоритм Δ построен. Тогда путем некоторого изменения системы подстановок алгоритма Δ можно построить новый алгоритм Δ' , который всякую запись несамоприменимого алгоритма по-прежнему перерабатывает в слово L , а ко всякой записи самоприменимого алгоритма неприменим (остановка никогда не наступает). Это приводит к противоречию. Действительно, если Δ' самоприменим, то он применим к собственной записи в виде слова (остановка наступает). Но по построению алгоритма Δ' это свидетельствует как раз о том, что Δ' несамоприменим. Если же Δ' несамоприменим, то по построению он применим к своей записи, так как он применим к любой записи несамоприменимого алгоритма. Но это как раз означает, что Δ' самоприменим.

Полученное противоречие показывает, что алгоритм Δ' не может быть построен. Отсюда вытекает, что проблема распознавания самоприменимости алгоритмически неразрешима.

Итак, решая какую-либо задачу, приходится считаться с тем, что алгоритм для ее решения может существовать, а может и не существовать. Поэтому важно как построение самого алгоритма, так и доказательство невозможности его построения. Иначе говоря, есть такие проблемы, которые нельзя решать только с помощью формальных рассуждений и вычислений и которые требуют творческого мышления (интуиции, предвосхищения и так далее).

В заключение раздела отметим, что нормальные алгоритмы Маркова являются хорошей моделью логических алгоритмов. Покажем, что любой логический алгоритм можно свести к вычислительному, т.е. любую алгоритмическую проблему можно свести к вычислению некоторой целочисленной функции от целочисленных значений аргументов.

Пусть все перерабатываемые алгоритмом Γ условия сведены в последовательность и пронумерованы целыми числами: A_0, A_1, \dots, A_n . Множество решений объединим в последовательность B_0, B_1, \dots, B_m .

Мы можем теперь сказать, что любой алгоритм, перерабатывающий запись A_n в B_m , сводится к вычислению значений числовой функции $m = y(n)$, в которой (после перенумерации) мы можем иметь дело только с номерами записей и решений, т.е. алгоритм перерабатывает номер записи условий в номер записи решений. Иначе говоря, мы имеем уже не логический, а численный алгоритм.

Очевидно, что если есть некоторый алгоритм, решающий исходную задачу, то есть и алгоритм вычисления значений соответствующей функции. Действительно, чтобы найти значение $y(n)$ при $n = n^*$, можно по n^* восстановить запись условий задачи. Затем с помощью имеющегося алгоритма найти запись решения и по записи решения определить номер m^* . Следовательно, $y(n^*) = m^*$. Обратно, если есть алгоритм вычисления функции $y(n)$, то, стало быть, имеется и алгоритм решения исходной задачи. Действительно, по записи условий задачи можно найти соответствующий ей номер n^* , затем вычислить и по m^* определить запись решения.

Таким образом, возможность построения любого алгоритма сводится к понятию вычислимости функции. Понятие вычислимости функции можно формализовать с помощью машин Тьюринга.

2. МАШИНЫ ТЬЮРИНГА

2.1. Состав и работа машины Тьюринга

Машина Тьюринга (МТ) состоит из устройства управления, головки и бесконечной ленты.

Устройство управления способно принимать некоторое конечное множество состояний. Лента разбита на ячейки, в которые заранее вносятся обрабатываемые данные, записанные в символах некоторого алфавита (в

простейшем случае в одну ячейку заносится один символ). Головка считывает данные из ячейки, которая находится прямо перед ней (такая ячейка называется рабочей) и осуществляет запись на ленту промежуточных результатов и сдвиг ленты вправо или влево (в простейшем случае - на одну ячейку).

Процесс работы МТ состоит в следующем. Устройство управления находится в одном из состояний, головка обзревает рабочую ячейку и считывает символ, записанный в ней. Устройство управления анализирует этот символ и выдает команду, в результате выполнения которой может быть изменено содержимое ячейки, а лента сдвинута вправо или влево. Конкретные действия определяются ситуацией, в которой находится машина, и командами перехода.

Пусть x_0, x_1, \dots, x_n - символы входного алфавита X , буквы которого могут быть записаны на ленте. Символ x_0 отождествляется с нулевым символом. Иногда он обозначается B и называется пробелом. Считается, что входное слово записано в ячейках ленты сплошным массивом, а все незанятые ячейки ленты заполнены пробелами.

Пусть $Q = \{q_0, q_1, \dots, q_m\}$ - алфавит внутренних состояний устройства управления МТ. Состояние q_0 - начальное состояние. Предполагается, что в исходном состоянии МТ находится в состоянии q_0 , а головка обзревает самый левый символ входной цепочки.

Пусть $I = \{l, r, s\}$ - множество инструкций управления движением ленты. Символ l означает сдвиг ленты на одну ячейку влево, r - вправо, s - отсутствие движения или остановка ленты. Заметим, что символы l и r можно употреблять в противоположном значении, если считать, что вместо ленты передвигается головка.

МТ условно можно представить так, как показано на рис. 2.1.

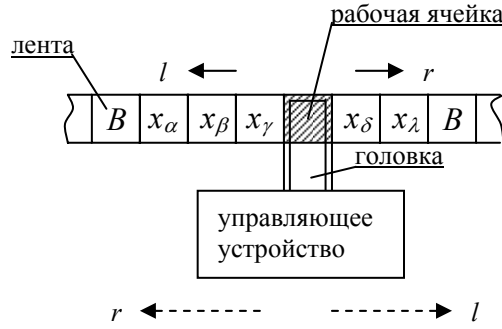


Рис. 2.1.

Ситуацией называется пара $q_i x_j$; $q_i \in Q$, $x_j \in X$. Очевидно, начальная ситуация МТ, указанная на рис. 2.1, есть $q_0 x_1$. Выражением будем называть конечную последовательность символов в алфавите $\{x_0, x_1, \dots, x_n, q_0, q_1, \dots, q_m, l, r, s\}$. Командой называется выражение одного из следующих типов: $q_i x_k x_l q_j$; $q_i x_k r q_j$; $q_i x_k l q_j$; $q_i x_k s q_j$.

Конечная последовательность команд, задающая функционирование МТ, называется таблицей Тьюринга. Если в этой таблице все команды имеют попарно различные ситуации, то соответствующая ей МТ называется детерминированной. Кроме того, функционирование МТ можно задать с помощью таблиц Айзермана, определяющих отображения $\delta : Q \times X$ в Q , $\mu : Q \times X$ в X и $\nu : Q \times X$ в I и называемых соответственно таблицей преобразования сигнала записи ленты и таблицей преобразования перемещения ленты. Эти три таблицы можно объединить в одну обобщенную таблицу.

Конфигурацией называется выражение вида $q_i x_j x_k \dots x_l$, в котором q_i не может занимать крайней правой позиции. Конфигурация однозначно определяет работу МТ. Пусть α, β - конфигурация некоторой МТ Z , а P, R - произвольные последовательности символов на ленте. МТ переходит из α в β ($\alpha \rightarrow \beta$) в одном из трех случаев:

1) по команде $q_i x_k x_l q_j$. В этом случае символ x_k в рабочей ячейке меняется на x_l , состояние q_i заменяется состоянием q_j , а лента остается без движения: $\alpha = P q_i l R$, $\beta = P q_j x_l R$;

2) по команде $q_i x_k l q_j$. Лента передвигается на одну ячейку влево, состояние q_i меняется на q_j : $\alpha = P q_i x_k R$, $\beta = P x_k q_j R$;

3) по команде $q_i x_k r q_j$. Лента передвигается на одну ячейку вправо, состояние q_i заменяется на q_j : $\alpha = P x_k q_j x_l R$, $\beta = P q_j x_k x_l R$.

Конфигурация α называется заключительной, если не существует такой конфигурации β , что $\alpha \rightarrow \beta$. Вычислением в МТ называется последовательность конфигураций $\alpha, \beta, \gamma, m, \dots, n, l$, таких, что $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$, $\gamma \rightarrow m$, \dots , $n \rightarrow l$, где l - заключительная конфигурация.

2.2. Анализ и синтез машин Тьюринга

Проблема анализ МТ состоит в том, чтобы для данной МТ определить в каждом шаге ее реакцию на заданное входное слово, т.е. в построении вычисления в МТ. Проведем анализ на примере конкретной МТ.

Пример 2.1. Пусть МТ задана таблицей Тьбрина:

$q_0 x_0 l q_0$; $q_1 x_0 l q_0$;
 $q_0 x_2 l q_0$; $q_1 x_2 l q_0$;
 $q_0 x_1 l q_1$; $q_2 x_1 x_2 q_2$.
 $q_1 x_1 r q_2$;

Заметим, что эту же МТ можно задать с помощью следующих таблиц Айзермана:

δ :

$Q \backslash x$	x_0	x_1	x_2
q_0	q_0	q_1	q_0
q_1	q_0	q_2	q_0
q_2	-	q_2	-

μ :

$Q \backslash x$	x_0	x_1	x_2
q_0	x_0	x_1	x_2
q_1	x_0	x_1	x_2
q_2	-	x_2	-

ν :

$Q \backslash x$	x_0	x_1	x_2
q_0	l	l	l
q_1	l	r	l
q_2	-	s	-

или с помощью обобщенной таблицы:

$Q \backslash x$	x_0	x_1	x_2
q_0	q_0, x_0, l	q_1, x_1, l	q_0, x_2, l

q_1	q_0, x_0, l	q_1, x_1, r	q_0, x_1, l
q_2	-	q_2, x_2, s	-

Пусть на ленте записано входное слово $x_1x_0x_2x_1x_1x_0x_1$, машина находится в начальном состоянии q_0 , и головка расположена против самого левого символа входного слова. Процесс анализа записывается следующими конфигурациями:

$q_0 x_1$	x_0	x_2	x_1	x_1	x_0	x_1
x_1	$q_1 x_0$	x_2	x_1	x_1	x_0	x_1
x_1	x_0	$q_0 x_2$	x_1	x_1	x_0	x_1
x_1	x_0	x_2	$q_0 x_1$	x_1	x_0	x_1
x_1	x_0	x_2	x_1	$q_1 x_1$	x_0	x_1
x_1	x_0	x_2	$q_1 x_1$	x_1	x_0	x_1
x_1	x_0	x_2	$q_2 x_1$	x_1	x_0	x_1

В последней конфигурации МТ останавливается, так как в таблице Тьюринга не существует команды для ситуации $q_2 x_2$ или, что то же самое, в таблицах Айзермана не определен переход $q_2 x_2$.

Таким образом, анализируемая МТ отсчитала четвертую букву входного слова (x_1), заменила ее на x_2 и остановилась.

Пусть теперь исходным словом будет слово $x_1x_2x_2$. Процесс вычисления выглядит следующим образом:

$q_0 x_1$	x_2	x_2	
x_1	$q_1 x_2$	x_2	
x_1	x_2	$q_0 x_2$	B
x_1	x_2	x_2	$q_0 B \quad B \dots$ и так далее

Так как ***, то в соответствии с первой командой МТ будет работать вечно и никогда не остановится.

Рассмотрим сущность проблемы синтеза МТ и идею проведения синтеза МТ на примере так называемых числовых МТ. В таких МТ входной алфавит состоит из двух символов: B и $|$ (палочка). Произвольные (целые) числа, представленные в числовой МТ, кодируются унарным кодом

следующим образом: $\bar{0} = |$, $\bar{1} = ||$, $\bar{2} = |||$, ..., $\bar{n} = \underbrace{*****}_{n=1}$ (черта сверху означает кодированное число).

Построим простейшую МТ, вычисляющую сумму $x + y$ любых двух целых неотрицательных чисел. На ленту занесем сначала первое число x , а затем через пробел (B) - второе число y . Вычисление начнем с крайней левой позиции, предположив, что машина находится в начальной ситуации q_0B . Будем считать, что результат получен, если на ленте осталось столько палочек, записанных не обязательно подряд, чему равно истинное значение суммы. Это значение таково: $x + y = \bar{x} + \bar{y} - 2$.

Для построения МТ используем следующую процедуру. Сначала выделяем первую палочку кода \bar{x} и стираем ее. Затем выделяем первую палочку кода \bar{y} и также стираем ее. В результате на ленте остается столько палочек, сколько их должно быть в числе $x + y$. Заметим, что тот же результат может быть получен другими способами, например, выделяя и вычеркивая первую палочку кода \bar{x} и последнюю в коде \bar{y} , две первые в коде \bar{x} и тому подобное. Исходя из этой идеи, можно синтезировать множество команд, определяющих требуемую МТ. Действительно, по команде q_0Blq_1 МТ перейдет из начальной ситуации q_0B в рабочее состояние q_1 , а в рабочей ячейке будет находиться первая палочка кода \bar{x} . Далее машина должна стереть эту палочку. Это выполняется командой $q_1|Bq_1$. Теперь машина находится в ситуации q_1B . Сдвиг ленты на одну ячейку влево приведет к ситуации $q_2|$. Это выполняется командой q_1Blq_2 . После выполнения команды $q_2|lq_2$ в рабочей ячейке скажется пробел B , предшествующий коду \bar{y} . Сдвиг ленты влево на одну ячейку по команде q_2Blq_3 приведет к ситуации $q_3|$. Команда $q_3|Bq_3$ сотрет первую палочку кода \bar{y} . В итоге мы построили следующую таблицу Тьюринга:

- | | |
|-----------------|-----------------|
| 1) q_0Blq_1 ; | 4) $q_2 lq_2$; |
| 2) $q_1 Bq_1$; | 5) q_2Blq_3 ; |
| 3) q_1Blq_2 ; | 6) $q_3 Bq_3$, |

которая задает МТ, выполняющую сложение чисел в унарном коде.

2.3. Диаграммы Тьюринга

В предыдущем разделе мы рассмотрели пример синтеза машины Тьюринга. Таблица этой МТ содержала шесть команд и описывала простейшие операции по просмотру массива унарного кода и стирание двух его элементов. При выполнении более сложных операций или использовании более сложных кодов число команд таблицы существенно возрастает, и поэтому усложняется процесс синтеза. Для облегчения синтеза используют композицию элементарных МТ. Набор элементарных МТ, как правило, ограничен, и все они имеют фиксированную систему команд. Если каким-то образом обозначить элементарные МТ и указать связи между ними, то результат их композиции и будет представлять собой диаграмму Тьюринга (ДТ).

Пусть $X = (x_0, x_1, \dots, x_n)$ - входной алфавит, r, l - сдвиг ленты соответственно вправо и влево, s - стоп, x_0 или B - пустая буква (пробел), X^* - множество слов над X , A - слово в алфавите X . Будем считать, что двухбуквенный алфавит X состоит из B и 1. Будем обозначать через \sim произвольную последовательность букв, несущественную в данном контексте, а через \sim - несущественную в данном контексте букву. Пусть \uparrow указывает букву, которую в данном шаге просматривает головка.

Приведем теперь несколько примеров простых МТ:

а) МТ x_i . Примененная к произвольной позиции, печатает в рабочей ячейке букву x_i . Очевидно, что таблица Тьюринга для такой МТ имеет вид:

$$q_0 x_0 x_i q_1,$$

$$q_0 x_n x_i q_1;$$

б) МТ r . Примененная к произвольной позиции, сдвигает ленту на одну ячейку вправо и останавливается;

в) аналогично работающая, но выполняющая сдвиг влево МТ l . Для МТ r и МТ l соответственно имеем таблицы Тьюринга

$q_0x_0rq_1$		$q_0x_0lq_1$
\vdots	и	\vdots
$q_0x_nrq_1$		$q_0x_nlx_1;$

г) МТ, выполняющая трехкратный сдвиг ленты вправо и печать на ней буквы x_0 :

$q_0x_0rq_1$	$q_2x_0rq_3$
\vdots	\vdots
$q_0x_nrq_1$	$q_2x_nrq_3$
$q_1x_0rq_1$	$q_3x_0x_0q_4$
\vdots	\vdots
$q_1x_nrq_2$	$q_3x_nx_0q_4;$

д) МТ K , переводящая начальную позицию $BAB \uparrow$ в позицию $BABAB \uparrow$.

1) q_0Brq_1	6) q_4Blq_5	11) $q_5B q_7$	16) q_2Blq_9
2) $q_1 rq_1$	7) $q_5 lq_5$	12) $q_7 rq_7$	17) $q_9 lq_9$
3) q_1Blq_2	8) q_5Blq_6	13) q_7Brq_8	
4) $q_2 Bq_3$	9) $q_6 lq_6$	14) $q_8 rq_8$	
5) q_3BBq_4	10) $q_6B q_7$	15) $q_8B q_2$	

Очевидно, что эта МТ копирует после пробела данное слово A .

Машины x_i , r , l , K примем за элементарные машины. Тогда, например, машину пункта г) можно собрать из трех машин r и машины x_0 .

Пусть M_1, \dots, M_n - некоторые элементарные МТ с общим алфавитом X . Диаграмма Тьюринга включает помимо символов элементарных машин символ \cdot (точка) и стрелки, взвешенные буквами алфавита X . Причем символ \cdot (точка) может встречаться в ДТ только один раз (он определяет начало работы), а из любого символа может выходить не более одной стрелки, взвешенной одной и той же буквой. Теперь можно заменить таблицу Тьюринга для МТ из примера г) диаграммой, показанной на рис. 2.2.

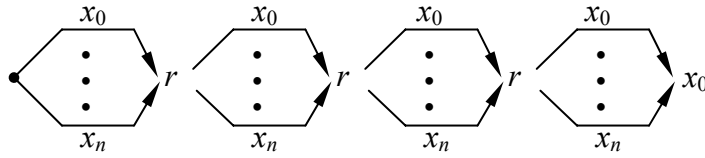


Рис. 2.2.

На рис. 2.3,а,б показаны диаграммы еще двух элементарных МТ R и L , выполняющих соответственно действия $\sim AB \sim \Rightarrow \sim AB \sim$ и $BAB \Rightarrow B AB$.

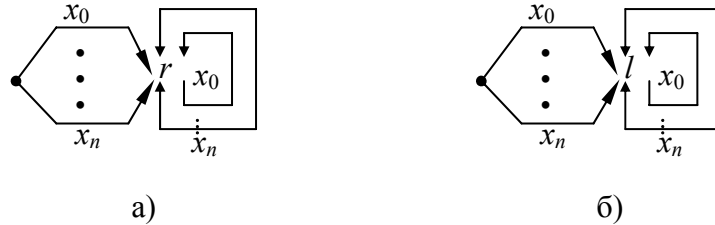


Рис. 2.3.

Изображение диаграмм можно упростить следующим образом. Заменяем все стрелки для всех $j = 0, 1, \dots, n$ одной стрелкой без букв. Если среди всевозможных стрелок для всех $j = 0, 1, \dots, n$ отсутствует лишь некоторая, взвешенная буквой x_j , то заменяем все стрелки одной, указав над ней $\neq x_j$. Опустим точку, если она соединена всеми стрелками только с одним символом. Опустим ненадписанные стрелки между двумя символами и будем писать M^4 вместо $\underbrace{M - M - \dots - M}_{n \text{ раз}}$. Тогда первая из приведенных выше диаграмм примет вид $r^3 x_6$, вторая $\boxed{\rightarrow r \mid}$, а третья $\boxed{\rightarrow l \mid}$.

Из приведенного способа построения ДТ следует, что переход от таблицы Тьюринга к ДТ осуществляется однозначно. Легко показать, что и обратный переход однозначен. Для этого необходимо выполнить следующее. Заменить символ каждой элементарной машины соответствующей ей таблицей. Определенным образом ввести переобозначение состояний в командах всех элементарных машин. Ввести сквозную нумерацию в полученной общей таблице.

2.4. Машины Тьюринга и вычислимые функции

Пусть заданы некоторая МТ Z с состояниями q_0, q_1, \dots, q_m и натуральный ряд чисел N (включая ноль). Поставим в соответствие каждой n -ке (a_1, a_2, \dots, a_n) , где $a_i \in N, i=1,2,\dots,n$, конфигурацию $\alpha_1 = q_0 a_1 a_2 \dots a_n$. Если для Z существует вычисление, начинающееся в конфигурации α_1 и доходящее до заключительной конфигурации α_p : $\alpha_1 \alpha_2 \dots \alpha_p$, то число $p \in N$, сопоставляемое α_p , есть значение функции от Z и начальной n -ки, т.е. $p = \psi_z(a_1, a_2, \dots, a_n)$. Если не существует такого числа p , что конфигурация α_p является заключительной, то функция ψ_z не определена для рассматриваемой n -ки и Z работает вечно. Таким образом, исходя из Z , мы пришли к функции с областью определения на множестве N^n (или, быть может, на некоторой подмножестве N^n), где N^n - множество всевозможных упорядоченных n -ок из N .

Будем исходить теперь из функций, определенных на N^n или его подмножестве. Функция f , определенная на некотором подмножестве множества N^n , является частично вычислимой, если существует МТ Z , такая, что для всякой n -ки (x_1, x_2, \dots, x_n) , которой отвечает некоторое значение f , выполняется равенство

$$f(x_1, x_2, \dots, x_n) = \psi_z(x_1, x_2, \dots, x_n).$$

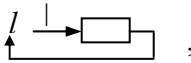
Функция f называется вычислимой, если она определена на N^n и является частично вычислимой.

Примером вычислимой функции является функция $f(x, y) = x + y$, поскольку в разделе 2.2 для нее мы построили МТ, исчисляющую значения $f(x, y)$ для любых x, y .

Пример 2.2. Покажем, что функция $f(x_1, x_2) = x_1 \times x_2$ является вычислимой. Доказательство заключается в построении МТ, заданной диаграммой. Пусть на ленте записано сначала \bar{x}_1 , а через пробел - \bar{x}_2 . Будем считать, что результат получен, если на ленте осталось столько подряд стоящих палочек, чему равно истинное значение произведения $x_1 \times x_2$. Итак, первоначально на ленте записано следующее:

$$\underbrace{B||\dots|B||\dots|B}_{\bar{x}_1} \quad \underbrace{B||\dots|B}_{\bar{x}_2} \quad B \uparrow$$

и положение головки указано стрелкой. МТ должна сначала перейти от кода \bar{x}_1 к числу x_1 и от кода \bar{x}_2 к числу x_2 . Это выполняется следующей ДТ: rBR^2lB .

В результате будет стерта первая палочка в \bar{x}_1 и последняя - в \bar{x}_2 . Затем машина копирует число x_2 столько раз, сколько имеется палочек в x_1 . Для этого используется цикл ,

при котором после стирания очередной палочки в x_1 копируется слово x_2 . Это выполняет ДТ в прямоугольнике, имеющая вид: BL^2KR^2 . В результате после того, как будет стерта последняя палочка в x_1 , а ленте будет записана последовательность $B \underbrace{B\dots B}_{x_1} B \underbrace{||\dots|B||\dots|B}_{x_2} B$. Теперь необходимо стереть число x_2 .

Это выполняет ДТ .

Для перевода головки в крайнее правое положение используем ДТ: L . Объединяя все приведенные диаграммы в одну, получим ДТ, показанную на рис. 2.4.

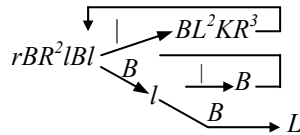


Рис. 2.4

Таким образом, на ленте будет записано число $B \underbrace{||\dots|B}_{x_1 \times x_2}$. Полученное

построение доказывает вычислимость функции $f(x_1, x_2) = x_1 \times x_2$.

Пример 2.3. Покажем, что функция $f(x_3, x_4) = x_3 - x_4$, где $x_3, x_4 \in N$, а пары $(x_3, x_4) \in N^2$, является частично вычислимой.

Предположим, что $x_3 \geq x_4$. Построим МТ, выполняющую вычитание при указанном условии. Пусть на ленте расположены коды \bar{x}_3 и \bar{x}_4 , разделенные пробелом и головка обзореваает пробел перед \bar{x}_3 . Организуем работу машины таким образом, чтобы она выполняла последовательные циклы. В каждом цикле машина просматривает записи на ленте слева направо и стирает самую левую палочку в \bar{x}_3 и самую правую в \bar{x}_4 , а затем

2.5. Гёделевский номер машины Тьюринга

Пусть p_1, p_2, \dots, p_k - соответственно первое, второе и так далее k -е простое число. Из теории чисел известно, что любое целое неотрицательное число $n \geq 1$ можно однозначно представить в форме:

$$n = p_1^{a_1} \times p_2^{a_2} \times \dots \times p_k^{a_k},$$

где a_1, a_2, \dots, a_k - некоторые целые неотрицательные числа, включая ноль. Например, $13 = 2^0 \cdot 3^0 \cdot 5^0 \cdot 7^0 \cdot 11^0 \cdot 13^1$, $60 = 2^2 \cdot 3^1 \cdot 5^1$. Очевидно, что для любого n однозначно определяется набор a_1, a_2, \dots, a_k и, наоборот, по любому набору a_1, a_2, \dots, a_k однозначно восстанавливается соответствующее ему число n .

Если для произвольного слова в некотором алфавите ввести определенным образом числовое кодирование входящих в него букв, то для данного слова можно найти однозначно определяющее его число n , которое по сути дела будет являться его числовым именем. В частности, каждой команде или набору команд, задающих МТ. можно сопоставить некоторое целое положительное число. Это число называется гёделевским номером МТ по имени математика Курта Геделя, который впервые применил числовое кодирование для обозначения нечисловых объектов в доказательстве своей знаменитой теоремы о неполноте арифметики.

Гедель предложил использовать следующий способ кодирования цепочек. Поясним его идею на примере кодирования команд МТ. Пусть задана команда $R = q_1 x_1 l q_2$. Для того, чтобы закодировать эту команду, необходимо ввести кодирование символов и мест, которые они занимают в команде. Будем кодировать символы нечетными числами большими единицы:

l	r	s	x_0	q_0	x_1	q_1	x_2	q_2	x_3	q_3	x_4	q_4	\dots
3	5	7	9	11	13	15	17	19	21	23	25	27	\dots

а номера мест - простыми числами:

1	2	3	4	\dots	k	\dots
2	3	5	7	\dots	p_k	\dots

Тогда команде Q будет однозначно соответствовать положительное число $nq(Q) = 2^{15} \cdot 3^{13} \cdot 5^3 \cdot 7^{19}$, которое и есть гёделевский номер команды Q .

Если задана упорядоченная последовательность команд $Q_1, Q_2, \dots, Q_j, \dots, Q_n$, которая задает некоторую МТ Z , то эта последовательность может быть закодирована следующим числом:

$$nq(Z) = 2^{nq(Q_1)} \times 3^{nq(Q_2)} \times \dots \times p_j^{nq(Q_j)} \times \dots \times p_n^{nq(Q_n)},$$

которое представляет собой гёделевский номер МТ Z .

Пример 2.4. Обратимся к синтезированной в разделе 2.2 МТ Z , выполняющей сложение двух чисел. Учитывая, что $B = x_0$, а $| = x_1$, получаем:

$$\begin{aligned} Q_1 &= q_0 x_0 l q_1; & nq(Q_1) &= 2^{11} \cdot 3^9 \cdot 5^3 \cdot 7^{15}; \\ Q_2 &= q_1 x_1 x_0 q_1; & nq(Q_2) &= 2^{15} \cdot 3^{13} \cdot 5^9 \cdot 7^{15}; \\ Q_3 &= q_0 x_0 l q_2; & nq(Q_3) &= 2^{15} \cdot 3^9 \cdot 5^3 \cdot 7^{19}; \\ Q_4 &= q_2 x_1 l q_2; & nq(Q_4) &= 2^{19} \cdot 3^{13} \cdot 5^3 \cdot 7^{19}; \\ Q_5 &= q_2 x_0 l q_3; & nq(Q_5) &= 2^{19} \cdot 3^9 \cdot 5^3 \cdot 7^{23}; \\ Q_6 &= q_3 x_1 x_0 q_3; & nq(Q_6) &= 2^{23} \cdot 3^{13} \cdot 5^9 \cdot 7^{23}. \end{aligned}$$

Поскольку гёделевские номера оказались упорядоченными в соответствии с нумерацией команд, гёделевский номер МТ Z равен:

$$nq(Z) = 2^{2^{11} \cdot 3^6 \cdot 5^3 \cdot 7^{15}} \times 3^{2^{15} \cdot 3^{13} \cdot 5^9 \cdot 7^{15}} \times 5^{2^{15} \cdot 3^9 \cdot 5^3 \cdot 7^{19}} \times 7^{2^{19} \cdot 3^{13} \cdot 5^3 \cdot 7^{19}} \times 11^{2^{19} \cdot 3^9 \cdot 5^3 \cdot 7^{23}} \times 13^{2^{23} \cdot 3^{13} \cdot 5^9 \cdot 7^{23}}.$$

Таким образом, каждая МТ получает свой гёделевский номер. Поскольку процедура приписывания номеров является механической, можно поручить ее в полнение некоторой МТ G_1 . Замети м, что G_1 вычисляет, в частности, и свой собственный гёделевский номер $nq(Q_1)$. Обратно, существует такая МТ G_2 , которая для любого положительного числа выдает один из двух следующих ответов:

- n не является гёделевским номером никакой МТ;
- n является гёделевским номером МТ Z , имеющей следующие команды....

Отметим, что если G_2 получает на вход число $n = nq(G_2)$, то она выдает ответ второго типа и печатает набор своих команд (точнее говоря, коды команд).

2.6. Рекурсивный и рекурсивно перечислимые множества

Рассмотрим некоторую функцию $f(x_1, x_2, \dots, x_n) = \psi_Z(x_1, x_2, \dots, x_n)$. По определению она частично вычислима, если существует МТ Z , которая эту функцию вычисляет. Так как машине Z однозначно сопоставлен геделевский номер $z = nq(Z)$, то всякой частично вычислимой функции $f(x_1, x_2, \dots, x_n)$ можно сопоставить геделевский номер именно той МТ, которая эту функцию вычисляет.

Введем функцию $\Phi_z(x_1, x_2, \dots, x_n)$, где z - целое неотрицательное число, а (x_1, x_2, \dots, x_n) - некоторая n -ка из N^n , следующим образом. Если $z = nq(Z)$, причем Z - МТ, вычисляющая значения f на n -ке (x_1, x_2, \dots, x_n) , то значение функции $\Phi_z(x_1, x_2, \dots, x_n)$ совпадают со значениями функции $f(x_1, x_2, \dots, x_n)$. Если не является геделевским номером никакой МТ, то значение функции $(x_1, x_2, \dots, x_n) = 0$. В результате мы получили вычислимую функцию, которая и можем теперь записать последовательность $\Phi_0, \Phi_1, \dots, \Phi_n, \dots$, которая перечисляет (быть может с повторением) множество частично вычислимых функций от n аргументов. Таким образом, мы можем в принципе перечислять области определения частично вычислимых функций.

Пусть заданы теперь произвольные z и (x_1, x_2, \dots, x_n) . Тогда, чтобы вычислить значения функции $\Phi_z(x_1, x_2, \dots, x_n)$, необходимо сначала выяснить, является ли z геделевским номером какой-либо МТ. Эту задачу решает МТ G_2 , введенная в разделе 2.5. Если z не является геделевским номером, то $\Phi_z(x_1, x_2, \dots, x_n) = 0$. В противном случае МТ Z , команды которой распечатываются МТ G_2 , предлагается (x_1, x_2, \dots, x_n) в качестве исходного задания. Соединяя G_2 и Z , мы получаем универсальную МТ, вычисляющую функцию $\Phi_z(x_1, x_2, \dots, x_n)$. Если на ленте этой машины поместить число $z = nq(Z)$, то она сможет вычислять функцию, соответствующую этому числу.

Пусть x - произвольный элемент множества M и $A \subset M$. Характеристическая функция подмножества A , обозначаемая C_A , определяется следующим

$$C_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ 0, & \text{если } x \notin A. \end{cases}$$

Будем говорить, что множество называется рекурсивным, если его характеристическая функция вычислима. Если множество рекурсивно, то существует МТ, которая, получив на вход элемент этого множества, всегда ставит ему в соответствие некоторый ответ: либо 1 (и тогда $x \in A$), либо 0 ($x \notin A$).

Будем говорить, что множество называется рекурсивно перечислимым, если оно является областью определения некоторой частично вычислимой функции. Теоретически можно перечислить частично вычислимые функции от одной целочисленной переменной, от двух целочисленных переменных и так далее. Следовательно, в принципе мы умеем перечислять рекурсивно перечислимые множества: $G_0, G_1, \dots, G_i, \dots$

Теорема 2.1. Множество является рекурсивным тогда и только тогда, когда оно само и есть дополнение рекурсивно перечислимы.

Доказательство. Пусть M - произвольное множество, $A \subset M$ и A является рекурсивным множеством. Тогда характеристическая функция C_A множества A является вычислимой определению рекурсивности множества A . Представим $C_A(x)$ в виде объединения двух функций:

$$C'_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ \text{не определена,} & \text{если } x \in A, \end{cases} \quad C''_A(x) = \begin{cases} 0, & \text{если } x \in A \text{ (т.е. } x \in M \setminus A), \\ \text{не определена,} & \text{если } x \in A. \end{cases}$$

Каждая из указанных функций является частично вычислимой функцией по определению частично вычислимой функции. Отсюда на основании определения рекурсивно перечислимых множеств множества A и \bar{A} являются рекурсивно перечислимыми множествами.

Предположим теперь, что множества A и \bar{A} рекурсивно перечислимы. Тогда они являются областями определения частично вычислимых функций

$f(x)$ и $\bar{f}(x)$, совпадающих соответственно с $C'_A(x)$ и $C'_{\bar{A}}(x)$. Для этих функций можно построить вычисляющие их МТ соответственно Z и \bar{Z} . Пусть на входы машин Z и \bar{Z} поступает элемент $x \in M$. Тогда одна из машин обязательно выдаст некоторый результат, а другая нет. Действительно, если результат дает машина Z , то $C'_A(x) = 1$ и $x \in \bar{A}$. Если же результат дает машина \bar{Z} , то $C'_{\bar{A}}(x) = 0$ и $x \in \bar{A}$. Пара машин Z, \bar{Z} эквивалентна некоторой одной МТ Z' , вычисляющей значения функции $C_A(x)$. Поэтому множество является рекурсивным. Более того, множество \bar{A} также рекурсивно, поскольку Z' вычисляет и значения характеристической функции множества \bar{A} :

$$C_{\bar{A}}(x) = \begin{cases} 0, & \text{если } x \in A, \\ 1, & \text{если, } x \notin A (\text{т.е. } x \notin M \vee x \in A). \end{cases}$$

Этим теорема 2.1 доказана.

Теорема 2.1 утверждает, что любое рекурсивное множество является рекурсивно перечислимым, но оставляет открытым вопрос о том, любое ли рекурсивно перечислимое множество является рекурсивным? Ниже будет показано, что существуют множества, которые являются рекурсивно перечислимыми, но не рекурсивными.

2.7. Неразрешимость проблемы остановки для произвольных машин Тьюринга

Сформулируем следующую проблему: существует ли машина Тьюринга Z , для которой входными заданиями являются пары (z, x) , где $z = nq(Z)$, а x - произвольное натуральное число, такая, что при поступлении на вход МТ Z исходных данных x , она отвечала бы: либо «да, ТМ Z применима к x , т.е. начав вычисления с x , выдает некоторый результат и остановится», либо «нет, предлагать МТ Z заданные x не следует, так как она никогда не остановится». (В этом случае говорят, что МТ Z не применима к

х.) Эта проблема известна под названием проблемы остановки для произвольных машин Тьюринга.

Теорема 2.2. Проблема остановки для произвольных машин Тьюринга алгоритмически неразрешима.

Для доказательства теоремы 2.2 покажем вначале, что существуют множества, которые являются рекурсивно перечислимыми, но не рекурсивными.

Рассмотрим множество машин Тьюринга, вычисляющих функции от одного аргумента. Каждой из этих машин Z будем предлагать в качестве исходного задания ее собственный геделевский номер $z = nq(Z)$. Если машина Z остановится и выдаст некоторый результат, то будем называть ее самоприменимой. Если машина Z никогда не остановится, то такую машину будем называть несамоприменимой. Таким образом, на множестве всех машин мы определили некоторую функцию от z , которая, очевидно, есть частично вычислимая функция. Для нее можно построить конкретную МТ, вычисляющую эту функцию. Для этого необходимо построить сначала универсальную МТ, вычисляющую значения указанной функции для любых чисел. В результате множество G геделевских номеров самоприменимых машин оказывается рекурсивно перечислимым множеством.

Предположим теперь, что множество G является рекурсивным. Это равносильно рекурсивной перечислимости его дополнения \bar{G} . В таком случае должна существовать МТ, перечисляющая \bar{G} в перечислении рекурсивно перечислимых множеств $\Gamma_0, \Gamma_1, \dots, \Gamma_i, \dots$. Предположим, что множество \bar{G} имеет в этом перечислении номер λ . Поэтому запишем $\bar{G} = \Gamma_\lambda$. Тогда для любого натурального числа $x \in \bar{G}$ имеет место $x \in \Gamma_\lambda$ и наоборот, т.е. $x \in G \leftrightarrow x \in \Gamma_\lambda$, где \leftrightarrow - знак логической эквивалентности или равнозначности. Однако из определения самоприменимости следует $x \in \bar{G} \leftrightarrow x \notin \Gamma_x$, где Γ_x - множество самоприменимых машин, имеющих в

перечислении $\Gamma_0, \Gamma_1, \dots, \Gamma_i, \dots$ номер x . Отсюда вытекает $x \in \Gamma_\lambda \leftrightarrow x \notin \Gamma_x$. Ести теперь положить, что $x = \lambda$, то имеем

$$\lambda \in \Gamma_\lambda \leftrightarrow \lambda \notin \Gamma_\lambda.$$

Полученное противоречие доказывает, что множество G не является рекурсивным. Итак, действительно, существуют множества, которые являются рекурсивно перечислимыми, но не рекурсивными.

Предположим теперь, что существует машина T , которая для произвольной пары (z, x) решает проблему остановки. Пусть G - рекурсивно перечислимое, но не рекурсивное множество. Множество G есть область определения некоторой частично вычислимой функции $f(x) = \psi_z(x)$, вычисляемой машины Z . Тогда машина T для любого натурального числа x даст один из двух ответов: «да, $x \in G$ и Z остановится», «нет, $x \notin G$ и Z никогда не остановится». Это означало бы, что G - рекурсивное, а не рекурсивно перечислимое множество. Получим противоречие, так как мы предполагали, что G - рекурсивно перечислимое, а не рекурсивное множество. Этим теорема 2.2 доказана.

Нетрудно видеть, что теорема 2.2 эквивалентна доказанному в разделе 1.4 утверждению о том, что проблема распознавания самоприменимости алгоритма неразрешима.

Легко видеть, что существует аналогия между универсальной машиной Тьюринга и универсальной ЭВМ. В этом случае проблема остановки обретает следующий конкретный смысл. Пусть имеется некоторая программа z и исходные данные x , которые вводятся в ЭВМ. Спрашивается, существует ли такая общая программа P , которая позволяет заранее выяснить, остановится ли ЭВМ, начав вычисление по программе с исходными данными x , или нет? На основании теоремы 2.2 ответ является отрицательным: подобной программы P не существует. Это означает, что никогда не будет написана программа, с помощью которой можно было бы обнаруживать в произвольных программах ошибки, приводящие к бесконечным вычислениям.

В заключение раздела отметим, что понятия вычислимости и рекурсивности эквивалентны. Понятие вычислимости ввел А. Тьюринг, а понятие рекурсивности - С. Клини. Оба эти понятия, определенные их авторами независимо друг от друга, были призваны формализовать интуитивное представление о вычислимости. Примечательно, что все другие понятия, введенные с той же целью также независимо друг от друга К.Геделем, А. Черчем, Э. Постом, А. Марковым, оказались эквивалентными друг другу. Это позволило А. Черчу сформулировать свой знаменитый тезис о том, что все введенные понятия правильно формализуют интуитивное представление об алгоритме и вычислимости и что их нельзя обобщить. Поэтому из всех эквивалентных определений можно выбрать одно, отождествить его с одним из точных определений и на основании последнего строить некоторую теорию.

2.8. Комбинаторные системы и машины Тьюринга

Покажем, что можно построить машину Тьюринга, в которой любое вычисление, приводящее к конечному результату, соответствует выводу некоторой теоремы в комбинаторной системе и наоборот. Пусть имеется некоторое входное слово n , заданное унарным кодом \bar{n} (см. раздел 2.2). Если предложить машине Z в качестве исходного задания \bar{n} , то начав вычисление с конфигурации q_0n , машина либо выдаст результат, либо будет работать вечно. Будем считать, что Z включает все промежуточные результаты вычислений между двумя маркерами h . Паре (Z, n) поставим в соответствие полутуэвскую систему Π с аксиомой $hq_0\bar{n}h$. Подстановки в Π выберем таким образом, чтобы ее промежуточные формулы соответствовали промежуточным конфигурациям МТ Z , а заключительная формула hsh - заключительной конфигурации МТ Z , означающей, что соответствующее вычисление завершено.

Теорема 2.3. Если МТ Z с начальным состоянием q_0 применима к входному слову n , то это эквивалентно тому, что в полутуэвской системе Π с аксиомой $hq_0\bar{n}h$ выводима формула hsh .

При доказательстве теоремы 2.3 ограничимся только доказательством того, что, если ТМ Z применима к n , то из этого следует, что в полутуэвской системе Π выводима формула hsh . Обратное доказательство проводить не будем.

Для этого покажем, как по командам МТ Z построить полутуэвскую систему Π . Алфавит системы Π образуется объединением входного алфавита состояний МТ Z и маркера h . Аксиомой системы является начальная ситуация МТ Z $q_0\bar{n}$ с маркерами h слева и справа, т.е. $hq_0\bar{n}h$. Команде МТ Z $q_ix_jx_kq_l$ в системе Π сопоставляется подстановка $Pq_ix_jQ \rightarrow Pq_lx_kQ$, которая обеспечивает переход от конфигурации, которая имела место в МТ Z до применения команды, к конфигурации, в которую переходит МТ Z после применения команды. Здесь P и Q - некоторые слова (контекст). Команде $q_ix_jzq_l$ соответствует подстановка $Px_kq_ix_jQ \rightarrow Pq_lx_kx_jQ$. Команде $q_ix_jlq_l$ соответствует подстановка $Pq_ix_jQ \rightarrow Px_jq_lx_lQ$.

Кроме того, введем в систему Π дополнительные подстановки, которые не соответствуют никаким командам МТ Z и применяются только тогда, когда МТ Z переходит в заключительную конфигурацию и останавливается (в этом случае МТ Z встретила ситуацию q_ix_j , которой нет ни в одной команде или выполнила команду $q_ix_jsq_0$). Подстановки $hPq_ix_jQh \rightarrow hPq_ix_jQh$, которые для всех пар (q_ix_j) , на встречающихся в левых частях команд МТ Z , заменяют символ состояния q_i маркером q . Подстановки $hPq_x_jQh \rightarrow hPqQh$, стирающие все символы (кроме h) справа от q . Подстановки $hPqh \rightarrow hPsh$ вводят символ s слева от правого h . Подстановки $hPx_jsh \rightarrow hPsh$ стирают все символы между левым h и s . И, наконец, подстановки $hx_0sh \rightarrow hsh$, которые перемещают левый маркер h в позицию, предшествующую символу s .

Пока МТ Z выполняет вычисления над числами n в соответствии с командами, начиная с аксиомы $hq_0\bar{n}h$ формируется вывод в полутуэвской системе по принципу: одна переменная команда МТ Z - один шаг вывода в системе. Как только МТ Z остановилась, к полученной формуле системы применяются дополнительные подстановки, которые в конечном итоге приводят к формулам hsh . Тем самым теорема доказана.

Заметим, что в системе Π от числа n зависит только аксиома $hq_0\bar{n}h$, а алфавит и подстановки от n не зависят. Если n принимает значения из множества целых неотрицательных чисел, то действуя таким образом, как указано в доказательстве теоремы 2.3, мы получим семейство полутуэвских систем типа Π . С помощью этого семейства множеству тех чисел n , к которым применима МТ Z , ставится в соответствие единственная формула hsh . Это наводит на мысль о том, что можно построить систему Π , которая, исходя из аксиомы hsh , выдавала бы в качестве заключительных формул все целые числа, к которым применима МТ Z . Действительно, для построения в качестве аксиомы можно выбрать hsh , а в качестве системы подстановок - подстановки, обратные подстановкам системы Π . Если объединить системы Π и Π' (при этом в качестве аксиомы объединенной системы принять hsh), то легко видеть, что МТ Z будет соответствовать туэвская система T .

В заключение раздела отметим следующее. Построенные комбинаторные системы Π , Π' и T , соответствующие МТ Z , сопоставляют входным словам машины Z формулы в указанных системах. Выбрав в качестве Z такую машину, для которой неразрешима проблема остановки, мы приходим к выводу о том, в соответствующих полутуэвской и туэвской системах неразрешима проблема доказуемости. Суть этой проблемы состоит в следующем: можно ли построить алгоритм, который бы однозначно определял, является или нет некоторое слово формулой данной системы? Очевидно, что проблема доказуемости неразрешима и для нормальных и постовских систем.

Обратимся теперь к проблеме эквивалентности слов в комбинаторной системе, сущность которой была рассмотрена в разделе 1.1. Ясно, что установить, эквивалентно ли слово A слову B , или выяснить, является ли B формулой данной туэвской системы, равносильно. Возьмем построенную выше туэвскую систему с неразрешимой проблемой доказуемости и поставим ей в соответствие ассоциативное исчисление, полученное путем замены каждой пары туэвских подстановок соответствующим соотношением Туэ. Легко видеть, что для этого ассоциативного исчисления проблема эквивалентности слов неразрешима.

3. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ЯЗЫКИ

3.1. Порождающие грамматики

Любой формальный язык можно рассматривать как набор правильных предложений, построенных по правилам синтаксиса языка и имеющих определенный смысл. Для описания синтаксиса средств самого языка не хватает. Поэтому для этой цели используют метаязыки, в которые помимо символов самого языка входят так называемые метасинтаксические (или металингвистические) переменные. Язык можно задать двумя методами: описанием процедуры построения правильных конструкций (этот процесс называется порождением) или распознаванием правильности конструкции. Наиболее часто используют первый метод. Второй метод, как правило, реализуется аппаратом распознающих автоматов, которые являются разновидностью машин Тьюринга с различными ограничениями.

Для описания синтаксиса алгоритмических языков можно использовать язык нормальных форм Бэкуса (НФБ). Этот язык был введен для описания синтаксиса языка Алгол-60. Его развитие использовано для описания языков Фортран, ПЛ/1, Алгол-68. НФБ позволяет формально строить предложения Алгола-60. Помимо символов моделируемого языка НФБ включает следующие металингвистические переменные: $::=$ - связка, которая означает

«быть», «равно по определению», «это есть»; разделитель $|$, означающий «или»; $< >$ - скобки, в которых указываются конструкции языка.

Пример записи в НФБ конструкции «идентификатор» Алгола-60:

$$\begin{aligned} \langle \text{идентификатор} \rangle &::= \langle \text{буква} \rangle | \langle \text{буква} \rangle \langle \text{цифра} \rangle | \langle \text{идентификатор} \rangle \langle \text{буква} \rangle \\ &| \langle \text{идентификатор} \rangle \langle \text{цифра} \rangle \\ \langle \text{буква} \rangle &::= a | \dots | z \\ \langle \text{цифра} \rangle &::= 0 | 1 | \dots | 9. \end{aligned}$$

Вместо названий конструкций в скобках иногда пишут обозначающие их большие латинские буквы, а вместо связки $::=$ - знак \rightarrow . Тогда, если ввести обозначения: $\langle \text{идентификатор} \rangle = I$; $\langle \text{цифра} \rangle = Ц$; $\langle \text{буква} \rangle = Б$, а вместо разделителя $|$ записать соответствующее

$$\begin{array}{llll} I \rightarrow Б | БЦ | IB | IC & & I \rightarrow Б & B \rightarrow a & Ц \rightarrow 0 \\ B \rightarrow a | \dots | Z & \text{или} & I \rightarrow БЦ & \vdots & \vdots \\ Ц \rightarrow 0 | \dots | 9 & & I \rightarrow IC & B \rightarrow Z & Ц \rightarrow 9 \\ & & I \rightarrow IB & & \end{array}$$

Особенностью некоторых правил в представлении их в виде НФБ является то, что для описания некоторых конструкций используются сами описываемые конструкции. Говорят, что в этом случае имеет место рекурсия.

Теперь, когда мы имеем язык для формального описания синтаксиса, определим понятие порождающей грамматики.

Порождающей грамматикой G называется четверка

$$G = \{V_T, V_A, S, R\},$$

где V_T - конечный терминальный или основной алфавит (словарь), V_A - конечный вспомогательный или нетерминальный алфавит (словарь), S - аксиома, $S \in V_A$, R - конечное множество правил вывода вида $\xi \rightarrow \eta$, причем $\xi, \eta \in \{V_A \cup V_T\}^*$. Здесь, как и ранее, V_A^* , V_T^* - множества слов в алфавитах V_A , V_T соответственно. Предполагается, что из S выводима любая синтаксически правильная цепочка языка. Для Алгола-60 аксиомой является символ S в правиле $S ::= \langle \text{программа} \rangle$.

Введем некоторые определения. Говорят, что цепочка ω_1 непосредственно выводима из цепочки ω , если $\xi_1 \omega \xi_2 \rightarrow \xi_1 \omega_1 \xi_2$, где $\xi_1, \xi_2 \in \{V_A \cup V_T\}$. Говорят, что цепочка ω_N выводима из цепочки ω_0 , если существует последовательность непосредственных выводов: $\omega_i \rightarrow \omega_{i+1}$, $i = \overline{0, n-1}$. В этом случае будем записывать

$$\omega_0 \xrightarrow{G} \omega_N.$$

Если $\omega_0 = S$, то говорят, что цепочка ω_N выводима из аксиомы.

Терминальную цепочку, т.е. цепочку символов терминального алфавита, выводимую из некоторого нетерминального символа, будем называть терминальным порождением этого символа. Особо подчеркнем, что множество всех терминальных цепочек грамматики G , выводимых из аксиомы S , образует язык $L(G)$, порождаемый этой грамматикой.

Одной из основных проблем, связанных с использованием грамматик является проблема распознавания. Эта проблема разрешима, если существует такой алгоритм, который за конечное число шагов дает ответ на вопрос: входит ли цепочка основных символов языка в язык, порождаемый этой грамматикой? Если число шагов такого алгоритма можно подсчитать до его выполнения (такой алгоритм называется эффективным), то язык называется легко распознаваемым.

Длиной цепочки $l(\omega)$ называется число ее символов. Если $l(\omega) = 0$, то мы имеем пустую цепочку, которая обозначается λ . Порождающая грамматика называется неукорачивающей, если для всех ее правил $\xi \rightarrow \eta$ имеет место $l(\xi) \leq l(\eta)$.

Американским математиком Н. Хомским была предложена следующая классификация порождающих грамматик.

1-й класс составляют грамматики непосредственных составляющих (НС-грамматики). НС-грамматики - это неукорачивающие грамматики с правилами вывода типа $\xi_1 A \xi_2 \rightarrow \xi_1 \eta \xi_2$, где ξ_1, ξ_2, η - произвольные цепочки из

множества $\{V_A \cup V_T\}^*$, $A \in V_A$ и $\eta \neq \lambda$; ξ_1 и ξ_2 называют соответственно левым и правым контекстом. Поэтому НС-грамматики называют еще контекстно зависимыми грамматиками. Класс языков, порождаемых НС-грамматиками, совпадает с классом языков, порождаемых неукорачивающими грамматиками.

2-й класс составляют контекстно-свободные (КС) грамматики, или бесконтекстные грамматики, которые являются сужением класса НС-грамматик. КС-грамматики - это неукорачивающие грамматики с правилами вывода типа $A \rightarrow \eta$, в которых $A \in V_A$, $\eta \in \{V_A \cup V_T\}^*$, причем $\eta \neq \lambda$. КС-грамматики порождают КС-языки, являющиеся хорошей моделью языков программирования.

3-й класс составляют укорачивающие КС-грамматики (УКС-грамматики). Правила вывода УКС-грамматик имеют тот же вид, что и в КС-грамматиках, но цепочка η может быть пустой.

4-й класс образуют автоматные грамматики (А-грамматики). Правила вывода в них наиболее просты и имеют вид $A \rightarrow vB$ и $A \rightarrow v$, где $A, B \in V_A$, $v \in V_T$. Такие А-грамматики называются правосторонними. Если грамматики содержат правила $A \rightarrow Bv$, то они называются левосторонними. В них могут включаться также правила вывода $A \rightarrow \lambda$. А-грамматики порождают А-языки и интересны тем, что с их помощью можно описать преобразователи информации, называемые конечными автоматами.

Приведенная классификация, естественно, не охватывает все типы порождающих грамматик. Существуют промежуточные классы, например, между НС- и КС-грамматиками, между КС- и А-грамматиками. Однако, специально на этом мы останавливаться не будем.

3.2. Контекстно-свободные грамматики и языки

КС-грамматике соответствует полутуэвская система, в которой имеются терминальные и нетерминальные формулы. Вывод их можно представить с помощью графа.

Пусть задана КС-грамматика $G_m = \{V_T, V_A, S, R\}$, где $V_A = \{A\}$, а правила вывода R имеют вид: 1) $S \rightarrow asa$; 2) $S \rightarrow vsb$; 3) $S \rightarrow c$. Гра грамматики показан на рис. 3.1.

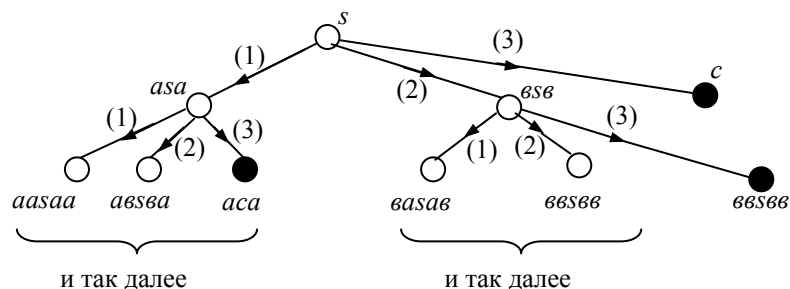


Рис. 3.1.

Вывод конкретной цепочки соответствует пути на графе, который начинается в вершине S и заканчивается вершиной, помеченной данной цепочкой.

Например, слову $aavsvaa$ в грамматике G_m соответствует вывод: $S \xrightarrow{1} aSa \xrightarrow{1} \xrightarrow{1} aaSaa \xrightarrow{2} aav Sv aa \xrightarrow{3} aavsvaa$. Поэтому можно записать $S \xrightarrow{G_m} aavsvaa$.

Аналогично $S \xrightarrow{G_m} vavavavav$. Ясно, что грамматика G порождает в точности те слова языка (терминальные цепочки), которые построены по схеме xcx^{-1} , где x - любая цепочка из символов a, v , а x^{-1} - ее обращение. Сокращенно это можно записать так $L_m = L(G_m) = \{xcx^{-1} \mid x \in \{a, v\}^*\}$.

Однако указанный способ представления выводов не отражает синтаксической структуры выводимых цепочек. Более наглядно такая структура представляется при изображении выводов в виде деревьев. Принцип построения деревьев вывода проиллюстрируем на примере.

Пример 3.2. Пусть задана грамматика $G_1 = \{V_T, V_A, S, R\}$, где $V_T = \{a, v, c, *, /, -, +, (,)\}$, $V_A = \{S, E, T, F, I\}$, S - аксиома, правила вывода имеют вид:

$$\begin{array}{l}
 S ::= E \\
 E ::= E + T \mid E - T \mid T \\
 R : T ::= T * F \mid T / F \mid F \\
 F ::= (E) \mid I \\
 I ::= a \mid v \mid c
 \end{array}$$

Большие латинские буквы обозначают следующие конструкции: S - арифметическое выражение, E - ***, T - терм, F - множитель, I - идентификатор.

Возьмем в качестве примера арифметическое выражение $(a + v) * c$. Дерево вывода этого выражения представлено на рис. 3.2.

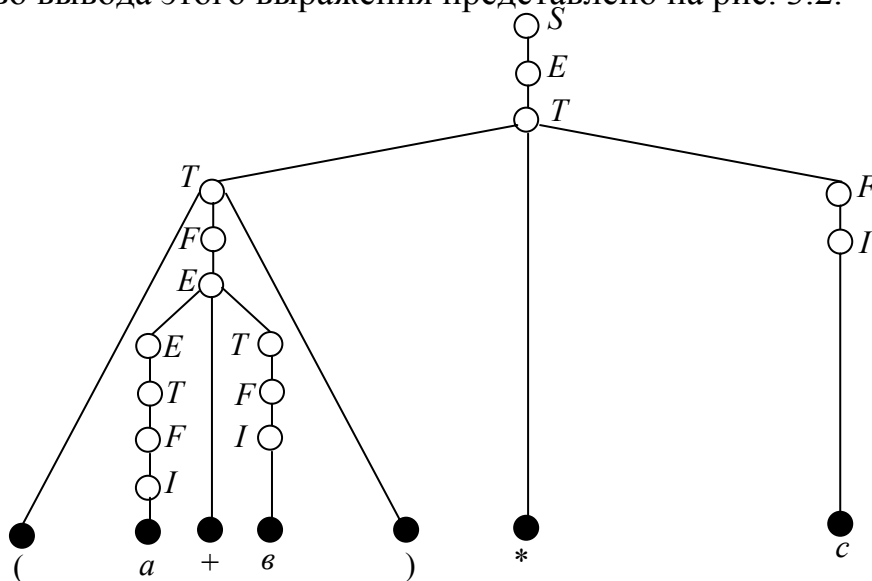


Рис. 3.2.

Всякой КС-грамматике однозначно соответствует некоторый КС-язык, быть может, пустой. Важно подчеркнуть, что соответствие между грамматиками и языками не является взаимно однозначным: один и тот же язык может быть порожден различными грамматиками. Грамматика, порождающая один и тот же язык, называется эквивалентным. Отсюда вытекает возможность эквивалентного преобразования грамматик. Не рассматривая глубоко этот вопрос, укажем только, что преобразования грамматик можно проводить различным образом. В частности, можно сократить число правил вывода и упростить грамматику, выбросив те правила, которые включают непроизводящие символы (т.е. символы, не имеющие терминального порождения), а также правила, включающие

недостижимые символы (т.е. символы, не встречающиеся ни в одном терминальном порождении). Можно, увеличив число правил вывода, ускорить вывод цепочек языка. Можно, наконец, представить правила вывода в некоторой стандартной форме. Возможны и другие преобразования грамматик. Покажем в качестве примера возможность получения из некоторой грамматики эквивалентной ей с ускоренным выводом цепочек.

Пример 3.3. Грамматика $G_n = \{V_T, V_A, S, R\}$, в которой $V_T = \{a, e, c\}$, $V_A = \{S, A\}$,

$R : \left\{ \begin{array}{ll} S \rightarrow asa & (1) \\ S \rightarrow esv & (2) \\ S \rightarrow c & (3) \\ S \rightarrow avsva & (4) \\ S \rightarrow cA & (5) \\ S \rightarrow aA & (6) \end{array} \right\}$ порождает тот же язык, что и грамматика G_m , но некоторые

его цепочки порождаются за меньшее число шагов вывода. Например, слово $aavsvaa$ выводится в G_n за три шага: $S \xrightarrow{(1)} aSa \xrightarrow{(4)} aavsvaa \xrightarrow{(3)} aavsvaa$, в то время как в G_m оно выводится за четыре шага. Заметим попутно, что грамматику G_n можно упростить, выбросив правила (5) и (6), так как они включают непроизводящие нетерминальные символы A .

Пусть в некоторой грамматике G имеется вывод $A \xrightarrow[G]{} u$, где A - нетерминальный символ, а u - произвольная цепочка. Очевидно, что не изменяя язык, порождаемый этой грамматикой, можно добавить к ней правило вывода вида $A \rightarrow u$. Говорят, что это правило, получено сокращением вывода. Нетрудно видеть, что к грамматике G_n таким правилом является правило $S \rightarrow avsva$. Ясно, что в грамматику G_n может быть добавлено сколь угодно правил такого типа.

Грамматика, в которой существует более одного вывода хотя бы одной терминальной цепочки, называется неоднозначной. Если некоторая цепочка имеет p равных выводов в данной грамматике, то говорят, что степень ее неоднозначности равна p . Язык, порождаемый неоднозначной грамматикой,

называется синтаксически неоднозначным. В графе неоднозначной грамматики для цепочки со степенью неоднозначности p имеется ровно p путей, идущих из вершины, помеченной аксиомой, в вершину, помеченную данной цепочкой. Эта цепочка может быть выведена p способами, каждый из которых задается своим деревом вывода.

Пример 3.4. Пусть задана грамматика $G_2 = \{V_T, V_A, S, R\}$, где $V_T = \{a, c\}$, $V_A = \{S\}$, S - аксиома, правила вывода имеют вид:

$$R: \left\{ \begin{array}{l} S \rightarrow a^{p_i} S, \quad p_i \geq 1 \\ S \rightarrow a^{p_j} S^* a^{q_j}, \quad p_j, q_j \geq 0 \end{array} \right\}.$$

Эта грамматика обязательно содержит правила вывода $S \rightarrow aS$, $S \rightarrow aSa$, которые вместе с правилами $S \rightarrow c$ порождает язык $L_2 = \{a^m c a^n\}$, где $m \geq n \geq 0$. Пусть, например, мы хотим получить цепочку $a^m c a^n$. Применим k раз правило $S \rightarrow aS$ и l раз правило $S \rightarrow aSa$. Имеем $k + l = m$ и $l = n$. Дерево вывода зависит от того, в каком порядке применялись правила: не различных последовательностей применения правил существует столько, сколькими способами можно выбрать n элементов из m элементов, т.е. C_m^n . Итак, степень неоднозначности цепочки $a^m c a^n$ равна C_m^n . Для цепочки $a^3 c a^2$ получим $C_3^2 = 3$ выводов, которые показаны на рис. 3.3.

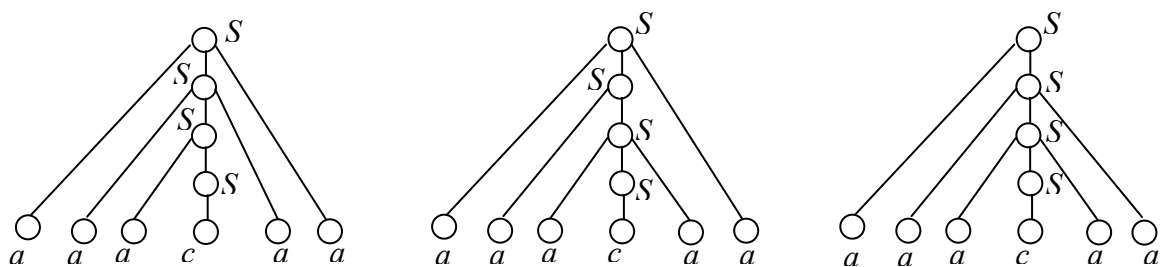


Рис. 3.3.

В некоторых случаях неоднозначность в грамматике можно устранять путем эквивалентного преобразования грамматики, однако в общем случае проблема устранения неоднозначности алгоритмически неразрешима. Это затрудняет трансляцию. Практически от неоднозначности избавляются путем

задания словесных ограничений, называемых контекстами, условиями языка, которые включаются в его семантику.

Любая КС-грамматика G может порождать язык $L(G)$, который может быть пустым, конечным или бесконечным. Для того, чтобы язык $L(G)$ был непустым, необходимо и достаточно, чтобы в грамматике существовало по крайней мере одно правило вывода вида $A \rightarrow \tau$, где A - нетерминал, τ - терминальная цепочка. Правила такого вида называются терминальными. В противном случае язык $L(G)$ является пустым.

Для того, чтобы язык $L(G)$, порождаемый грамматикой G , был бесконечным, необходимо и достаточно, чтобы хотя бы для одного нетерминального символа существовал вывод $A \xrightarrow{G} yA\psi$, причем $|y| + |\psi| \neq 0$.

В заключение отметим, что, строго говоря, все существующие языки программирования порождаются не УКС-, а КС-грамматиками. УКС-грамматики отличаются от КС-грамматик тем, что в них встречаются укорачивающие правила. Такие правила имеют место тогда, когда использование отдельных конструкций необязательно или зависит от задачи. Очевидно, что проблема описания УКС-грамматик легко решается, так как для любой УКС-грамматики можно построить почти эквивалентную ей КС-грамматику. Поэтому очень часто в качестве модели для описания синтаксиса языков в программировании используют КС-грамматики. Только что мы сказали о том, что УКС- и КС-грамматики можно использовать в качестве модели языков программирования. Однако они все-таки не полностью отражают некоторые правила построения программ. Правила языков программирования, которые не описываются средствами УКС- и КС-грамматик, называют контекстными условиями. Большинство контекстных условий (КУ) связаны с двумя особенностями языков программирования: необходимостью представления объектов различных типов и наличием так называемых областей действия. Первая особенность учитывается путем описания типов переменных. Например, в Алголе КУ этого типа описывают все переменные как *real*, *integer* или *Boolean*. Знание типов объектов

необходимо для правильного исчисления операторов. КУ, связанные со второй особенностью, используются для распределения памяти. Во многих языках для этой цели используются конструкции: типа «блок» языка Алгол. В этих конструкциях вводятся в употребление каждая переменная, используемая в блоке.

Все КУ можно свести к четырем типам. 1-й тип - КУ, описывающие тип переменных. Ими описываются, например, в Алголе: простые переменные, массивы, переключатели. 2-й тип - КУ, определяющие взаимосвязь между определяющими и использующими вхождениями. Определяющее вхождение указывает правило использования символа, а использующее - место его вхождения. 3-й тип - КУ, определяющие соотношение типов величин. Для проверки КУ этого типа необходимо знать тип переменных, а для этого их надо идентифицировать, т.е. основная часть проверки данных КУ - исполнение идентификации. 4-й тип - КУ, определяющие ограничения, накладываемые конкретной ЭВМ, транслятором. Можно сказать, что КУ этого типа определяются входным языком транслятора.

3.3. Алгоритмы с магазинной памятью (МПА)

МПА - подкласс машин Тьюринга, предназначенных для распознавания КС-языков. МПА состоит из входной ленты N , бесконечной в обе стороны, рабочей ленты M , бесконечной вправо, управляющего устройства A , регулирующего работу двух головок - входной и рабочей. Входная головка читает символы входной ленты, рабочая - символы рабочей ленты. Кроме того, рабочая головка может стирать и записывать символы на рабочей ленте. Каждая головка передвигает свою ленту. На входную ленту заносятся символы из входного словаря $V_T = \{a_1, a_2, \dots, a_m, e\}$, где e - единичный символ. Входное слово справа и слева окружается пробелами. На рабочую ленту заносятся символы из рабочего словаря $V_A = \{A_1, A_2, \dots, A_p, \sigma\}$, где σ -

маркер или из входного словаря V_T . Управляющее устройство может принимать одно из состояний множества $Q = \{q_0, q_1, \dots, q_n\}$.

Схематический МПА представлен на рис. 3.3.

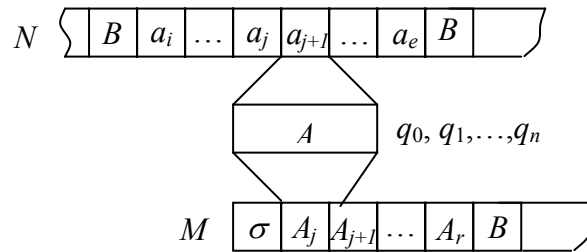


Рис. 3.3.

Физической ситуацией называется тройка (a_i, q_j, v_k) , где $a_i \in V$, q_j - состояние, $v_k \in V_A \cup V_T$. Ситуацией МПА называется либо физическая ситуация, либо тройка, полученная из нее заменой a_i , или v_k , или обоих сразу на единый символ e . Если МПА находится в ситуации $\sum = (a_i, q_j, v_k)$, считается, что он находится одновременно в четырех ситуациях: 1) (a_i, q_j, v_k) , 2) (e, q_j, v_k) , 3) (a_i, q_j, e) , 4) (e, q_j, e) .

Командой МПА называется выражение типа $\sum \rightarrow (q_i, x)$, где x - либо e , либо σ , либо цепочка в $V_A \cup V_T$. Для задания некоторого МПА необходимо задать множество команд $\sum_i \rightarrow (q_i, x_k)$. В общем случае МПА может быть как детерминированным, так и недетерминированным автоматом.

МПА работает следующим образом. Перед началом работы на ленту N заносится анализируемая цепочка из входного алфавита. Автомат находится в начальном состоянии q_0 . Входная головка рассматривает самый левый символ цепочки, рабочая головка - символ σ , записанный в самой левой ячейке ленты. Таким образом, (a_i, q_0, σ) - начальная ситуация МПА. Пусть автомат перешел в некоторую ситуацию \sum . Если \sum такова, что среди команд автомата нет команды, начинающейся с \sum , то автомат останавливается. Если в МПА имеется несколько команд, начинающихся с \sum , то автомат выбирает любую из них. В этом случае он работает как недетерминированный автомат. Пусть выбрана команда $\sum \rightarrow (q_m, x)$. Тогда устройство управления переходит

в состояние q_m . Если просматриваемый символ входной цепочки - e , то головка не передвигает ленту N . Если же это символ a_i , то лента передвигается влево таким образом, чтобы головка просматривала следующий за ним символ. В зависимости от того, какое значение принимает x , рабочая головка оперирует с рабочей лентой следующим образом:

- а) если $x = \sigma$, то лента M передвигается на одну ячейку вправо, предыдущий символ стирается;
- б) если $x = e$, то лента M не передвигается, символ не меняется;
- в) если $x = y$, где y - последовательность символов длиной m , то y заносится на M , лента сдвигается на m ячеек влево и рабочая головка читает последний символ y .

Различают два типа МПА: допускающие и порождающие языки.

Говорят, что входная цепочка $\alpha \in V_T^*$ является допустимой или представимой в МПА, если вычисление, начинающееся с этой цепочки, заканчивается ситуацией (B, q_0, B) . В противном случае входная цепочка не допустима или не представима в МПА. Множество цепочек, представимых в автомате, образуют язык, допускаемый автоматом.

Рассмотрим работу автомата, допускающего язык $L_m = \{x c x^{-1} / x \in \{a, b\}^*\}$. Любая цепочка языка L_m , допуская вольность речи, симметрична относительно маркера c . Поэтому работу автомата можно организовать следующим образом. Вначале скопировать на рабочую ленту подцепочку x , а затем сравнить содержимое рабочей ленты с подцепочкой x^{-1} . Если они совпадают, то содержимое рабочей ленты стирается, автомат переходит в ситуацию (B, q_0, B) . В том случае, если цепочка не принадлежит языку L_m , на рабочей ленте после остановки автомата остаются некоторые символы, отличные от B . Описанную процедуру задает следующее множество команд, однозначно определяющее МПА.

- 1) $(a, q_0, \sigma) \rightarrow (q_1, \sigma)$; 6) $(b, q_1, b) \rightarrow (q_1, b)$;
- 2) $(b, q_0, \sigma) \rightarrow (q_1, b)$; 7) $(c, q_1, a) \rightarrow (q_2, e)$;

- 3) $(a, q_1, a) \rightarrow (q_1, a)$; 8) $(c, q_1, \epsilon) \rightarrow (q_2, \epsilon)$;
 4) $(a, q_1, \epsilon) \rightarrow (q_1, a)$; 9) $(a, q_2, a) \rightarrow (q_2, B)$;
 5) $(\epsilon, q_1, a) \rightarrow (q_1, \epsilon)$; 10) $(\epsilon, q_2, \epsilon) \rightarrow (q_2, B)$;
 11) $(B, q_2, \sigma) \rightarrow (q_0, B)$.

Пример 3.5. Пусть приведенному МПА предъявляется цепочка $\alpha = авсва$. Последовательность вычислений МПА приведена на рис. 3.4.

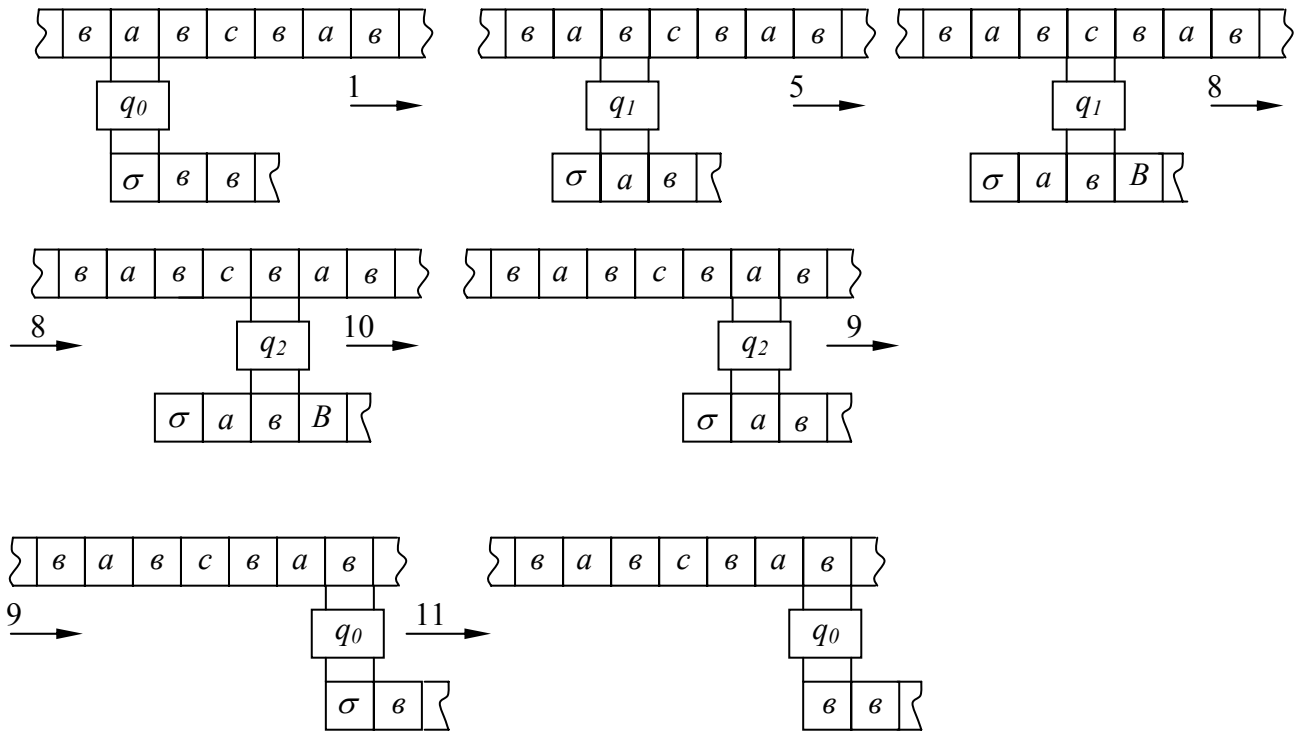


Рис. 3.4.

Поскольку автомат перешел в заключительную ситуацию (B, q_0, B) , то $\alpha \in L_m$. Заметим, что если автомату предъявить цепочку $\beta = авсвв$, то в четвертом шаге он будет находиться в ситуации (ϵ, q_2, a) , которой нет среди множества команд. Поэтому МПА остановится, а на рабочей ленте останется запись σa . Таким образом, $\beta \notin L_m$. Нетрудно видеть, что описанный МПА является детерминированным.

Говорят, что входная цепочка $\alpha \in V_T^*$ порождается автоматом, если вычисление, начинающееся с ситуации (B, q_0, σ) , приводит к записи α на входной ленте. Поскольку входная лента N в начальный момент всегда пустая, то начальная ситуация фактически определяется парой (q_0, σ) . И

вообще, в каждом шаге работа порождающего МПА определяется частичной ситуацией (q, A_k) . Команда порождающего МПА имеет вид $(q_i, A_k) \rightarrow (a_i, q_m, x)$, где q_i, q_m - состояния МПА, $A_k \in V_A$, $x \in V_T \cup V_A$, а буква a_i порождается (генерируется) на входной ленте. Порождающий МПА выдает на входную ленту правильную цепочку всякий раз, когда, исходя из записанной выше начальной ситуации, он возвращается в состояние q_0 , причем в этот момент рабочая лента M оказывается пустой и МПА оказывается в заключительной ситуации (B, q_0, B) .

Рассмотрим работу автомата, порождающего цепочки языка L_m . Множество команд такого автомата имеет вид:

- 1) $(q_0, \sigma) \rightarrow (a, q_1, a)$; 7) $(q_1, a) \rightarrow (c, q_2, e)$;
- 2) $(q_0, \sigma) \rightarrow (e, q_1, e)$; 8) $(q_1, e) \rightarrow (c, q_2, e)$;
- 3) $(q_1, a) \rightarrow (a, q_1, a)$; 9) $(q_2, a) \rightarrow (a, q_2, B)$;
- 4) $(q_1, e) \rightarrow (a, q_1, a)$; 10) $(q_2, e) \rightarrow (e, q_2, B)$;
- 5) $(q_1, a) \rightarrow (e, q_1, e)$; 11) $(q_2, \sigma) \rightarrow (B, q_0, B)$.
- 6) $(q_1, e) \rightarrow (e, q_1, e)$;

Пример 3.6. Запишем процесс порождения цепочки $\alpha = avcva$ МПА, имеющим описанное выше множество команд. Последовательность вычислений МПА приведена на рис. 3.5.

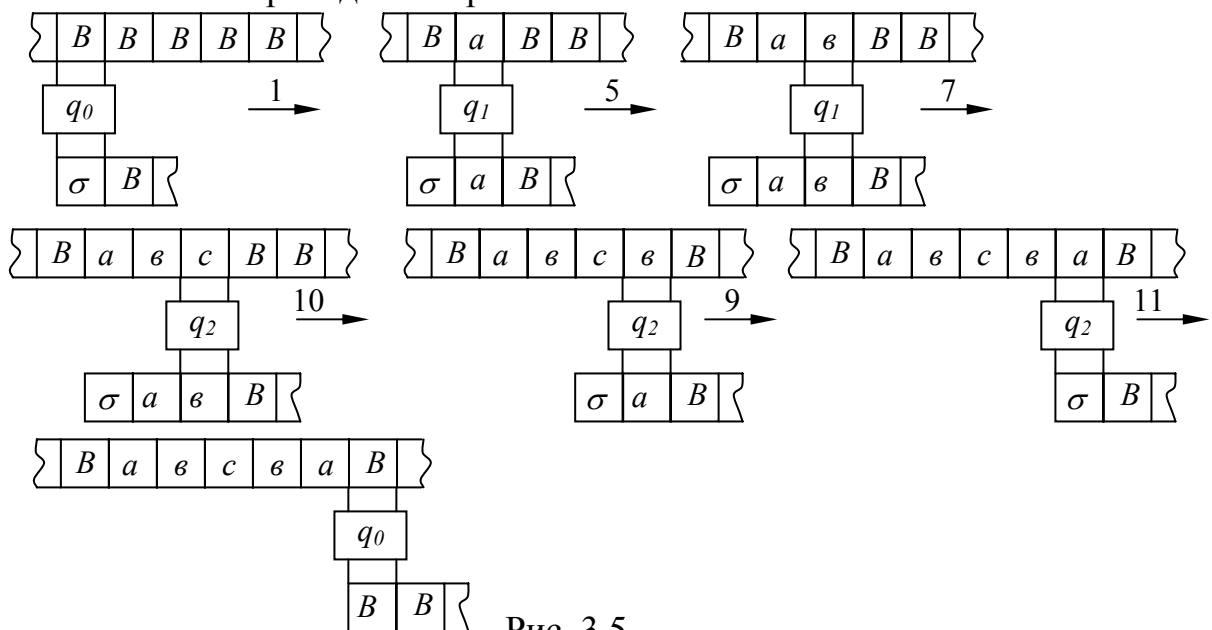


Рис. 3.5.

Поскольку МПА оказался в заключительной ситуации (B, q_0, B) их входной ленте порождена цепочка $\alpha \in L_m$. Нетрудно видеть, что описанный МПА никогда не может породить, например, цепочку $\beta = avcswv$, такую, что $\beta \notin L_m$. Другими словами, если автомат оказывается в ситуации (B, q_0, B) , то на входной ленте всегда записана правильная цепочка данного языка.

Очевидно, что в общем случае порождающий МПА не является детерминированным. Недетерминированность автомата проявляется при порождении подцепочки x слова α . При порождении подцепочки x^{-1} , автомат становится детерминированным.

Нетрудно показать, что в общем случае для любой КС-грамматики G можно построить МПА, допускающий (порождающий) язык $L(Q)$, и наоборот, язык, допускаемый (порождаемый) МПА является КС-языком. Поэтому справедливо утверждение о том, что класс языков, допускаемых (порождаемых) МПА, совпадает с классом КС-языков.

3.4. Автоматные грамматики и языки

Автоматные грамматики ***** КС-грамматик. Для задания А-грамматики *****

Для любой левосторонней грамматики можно построить эквивалентную правостороннюю и наоборот. Любая А-грамматика порождает автоматный язык (А-язык), быть может пустой.

Пример 3.7. Пусть задана правосторонняя грамматика *****,
 $V_A = \{A_1, A_2\}$, $V_T = \{x_1, x_2\}$.

$$R: \left| \begin{array}{ll} A_1 \rightarrow x_1 A_1 & (1) \\ A_1 \rightarrow x_1 A_2 & (2) \\ A_2 \rightarrow x_2 A_2 & (3) \\ A_2 \rightarrow x_2 & (4) \end{array} \right|.$$

Грамматика G_n задает язык, состоящий из всевозможных слов, которые начинаются последовательностью из x^{*****} *** последовательностью из x_2 ,

т.е. $L(G_n) = \{x_1^m, x_2^n\}$, где ***. Например, слово $x_1x_1x_1x_1x_2x_2(x_1^4x_2^2)$ может быть получено выводом: $A_1 \xrightarrow{1)} x_1A_1 \xrightarrow{1)} x_1x_1A_1 \xrightarrow{11)} x_1x_1x_1A_1 \rightarrow x_1x_1x_1x_1h_2 \rightarrow x_1x_1x_1x_1x_2A_2 \rightarrow$

$\rightarrow x_1x_1x_1x_1x_2x_2$. Этот же язык может быть получен левосторонней грамматикой

$G_A = \{V_A, V_T, A_1, R\}$, где $V_A = \{A_1, A_2\}$, $V_T = \{x_1, x_2\}$ с правилами вывода A вида:

$$R: \left\{ \begin{array}{ll} A_1 \rightarrow A_1x_2 & (1) \\ A_1 \rightarrow A_2x_2 & (2) \\ A_2 \rightarrow A_2x_1 & (3) \\ A_2 \rightarrow x_1 & (4) \end{array} \right.$$

Действительно, цепочка $x_1^4x_2^2$ может быть получена в грамматике G_A

следующим выводом: $A_1 \xrightarrow{(1)} A_1x_2 \xrightarrow{(2)} A_2x_2x_2 \xrightarrow{(3)} A_2x_1x_2x_2 \xrightarrow{(3)} A_2x_1x_1x_2x_2 \xrightarrow{(3)} A_2x_1x_1x_1x_2x_2 \xrightarrow{(4)}$

$\xrightarrow{(4)} x_1x_1x_1x_1x_2x_2$. Поэтому $L(G_n) = L(G_A)$.

Любой конечный язык является А-языком. Действительно, множество, содержащее n слов $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$, где слово $\alpha_i = x_{i_1}x_{i_2}, \dots, x_{i_k}$, может быть описано следующей А-грамматикой с правилами вывода вида: $A_1 \rightarrow x_{i_1}A_{i_1}$; $A_{i_1} \rightarrow x_{i_2}A_{i_2}$; \dots , $A_{i_{k-2}} \rightarrow x_{i_{k-1}}A_{i_{k-1}}$; $A_{i_{k-1}} \rightarrow x_{i_k}$, где $1 \leq i \leq n$.

Важной особенностью А-грамматики является возможность представления их с помощью конечных графов. По графу грамматики легко отыскивается вывод нужного слова. Для представления грамматики в виде графа необходимо преобразовать ее к виду так называемой модифицированной грамматики. Пусть имеется А-грамматика $G = \{V_A, V_T, A_1, R\}$. Дополним алфавит V_A символом k (k - концевой символ). Получим $V'_A = V_A \cup \{k\}$. Все терминальные правила вывода $A_i x_j$ заменим правилами $A_i \rightarrow x_jk$, $k \rightarrow \lambda$. Получим новое множество правил и модифицированную А-грамматику $G' = \{V'_A, V_T, A_1, R'\}$. Каждому символу из V'_A сопоставим вершину графа. Каждому правилу вывода $A_i \rightarrow x_jA_k$ сопоставим дугу из вершины A_i в вершину A_k и пометим ее буквой x_j . Каждому правилу вывода $A_i \rightarrow x_jk$ сопоставим дугу, ведущую из A_i в k и помеченную буквой x_j . Любой вывод слова в А-грамматике соответствует пути в графе грамматики,

который начинается из вершины, помеченной аксиомой A_1 и заканчивается в конечной вершине k . Графы грамматик G_n и G_A показаны соответственно на рис. 3.6, а, б.

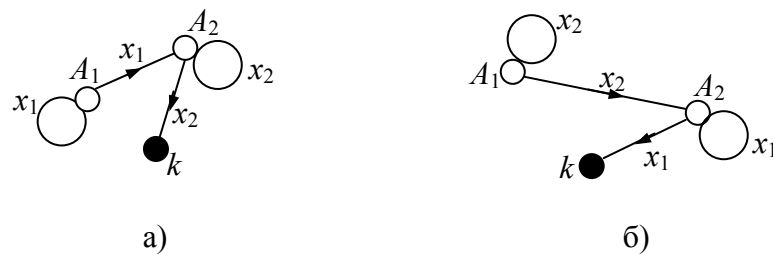


Рис. 3.6

Подчеркнем особо, что граф КС-грамматики в общем случае задается в виде бесконечного графа.

3.5. Конечные автоматы и способы их задания

Конечные автоматы (КА) - подкласс машин Тьюринга, которые распознают А-языки. Условно КА можно представить в виде входной ленты, ограниченной слева и неограниченной справа, и устройства управления (УУ) с головкой, читающей символы на входной ленте так, как показано на рис. 3.7.

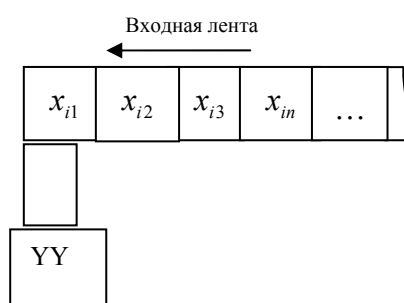


Рис.3.7

КА имеет конечный входной алфавит $V_T = \{x_1, x_2, \dots, x_n\}$, конечное множество состояний $Q = \{q_0, q_1, \dots, q_k\}$, где q_0 - начальное состояние. Функционирование КА можно описать следующим образом. На ленту записывается входное слово $d \in V_T^n$. Считается, что справа от d записаны пробелы ε , автомат находится в состоянии q_0 , а головка обзревает самый левый символ x_1 слова d . То есть КА находится в начальной ситуации (q_0, x_1) .

Прочтя x_i , КА переходит в новое состояние $q_j \in Q$, сдвигает ленту на одну ячейку влево и тем самым оказывается в новой ситуации и так далее. Таким образом, команды КА имеет вид $(q_j, x_i) \rightarrow q_k$. Число команд КА конечно. Если после выполнения начислений над входом словом d КА возвращается к ситуации (q_0, ε) , то говорят, что автомат допускает слово d . Если КА возвращается к ситуации (q_0, ε) , то говорят, что автомат допускает слово d . Множество слов, допустимых автоматом, есть по определению язык, допустимый этим автоматом. Покажем, что класс языков, допускаемых конечными автоматами, совпадает с классом А-языков.

Пусть задана A -грамматика $G = (V_A, V_T, A, R)$ и пусть $L(G)$ - язык порождаемый ею. Построим КА, допускающий слова языка $L(G)$. Каждому символу $A_i \in V_A$ сопоставляем $q_i \in Q$. Кроме этого, для автомата введем начальное состояние q_0 , которое не соответствует никакому нетерминальному символу грамматики G вида $A_i \rightarrow x_j$ сопоставим команду вида $(q_k, x_j) \rightarrow q_\varepsilon$, которая соответствует переходу автомата из состояния q_k в q_ε по букве x_j . Заключительной командой КА будет $(q_1, \varepsilon) \rightarrow q_0$, переводящая автомат из последнего текущего состояния (соответствующего аксиоме A_1 грамматики G) в начальное состояние q_0 по пробелу ε . Таким образом, по грамматике G построен конечный автомат, заданный описанным множеством команд.

Нетрудно показать, что по заданному КА, допускающему $L(G)$, можно однозначно построить правила вывода A -грамматики G .

Пример 3.8. Для грамматики G_A , порождающей $L(G_A) = \{x_1^m, x_2^n\}$, построим множество команд, описывающих функционирование конечного автомата A , допускающего язык $L(G_A)$. Грамматика G_A имеет одно терминальное $A_2 \rightarrow x_1$. Вводим начальное состояние q_0, ε, Q и записываем команду $(q_0, x_1) \rightarrow q_2$, сопоставляем символу A_2 состояние $q_2 \in Q$. Далее по правилу $A_1 \rightarrow A_1 x_2$ записываем команду $(q_2, x_1) \rightarrow q_1$, сопоставляем A_1 состоянию q_1 . Затем по правилам вывода (2) и (3) грамматики G_A записываем команды $(q_2, x_2) \rightarrow q_1$ и $(q_2, x_1) \rightarrow q_2$ соответственно. После этого записываем заключительную команду $(q_1, \varepsilon) \rightarrow q_0$. Таким образом, автомат A , допускающий язык $L(G_A)$, задается набором команд:

$(q_0, x_1) \rightarrow q_2; (q_2, x_2) \rightarrow q_1; (q_1, \varepsilon) \rightarrow q_0.$
 $(q_1, x_2) \rightarrow q_1; (q_2, x_1) \rightarrow q_2$

Для слова $d = x_1 x_1 x_1 x_1 x_2 x_2$ последовательность вычислений (по аналогии с машиной Тьюринга) автомата A имеет вид

q_0	x_1		x_1		x_1		x_1		x_2		x_2	
	x_1	q_2	x_1		x_1		x_1		x_2		x_2	
	x_1		x_1	q_2	x_1		x_1		x_2		x_2	
	x_1		x_1		x_1		x_1		x_2		x_2	
	x_1		x_1		x_1	q_2	x_1		x_2		x_2	
	x_1		x_1		x_1		x_1	q_2	x_2		x_2	
	x_1		x_1		x_1		x_1		x_2	q_1	x_2	$q_1 \varepsilon$
	x_1		x_1		x_1		x_1		x_2	x_2		$q_0 \varepsilon$

Поскольку автомат оказался в ситуации (q_0, ε) , то слово d допустимо автоматом A . Легко проверить, что любое слово языка $L(G_A)$ допустимо автоматом A .

Заметим, что описанным способом может быть построена КА, допускающий любой A - язык, порождаемый левосторонней грамматикой. Поэтому такие КА можно назвать левосторонними автоматами. Для правосторонних грамматик и языков, или порождаемых, необходимо ввести правосторонние КА. Входная лента такого КА ограничена справа, неограниченна слева и сдвигается слева направо. Входное слово на ленте читается справа налево в отличие от общепринятого направления чтения.

Для правосторонней грамматики G_n , используя описанную ниже методику, можно построить правосторонний КА ε , допускающий язык $L(G_n)$, который имеет следующую последовательность команд:

$$(x_2, q_0) \rightarrow q_2; \quad (x_1, q_2) \rightarrow q_1, \quad (\varepsilon, q_1) \rightarrow q_2.$$

$$(x_1, q_1) \rightarrow q_1, \quad (x_2, q_2) \rightarrow q_2$$

Для слова $d = x_1 x_1 x_1 x_1 x_2 x_2$ последовательность вычислений КА имеет вид:

	x_1	x_1	x_1	x_1	x_2	$x_2 q_0$
	x_1	x_1	x_1	x_1	$x_2 q_2$	x_2
	x_1	x_1	x_1	$x_1 q_2$	x_2	x_2
	x_1	x_1	$x_1 q_2$	x_1	x_2	x_2
	x_1	$x_1 q_2$	x_1	x_1	x_2	x_2
	$x_1 q_1$	x_1	x_1	x_1	x_2	x_2
εq_1	x_1	x_1	x_1	x_1	x_2	x_2
εq_0	x_1	x_1	x_1	x_1	x_2	x_2

В дальнейшем, если не оговорено противное, будем рассматривать левосторонние КА.

Легко понять, что множество команд КА представляет собой функцию, называемую функцией перехода автомата, которую обозначим через Y . Она определена на множестве (точнее подмножестве) декартового произведения $Q \times V_Y$ и принимает значения в Q . В общем виде её можно записать $q_k = Y(q_j, x_i)$. Функция переходов может быть задана матрицей переходов, представляющей собой прямоугольную матрицу, строки которой помечены символами $q_j \in Q$, а на пересечении q_i строки и q_j столбца ставится значение функции q_k . Функцию Y можно доопределить на множестве $Q \times V_Y^*$ следующим образом. Пусть $d \in V_T^*$, причем $d = d_1 x_i$. Тогда $Y(q_j d) = Y(Y(q_j d_1), x_i)$.

Таким образом, формально любой конечный автомат A можно записать в виде пятерки множеств: $A = \{V_k, Q, q_0, Y(q, x), Q'\}$, где V_k - входной алфавит, Q - алфавит состояний, $q_0 \in Q$ - начальное состояние, $Y(q, x)$ - функция переходов, а $Q' \equiv Q$ - множество заключительных состояний автомата.

КА можно также задать в виде ориентированного нагруженного графа. Каждому состоянию $q_j \in Q$ соответствует вершина графа, а каждой команде $(q_j, x_i) \rightarrow q_k$ сопоставляется дуга, ведущая из вершины q_j в вершину q_k и помеченную символом x_i .

Пример 3.9. На рис. 3.8 (а,б) приведены соответственно таблицы переходов автомата А и В, а на рис.3.9 (а,б) – их графы

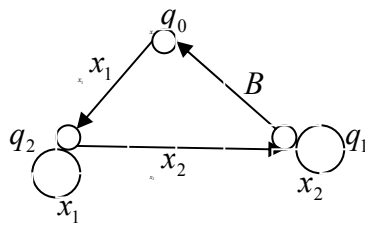
$V_T \backslash Q$	q_0	q_1	q_2
x_1	q_2	-	q_2
x_2	-	q_1	q_1
B	-	q_0	-

а)

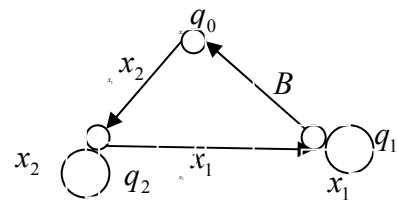
$V_T \backslash Q$	q_0	q_1	q_2
x_1	-	q_1	q_1
x_2	q_2	-	q_2
B	-	q_0	-

б)

рис.3.8



а)



б)

рис.3.9

Итак, мы рассмотрели конечные автоматы, допускающие языки. Если снабдить КА выходной лентой, ограниченной слева и неограниченной справа и передвигающейся справа налево, на которой записываются буквы из некоторого выходного алфавита автомата, то мы получим односторонний конечный преобразователь или конечный автомат с выходами. Такие автоматы реализуют в общем случае частичное отображение множества всех слов входного алфавита и множество всех слов выходного алфавита.

Рассмотрим способы задания и функционирования таких автоматов. Пусть $V_k = \{x_1, x_2, \dots, x_m\}$ - алфавит входных символов автомата, а V_k^* - множество всех слов в V_k . Любое подмножество множества называют событием в этом алфавите. Пусть $V_y = \{y_1, y_2, \dots, y_n\}$ - алфавит выходных символов автомата. Примем, как и ранее, что алфавит состояний автомата $Q = \{q_0, q_1, \dots, q_k\}$. Конечный автомат с выходами работает аналогично конечному автомату без выходов с той разницей, что в каждом такте работы в ячейку выходной ленты записывается некоторая буквы выходного алфавита, после чего выходная лента, также как и входная, сдвигается на одну ячейку влево. Формально функционирование конечного автомата с выходами можно описать командами вида $(q_l, x_i) \rightarrow (q_k, y_j)$ и задать двумя функциями Y и Ψ , называемыми соответственно функцией переходов и функцией выходов, которые имеют вид $q_k = Y(q_l, x_i), y_j = \Psi(q_l, x_i)$. Каждую из них можно задавать в виде таблиц. Таблица переходов строится так же, как и для КА без выходов, строки которой помечены входными буквами, а столбцы-состояниями, содержит на пересечении x_i строки и q_l столбца выходную букву y_i . Кроме таблиц переходов и выходов конечный автомат с выходами можно задать в виде ориентированного нагруженного графа, дуги которого помечаются кроме входных и выходными буквами.

Пример 3.10. пусть задан конечный автомат с выходами $A = \{V_x, V_y, Q, q_0, Y(q, x), \psi(q, x)\}$, где $V_x = \{x_1, x_2, x_3\}, V_y = \{y_1, y_2\}, Q = \{q_0, q_1, q_2\}$, а функции $Y(q, x)$ и $\psi(q, x)$ заданы таблицами, показанными соответственно на рис.3.10 (а,б):

$V_x \backslash Q$	q_0	q_1	q_2
x_1	q_1	q_2	q_1
x_2	q_0	q_1	-
x_3	q_2	-	q_0

а)

$V_x \backslash Q$	q_0	q_1	q_2
x_1	y_2	y_1	q_2
x_2	y_1	y_2	-
x_3	y_1	-	y_1

б)

рис.3.10

Граф автомата A показан на рис.3.11:

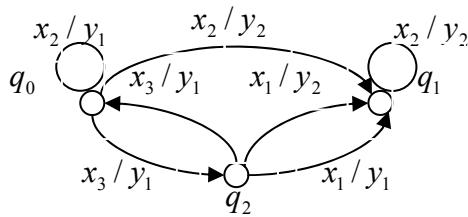


рис.3.11

Пусть на вход автомата, установленного в начальное состояние $q_0 \in Q$ подается входное слово $d = x_2 x_1 x_2 x_1 x_1 x_1 x_3$. Автомат переработает d в выходное слово

$\beta = y_1 y_2 y_2 y_1 y_2 y_1 y_1$ и окажется снова в состоянии q_0 . Поэтому можно записать

$\beta = f(d)$. Если на вход автомата подать входные слова $d_1 = x_1 x_2 x_3$ или $d_2 = x_3 x_1 x_2 x_1 x_2$,

то на выходе не образуется выходных слов, так как переходы по букве x_3 в слове d_1 и по последней букве в x_2 в слове d_2 не определены в таблице 3.10 (а). Поэтому слова d_1 и d_2 являются недопустимыми и относятся к области запрета. В качестве заключительного состояния автомата A выбрано состояние q_0 . Отметим еще раз, что в качестве заключительных состояний может быть выбрано любое подмножество $Q' \equiv Q$.

Говорят, что конечный автомат с выходами индуцирует (или реализует) частичное отображение f множества V_x^* в V_y^* , которое удовлетворяет следующим условиям:

- 1) любому допустимому входному слову $\alpha \in V_x^*$ отображение f сопоставляет выходное слово $f(\alpha) \in V_y^*$, имеющее одинаковую длину с α ;
- 2) если α - допустимое входное слово, а α_1 - начальный отрезок (вхождение) слова α , то $f(\alpha_1)$ существует и совпадает с некоторым начальным отрезком слова $f(\alpha)$.

Эти условия называются условиями автоматности отображения.

3.6 Алгебра событий. Представление событий в автоматах.

Пусть A - конечный автомат с входным алфавитом $V_x = \{x_1, x_2, \dots, x_m\}$ и выходным алфавитом $V_y = \{y_1, y_2, \dots, y_n\}$, который индуцирует частичное автоматное отображение f множества V_x^* в V_y^* .

Событием P_i , представленным (допустимым) в автомате A выходным сигналом y_i ($i \in I = \{1, 2, \dots, n\}$), называется множество всех входных слов $\alpha \in V_x^*$ этого автомата, для которых слово $f(\alpha)$ определено и оканчивается y_i . Если $Y' \equiv V_y$ - некоторое подмножество выходных сигналов, то событием, представленным в автомате A множества Y' , называется объединение событий, представляемых всеми элементами этого множества. В том случае, когда Y' совпадает с V_y , соответствующие ему событие называется каноническим множеством событий P_1, P_2, \dots, P_n автомата A . Например $P_1 = \{x_2, x_3; x_3 x_3, x_2 x_1 x_2 x_1, x_3 x_1 x_2 x_1 x_3, x_2 x_1 x_2 x_1 x_3 x_1 x_3\}$ и $P_2 = \{x_1; x_2 x_1, x_1 x_2, x_1 x_2 x_1 x_1, x_3 x_1 x_3 x_1 x_2\}$ - события, представленные выходными буквами и в автомате A из примера 3.10.

Сформулируйте предложение, которое подведет базу под изучение автоматных отображений.

Теореме 3.1. Задание частичного автоматного отображения f множества V_x^* в V_y^* произвольного автомата A с входным алфавитом $V_x = \{x_1, x_2, \dots, x_m\}$ и выходным алфавитом $V_y = \{y_1, y_2, \dots, y_n\}$ эквивалентно заданию канонического множества событий P_1, P_2, \dots, P_n данного автомата.

Доказательство. Вначале покажем, что задание f однозначно определяет каноническое множество событий автомата A .

Пусть f - частичное автоматное отображение V_x^* в V_y^* и α - произвольное слово из V_x^* . Если образ $f(\alpha)$ существует и оканчивается буквой y_j . Если хотя бы для одной входной буквы слова α не определена выходная буква, то слово α является запрещенным и его относим к множеству R - области запрета автомата A . Легко видеть, что в результате просмотра всех слов $\alpha \in V_x^*$ множеств V_x^* разбивается на $n+1$ попарно непересекающихся подмножества P_1, P_2, \dots, P_n являются событиями, представленными в автомате выходными сигналами y_1, y_2, \dots, y_n , и образуют каноническое множество событий автомата A , а множество R - событие, состоящее из всех тех слов $\alpha \in V_x^*$, которые не вошли ни в одно из событий $P_j, j \in I$.

Покажем теперь, что задание канонического множества событий автомата A однозначно задает частичное отображение f , индуцируемое автоматом A . Пусть дано каноническое множество событий P_1, P_2, \dots, P_n автомата A . Покажем, каким образом определить отображение f не прибегая к помощи переходов и выходов автомата A . Предположим, что $\alpha = x_{i_1}x_{i_2}\dots x_{i_k}$ - произвольное входное слово из V_x^* . Для каждого $x_{i_l} (1 \leq l \leq k)$ найдем выходную букву y_{j_l} по следующему правилу: y_{j_l} есть выходной сигнал, представляющий в автомате A событие P_{j_l} , которое содержит начальный отрезок $x_{i_1}x_{i_2}\dots x_{i_l}$ длины l входного слова α . Если для всех $l = 1, 2, \dots, k$ существует соответствующие им y_{j_l} , то полагаем $f(P) = f(x_{i_1}x_{i_2}\dots x_{i_k}) = y_{j_1}y_{j_2}\dots y_{j_k}$. Если хотя бы для одного l не существует y_{j_l} , то полагаем, что отображение f на слове α не определено $P \in R$. Этим теорема доказана.

Таким образом из теоремы 3.1 вытекает, что произвольные автоматные отображения можно задавать с помощью разбиений множества V_x^* всех слов во входном алфавите на конечное число попарно непересекающихся событий. Для эффективного описания конечных и некоторых классов бесконечных событий рассмотрим так называемую алгебру (регулярных) событий, предложенную С. Клини.

Алгеброй событий в алфавите V_x называется множество всех событий в этом алфавите, на котором задана система трех операций: дизъюнкции (\vee), умножения (\bullet) и итерации ($\{ \}^*$), которые определяются следующим образом.

Событие P , равное дизъюнкции событий $P_1 \vee P_2$, определяется как множество слов, принадлежащих событиям P_1 или P_2 .

Событие R , равное умножению $R_1 \cdot R_2$, определяется как конкатенация (приписывание справа) к любому слову события R_1 любого слова события R_2 .

Итерацией события P по определению является множество всех слов, которые могут быть получены по следующей формуле:

$$\{P\}^* = \lambda \vee P \vee P P \vee P P P \vee \dots P P \dots P \vee \dots$$

Легко видеть, что если исходные события конечны, то в результате дизъюнкции и умножения получаем события, состоящие из конечного множества слов. Применение итерации приводит к появлению событий, состоящих из бесконечного множества слов.

Пусть $V_x = \{x_1, x_2\}$ и заданы события $P = \{x_1, x_2, x_1\}$ и $R = \{x_2, x_2, x_1, x_2\}$ в алфавите V_x . На основании определений операций можно построить события $P \vee R, P \cdot R, \{P\}^*$, имеющие вид:

$$P \vee R = \{x_1, x_2, x_1, x_2, x_2, x_1, x_2\},$$

$$P \cdot R = \{x_1, x_2, x_2, x_1, x_1, x_2, x_2, x_1, x_2, x_2, x_1, x_1, x_2\}$$

$$\{P\}^* = \{\lambda, x_2 x_1, x_1 x_1, x_1 x_2 x_1, x_2 x_1 x_1, x_2 x_1 x_1, x_2 x_1 x_2 x_1, x_1 x_1 x_1, x_1 x_2 x_1 x_1, x_2 x_1 x_1 x_1, x_2 x_1 x_2 x_1, x_1 x_1 x_2 x_1, x_1 x_2 x_1 x_2 x_1, \dots\}$$

Заметим, что одноэлементные события λ , образованные пустым словом λ , состоит из одного слова нулевой длины. Условимся не считать события различными, отличающиеся друг от друга только пустыми словами λ . Кроме события λ будем также рассматривать пустое событие \emptyset , которое состоит из пустого множества букв входного алфавита и поэтому является частью любого события.

Пусть $V_x = \{x_1, x_2, \dots, x_m\}$ - произвольный алфавит. Введем теперь понятие регулярного выражения в алфавите V_x , которое определяется следующим образом:

- 1) символы x_1, x_2, \dots, x_m и \emptyset являются регулярными выражениями;
- 2) если P и R - регулярные выражения, то таковыми являются $P \vee R, P \cdot R, \{P\}^*$ и $\{R\}^*$;
- 3) никакое другое выражение не является регулярным, если оно не получено путем конечного числа применений правил 1) и 2).

Таким образом регулярное отношение – формула в алгебре событий, причем одно и то же событие может быть по разному выражено через одноэлементные события и операции $\vee, \cdot, \{ \}^*$. Те события, для которых существует регулярные выражения, называются регулярными событиями. В противном случае событие называется нерегулярным.

Законы эквивалентности преобразования регулярных выражений играют важную роль в алгебре событий. Пусть P, R, T - регулярные события из V_x^* . Тогда справедливы следующие соотношения:

$$P \vee \emptyset = \emptyset \vee P = P;$$

$$P \cdot \emptyset = \emptyset \cdot P = \emptyset;$$

$$\{\emptyset\}^* = \lambda;$$

$$\{\lambda\}^* = \lambda;$$

$$P \vee R = R \vee P;$$

$$P \cdot \{P\}^* = \{P\}^* \cdot P;$$

} коммутативность \vee и $\{ \}^*$

$$P \vee (R \vee T) = (P \vee R) \vee T$$

$$P \cdot (R \cdot T) = (P \cdot R) \cdot T$$

} ассоциативность \vee и \cdot

$$P \cdot (R \vee T) = (P \cdot R) \vee (P \cdot T)$$

$$(P \vee R) \cdot T = (P \cdot T) \vee (R \cdot T)$$

} левая и правая дистрибутивность относительно \vee

$$\{P\}^* = \lambda \vee P \cdot \{P\}^* \text{ - разворачивание итерации}$$

$$\left. \begin{aligned} PvP &= P \\ \{ \{P\}^* \}^* &= \{P\}^* \end{aligned} \right\} \text{идемпотентность } v \text{ и } \{ \}^*$$

$$\{P\}^* vP = \{P\}^* - \text{дизъюнктивное поглощение } \{ \}^*$$

$$\{P\}^* \cdot \{P\}^* = \{P\}^* - \text{мультипликативное поглощение } \{ \}^*.$$

При отсутствии обычных скобок в регулярных выражениях первыми выполняются итерации, затем умножения и потом дизъюнкции.

Рассмотрим некоторые примеры регулярных событий в алфавите $V_x = \{x_1, x_2, x_3\}$.

- 1) Событие называемое универсальным, которое состоит из всех слов алфавита V_x , записывается в виде $P = \{x_1, x_2, x_3\}^*$.

- 2) Событие, которое содержит только трех буквенные слова в V_x :

$$R = (x_1vx_2vx_3) \cdot (x_1vx_2vx_3) \cdot (x_1vx_2vx_3).$$

- 3) Событие, состоящие из всех слов, в которых хотя бы один раз встречается отрезок $x_1x_2x_3$:

$$T = \{x_1vx_2vx_3\}^* \cdot x_1x_2x_3 \cdot \{x_1vx_2vx_3\}^*.$$

- 4) Событие, состоящие из всех слов, которые начинаются буквой x_3 или x_1 , а оканчиваются отрезком x_1x_2 :

$$Q = (x_1vx_3) \cdot \{x_1vx_2vx_3\}^* \cdot x_1x_2.$$

- 5) Событие, состоящие из слов, начало которых есть любая цепочка из букв x_1 или x_2 , затем следует двухбуквенная цепочка из x_2 или x_3 и, наконец, окончание содержит, по крайней мере, одну букву x_1 :

$$W = \{x_1vx_2\}^* \cdot (x_2vx_3) \cdot (x_2vx_3) \cdot x_1 \cdot \{x_1\}^*.$$

Из рассмотрения операций $v, ;, \{ \}^*$ алгебры событий вытекает, что все конечные события регулярны. Это следует из того, что любая цепочка выражается произведением букв, а любое конечное событие – дизъюнкцией образующих его цепочек.

Применение итерации приводит к появлению бесконечных, регулярных событий. Это бесконечные, в конечном счете, повторяющиеся цепочки. Однако, как легко заметить, большинство бесконечных событий являются нерегулярными событиями. Примерами таких событий являются цепочки, в которых длина слов $n_1, n_2, \dots, n_i, \dots$ такова, что разности $n_{i+1} = n_i$ ($i = 1, 2, \dots$) не ограничены в совокупности. (Допустим, увеличивается в два раза, в квадрат, в куб и т.д.).

Фундаментальную роль в анализе и синтезе конечных автоматов сыграла теорема Клини.

Класс событий, представленных в конечных автоматах, совпадает с классом регулярных событий.

Конструктивное доказательство этой теоремы приводит к построению алгоритмов анализа и синтеза произвольных конечных автоматов.

3.7 Взаимосвязь автоматов и языков

В предыдущих разделах мы рассмотрели три класса автоматов: машина Тьюринга, автоматы с магазинной памятью и конечные автоматы. Какова взаимосвязь между этими автоматами и классами языков, которые они определяют? Вообще говоря, следуя Хомского, формальные языки можно охарактеризовать следующим образом:

- 1) Язык L является рекурсивно перечислимым тогда и только тогда, когда он определяется некоторой машиной Тьюринга.
- 2) Язык L является контекстно- свободным (контекстно- независимым) тогда и только тогда, когда он определяется автоматом с магазинной памятью.
- 3) Язык L является автоматом (левосторонним или правосторонним), если он определяется конечным автоматом.

Заметим, что хотя им и не определили этого специально, язык является контекстно – зависимым (КС-языком) тогда и только тогда, когда он определяется так называемым недетерминированным линейно ограниченным автоматом. Такой автомат представляет собой недетерминированную машину Тьюринга, на которую наложено дополнительное ограничение, которое заключается в том, что в процессе работы не должна увеличиваться длина рабочей цепочки.

Мы показали, что в общем случае проблема остановки машины Тьюринга алгоритмически неразрешима. Из этого вытекает, что такие свойства языков, определяемых машиной Тьюринга, как распознаваемость, эквивалентность слов, пустота пересечения, бесконечность и ряд других, которые связаны в конечном счете с этой проблемой, также алгоритмически неразрешимы. Указанные свойства распространяются и на линейно ограниченные автоматы, а значит, и на КС-языки.

Что касается проблемы остановки для МПА и КА, то для них она разрешима, поскольку множества слов, порождаемых КС- и А- грамматиками, являются рекурсивными множествами. Поэтому для КС- и А- языков алгоритмически разрешима проблема пустоты, конечности и бесконечности ,распознаваемости принадлежности произвольного слова к языку и проблема эквивалентности слов языка. В то же время и для КС- языков существует алгоритмически неразрешимые проблемы. Это прежде всего проблемы, связанные с пересечением КС-языков, также как не пустоты пересечения, автоматности пересечения и контекстной свободности пересечения, а также проблемы, связанные с неоднозначностью вывода в КС-языках. Иерархия автоматов и соответствующих им формальных языков показана на рис.3.11. Очевидно, что каждый из автоматов определенного уровня иерархии, кроме указанного соответствующего класса языков, распознаёт также все языки более низкого уровня иерархии, разумеется естественно, если проблема распознавания для данного языка разрешима. В этом смысле автоматы с магазинной памятью распознают КС- и А-языки, а машина Тьюринга- все перечисленные типы языков.

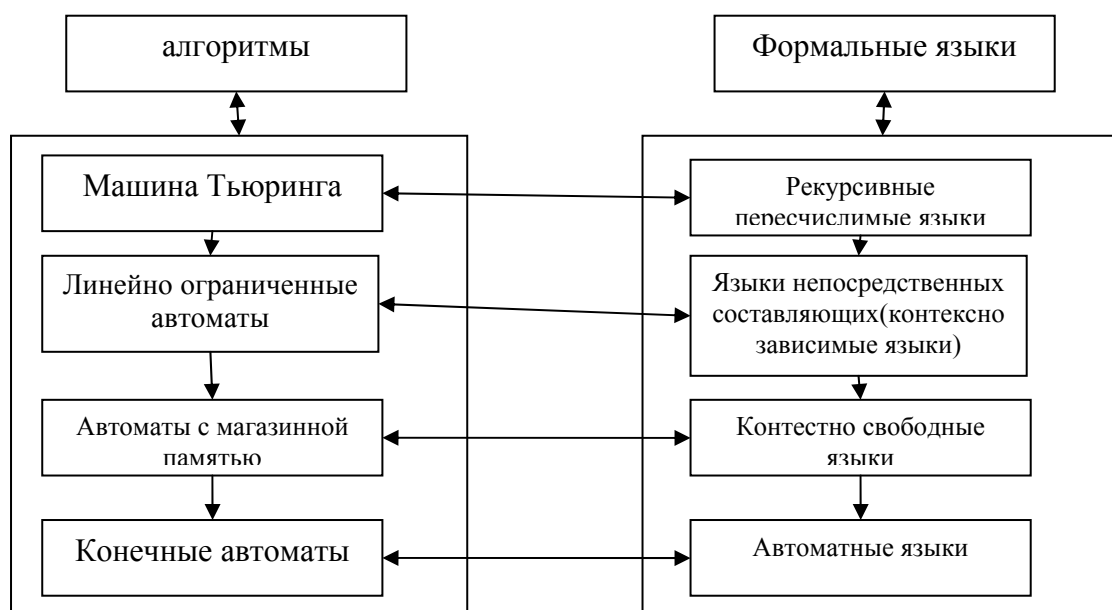


Рис.3.11

В качестве примера покажем алгоритмическую разрешимость проблемы пустоты и проблемы принадлежности для А-языков, используя для этого модель конечного автомата.

Проблемы пустоты А-языка можно сформулировать следующим образом: задан КА А, допускающий язык. Пустой ли этот язык? Приведем конструктивное доказательство разрешимости проблемы пустоты. Пусть дан КА $A = \{V_x, Q, q_0, I(q, x), Q'\}$, где V_x - входной алфавит, Q - алфавит состояний, $q_0 \in Q$ - начальное состояние, $I(q, x)$ - функция переходов, а $Q' \subseteq Q$ - множество заключительных состояний.

Если язык $L(A) = \emptyset$, то будем говорить «да» (язык пуст, если) $L(A) \neq \emptyset$, то будем говорить «нет» (язык не пуст). Алгоритм решения проблемы пустоты заключается в следующем. Вычисляется множество состояний, достижимых из q_0 . Если это множество не содержит ни одного состояния Q' , то можем сказать «да» (язык пуст) в противном случае говорим «нет» (язык не пуст). Заметим, что состояние $q_{jk} \in Q$ называется достижимым из $q_0 \in Q$, если при подаче на вход автомата некоторого входного слова $\alpha = x_{i1}x_{i2} \dots x_{ik}$ на графе автомата существует последовательность $q_0 x_{i1} q_{j1} x_{i2} q_{j2} \dots x_{ik} q_{jk}$, которая начинается в начальном состоянии q_0 и заканчивается в состоянии q_{jk} .

Проблему принадлежности можно сформулировать следующим образом: задан КА А, допускающий язык $L(A)$ и слово α ; принадлежит ли α языку $L(A)$? Доказательство разрешимости этой проблемы также проведем конструктивно. Пусть задан тот же автомат $A = \{V_x, Q, q_0, I(q, x), Q'\}$ и слово $\alpha \in V_x^*$. Будем говорить «да», если слово $\alpha \in L(A)$, и «нет», если $\alpha \notin L(A)$. Алгоритм решения проблемы принадлежности состояния $q_1 = I(q_0, x_1), q_2 = I(q_1, x_2), \dots, q_n = I(q_{n-1}, x_n)$. Если $q_n \in Q'$, то говорим «да», если $q_n \notin Q'$, то говорим «нет».

В заключение раздела отметим, что нами были рассмотрены не только конечные автоматы, которые можно использовать для распознавания правильности конструкций языка, но и конечные автоматы с выходами, которые являются преобразователями слов одного алфавита в слова другого алфавита или физически реализуемыми трансляторами А-языков. По аналогии с односторонними ленточными преобразователями можно ввести выходной алфавит и соответствующую ему выходную ленту для других типов автоматов. В этом случае получим преобразователи соответствующих типов языков.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели основные понятия теории формальных языков и показали их взаимосвязь с такими понятиями, как алгоритм, машина Тьюринга, формальная система. Мы выяснили, что грамматики Хомского являются хорошей моделью языков программирования. Однако все сказанное относится исключительно к синтаксису формальных языков (точнее говоря, языков Хомского). Фактически мы не касались одной важной их составляющей – семантики. Мало было сказано и об одном из важнейших разделов математического обеспечения – разработке трансляторов.

Семантика языка- правила, объясняющие смысл его синтаксически правильных конструкций. Семантику программ необходимо знать, чтобы знать, как исполнять данную программу на ЭВМ. Семантические правила можно записать на некотором формальном (семантическом) языке и оформить в виде семантических подпрограмм. С каждой синтаксической конструкцией тогда можно связать свою семантическую подпрограмму. Основная трудность, возникающая при таком подходе- сложность формального задания всех семантических правил языка. Это обстоятельство, так же как и сложность

формального задания контекстных условий языка, затрудняет построение трансляторов. Практически очень часто и те и другие задаются неформально, на естественном языке.

Структура транслятора зависит от уровня (типа) входного языка, а также от объема памяти и типа ЭВМ. Наиболее часто для трансляции с языков высокого уровня используются трансляторы компилирующего типа (или просто компиляторы), реализующие синтаксические методы трансляции.

Наиболее простые, так называемые однопросмотровые компиляторы осуществляют следующие действия: лексический анализ, синтаксический анализ, генерацию промежуточного кода, оптимизацию, семантический анализ, генерацию объектного кода (машинной программы).

На этапе лексического анализа определенных терминальные символы группируются в так называемые лексемы – цепочки терминальных символов языка, с которыми связывают его лексическую структуру. Лексемы – это пары, состоящие из двух компонент, одна из которых определяет синтаксическую категорию, например «константа», «идентификатор», а вторая – указатель, задающий информацию о лексеме. Для любого языка число лексем конечно. Лексический анализатор анализирует цепочки символов языка и выделяет в них лексемы, число которых в программе обычно очень большое.

Особенностью лексического анализа является то, что синтаксис лексем задается А-грамматиками, а значит, для целей лексического анализа могут быть использованы конечные автоматы. Поскольку число лексем в реальных программах велико, лексический анализ, формально являясь частью синтаксического анализа, выносится в отдельный этап. Это позволяет упростить последующий синтаксический анализ.

На этапе собственно синтаксического анализа исследуется цепочка лексем, устанавливается ее соответствие синтаксису языка, а также проверяются контактные условия языка. Синтаксис языков программирования можно задать с помощью КС-грамматик, а значит, синтаксический анализ можно выполнить с помощью автоматов (преобразователей) с магазинной памятью. Контекстные условия (вместе с синтаксисом) языка можно задать с помощью КС-грамматик. Однако при этом получаются грамматики, содержащие настолько много правил, что построить анализатор, приемлемый по сложности, не удастся. Поэтому наиболее перспективен путь построения грамматик, являющихся расширением КС-грамматик и порождающих языки уже НС-языков, но шире КС-языков. Правила вывода в таких грамMATиках похожи на правила вывода в КС-грамматиках, но их применение зависит от того, какие правила были применены на последующем шаге. Отметим, что для распознавания языков, порождаемых ими, можно использовать слегка модифицированные анализаторы с КС-языков (то есть автоматы с магазинной памятью), к числу действий которых добавляются операции по проверке КУ.

В результате синтаксического анализа мы имеем дерево вывода входной цепочки. По нему можно было бы с учетом смысла разобранной конструкции сформулировать (сгенерировать) машинную программу, однако предварительно производится оптимизация. Из практических соображений удобно использовать при этом некое промежуточное представление программы.

Семантический анализ выявляет смысл компонент опознанной при синтаксическом анализе конструкции и определяет способ ее исполнения.

На всех этих этапах компиляции широко используются таблицы, подпрограммы и применяются специальные методы их организации и работы с ними.

В принципе, возможно, создать компилятор, универсальный для целого класса языков. Такой компилятор по формально заданному синтаксису и семантике языка при неизменной своей структуре, не зависящей от исходного языка, формирует машинную программу.

Компилятор состоит из трех блоков: синтаксического загрузчика, семантического загрузчика и ядра. Взаимосвязь блоков показана на рис.4.1.



рис 4.1

Основная трудность создания универсального компилятора- трудность формального описания семантики.

ЛИТЕРАТУРА:

1. Айзерман М.А., Гусев Л.А., ***** Логика, автоматы, алгоритмы- М: Физмат***, 1963- 231с.
2. Глушков В.М. Синтаксис цифровых автоматов. – М: Физмат***, 1962*476с.
3. Мелихов А.Н. Ориентированные графы и конечные автоматы.- М: Наука, 1971-416с.
4. Мальцев А.Н. Алгоритмы и рекурсивные функции.- М: Наука, 1965- 391с.
5. Эббинхауз Г.Д., Якобс К. и др. Машины Тьюринга и рекурсивные функции. М.: Мир, 1972-264с.
6. Трахтенброт Б.А. Алгоритмы и машинное решение задач.- М: Наука, 1960-321с.
7. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции.- М.: Мир, 1978. Т.1-611с.
8. Братчикова И.Л. Синтаксис языков программирования.- М.:Физмат***, 1975-231с.
9. Грисс Д. Конструирование компиляторов для *** - М.: Мир, 1975.-544с.
10. Гросс М., Лантом Д. Теория формальных грамматик. – М.: Мир, 1971-294с.
11. Лебедев В.Н. Введение в системы программирования. – М.:Статистика, 1976-309с.
12. Гладкий А.В. Формальные грамматики и языки.- Наука, 1973-368с.

ОГЛАВЛЕНИЕ:

Введение

0.1 Объекты изучения и задачи курса.....	4
0.2 Понятие формальной системы.....	5
0.3 Интуитивное понятие алгоритма.....	6
0.4 Понятие языка.....	8
0.5 Понятие программы.....	10
0.6 Понятие транслятора.....	11

1. Алгоритмические системы

1.1 Основные понятия и определения.....	12
1.2 Комбинаторные системы.....	15
1.3 Соответствие между полутьюэвской и нормальной системами.....	17
1.4 Нормальные алгоритмы Маркова.....	20

2. машина Тьюринга

2.1 Состав и работа машины Тьюринга.....	23
2.2 Анализ и синтез машин Тьюринга.....	25
2.3 Диаграммы Тьюринга.....	27
2.4 Машины Тьюринга и вычислимые функции.....	30
2.5 Геделевский номер Машины Тьюринга.....	32
2.6 Рекурсивные и рекурсивно перечислимые множества.....	34
2.7 Неразрешимость проблемы остановки для произвольных машин Тьюринга.....	36
2.8 Комбинаторные системы и машины Тьюринга.....	38

3. Формальные грамматики и языки

3.1 Порождающие грамматики.....	40
3.2 Контекстно свободные грамматики.....	43
3.3 Автоматы с магазинной памятью.....	48
3.4 Автоматные грамматики и языки.....	52
3.5 Конечные автоматы и способы их задания.....	53
3.6 Алгебры событий. Представление событий в автоматах.....	59
3.7 Взаимосвязь автоматов и языков.....	63
ЗАКЛЮЧЕНИЕ.....	65
ЛИТЕРАТУРА.....	68