Министерство образования Российской Федерации
Восточно-Сибирский государственный технологический университет

Бильгаева Н.Ц.

Теория алгоритмов, формальных языков, грамматик и автоматов

Учебное пособие для студентов специальности
220400 -"Программное обеспечение вычислительной
техники и автоматизированных систем"

Издательство ВСГТУ
Улан-Удэ 2000

УДК 681.5.01:512

Бильгаева Н.Ц. Теория алгоритмов, формальных языков, грамматик и автоматов: Учебное пособие. Улан-Удэ: Изд-во ВСГТУ, 2000. - с.

В учебном пособии рассмотрены основные понятия теории алгоритмов, формальных языков, грамматик и автоматов; рассмотрены формальные модели алгоритмов, дается классификация формальных грамматик, описаны используемые в практике программирования алгоритмы преобразования грамматик и синтеза автоматов. По каждому разделу приведен теоретический материал, даны методические рекомендации и примеры решения задач, а также задания для самостоятельной работы.

Рецензенты:
Найханова Л.В. - к.т.н., доцент, заведующий кафедрой систем информатики ВСГТУ.
Дармаев Т.Г. – к.ф.-м. н., заведующий лабораторией геоинформационных технологий БИП СО РАН.

Печатается по решению редакционно-издательского совета Восточно-Сибирского государственного технологического университета

ВВЕДЕНИЕ

В пособии рассмотрены основные понятия теории алгоритмов и теории формальных грамматик, языков и автоматов. Пособие ориентировано на студентов младших курсов, обучающихся по специальности 220400 - "Программное обеспечение вычислительной техники и автоматизированных систем". В процессе изучения одноименных дисциплин студенты сталкиваются с тем, что учебный материал разбросан по разным источникам, написан языком труднопонимаемым для первого знакомства, а для многих понятий существует множество терминов. Поэтому при работе над пособием автор старался совместить строгость изложения основных понятий с доходчивостью восприятия.

Теоретический материал по каждому разделу сопровождается методикой решения задач, примерами, а также приводятся задания для самостоятельной работы. При написании учебного пособия широко использовались книги и монографии, указанные в списке литературы, без специальных ссылок на них в тексте пособия.

Пособие содержит введение, пять разделов и заключение. В первом разделе рассмотрены основные понятия теории алгоритмов и необходимость математического определения алгоритма. Во втором разделе рассмотрены рекурсивные функции, приводится понятие простейших функций и приемы построения сложных арифметических функций с помощью операций суперпозиции, примитивной рекурсии и минимизации. В третьем разделе дано описание машин Тьюринга, рассмотрены способы их представления, операции над машинами Тьюринга, рассмотрены алгоритмически неразрешимые проблемы теории алгоритмов. В четвертом разделе рассмотрены основные понятия формальных грамматик и языков, приводится классификация грамматик, стратегии грамматического разбора, а также эквивалентные преобразования КС-грамматик. В пятом разделе рассмотрены различные типы автоматов (конечные автоматы, автоматы с магазинной памятью, автоматы Мили и Мура) и их связь с грамматиками и языками.

Автор считает своим долгом выразить признательность Бриковой М., Жаргалову Б., Крюкову Е., которые выполнили компьютерный набор первоначального варианта текста пособия, оказали помощь при подборе заданий для самостоятельной работы.

Все замечания и пожелания по пособию автор просит направлять по адресу: 670013, г.Улан-Удэ, ул. Ключевская, 40-а, ВСГТУ.

1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ

1.1. Предварительные сведения

Понятие алгоритма, являющееся одним из основных понятий математики, возникло задолго до появления вычислительных машин. На протяжении многих веков люди пользовались интуитивным понятием алгоритма. Арабский математик IX века Мухаммед ибн Муса Аль-Хорезми впервые выдвинул идею о том, что решение любой поставленной математической и философской задачи может быть оформлено в виде последовательности механически выполняемых правил, т.е. может быть алгоритмизировано. Этого же мнения придерживались Декарт, Лейбниц, Гильберт.

Приведем интуитивное определение алгоритма.

Алгоритм - это строгая и четкая конечная система правил решения некоторого класса задач, определяющая процесс преобразования исходных данных в искомый результат.

В рамках данного определения понятие алгоритма отождествлялось с понятием метода вычисления, традиции организации вычислений складывались веками и стали составной частью общей научной культуры, формулировались и успешно применялись на практике различные вычислительные алгоритмы. Поэтому понятие метода вычислений считалось изначально ясным и потребности в изучении самого этого понятия не возникало.

1.2. Основные требования к алгоритмам

Каждый алгоритм имеет дело с *данными* - входными, промежуточными и выходными. Данные как объекты, с которыми могут работать алгоритмы, должны быть четко определены и отличимы как друг от друга, так и от другой информации.

Данные для своего размещения требуют *памяти*. Память обычно считается однородной и дискретной. Единицы измерения объема данных и памяти согласованы, при этом память может быть бесконечной.

Если выполнение алгоритма заканчивается получением результатов, то говорят, что он применим к рассматриваемой совокупности исходных данных. Любой применимый алгоритм имеет следующие основные свойства, раскрывающие его определение.

1. *Дискретность*. Это свойство состоит в том, что алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. При этом для выполнения каждого шага алгоритма требуется некоторый конечный отрезок времени, то есть преобразование исходных данных в результат осуществляется во времени дискретно.

2. *Определенность* (или детерминированность). Это свойство состоит в том, что каждое правило алгоритма должно быть четким и однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

3. *Результативность* (или конечность). Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.

4. *Массовость*. Это свойство состоит в том, что алгоритм решения задачи разрабатывается в общем виде и должен быть применим для некоторого класса задач, различающихся лишь исходными данными.

Для задания алгоритма необходимо выделить и описать следующие его элементы:

- набор объектов, составляющих совокупность данных: исходных, промежуточных и конечных;
- правило начала;
- правила непосредственной переработки информации;
- правило окончания;
- правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. Если исполнителем является компьютер, то алгоритм должен быть записан на языке программирования.

1.3. Математическое определение алгоритма

Интуитивное определение алгоритма не позволяет рассматривать свойства алгоритмов как свойства формальных объектов. Поэтому математическое определение алгоритма необходимо по следующим причинам:

1) только при наличии формального определения алгоритма можно сделать вывод о разрешимости или неразрешимости каких-либо проблем;

2) это дает возможность для сравнения алгоритмов, предназначенных для решения одинаковых задач;

3) это дает возможность для сравнения различных проблем по сложности алгоритмов их решения.

Одна из причин расплывчатости интуитивного определения алгоритма состоит в том, что объектом алгоритма может оказаться все, что угодно. Поэтому естественно было начать с формализации понятия объекта. В вычислительных алгоритмах объектами работы являются числа, в алгоритме шахматной игры - фигуры и позиции, при алгоритмизации производственных процессов объектами служат, например, показания приборов. Однако алгоритмы оперируют не с объектами реального мира, а с изображениями этих объектов.

Поэтому алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между изображениями объектов в абстрактных алфавитах.

Абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов нас совершенно не интересует. Символом абстрактных алфавитов можно считать буквы алфавита какого-либо языка, цифры, любые значки и даже слова некоторого конкретного языка. Основным требованием к алфавиту является его конечность. Таким образом, можно утверждать, что алфавит - это конечное множество различных символов. Алфавит, как любое множество, задается перечислением его элементов.

Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова. Тогда можно сформулировать следующее определение.

Алгоритм есть четкая конечная система правил для преобразования слов из некоторого алфавита в слова из этого же алфавита.

Слово, к которому применяется алгоритм, называется *входным*. Слово, вырабатываемое в результате применения алгоритма, называется *выходным*.

Совокупность слов, к которым применим данный алгоритм, называется областью применимости этого алгоритма.

Формальные определения алгоритма появились в 30-х - 40-х годах XX века. Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм. Первый тип связывает понятие алгоритма с наиболее традиционными понятиями математики - вычислениями и числовыми функциями. Наиболее развитая и изученная модель этого типа - рекурсивные функции - является исторически первой формализацией понятия алгоритма. Эта модель основана на функциональном подходе и рассматривает понятие алгоритма с точки зрения того, что можно вычислить с его помощью.

Второй тип основан на представлении алгоритма как некоторого детерминированного устройства, способного выполнять в каждый отдельный момент некоторые примитивные операции, или инструкции. Такое представление не оставляет сомнений в однозначности алгоритма и элементарности его шагов. Основной теоретической моделью этого типа, созданной в 30-х годах, является машина Тьюринга, которая представляет собой автоматную модель, в основе которой лежит анализ процесса выполнения алгоритма как совокупности набора инструкций.

Третий тип алгоритмических моделей - это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т.е. замены части слова (подслова) другим словом. Преимущество этого типа состоит в его максимальной абстрактности и возможности применить понятие алгоритма к объектам произвольной природы. Модели второго и третьего типа довольно близки и отличаются в основном эвристическими подходами. Примерами моделей этого типа являются нормальные алгоритмы Маркова и канонические системы Поста.

1.4. Понятие алфавитного оператора

Понятие алфавитного оператора является наиболее общим. К нему фактически сводятся любые процессы преобразования информации. К изучению алфавитных операторов фактически сводится теория любых преобразователей информации. Основой теории алфавитных операторов являются способы их задания. Если область определения алфавитного оператора конечна, то оператор может быть задан простой таблицей соответствия. В случае бесконечной области определения алфавитного оператора он задается системой правил, позволяющей за конечное число шагов найти выходное слово, соответствующее заданному входному слову. Алфавитные операторы, задаваемые с помощью конечной системы правил, называются алгоритмами.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются *численными алгоритмами*.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к логическим действиям, называются *логическими алгоритмами*.

Различают однозначные и многозначные алфавитные операторы.

Под *однозначным алфавитным оператором* понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие не более одного выходного слова.

Под *многозначным алфавитным оператором* понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие более одного выходного слова.

Алфавитный оператор, не сопоставляющий данному входному слову a_i никакого выходного слова b_j (в том числе и пустого), не определен на этом слове.

Совокупность всех слов, на которых алфавитный оператор определен, называется *областью его определения*.

Два алфавитных оператора считаются *равными*, если равны соответствующие им алфавитные операторы и совпадает система правил, задающих действие этих алгоритмов на выходные слова.

Два алгоритма считаются *эквивалентными*, если у них совпадают алфавитные операторы, но не совпадают способы их задания (система правил). Обычно в теории алгоритмов рассматриваются лишь такие алгоритмы, которым соответствуют однозначные алфавитные операторы.

1.5. Задания для самостоятельной работы

Во всех заданиях необходимо разработать схемы алгоритмов и проанализировать процесс реализации алгоритма, т.е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным исходным данным.

1. Разработать словесные алгоритмы вычитания из некоторого числа A последовательности n чисел $b_1, b_2, ..., b_n$:

а) алгоритм вычисления по формуле:

$$C=(...((A - b_1) - b_2) - ... - b_n)$$

б) алгоритм вычисления по формуле:

$$C = A - \sum_{i=1}^{n} b_i .$$

2. Разработать алгоритм вычисления по формуле:

$$C_k = \sum_{i=1}^{n} a_i - b_k (1 \le i \le n, 1 \le k \le m) .$$

3. Разработать алгоритм вычисления по формуле:

$$C_i = \prod_{k=1}^{m} a_k \times b_i (1 \le i \le m, 1 \le k \le m) .$$

4. Разработать алгоритм поиска максимального (минимального) элемента из последовательности, заданной в виде одномерного массива $A = \{a_1, a_2, ... , a_k\}$.

5. Разработать алгоритм определения количества одинаковых чисел в последовательности, заданной в виде одномерного массива $A=\{a_1, a_2, . . . , a_k\}$.

6. Разработать алгоритм подсчета количества одинаковых элементов в матрице B размерностью $n \times m$. Размерность матрицы вводится с клавиатуры.

7. Разработать алгоритм умножения матрицы на вектор.

8. Разработать алгоритм умножения матрицы на матрицу.

9. Разработать алгоритм транспонирования матрицы.

10. Разработать алгоритм, реализующий операцию объединения следующих двух множеств:

$A = \{a_1, a_2, . . ., a_k\}$ и $B= \{b_1, b_2, ... , b_n\}$

2. РЕКУРСИВНЫЕ ФУНКЦИИ

2.1. Общие сведения

Первой алгоритмической системой была система, основанная на использовании конструктивно определяемых арифметических (целочисленных) функций, получивших специальное название *рекурсивных функций*.

Рекурсией называется способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается известным образом через значения определяемой функции для меньших значений аргументов.

Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Наиболее просто такую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, а слова, имеющие одинаковую длину, - в произвольном порядке.

После нумерации входных и выходных слов в произвольном алфавитном операторе этот оператор превращается в функцию $y = f(x)$, в которой аргумент x и функция y принимают неотрицательные целочисленные значения. Функция $f(x)$ может быть определена не для всех значений x, а лишь для тех, которые составляют область определения этой функции.

2.2. Понятие простейших функций

Числовые функции, значение которых можно установить посредством некоторого алгоритма, называются вычислимыми функциями.

Для того чтобы описать класс функций с помощью рекурсивных определений, рассмотрим набор простейших функций:

1) $Z(x_1, x_2, ..., x_n) = 0$ - нуль-функция, которая определена для всех неотрицательных значений аргумента;

2) $s(x) = x+1$ - функция непосредственного следования, также определенная для всех целых неотрицательных значений своего аргумента;

3) $I^n_m(x_1, x_2, \ldots, x_m, \ldots, x_n) = x_m$ - функция выбора (тождества), повторяющая значения своих аргументов.

Используя простейшие функции в качестве исходных функций, можно с помощью небольшого числа общих конструктивных приемов строить сложные арифметические функции. В теории рекурсивных функций особо важное значение имеют три операции: суперпозиции, примитивной рекурсии и минимизации.

2.2.1. Оператор суперпозиции

Оператором суперпозиции S называется подстановка в функцию от m переменных m функций от n одних и тех же переменных. Она дает новую функцию от n переменных. Например, из функций $f(x_1, x_2, ..., x_m)$, $f_1(x_1, x_2, ..., x_n)$, $f_2(x_1, x_2, ..., x_n)$, . . ., $f_m(x_1, x_2, ..., x_n)$ можно получить новую функцию:

$$S^{m+1}(f, f_1, f_2, ..., f_m) = g(x_1, x_2, ..., x_n) = f(f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_m(x_1, ..., x_n)). \quad (1)$$

В операции суперпозиции S^{m+1} индекс сверху указывает на число функций.

Таким образом, при помощи оператора суперпозиции и функции выбора можно выразить любую подстановку функции в функцию.

Например, осуществляя операцию суперпозиции функций $f(x) = 0$ и $g(x) = x+1$, получим функцию:

$h(x) = g(f(x)) = 0 + 1 = 1$.

При суперпозиции функции g(x) с этой же функцией получим функцию h(x) = g(g(x)) = x + 2.

Используя подстановку и функции тождества, можно переставлять и отождествлять аргументы в функции:

$$f(x_2, x_1, x_3, \ldots, x_n) = f(I_2^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \ldots, x_n);$$

$$f(x_1, x_1, x_3, \ldots, x_n) = f(I_1^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \ldots, x_n).$$

Таким образом, если заданы функции тождества и операторы суперпозиции, то можно считать заданными всевозможные операторы подстановки функций в функции, а также переименования, перестановки и отождествления переменных.

2.2.2. Оператор примитивной рекурсии

Оператор примитивной рекурсии R_n позволяет определить (n+1) - местную функцию f по двум заданным функциям, одна из которых является n- местной функцией g, а другая (n+2) - местной функцией h.

Функция $f(x_1, x_2, \ldots, x_n, y)$ получается оператором примитивной рекурсии из функций $g(x_1, x_2, \ldots, x_n)$ и функции $h(x_1, x_2, \ldots, x_n, y, z)$, если:

$$f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n); \qquad (2)$$

$$f(x_1, x_2, \ldots, x_n, y+1) = h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y)).$$

Приведенная пара равенств (2) называется схемой примитивной рекурсии. Для понимания операции примитивной рекурсии необходимо заметить, что всякую функцию от меньшего числа аргументов можно рассматривать как функцию от большего числа аргументов. Существенным в операторе примитивной рекурсии является то, что независимо от числа переменных в f рекурсия ведется только по одной переменной y. Остальные n переменных $x_1, x_2, \ldots, x_n$ на момент применения схемы (2) зафиксированы и играют роль параметров.

2.2.3. Оператор минимизации

Отношение $P(x_1, x_2, \ldots, x_n)$ называется примитивно-рекурсивным, если примитивно-рекурсивна его характеристическая функция:

$$\chi(x_1, x_2, \ldots, x_n) = \begin{cases} 1, \text{если } P(x_1, x_2, \ldots, x_n) - \text{"истина"}, \\ 0, \text{если } P(x_1, x_2, \ldots, x_n) - \text{"ложь"}. \end{cases}$$

Предикат называется примитивно-рекурсивным, если его характеристическая функция примитивно-рекурсивна.

Функция $f(x_1, x_2, \ldots, x_n)$ получается посредством оператора минимизации из предиката $P(x_1, x_2, \ldots, x_n, z)$, если в любой точке значением функции $f(x_1, x_2, \ldots, x_n)$ является минимальное значение z, обращающее предикат $P(x_1, x_2, \ldots, x_n, z)$ в значение «истина»:

$$f(x_1, x_2, \ldots, x_n) = \mu_z (P(x_1, x_2, \ldots, x_n, z)),$$

где μ_z – оператор минимизации.

2.2.4. Ограниченный оператор минимизации

Функция $f(x_1, x_2, \ldots, x_n)$ получается ограниченным оператором минимизации из предиката $P(x_1, x_2, \ldots, x_n, z)$ и функции $U(x_1, x_2, \ldots, x_n)$, если в любой точке значение этой функции определено следующим образом:

1) для любого $z < U(x_1, x_2, \ldots, x_n)$ такого, что $P(x_1, x_2, \ldots, x_n, z)$ = "истина", значение функции $f(x_1, x_2, \ldots, x_n) = \mu_z(P(x_1, \ldots, x_n, z))$,

2) не для любого $z < U(x_1, x_2, \ldots, x_n)$ такого, что $P(x_1, x_2, \ldots, x_n, z)$ = "истина", значение функции $f(x_1, x_2, \ldots, x_n) = U(x_1, x_2, \ldots, x_n)$.

Данное определение записывается следующим образом:

$$f(x_1, x_2, ..., x_n) = \mu_{z < U(x)} (P(x_1, x_2, ..., x_n, z)).$$

Ограничение z в ограниченном операторе минимизации дает гарантию окончания вычислений, поскольку оно оценивает сверху число вычислений предиката P. Возможность оценить сверху количество вычислений является существенной особенностью примитивно-рекурсивных функций.

2.3. Примитивно-рекурсивные и частично-рекурсивные функции

Большинство вычислимых функций относится к классу примитивно-рекурсивных функций.

Функция называется *примитивно-рекурсивной*, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции и примитивной рекурсии.

Если некоторые функции являются примитивно-рекурсивными, то в результате применения к ним операторов суперпозиции или примитивной рекурсии можно получить новые примитивно-рекурсивные функции.

Существует три возможности доказательства того, что функция является примитивно-рекурсивной:

а) показать, что заданная функция является простейшей;

б) показать, что заданная функция построена с помощью оператора суперпозиции;

в) показать, что заданная функция построена с помощью оператора примитивной рекурсии.

Тем не менее примитивно-рекурсивные функции не охватывают все вычислимые функции, которые могут быть определены конструктивно. При построении этих функций могут использоваться другие операции. Функция называется *частично-рекурсивной*, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции, примитивной рекурсии и минимизации. Указанные операции могут быть выполнены в любой последовательности и любое конечное число раз. Если такие функции оказываются всюду определенными, то они называются общерекурсивными функциями. Частично-рекурсивные функции вычислимы путем определенной процедуры механического характера. Практически понятием частично-рекурсивных функций пользуются для доказательства алгоритмической разрешимости или неразрешимости проблем. Приведем тезис Черча, который связывает понятие алгоритма и строгое математическое понятие частично-рекурсивной функции. Тезис Черча: всякий алгоритм может быть реализован частично-рекурсивной функцией. Утверждение о несуществовании частично-рекурсивной функции эквивалентно несуществованию алгоритма решения соответствующей задачи.

2.4. Типы рекурсивных алгоритмов

Эффективность разработки рекурсивного алгоритма определяется наличием некоторых условий:

1) если исходные данные имеют рекурсивную структуру, то процедуры анализа таких структур будут более эффективны, если они сами рекурсивны;

2) если алгоритм обработки некоторого набора данных построить, разбивая данные на части и обрабатывая в отдельности каждую часть, то из частичных решений можно получить общее;

3) если по условию задачи необходимо выбрать оптимальный вариант из некоторого множества решений, то искомое решение может быть найдено через конечное число шагов. При этом на каждом шаге удаляется часть информации, и далее задача решается на меньшем объеме данных. Поиск решения завершается после окончания данных либо при нахождении искомого решения на текущем наборе данных.

2.5. Методика решения задач

2.5.1. Использование оператора примитивной рекурсии

Пример 1. Доказать примитивную рекурсивность функции $f(x, y) = x + y$.

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(x, 0) = x$,

$f(x, y+1) = x + y + 1 = f(x, y) + 1$.

Чтобы показать соответствие данной рекурсивной схемы схеме примитивной рекурсии, воспользуемся функциями выбора и следования:

$f(x, 0) = x = I(x)$,

$f(x, y+1) = f(x, y) + 1 = S(f(x, y)) = S(I_3(x, y, f(x, y)))$.

Пример 2. Доказать примитивную рекурсивность функции $f(x, y) = x \cdot y$.

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(x, 0) = 0$,

$f(x, y+1) = x \cdot (y + 1) = x \cdot y + x = f(x, y) + x$,

из которого следует, что $Z(x) = x \cdot 0$.

Обозначим $x \cdot y = p(x, y)$, тогда:

$p(x, 0) = Z(x)$;

$p(x, y+1) = p(x, y) + x = S(p(x, y), x) = S(I_1^3(x, y, p(x, y)), I_3^3(x, y, p(x, y)))$.

Пример 3. Доказать примитивную рекурсивность функции

$$Sg(x) = \begin{cases} 0, \text{если } x = 0, \\ 1, \text{если } x \neq 0. \end{cases}$$

Решение. Рассмотрим способ рекурсивного определения данной функции:

$Sg(0) = 0 = Z(x)$,

$Sg(x+1) = Z(x) + 1 = 1 = S(Z(x))$.

Пример 4. Доказать примитивную рекурсивность функции $f(x) = 2^x$.

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(0) = 1 = S(Z(x))$,

$f(x+1) = 2 \cdot 2^x = 2 \cdot f(x) = S(S(Z(x)))$.

Пример 5. Доказать примитивную рекурсивность функции $f(x, y) = x^y$.

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(x, 0) = 1 = S(Z(x))$;

$f(x, y+1) = x^{y+1} = x^y \cdot x = f(x, y) \cdot I_1^2(x, y) = p(I_3^3(x, y, f(x, y)), I_1^3(x, y, f(x, y)))$.

2.5.2. Использование оператора минимизации

Пример 1. Пусть $f(x, y) = \mu_z(2 \cdot x + z = y+1)$, откуда предикат $P(x, y, z) = 2 \cdot x + z = y + 1$.

Покажем примитивную рекурсивность предиката P. Его характеристическая функция может быть представлена следующим выражением:

$Sg(Sg((2 \cdot x + z) \div (y + 1)) + Sg((y + 1) \div (2 \cdot x + z))$.

Найдем значение функции в точке (1, 5):

При $z = 0$: $P(1, 5, 0) = 2 \cdot 1 + 0 = 5 + 1$ - "ложь",

$z = 1$: $P(1, 5, 1) = 2 \cdot 1 + 1 = 5 + 1$ - "ложь",

$z = 2$: $P(1, 5, 1) = 2 \cdot 1 + 2 = 5 + 1$ - "ложь",

$z = 3$: $P(1, 5, 1) = 2 \cdot 1 + 3 = 5 + 1$ - "ложь",

$z = 4$: $P(1, 5, 4) = 2 \cdot 1 + 4 = 5 + 1$ - "истина".

Таким образом, минимальное значение переменной z, обращающее предикат в "истину", дает значение функции в точке (1, 5), и оно равно 4.

2.5.3. Использование ограниченного оператора минимизации

Ограниченный оператор минимизации является удобным средством для построения обратных функций.

Пример 1. С помощью ограниченного оператора минимизации определить функцию y(x, z) = $\left[\frac{z}{x}\right]$ - "целая часть от деления z на x" - как функцию, обратную умножению.

Решение.

z / x = y, тогда y · x = z,

z / x < y + 1, поэтому z < x(y + 1).

С использованием ограниченного оператора минимизации запишем исходную функцию через обратную:

$y(x, z) = \left[\frac{z}{x}\right] = \mu_{z \geq y}(P(x, y, z))$,

предикат P(x, y, z) = z < x(y + 1).

Тогда $\left[\frac{z}{x}\right] = \mu_{z \geq y}(z < x(y+1))$.

Вычислим значение функции при z = 11, x = 2.

При y = 0: P(2, 11, 0) = 2(0+1) < 11 - "ложь",

y = 1: P(2, 11, 1) = 2(1+1) < 11 - "ложь",

. . .

y = 5: P(2, 11, 5) = 2(5+1) > 11 - "истина".

Следовательно, $\left[\frac{z}{x}\right] = 5$.

Пример 2. С помощью ограниченного оператора минимизации определить функцию $f(x) = \lfloor\sqrt{x}\rfloor$ - "целая часть от корня из x".

Решение.

Пусть $\lfloor\sqrt{x}\rfloor = z$. Тогда $\sqrt{x} < z + 1$, а предикат P равен:

$P(x, z) = x < (z + 1)^2$.

С использованием ограниченного оператора минимизации запишем исходную функцию через обратную: $\lfloor\sqrt{x}\rfloor = \mu_{z \leq x}(x < (z+1)^2)$.

Вычислим значение функции при x = 17.

При z = 0: P(17, 0) = 17 < 1 - "ложь",

z = 1: P(17, 1) = 17 < 4 - "ложь",

. . .

z = 4: P(17, 4) = 17 < 25 - "истина".

Следовательно, $\lfloor\sqrt{x}\rfloor = 4$, что и требовалось доказать.

Пример 3. С помощью ограниченного оператора минимизации определить функцию f(x, y) = $\lfloor \log_y x \rfloor$ - "целая часть от $\log_y x$".

Решение.

Пусть $y^z = x$. Тогда $y^{z+1} > x$.

С использованием ограниченного оператора минимизации запишем исходную функцию как:

$\lfloor \log_y x \rfloor = \mu_{z < x}(y^{z+1} > x)$.

Предикат $P(x, y, z) = y^{z+1} > x$.

Вычислим значение функции при $x = 8$, $y = 2$.

При $z = 0$: $P(8, 2, 0) = 1 > 8$ - "ложь",

$z = 1$: $P(8, 2, 1) = 2 > 8$ - "ложь",

$z = 2$: $P(8, 2, 2) = 8 > 8$ - "ложь",

$z = 3$: $P(8, 2, 3) = 16 > 8$ - "истина".

Следовательно, $\lfloor \log_y x \rfloor = 3$.

2.6. Задания для самостоятельной работы

Доказать примитивную рекурсивность следующих функций.

1. Усеченное вычитание

$$x_1 \div x_2 = \begin{cases} x_1 - x_2, \text{если } x_1 > x_2, \\ 0, \qquad \text{если } x_1 < x_2. \end{cases}$$

2. Функция знака

$$Sg(x) = \begin{cases} 0, \text{если } x = 0, \\ 1, \text{если } x \neq 0. \end{cases}$$

3. Нахождение минимального значения из двух заданных чисел $f(x, y) = \min(x, y)$.
4. Нахождение максимального значения из двух заданных чисел $f(x, y) = \max(x, y)$.
5. Усеченное вычитание

$$x \div 1 = \begin{cases} x - 1, \text{если } x > 1, \\ 0, \qquad \text{если } x < 1. \end{cases}$$

6. Функция r(x, y) - остаток от деления y на x.
7. Функция q(x, y) - частное от деления y на x.
8. Доказать, что предикат $Pd_n(x)$ "x делится на n" примитивно-рекурсивен для любого n.
9. Доказать, что предикат $Pd_{n,m}(x)$ "x делится на n и m" примитивно-рекурсивен для любых n и m.
10. Доказать, что отношение $x_1 > x_2$ примитивно-рекурсивно.
11. Доказать, что предикат $f(x) = g(x)$ примитивно-рекурсивен, если функции f(x) и g(x) примитивно-рекурсивны.
12. Дано множество слов одинаковой длины, причем первые два слова выделены. Построить цепь от первого выделенного слова ко второму так, чтобы все слова этой цепи были только из заданного множества. Соседние слова построенной цепи должны отличаться только одной буквой.
13. Дано множество слов. Построить из них кроссворд заданной конфигурации (число слов может быть больше требуемого количества для заполнения кроссворда).
14. Грани кубика разбиты на клетки 5x5. Каждая из клеток выкрашена в белый или синий цвет. Переход из клетки в клетку допускается только через общую сторону при условии совпадения цветов этих клеток. Построить маршрут перехода из одной заданной клетки в другую заданную клетку этого же цвета.
15. Имеется клетчатая ткань размером NxN клеток. Разрезать ее на M квадратных частей, не нарушая целостности клеток.

3. МАШИНЫ ТЬЮРИНГА

3.1. Общие сведения

Одним из первых формальных определений алгоритма было определение английского математика А. Тьюринга, который в 1936 г. описал схему некоторой гипотетической

(абстрактной) машины и формализовал правила выполнения действий при помощи описания работы этой машины.

Машина Тьюринга является абстракцией, которую нельзя реализовать практически. Поэтому алгоритмы для машины Тьюринга должны выполняться другими средствами. Основным следствием формализации алгоритмов с использованием машины Тьюринга является возможность доказательства существования или несуществования алгоритмов решения различных задач.

Описывая различные алгоритмы для машин Тьюринга и доказывая реализуемость всевозможных композиций алгоритмов, Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции, что позволило ему выступить со следующим тезисом: “Всякий алгоритм может быть реализован соответствующей машиной Тьюринга”. Доказать тезис Тьюринга нельзя, так как в его формулировке не определено понятие “всякий алгоритм”. Его можно только обосновать, представляя различные алгоритмы в виде машин Тьюринга. Было доказано, что класс функций, вычислимых на этих машинах, совпадает с классом частично рекурсивных функций.

3.2. Неформальное определение машины Тьюринга

Машина Тьюринга представляет собой автомат,

имеющий бесконечную в обе стороны ленту, считывающую головку и управляющее устройство. Управляющее устройство может находиться в одном из состояний, образующих конечное множество $Q = \{q_0, q_1, ..., q_n\}$. Множество Q называют внутренним алфавитом машины Тьюринга.

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том, что ее запоминающее устройство представляет собой бесконечную ленту, из-за которой невозможна ее физическая реализация. Лента разделена на ячейки, в каждой из которых может быть записан один из символов конечного алфавита $A = \{a_0, a_1, . . . , a_m\}$, который называют входным алфавитом машины Тьюринга. Во время функционирования машины Тьюринга может быть заполнено конечное число ячеек. Считывающая головка в каждый момент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке и состояния управляющего устройства записывает в ячейку новый символ или оставляет его без изменения, сдвигается на ячейку влево или вправо или остается на месте. При этом управляющее устройство переходит в новое состояние или остается в старом. Среди состояний управляющего устройства выделены начальное состояние q_0 и заключительное состояние q_z.

Таким образом, за один такт работы машина Тьюринга может считать символ, записать вместо него новый или оставить его без изменения и сдвинуть головку на одну ячейку влево или вправо или оставить ее на месте.

3.3. Формальное определение машины Тьюринга

Алфавит A – это некоторое конечное множество символов.

Цепочкой над алфавитом называется последовательность символов из этого алфавита. Длина цепочки x представляет собой число символов в цепочке и обозначается $| x |$. Длина пустой цепочки равна нулю.

Конкатенацией цепочек X и Y называют цепочку, полученную приписыванием символов цепочки Y справа к цепочке X.

Машиной Тьюринга называется семерка вида

$T = (Q, A, \delta, p_0, p_z, a_0, a_1)$, где

Q – конечное множество состояний, в которых может находиться управляющее устройство;

A – входной алфавит;

δ – функция переходов, $\delta = Q \cdot A \rightarrow Q \cdot A \cdot S$, где

S = {R, L, E} - направления сдвига;

p_0 – начальное состояние; $p_0 \in Q$;

p_z – заключительное состояние; $p_z \in Q$;

a_0 – символ для обозначения пустой ячейки, $a_0 \in A$;

a_1 – специальный символ - разделитель цепочек на ленте, $a_1 \in A$.

Командой машины Тьюринга называется элемент функции переходов $qa \rightarrow pbr$, где q и $p \in Q$; a и $b \in A$; $r \in S$. Каждая команда описывает один такт работы машины Тьюринга.

Конфигурация машины Тьюринга представляется следующим образом: t = <CqaB>, где

C - цепочка, находящаяся слева от считывающей головки;

q - состояние машины Тьюринга;

a - символ, находящийся под головкой машины Тьюринга;

B - цепочка, находящаяся справа от головки машины Тьюринга.

Конфигурация <CqaB> непосредственно переходит в конфигурацию $<C_nq_na_nB_n>$, если новая конфигурация получена в результате применения одной команды к исходной конфигурации.

Обозначим непосредственный переход из одной конфигурации в другую следующим образом: $<CqaB> \Rightarrow <C_nq_na_nB_n>$.

Конфигурация, содержащая начальное состояние, называется начальной, а содержащая заключительное состояние - заключительной. Если цепочка C в конфигурации пустая, то начальная и заключительная конфигурации называются стандартными.

Машина Тьюринга перерабатывает цепочку x в цепочку y, если, действуя из начальной конфигурации и имея на ленте цепочку x, машина Тьюринга переходит в заключительную конфигурацию, имея на ленте цепочку y. Если начальная и заключительная конфигурации являются стандартными, то процесс переработки x в y является правильной переработкой.

3.4. Способы представления машины Тьюринга

Существует три способа представления машины Тьюринга: совокупностью команд, в виде графа, в виде таблицы соответствия.

3.4.1. Представление машины Тьюринга совокупностью команд

Совокупность всех команд, которые может выполнять машина, называется ее программой.

Машина Тьюринга считается заданной, если заданы ее внешний и внутренний алфавиты, программа, начальная конфигурация и указано, какие из символов обозначают пустую ячейку и заключительное состояние.

Чтобы записать совокупность команд, нужно воспользоваться следующими правилами:

1) начальному шагу алгоритма ставится в соответствие q_0 - начальное состояние;

2) циклы реализуются так, что последнее действие цикла должно соответствовать переходу в то состояние, которое соответствует началу цикла;

3) соседним шагам алгоритма соответствует переход в смежные состояния, которые соответствуют этим пунктам;

4) последний шаг алгоритма вызывает переход в заключительное состояние.

В качестве примера рассмотрим совокупность команд машины Тьюринга, которая инвертирует входную цепочку, записанную с использованием нулей и единиц.

Пусть алфавит машины Тьюринга задан множеством A={0, 1, ε}, где символ ε соответствует пустой ячейке, а число состояний устройства управления задано в виде множества $Q = \{q_0, q_1, q_z\}$.

Если, например, начальная конфигурация имеет вид $q_0$110011, то конечная конфигурация после завершения операции инвертирования должна иметь вид q_z001100. Для решения задачи машиной будет порождена следующая последовательность команд:

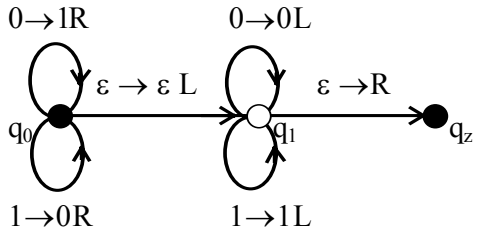
$q_0 1 \to q_0 0R$, $q_1 0 \to q_1 0L$,
$q_0 0 \to q_0 1R$, $q_1 1 \to q_1 1L$,
$q_0 \varepsilon \to q_1 \varepsilon L$, $q_1 \varepsilon \to q_z \varepsilon R$.

В стандартной начальной конфигурации головка стоит над первым символом слева, и устройство управления находится в начальном состоянии. На следующем такте машина Тьюринга, не меняя своего состояния, заменяет символ 0 на 1 или 1 на 0 и сдвигается вправо на один символ. После просмотра всей цепочки под головкой оказывается символ, указывающий на пустую ячейку. В этом случае машина Тьюринга переходит в новое состояние и сдвигается влево на один символ. На последующих тактах управляющее устройство не меняет своего состояния, оставляет без изменения символ под головкой и перемещается влево до тех пор, пока не встретит пустую ячейку. Встретив пустую ячейку, машина Тьюринга переходит в заключительное состояние и перемещается вправо на один символ, переходя в стандартную заключительную конфигурацию.

3.4.2. Представление машины Тьюринга графом

При представлении машины Тьюринга посредством графа необходимо каждому состоянию поставить в соответствие вершину графа, а каждой команде - помеченную дугу.

Машина Тьюринга из рассмотренного примера инвертирования цепочки, состоящей из символов 0 и 1, будет представлена в виде графа следующим образом:



Начальная и конечная вершины графа обычно выделяются; на рисунке они отмечены черным кружком. Если на очередном такте работы машины Тьюринга символ на ленте не изменяется, то в правой части команды его можно не писать (ε на ребре в состояние q_z).

3.4.3. Представление машины Тьюринга таблицей соответствия

При представлении машины Тьюринга данным способом составляется таблица, в которой каждому состоянию соответствует строка, а каждому символу из входного алфавита - столбец. В клетках таблицы на пересечении строки и столбца будет находиться действие (или правая часть команды).

Таблица соответствия для задания машины Тьюринга из рассмотренного примера будет выглядеть следующим образом:

	0	1	ε
q_0	q_0 1R	Q_0 0R	$q_1 \varepsilon$ L
q_1	q_1 0L	Q_1 1L	$q_z \varepsilon$L

Инвертирование входной цепочки можно выполнить программой машины Тьюринга, приведенной в таблице соответствия. Эта программа включает шесть команд.

3.5. Вычислимые функции

Говорят, что машина Тьюринга вычисляет функцию $f(x_1, x_2, ..., x_n)$, если выполняются следующие условия:

1) для любых x_1, x_2, ..., x_n, принадлежащих области определения функции, машина Тьюринга из начальной конфигурации, имея на ленте представление аргументов, переходит в заключительную конфигурацию, имея на ленте результат (представление функции);

2) для любых x_1, x_2, ..., x_n, не принадлежащих области определения функции, машина Тьюринга из начальной конфигурации работает бесконечно.

Если начальная и заключительная конфигурации машины Тьюринга являются стандартными, то говорят, что машина Тьюринга правильно вычисляет функцию f.

Функция называется вычислимой по Тьюрингу, если существует машина Тьюринга, вычисляющая ее.

Для того чтобы доказать вычислимость функции, а в дальнейшем и существование алгоритма, необходимо построить машину Тьюринга, реализация которой на практике зачастую представляет собой трудоемкую задачу. В связи с этим возникает необходимость разбиения алгоритма на отдельные задачи, каждая из которых будет решаться отдельной машиной Тьюринга. Если объединить программы этих машин, то получится новая программа, позволяющая решить исходную задачу.

Машины Тьюринга могут вычислять искомую функцию с восстановлением и без восстановления. Вычисление функции с восстановлением означает работу машины Тьюринга с сохранением исходных данных:

$$p_0 1^{X1} * \ldots * 1^{Xn} \Rightarrow^* p_z 1^{f(X1, X2, \ldots, Xn)} * 1^{X1} * \ldots * 1^{Xn}.$$

Приведенное определение позволяет получать на ленте сначала результат, а затем исходные данные. В отдельных случаях удобно сделать наоборот:

$$p_0 1^{X1} * \ldots * 1^{Xn} \Rightarrow^* 1^{X1} * \ldots * 1^{Xn} * p_z 1^{f(X1, X2, \ldots, Xn)}.$$

Вычисление функции без восстановления означает работу машины Тьюринга без сохранения исходных данных:

$$p_0 1^{X1} * \ldots * 1^{Xn} \Rightarrow^* p_z 1^{f(X1, X2, \ldots, Xn)}.$$

Справедливо утверждение, что всякая правильно вычислимая функция правильно вычислима с восстановлением.

3.6. Операции над машинами Тьюринга

1. Композиция машин Тьюринга. Пусть машины T_1 и T_2 имеют программы P_1 и P_2. Предположим, что внутренние алфавиты этих машин не пересекаются; пусть q_{z1} - заключительное состояние машины T_1, а q_{02} - начальное состояние машины T_2. Заменим всюду в программе P_1 заключительное состояние q_{z1} на начальное состояние q_{02} машины T_2 и полученную программу объединим с программой P_2. Новая программа P определяет машину T, называемую композицией машин T_1 и T_2 по паре состояний (q_{z1}, q_{02}). Композиция машин может быть обозначена $T_1 \cdot T_2$ или T_1T_2. Более подробно композиция машин записывается следующим образом:

$T = T(T_1, T_2, (q_{z1}, q_{02}))$, где

$T_1 = (Q_1, A_1, \delta_1, p_{01}, p_{z1}, a_{01}, a_{11})$,

$T_2 = (Q_2, A_2, \delta_2, p_{02}, p_{z2}, a_{02}, a_{12})$.

Пусть $a_{01} = a_{02} = a_0$ и $a_{11} = a_{12} = a_1$. Внешний алфавит композиции T_1T_2 является объединением внешних алфавитов машин T_1 и T_2:

$$T = (Q_1 \cup Q_2 \setminus \{p_{z1}\}, A_1 \cup A_2, \ |^{p_{02}}_{p_{z1}} (\delta_1 \cup \delta_2), p_{01}, p_{z2}, a_0, a_1).$$

Операция композиции, выполняемая над алгоритмами, позволяет получать новые, более сложные алгоритмы из ранее известных простых алгоритмов.

2. Итерация машины Тьюринга. Эта операция применима только к одной машине.

Пусть q_z - заключительное состояние машины T, а q_n - какое-либо состояние машины T, не являющееся заключительным. Заменим всюду в программе P машины T состояние q_z на q_n . Полученная программа определяет новую машину $T'(q_z, q_n)$, которая называется итерацией машины T по паре состояний (q_z, q_n). Если машина Тьюринга имеет одно заключительное состояние, то после выполнения итерации получается машина, не имеющая заключительного состояния.

3. Разветвление машин Тьюринга. Пусть машины Тьюринга T_1, T_2 и T_3 задаются программами P_1, P_2 и P_3 соответственно. Считаем, что внутренние алфавиты этих машин попарно не пересекаются. Пусть q_{z11} и q_{z12} - какие-либо различные заключительные состояния машины T_1. Заменим всюду в программе P_1 состояние q_{z11} начальным состоянием q_{02} машины T_2, а состояние q_{z12} начальным состоянием q_{03} машины T_3. Затем новую программу объединим с программами P_2 и P_3. Получим программу P, задающую машину Тьюринга и обозначаемую:

$T = T(T_1, (q_{z11}, q_{02}), T_2, (q_{z12}, q_{03}), T_3)$.

Машина T называется разветвлением машин T_2 и T_3, управляемым машиной T_1.

3.7. Примеры построения машин Тьюринга

Пример 1. Построить машину Тьюринга, которая правильно вычисляет функцию f(x) = x+1 по правилам двоичного сложения.

Решение. Исходя из формулировки задачи, требующей вычислить функцию по правилам сложения в двоичной системе сложения, выберем входной алфавит:

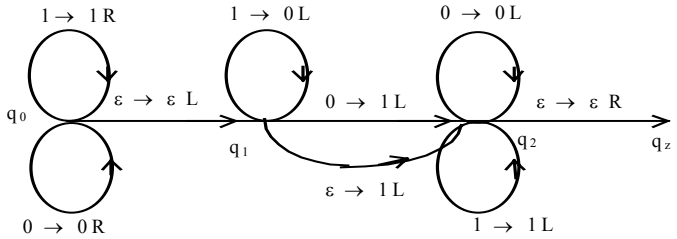
$A = \{0, 1, \varepsilon\}$.

Представим машины Тьюринга таблицей соответствия и графом.

Таблица соответствия:

	0	1	ε
q_0	q_0 0R	q_0 1R	Q_1 ε L
q_1	q_2 1L	q_1 0L	q_2 1L
q_2	q_2 0L	Q_2 1L	q_z ε R

Представление машины Тьюринга в виде графа:



Запишем программу построенной машины Тьюринга для случая, когда входная цепочка на ленте равна двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

1) $q_0 1 \rightarrow q_0 1R$ $\quad \varepsilon\underline{1}11\varepsilon$ $\quad$ 6) $q_1 1 \rightarrow q_1 0L$ $\quad \varepsilon 1\underline{1}0\varepsilon$

2) $q_0 1 \to q_0 1R$ ε1$\underline{1}$1ε 7) $q_1 1 \to q_1 0L$ ε$\underline{1}$00ε
3) $q_0 1 \to q_0 1R$ ε11$\underline{1}$ε 8) $q_1 ε \to q_2 1L$ $\underline{ε}$000ε
4) $q_0 ε \to q_1 εL$ ε111$\underline{ε}$ 9) $q_2\, ε \to q_z\, εR$ $\underline{ε}$1000ε
5) $q_1 1 \to q_1 0L$ ε11$\underline{1}$ε

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111, выполнив совокупность команд 1-9, перешла в стандартное заключительное состояние, имея на ленте результат 1000. Действительно:

$111_2 + 1_2 = 1000_2$.

Пример 2. Построить машину Тьюринга, выполняющую операцию (x div 2) и имеющую входной алфавит A = {0, 1, ε}.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	0	1	ε		0	1	ε
q_0	q_0 0R	q_0 1R	q_1 ε L	q_2	q_3 1L	q_2 0L	q_3 1L
q_1	q_3 ε L	q_2 ε L		q_3	q_2 0L	q_2 1L	q_z ε R

Операция (x div 2) реализована сдвигом цепочки вправо на 1 разряд.

Пример 3. Построить машину Тьюринга, которая выполняет копирование заданного аргумента. Выберем входной алфавит A={0,1,ε}. Представим данную машину таблицей соответствия:

	0	1	*	ε
q_0	$q_1$1L	$q_1$0R	q_0*L	q_zεR
q_1		$q_1$1R	q_2*R	q_2*R
q_2		$q_2$1R		$q_3$1L
q_3	$q_0$0R	$q_3$1L	q_3*L	

Запишем программу построенной машины для заданной входной цепочки на ленте, равной двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

1) $q_0 1 \to q_1 0R$ ε$\underline{1}$11ε 17) $q_3 1 \to q_3 1L$ ε00$\underline{1}$*11ε
2) $q_1 1 \to q_1 1R$ ε0$\underline{1}$1ε 18) $q_3 0 \to q_0 0R$ ε0$\underline{0}$1*11ε
3) $q_1 1 \to q_1 1R$ ε01$\underline{1}$ε 19) $q_0 1 \to q_1 0R$ ε00$\underline{0}$*11ε
4) $q_1 ε \to q_2 *R$ ε011$\underline{ε}$ 20) $q_1 * \to q_2 *R$ ε000$\underline{*}$11ε
5) $q_2 ε \to q_3 1L$ ε011*$\underline{ε}$ 21) $q_2 1 \to q_2 1R$ ε000*$\underline{1}$1ε
6) $q_3 * \to q_3 *L$ ε011$\underline{*}$1ε 22) $q_2 1 \to q_2 1R$ ε000*1$\underline{1}$ε
7) $q_3 1 \to q_3 1L$ ε01$\underline{1}$*1ε 23) $q_2 ε \to q_3 1L$ ε000*11$\underline{ε}$
8) $q_3 1 \to q_3 1L$ ε0$\underline{1}$1*1ε 24) $q_3 1 \to q_3 1L$ ε000*1$\underline{1}$1ε
9) $q_3 0 \to q_0 0R$ ε$\underline{0}$11*1ε 25) $q_3 1 \to q_3 1L$ ε000*$\underline{1}$11ε
10) $q_0 1 \to q_1 0R$ ε0$\underline{1}$1*1ε 26) $q_3 * \to q_3 *L$ ε000$\underline{*}$111ε
11) $q_1 1 \to q_1 1R$ ε00$\underline{1}$*1ε 27) $q_3 0 \to q_0 0R$ ε00$\underline{0}$*111ε
12) $q_1 * \to q_2 *R$ ε001$\underline{*}$1ε 28) $q_0 * \to q_0 *L$ ε000$\underline{*}$111ε
13) $q_2 1 \to q_2 1R$ ε001*$\underline{1}$ε 29) $q_0 0 \to q_0 1L$ ε00$\underline{0}$*111ε
14) $q_2 ε \to q_3 1L$ ε001*1$\underline{ε}$ 30) $q_0 0 \to q_0 1L$ ε0$\underline{0}$1*111ε

15) $q_3 1 \to q_3 1L$ ε001*1̲1ε 31) $q_0 0 \to q_0 1L$ ε0̲11*111ε
16) $q_3 * \to q_3 *L$ ε001*1̲1ε 32) $q_0 \varepsilon \to q_z 1R$ ε̲111*111ε

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111 и выполнив совокупность команд 1-32, перешла в стандартное заключительное состояние, имея на ленте результат ε̲111*111ε.

3.8. Машина Тьюринга с полулентой

В рассмотренных выше определениях машины Тьюринга использовалась бесконечная в обе стороны лента. Ограничим ленту с одной стороны и покажем, что машина Тьюринга с правой или левой полулентой эквивалентна машине Тьюринга с бесконечной лентой.

Говорят, что функция, правильно вычислимая на машине Тьюринга с обычной лентой, правильно вычислима и на машине Тьюринга с правой полулентой, т.е. для любой машины Тьюринга T существует эквивалентная ей машина с правой полулентой T_R.

Идея доказательства этого утверждения основана на следующих предпосылках:

а) рабочая область ленты ограничена двумя маркерами: неподвижным - слева и подвижным - справа;

б) на внутренней части ограниченной области машина Тьюринга с полулентой должна работать как обычная машина Тьюринга, а при выходе на маркеры она должна освобождать рабочее пространство, для этого правый маркер может сдвигаться вправо, а при выходе на левый маркер необходимо сдвинуть всю цепочку вправо;

в) полученный результат, находящийся между маркерами, в конце работы машины Тьюринга нужно сдвинуть вплотную к левому маркеру.

Машина Тьюринга может вычислять функцию с восстановлением, то есть с сохранением исходных данных. Это позволяет получить на ленте сначала результат, а затем - исходные данные или наоборот.

Рассмотрим пример построения машины Тьюринга с правой полулентой, вычисляющей функцию f(x) = 2x. Алгоритм вычисления данной функции состоит в приписывании нуля к входной цепочке справа.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	<	0	1	>	ε
q_0		q_0 0R	q_0 1R	q_1 0R	
q_1					q_2 >L
q_2	q_z < R	q_2 0L	q_2 1L		

Здесь символ “<“ обозначает левый маркер, а символ “>“ - правый маркер. Машина Тьюринга, начав работу из стандартной начальной конфигурации, после выполнения совокупности команд переходит в стандартную заключительную конфигурацию. Полученный результат располагается между маркерами и сдвинут вплотную к левому маркеру.

3.9. Универсальная машина Тьюринга

До сих пор мы имели дело со специализированными машинами Тьюринга, предназначенными для решения конкретных задач и отличающимися набором команд, внутренним и внешним алфавитами. Однако можно построить универсальную машину

Тьюринга, способную выполнять любой алгоритм, а значит работу любой машины Тьюринга. В ней входное слово должно включать изображение программы и входное слово интерпретируемой машины.

Чтобы получить изображение программы интерпретируемой машины, нужно последовательно, строка за строкой, закодировать эту программу в алфавите универсальной машины. Универсальная машина Тьюринга должна иметь фиксированный внешний алфавит, используемый при записи программы, включая и алфавит интерпретируемой машины.

Она должна быть приспособлена к приему в качестве исходной информации всевозможных состояний устройства управления и конфигураций, в которых могут встречаться буквы из разнообразных алфавитов со сколь угодно большим числом различных букв.

Это достигается путем кодирования конфигурации и программы любой заданной машины Тьюринга в символах входного алфавита универсальной машины. Само кодирование должно выполняться следующим образом:

1) различные буквы должны заменяться различными кодовыми группами, но одна и та же буква должна заменяться всюду, где бы она не встречалась, одной и той же кодовой группой;

2) строки кодовых записей должны однозначным образом разбиваться на отдельные кодовые группы;

3) должна существовать возможность распознать, какие кодовые группы соответствуют различным сдвигам управляющей головки (R, L, E), и различать кодовые группы, соответствующие буквам внутреннего алфавита и буквам внешнего алфавита.

Рассмотрим пример такого кодирования для машины Тьюринга T, имеющей внешний алфавит $A = \{a_1, a_2, \dots, a_k\}$ и внутренний алфавит $Q = \{q_1, q_2, \dots, q_z\}$. Если внешний алфавит состоит из символов $A = \{0, 1\}$, то условия кодирования будут соблюдены при следующем способе кодирования.

1. В качестве кодовых групп возьмем 3+k+m различных слов вида 100...01, где между единицами проставляются нули; k - число символов внешнего алфавита; m - число состояний устройства управления.

Тогда разбивка строк на кодовые группы производится путем выделения последовательностей нулей, заключенных между двумя единицами.

2. Сопоставление кодовых групп исходным символам внешнего и внутреннего алфавитов осуществляется согласно следующей таблице кодирования:

	Буква	Кодовая группа		
	R	101		
	E	1001		
	L	10001		
Внешний алфавит	a_1	100001	- 4 нуля	Четное число нулей, большее чем 2
	a_2	10000001	- 6 нулей	
	.....	.....		
	a_k	10.....01	2(k + 1) нулей	
Внутренний алфавит (состояния)	q_1	1000001	- 5 нулей	Нечетное число нулей, большее чем 3
	q_2	100000001	- 7 нулей	
	.....	.....		
	q_m	10.....01	2(m + 1)+1 нулей	

Например, для машины Тьюринга, которая перерабатывает слово bcad в слово bccd, входное слово в универсальной машине Тьюринга с данным кодом будет представлено следующей строкой:

10000001 1000000001 100001 100000000001,

где 100001 - a, 10000001 - b, 1000000001 - c, 100000000001 - d.

Программа же будет представлена следующими строками:

1) 1000001 10000001 1000001 10000001 101
($q_0 b \rightarrow q_0 b R$)
2) 1000001 1000000001 1000001 1000000001 101
($q_0 c \rightarrow q_0 c R$)
3) 1000001 100001 100000001 1000000001 101
($q_0 a \rightarrow q_1 c R$)
3) 100000001 100000000001 10000000001 100000000001 10001
($q_1 d \rightarrow q_2 d L$).

Рассмотрим еще один пример. Пусть на ленту универсальной машины Тьюринга поступает слово, составленное из букв английского алфавита. Задача машины Тьюринга - переставлять местами буквы **n** и **o** таким образом, чтобы сочетание **on** преобразовывалось в **no**. Таким образом, после переработки входного слова в нем не должно остаться ни одного буквосочетания **on**.

Возьмем слово **mnonnop**, которое должно преобразоваться универсальной машиной Тьюринга в новое слово **mnnnoop**.

Пусть внешний алфавит универсальной машины Тьюринга состоит из символов A = {0,1}, а внутренний алфавит Q = {q_0, q_1, q_2, q_3, q_z }, где q_z - заключительное состояние. Разбивка входных символов на кодовые группы и сопоставление кодовых групп исходным символам внешнего и внутреннего алфавитов осуществляется согласно приведенной выше таблице кодирования.

Тогда входное слово будет представлено следующим образом:

100001 10000001 1000000001 10000001 10000001 1000000001 100000000001,

где 100001 - m, 10000001 - n, 1000000001 - o, 100000000001 - p.

Ниже представлены команды универсальной машины Тьюринга, которые будут выполняться при обработке и преобразовании исходного слова. Напротив каждой команды приводится входное слово таким, какое оно есть в момент начала выполнения данной команды. Символ, на который указывает головка машины Тьюринга, показан как прописная буква.

$q_0 m \rightarrow q_0 m R$	**M n o n n o p**
$q_0 n \rightarrow q_0 n R$	**m N o n n o p**
$q_0 o \rightarrow q_1 o R$	**m n O n n o p**
$q_1 n \rightarrow q_2 o L$	**m n o N n o p**
$q_2 o \rightarrow q_0 n R$	**m n O o n o p**
$q_0 o \rightarrow q_1 o R$	**m n n O n o p**
$q_1 n \rightarrow q_2 o L$	**m n n o N o p**
$q_2 o \rightarrow q_0 n R$	**m n n O o o p**
$q_0 o \rightarrow q_1 o R$	**m n n n O o p**
$q_1 o \rightarrow q_0 o R$	**m n n n o O p**
$q_0 p \rightarrow q_z p E$	**m n n n o o P.**

Программа же будет выглядеть так:

1000001 100001 1000001 100001 101
1000001 10000001 1000001 10000001 101
1000001 1000000001 100000001 1000000001 101
100000001 10000001 10000000001 1000000001 10001
10000000001 1000000001 1000001 10000001 101
1000001 1000000001 100000001 1000000001 101

100000001 10000001 10000000001 1000000001 10001
10000000001 1000000001 1000001 10000001 101
1000001 1000000001 100000001 1000000001 101
100000001 1000000001 1000001 1000000001 101
1000001 100000000001 1000000000001 100000000001 1001.

Таким образом, если какая-либо машина Тьюринга **T** решает некоторую задачу, то и универсальная машина Тьюринга способна решить эту задачу при условии, что кроме кодов исходных данных этой задачи на ее ленту будет подан код программы машины **T**. Задавая универсальной машине Тьюринга T_u изображение программы любой данной машины Тьюринга T_n и изображение любого ее входного слова x_n, получим изображение выходного слова y_n, в которое машина T_n переводит слово x_n.

Если же алгоритм, реализуемый машиной T_n, не применим к слову x_n, то алгоритм, реализуемый универсальной машиной T_u, также не применим к слову, образованному из изображения x_n и программы машины T_n.

Таким образом, машина Тьюринга T_n может рассматриваться как одна из программ для универсальной машины T_u.

В связи с существованием универсальной машины Тьюринга таблицы соответствия, описывающие различные состояния устройства управления машины, имеют двоякое назначение:

1) для описания состояний устройства управления специальной машины Тьюринга, реализующей соответствующий алгоритм;

2) для описания программы, подаваемой на ленту универсальной машины Тьюринга, при реализации соответствующего алгоритма.

Современные ЭВМ строятся как универсальные; в запоминающее устройство наряду с исходными данными поставленной задачи вводится также и программа ее решения.

Однако в отличие от машины Тьюринга, в которой внешняя память (лента) бесконечна, в любой реальной вычислительной машине она конечна.

3.10. Алгоритмически неразрешимые проблемы

Свойство массовости алгоритмов означает, что они предназначены для решения некоторой массовой проблемы, формулируемой в виде отображения множества входных слов в соответствующие им выходные слова. Поэтому всякий алгоритм можно рассматривать как универсальное средство для решения целого класса задач.

В теории алгоритмов известны некоторые задачи, для решения которых не существует единого универсального приема. Однако алгоритмическая неразрешимость проблемы решения задач того или иного класса не означает невозможность решения любой конкретной задачи из этого класса. В данном случае речь идет о невозможности решения всех задач одного класса одним и тем же приемом.

Таким образом, задачи (проблемы) можно разделить на алгоритмически разрешимые и алгоритмически неразрешимые.

Обычно алгоритмическая неразрешимость новых задач доказывается методом сведения к этим задачам известных алгоритмически неразрешимых задач. Тем самым доказывается, что если бы была разрешима новая задача, то можно было бы решить и заведомо неразрешимую задачу. Применяя метод сведения, обычно ссылаются на искусственные задачи, которые не представляют самостоятельного интереса, но для которых легко непосредственно доказать их разрешимость. К числу таких задач относится проблема распознавания самоприменимости.

Самоприменимыми называются алгоритмы, которые, начав работу над собственным описанием, рано или поздно останавливаются. Если же алгоритм в таком случае зацикливается, он называется *несамоприменимым*. Аналогично можно говорить о

самоприменимости машин Тьюринга, имея в виду их применение к своим линейным изображениям.

Таким образом, возникает задача: как узнать, является ли самоприменимым тот или иной алгоритм.

Проблема распознавания самоприменимости алгоритмов состоит в том, чтобы найти единый конструктивный прием, позволяющий за конечное число шагов по схеме любого данного алгоритма узнать, является алгоритм самоприменимым или несамоприменимым.

Доказательство того, что общего алгоритма распознавания самоприменимости не существует, можно проводить, используя алгоритмические системы Тьюринга. На основе этого доказательства было показано, что проблема распознавания самоприменимости алгоритмически неразрешима.

3.11. Задания для самостоятельной работы

1. Построить машину Тьюринга, выполняющую операцию конкатенации двух цепочек, заданных во входном алфавите A = {0, 1, *, ε}.

2. Построить машину Тьюринга, выполняющую операцию копирования входной цепочки, заданной в алфавите A = {1, *, ε }, где символ “*” используется в качестве разделителя двух цепочек.

3. Построить машину Тьюринга в алфавите A={0, 1}, которая, начав работу с последней единицы массива из единиц, сдвигает его на одну ячейку влево, не изменяя остального содержимого ленты. Головка останавливается на первой единице перенесенного массива.

4. По заданной совокупности команд машины Тьюринга T и начальной конфигурации K найти заключительную конфигурацию.

4.1.

$q_0 1 \to q_0 1R$, $q_1 0 \to q_z 1E$,

$q_0 0 \to q_1 1R$, $q_1 1 \to q_1 1L$,

а) $K = 1101q_0 01$; б) $K = 101q_0 010$.

4.2.

$q_0 0 \to q_0 1L$, $q_1 1 \to q_0 0R$,

$q_0 1 \to q_1 1R$, $q_1 0 \to q_z 0L$,

а) $K = 1q_1\ 0111$; б) $K = 1q_1 1111$.

4.3.

$q_0 0 \to q_1 0L$, $q_1 0 \to q_1 1L$,

$q_0 1 \to q_0 0R$, $q_1 1 \to q_z 0R$,

а) $K = 1000q_1\ 01$; б) $K = 11q_1 11101$.

5. Построить машину Тьюринга, которая во входной цепочке, заданной в алфавите A = {0, 1, ε}, переставляет единицы и нули так, чтобы все единицы были в начале, а нули в конце цепочки.

6. Построить композицию $T_1 \cdot T_2$ машин Тьюринга T_1 и T_2 по паре состояний (q_{1z}, q_{20}) и найти результат применения композиции $T_1 \cdot T_2$ к слову D.

6.1. Машины T_1 и T_2 заданы таблицами соответствия:

T_1:

	q_{10}	q_{11}	q_{12}
0	$q_{1z}0L$	$q_{12}0R$	$q_{10}0R$
1	$q_{11}1R$	$q_{12}1R$	$q_{10}0R$

	q_{20}	q_{21}

T_2:

0	$q_{21}1L$	$q_{2z}0R$
1	$q_{21}1L$	$q_{20}0L$

а) D = 111100111011; б) D = 11010111.

6.2. Машины T_1 и T_2 заданы таблицами соответствия:

T_1:

	q_{10}	q_{11}	q_{12}
0	$q_{11}0R$	$q_{12}0R$	$q_{1z}1L$
1	$q_{10}1R$	$q_{10}1R$	

T_2:

	q_{20}	q_{21}	q_{22}
0	$q_{21}0L$	$q_{22}0L$	$q_{2z}0R$
1	$q_{20}1L$	$q_{21}1L$	$q_{22}1L$

а) D = 11000101001; б) D = 10100111110.

7. Найти результат применения итерации машины T по паре состояний (q_{z1}, q_i) к слову D. Заключительными состояниями машины T являются q_{z1} и q_{z2}. На ленте первоначально записаны нули, а в начальной конфигурации головка указывает на левый символ входной цепочки, состоящей из единиц.

7.1. Машина Тьюринга задана таблицей соответствия:

T:

	q_0	q_1	q_2	q_3	q_4
0	$q_{z1}0E$	q_30E	q_40E	q_41R	$q_{z2}1L$
1	q_10R	q_20R	q_00R		

а) i = 1, $D = 1^{3k}$, $k \geq 1$; б) i = 1, $D = 1^{3k+2}$, $k \geq 1$.

7.2. Машина Тьюринга задана таблицей соответствия:

T:

	q_0	q_1	q_2	q_3	q_4	q_5
0	$q_{z2}0R$	$q_{z2}0R$	q_30R	q_41L	q_50L	$q_{z1}0R$
1	q_10R	q_20R	q_21R	q_10E	q_41L	q_51L

а) i = 3, $D = 1^{2k+1}$, $k \geq 1$; б) i = 1, $D = 1^{3k+2}$, $k \geq 1$.

4. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ЯЗЫКИ

4.1. Общие сведения

В последние три десятилетия появилось большое количество работ по общей теории языков и грамматик. Можно выделить четыре научных направления, которые удалось объединить по методам их исследования в одну общую задачу теории языков.

Первое из этих направлений связано с построением формальной, или математической, лингвистики, которая начала особенно быстро развиваться в тот период, когда были сформулированы вопросы машинного перевода. Эта проблема требовала формализации понятий словарь, грамматика, язык, их классификации и умения относить конкретные словари, грамматики, языки к определенному классу.

Интерес к задачам такого рода еще более усилился с появлением искусственных языков программирования. С появлением трансляторов проблема перевода во многом определила построение общей теории вычислительных машин, а сами языки программирования стали все более приближаться к формально построенным математическим конструкциям.

Независимо от указанных двух направлений развивалось построение формальных моделей динамических систем. Для создания продуктивной теории эти модели должны были быть, с одной стороны, достаточно узкими, а с другой - достаточно широкими, чтобы охватить некоторый общий класс прикладных задач. Типичным примером такого рода является модель конечного автомата. Эта модель позволяет описать многие процессы, заданные на конечных множествах и развивающиеся в счетном времени, что позволило создать для нее продуктивную теорию. Если в эту модель ввести бесконечность по какому-либо параметру, то это приводит к общему классу систем, например, к машинам Тьюринга.

Независимо и параллельно развивалась общая теория алгоритмов как ветвь современной математики. Развитие вычислительной техники поставило перед математической теорией алгоритмов новую задачу: стало необходимым классифицировать алгоритмы по различным признакам. Эквивалентность понятий "алгоритм" и "машина Тьюринга" позволила предположить, что поиски классификации алгоритмов окажутся связанными с поисками промежуточных моделей между моделями конечного автомата и машиной Тьюринга.

Таким образом, перечисленные четыре направления оказались тесно связанными. Теория языков, порожденная чисто лингвистическими задачами, оказалась в центре интересов математиков, занимающихся теорией алгоритмов, и ученых, разрабатывающих абстрактные модели динамических систем и теоретические основы автоматики.

Теория формальных языков и грамматик является разделом математической лингвистики - специфической математической дисциплины, ориентированной на изучение структуры естественных и искусственных языков. По характеру используемого математического аппарата теория формальных грамматик и языков близка к теории алгоритмов и теории автоматов.

На интуитивном уровне язык можно определить как некоторое множество предложений с заданной структурой, имеющих определенное значение. Правила, определяющие допустимые конструкции языка, составляют синтаксис языка. Значение, или смысл фразы, определяется семантикой языка.

Если бы все языки состояли из конечного числа предложений, то не было бы проблемы синтаксиса. Но, так как язык содержит неограниченное (или довольно большое) число правильно построенных фраз, то возникает проблема описания бесконечных языков с помощью каких-либо конечных средств.

Различают следующие виды описания языков:

1) порождающее, которое предполагает наличие алгоритма, последовательно порождающего все правильные предложения языка;

2) распознающее, которое предполагает наличие алгоритма, определяющего принадлежность любой фразы данному языку.

4.2. Основные понятия порождающих грамматик

Алфавит - это непустое конечное множество. Элементы алфавита называются символами.

Цепочка над алфавитом $\Sigma=\{a_1, a_2, \ldots, a_n\}$ есть конечная последовательность элементов a_i. Множество всех цепочек над алфавитом Σ обозначается Σ^*.

Длина цепочки x равна числу ее элементов и обозначается |x|. Цепочка нулевой длины называется пустой и обозначается символом ε. Соответственно, непустая цепочка определяется как цепочка ненулевой длины. Пусть имеется алфавит $\Sigma=\{a, b\}$. Тогда множество всех цепочек определяется следующим образом: $\Sigma^*=\{$ ε, a, b, aa, ab, ba, bb, aaa, aab, aba, . . .$\}$.

Цепочки x и y равны, если они одинаковой длины и совпадают с точностью до порядка символов, из которых они состоят.

Над цепочками x и y определена операция сцепления (конкатенации, произведения), которая получается дописыванием символов цепочки y за символами цепочки x.

Язык L над алфавитом Σ представляет собой множество цепочек над Σ. Необходимо различать пустой язык L=0 и язык, содержащий только пустую цепочку: L={ε}≠0.

Формальный язык L над алфавитом Σ - это язык, выделенный с помощью конечного множества некоторых формальных правил.

Пусть M и L - языки над алфавитами. Тогда конкатенация LM = {xy | x∈L, y∈M}. В частности, {ε}L = L{ε}=L. Используя понятие произведения, определим итерацию L^* и усеченную итерацию L^+ множества L:

$$L^+ = \bigcup_{i=1}^{\infty} L^i ,$$

$$L^* = \bigcup_{i=1}^{\infty} L^i ,$$

где i - степень языка, L определяется рекурсивно следующим образом:

$L^0 = \{\varepsilon\}$;

$L^1 = L$;

$L^{n+1} = L^n L = L L^n$;

{ε}L=L{ε}=L.

Например, если задан язык L={a}, тогда L^*={ε, a, aa, aaa, …}, L^+={a, aa, aaa, …}.

Порождающая грамматика - это упорядоченная четверка $G= (V_T, V_N, P, S)$, где

V_T - конечный алфавит, определяющий множество терминальных символов;

V_N - конечный алфавит, определяющий множество нетерминальных символов;

P - конечное множество правил вывода, т.е. множество пар следующего вида:

$$u \to v, \quad \text{где } u, v \in (V_T \cup V_N)^*;$$

S - начальный символ (аксиома грамматики), $S \in V_N$.

Из терминальных символов состоят цепочки языка, порожденного грамматикой. Аксиомой называется символ в левой части первого правила вывода грамматики.

Для того чтобы различать терминальные и нетерминальные символы, принято обозначать терминальные символы строчными, а нетерминальные символы заглавными буквами латинского алфавита.

В грамматике G цепочка x непосредственно порождает цепочку y, если x = αuβ, y = αvβ и u→v ∈ P, т.е. цепочка y непосредственно выводится из x, что обозначается x => y.

Языком, порождаемым грамматикой G = (V_T, V_N, P, S), называется множество терминальных цепочек, выводимых в грамматике G из аксиомы:

$L(G) = \{x \mid x \in V_T^*; S => *x\}$, где =>*- выводимость.

Пример. Дана грамматика G = (V_T, V_N, P, S), в которой V_T = {a, b}, V_N = {S}, P = {S → aSb, S → *ab).* Определить язык, порождаемый этой грамматикой.

Решение. Используя рекурсию, выведем несколько цепочек языка, порождаемого данной грамматикой:

S → ab;

S → aSb => aabb;

S → aSb =>aaSbb => aaabbb; . . .

Определим язык, порожденный данной грамматикой:

$L(G) = \{a^n b^n \mid n > 0\}$.

Говоря о представлении грамматик, нужно отметить, что множество правил вывода грамматики может приводиться без специального указания множеств терминалов и нетерминалов. В таком случае обычно предполагается, что грамматика содержит в точности те терминальные и нетерминальные символы, которые встречаются в правилах вывода.

Также предполагается, что правые части правил, для которых совпадают левые части, можно записать в одну строку с использованием разделителя. Так, правила вывода из приведенного примера можно записать следующим образом: $S \to aSb \mid ab$.

4.3. Классификация грамматик

Правила порождающих грамматик позволяют осуществлять преобразования строк. Ограничения же на виды правил позволяют выделить классы грамматик. Рассмотрим классификацию, которую предложил Н. Хомский.

Грамматики типа 0 - это грамматики, на правила вывода которых нет ограничений.

Грамматики типа 1, или контекстные грамматики -это грамматики, все правила которых имеют следующий вид: $xAy \to x\varphi y$, где $A \in V_N$, $x, y, \varphi \in (V_N \cup V_T)^+$.

Грамматики типа 2 - это бесконтекстные, или контекстно-свободные грамматики (КС-грамматики). Правила вывода для этих грамматик имеют следующий вид: $A \to \varphi$, где $A \in V_N$, $\varphi \in (V_N \cup V_T)^*$.

Грамматики типа 3 - это автоматные грамматики, которые делятся на два типа:

а) леволинейные (леворекурсивные), правила вывода для которых имеют следующий вид: $A \to Aa \mid a$, где $A \in V_N$;

б) праволинейные (праворекурсивные), правила вывода для которых имеют следующий вид: $A \to Aa \mid a$.

Язык L называется языком типа i, если существует грамматика типа i, порождающая язык L.

4.3.1. Методика решения задач

Пример 1. Дана порождающая грамматика $G = (V_T, V_N, P, S)$, в которой $V_T = \{a, d, e\}$, $V_N = \{B, C, S\}$, $P = \{S \to aB, B \to Cd \mid dC, C \to e\}$. Выписать терминальные цепочки, порожденные данной грамматикой, и определить длину их вывода.

Решение. Получим следующие терминальные цепочки:

$S \to aB \to aCd \to aed$,

$S \to aB \to adC \to ade$,

длина вывода которых равна 3.

Пример 2. Дана грамматика $G = (V_T, V_N, P, C)$, в которой $V_T = \{a, b, c, d, e\}$, $V_N = \{A, B, C, D, E\}$, $P = \{A \to ed, B \to Ab, C \to Bc, C \to dD, D \to Ae, E \to bc\}$. Определить, принадлежит ли языку L(G) цепочка eadabcb.

Решение. Выведем возможные терминальные цепочки из аксиомы с помощью заданных правил вывода:

$C \to Bc \to Abc \to edbc$,

$C \to dD \to dAe \to dede$.

Так как первые же терминальные символы полученных цепочек не совпадают с заданной, можно сделать вывод о том, что цепочка eadabcb не принадлежит языку L(G).

Пример 3. Построить КС-грамматику (грамматику типа 2), порождающую цепочки из 0 и 1 с одинаковым числом тех и других символов.

Решение. Определим множества, задающие грамматику: $V_T = \{0, 1\}$; $V_N = \{S\}$; $P = \{S \to 0S1, S \to 1S0, S \to \varepsilon, S \to S01, S \to S10\}$.

Пример 4. Описать язык, порождаемый следующими правилами: $S \to bSS \mid a$.

Решение. Построим несколько терминальных цепочек, применяя заданные правила вывода:

$S \to a$;

$S \to bSS \to baa$;

$S \to bSS \to bbSSS \to bbaaa$;

$S \to bSS \to bbSSbSS \to bbaabaa;$

$S \to bSS \to bbSSbSS \to bbbSSbSSbbSSbSS \to ...$

Из полученных цепочек видно, что:

а) цепочки всегда начинаются с терминала b, кроме аксиомы, и заканчиваются терминалом a;

б) количество терминалов a в любой цепочке всегда на 1 больше, чем b.

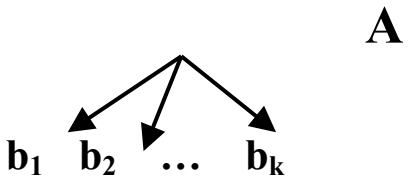
Исходя из этих выводов, определим язык, порождаемый заданными правилами, следующим образом:

$L(G) = \{\alpha \mid \alpha \in \{a, b\}^*\}$, $|a| = |b| + 1$, причем цепочки начинаются с терминала b и заканчиваются терминалом a.

4.4. Грамматический разбор

В КС-грамматике может быть несколько выводов, эквивалентных в том смысле, что в них применяются одни и те же правила к одинаковым в промежуточном выводе цепочкам, но в различном порядке. Например, для грамматики G с правилами вывода $S \to ScS|b|a$ возможны следующие эквивалентные выводы терминальной цепочки acb: $S \to ScS \to Scb \to acb$ и $S \to ScS \to acS \to acb$.

Деревом вывода цепочки x в КС-грамматике $G = (V_T, V_N, P, S)$ называется упорядоченное дерево, каждая вершина которого помечена символом из множества $V \cup V_N \cup \{\varepsilon\}$ так, что каждому правилу $A \to b_1b_2b_3 ... b_k$, использующемуся при выводе цепочки x, в дереве вывода соответствует поддерево с корнем A и прямыми потомками $b_1, b_2, b_3, ..., b_k$, которое приведено на рисунке:



В силу того, что цепочка $x \in L(G)$ выводится из аксиомы S, корнем вывода всегда является аксиома. Внутренние узлы вывода соответствуют нетерминальным символам. Концевые вершины дерева вывода - листья - это вершины, не имеющие потомков. Такие вершины соответствуют либо терминалам, либо пустым символам (ε) при условии, что среди правил грамматики имеются правила с пустой правой частью. При чтении слева направо концевые вершины дерева вывода образуют цепочку x.

Дерево вывода часто называют деревом грамматического разбора, или синтаксическим деревом, а процесс построения дерева вывода - грамматическим разбором (синтаксическим анализом). Одной цепочке языка может соответствовать больше одного дерева, так как эта цепочка может иметь разные выводы, порождающие разные деревья.

КС-грамматика $G == (V_T, V_N, P, S)$ называется неоднозначной (неопределенной), если существует цепочка $x \in L(G)$, имеющая два или более дерева вывода.

Рассмотрим пример.

Пусть даны две КС-грамматики:

$G_1 = (V_T, V_N, P_1, S)$, $V_N = \{S\}$,

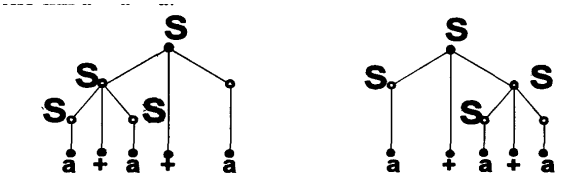
$P_1 = \{S \to S+S \mid S \mid S \cdot S \mid (S) \mid a\}$;

$G_2 = (V_T, V_N, P_2, S)$, $V_N = \{S, A, B\}$,

$P_2 = \{S \to S + A \mid A \mid A,\ A \to A \cdot B \mid A / B \mid B,\ B \to a \mid (S)\}$, содержащие в множестве терминальных символов знаки операций, круглые скобки и символ a. Определить, являются

ли грамматики однозначными. Если какая-либо из них неоднозначна, привести пример цепочки, для которой существует два различных дерева вывода.

Решение: Грамматика G_1 является неоднозначной, так как она имеет правила с правой частью, содержащей два вхождения нетерминала S и знак арифметической операции. Построим два дерева вывода для простейшей цепочки a + a + a:



Грамматика G_2 является однозначной, так как не содержит правил с двойным вхождением нетерминального символа. Так, для цепочки a + a + a она имеет только одно дерево вывода.

В некоторых случаях неоднозначность в грамматиках может устраняться путем эквивалентных преобразований. Однако в общем случае проблема устранения неоднозначности неразрешима. На практике от неоднозначности избавляются путем задания словесных ограничений, называемых контекстными условиями языка, которые включаются в его семантику.

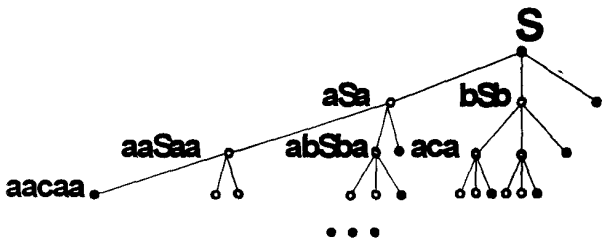
Различают две стратегии грамматического разбора: восходящую и нисходящую, которые соответствуют способу построения синтаксических деревьев. При нисходящей стратегии разбора дерево строится от корня аксиомы вниз, к терминальным вершинам. Главной задачей при нисходящем разборе является выбор того правила, которое следует применить на рассматриваемом шаге. При восходящем разборе дерево строится от терминальных вершин к корню дерева (аксиоме).

Преобразование цепочки, обратное порождению, называется редукцией.

4.4.1. Представление грамматики в виде графа

Дерево грамматического разбора не следует путать с представлением грамматики в виде графа. Граф грамматики в качестве вершин содержит сентенциальные формы (любые цепочки, выводимые из аксиомы).

Рассмотрим представление грамматики G в виде графа: $G = (V_T, V_N, P, S)$, в которой $V_T = \{a, b, c\}$, $V_N = \{S\}$, $P=\{S \rightarrow aSa \mid bSb \mid c\}$.



4.5. Преобразования КС-грамматик

Часто требуется изменить грамматику таким образом, чтобы она удовлетворяла определенным требованиям, не изменяя при этом порождаемый грамматикой язык. Для этого используются эквивалентные преобразования КС-грамматик, некоторые из которых рассмотрены ниже.

4.5.1. Удаление правил вида A → B

Преобразование первого типа состоит в удалении правил A → B, или <нетерминал> → <нетерминал>.

Покажем, что для любой КС-грамматики можно построить эквивалентную грамматику, не содержащую правил вида: A → B, где A и B - нетерминальные символы.

Пусть имеется КС-грамматика G=(V_T, V_N, P, S), где множество нетерминалов V_N={A_1, A_2, . . . , A_n}. Разобьем P на два непересекающихся множества: P = P_1 ∪ P_2. В P_1 включены все правила вида A_i → A_k, в P_2 включены все остальные правила, т.е. P_2 = P \ P_1. Затем для каждого A_i определим множество правил P(A_i), включив в него все такие правила A_i→φ, что A_i →* A_j и A_j → φ, где A_j → φ ∈ P_2. Построим эквивалентную КС-грамматику $G_э$ = (V_T, V_N, $P_э$, S), в которой множества терминальных и нетерминальных символов, а также аксиома совпадают с теми же объектами исходной грамматики G, а множество правил $P_э$ получено объединением правил множества P_2 и правил P(A_i) для всех 1≤ i ≤n:

$$P_э = \bigcup_{i=1}^{n} P(A_i) \cup P_2 .$$

Пример. Пусть задана грамматика G со следующими правилами вывода S → aFb | A; A → aA | B; B → aSb | S; F → bc | bFc.

Построим множества правил P2, P(S), P(A), P(B), P(F).

Определим правила для P2: P2 = {S → aFb; A → aA; B → aSb; F → bc | bFc}.

Определим правила для P(S): S => A => B или S =>*A; S=>B, где =>* обозначает непосредственную выводимость. P(S) = {S → aA; S → aSb).

Определим правила для P(A): A => B => S или A =>* B; A => S. P(A) = {A → aSb; A → aFb}.

Определим правила для P(B): B => S => A или B =>* S; B =>*A. P(B) = {B → aFb; B → aA}.

Определим правила для P(F): так как непосредственно выводимых нетерминалов не существует, то P(F) = 0.

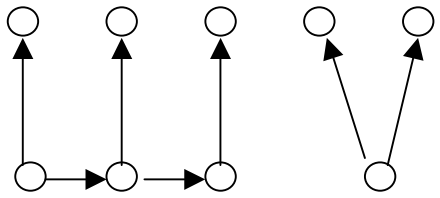
Объединив полученные правила, можно записать грамматику Gэ, эквивалентную исходной:

S → aFb | aSb | aA; A → aA | aSb | aFb;

B → aA | aSb | aFb; F → bc | bFc.

Графическая модификация метода

Аналитическое преобразование по рассмотренному алгоритму оказывается довольно сложным. При автоматизированном преобразовании грамматик проще применить графическую модификацию этого метода. С этой целью каждому нетерминальному символу и каждой правой части правил из множества P2 поставлена в соответствие вершина графа. Из вершины с меткой U в вершину с меткой V направлено ребро, если в грамматике существует правило U → V.



В эквивалентную грамматику будут включены правила вида $A \to w$, $A \in V_N$; $w \in (V_T \cup V_N)^*$, если из вершины с меткой A существует путь в вершину с меткой w.

$S \to aFb \mid aSb \mid aA;$

$A \to aA \mid aSb \mid aFb;$

$B \to aA \mid aSb \mid aFb;$

$F \to bc \mid bFc.$

Получено то же множество правил P, что и аналитическим методом.

4.5.2. Построение неукорачивающей грамматики

Грамматика, не содержащая правил с пустой правой частью, называется неукорачивающей грамматикой.

В грамматике с правилами вида $A \to \varepsilon$ длина выводимой цепочки при переходе от k-го шага к (k+1)-му уменьшается. Поэтому грамматики с правилами вида $A \to \varepsilon$ называются укорачивающими. Восходящий синтаксический разбор в укорачивающих грамматиках сложнее по сравнению с разбором в неукорачивающих грамматиках, т.к. при редукции необходимо отыскать такой фрагмент входной цепочки, в которую можно вставить пустой символ.

Покажем, что для произвольной КС-грамматики, порождающей язык без пустой цепочки, можно построить эквивалентную неукорачивающую КС-грамматику.

Построим множество всех нетерминальных символов грамматики $G=(V_T, V_N, P, S)$, из которых выводится пустая цепочка, выделив следующие множества:

$W_1 = \{A \mid A \to \varepsilon \in P\},$

$W_{n+1} = W_n \cup \{B \mid B \to \varphi \in P, \varphi \in W^*_n\}.$

Затем найдем множество правил эквивалентной грамматики в два этапа:

а) удалив из множества P исходной грамматики правила с пустой правой частью $P_1' = P \setminus \{A \to \varepsilon \mid A \to \varepsilon \in P\}$;

б) получив новые правила $A \to \varphi'$ после удаления из каждого правила исходной грамматики $A \to \varphi \in P$ те нетерминалы, которые вошли в множество W_n по правилу:

$P_1'' = \{ A \to \varphi' \mid A \to \varphi \in P'; \varphi = \varphi_1 B \varphi_2 \mid B \in W_n; \varphi' = \varphi_1 B \varphi_2\}.$

Повторив п.б) для каждого нетерминала, принадлежащего множеству W_n, получим эквивалентную грамматику $G = (V_T, V_N, P_э, S)$.

Пример. Пусть задана грамматика G со следующими правилами вывода: $S \to AbA \mid cAb \mid Bb$; $A \to aAb \mid \varepsilon$; $B \to AA \mid a$. Необходимо:

1) построить множество нетерминалов, из которых выводится ε;

2) построить неукорачивающую грамматику, эквивалентную исходной.

Для того чтобы построить множество всех нетерминалов грамматики, из которых выводится пустая цепочка, выделим следующие множества:

$W_1 = \{A \mid A \to \varepsilon \in P\};$

$W_{m+1} = W_m \cup \{B \mid B \to \varphi \in P, \varphi \in W^*_m\}.$

Так как мы имеем правило $A \to \varepsilon \in P$, то можно построить множество $W_1 = \{A\}$, включающее нетерминал A.

Построим множество W_2. С нетерминалом A связан нетерминал B, т.е. существует правило $B \to AA$ и $A \in W_1$. Следовательно, $W_2 = \{A,B\}$.

$W_3 = W_2$, т. к. множество $W_3 = \{B \mid B \to \varphi \in P, \varphi \in W^*_m\}$ является пустым.

Исключив правило, содержащее пустую цепочку в правой части, получим неукорачивающую грамматику G1 следующего вида:

$S \to AbA \mid cAb \mid Bb \mid bA \mid Ab \mid cb \mid b;$

$A \to aAb \mid ab;$

$B \to AA \mid A \mid a.$

4.5.3. Построение грамматики с продуктивными нетерминалами

Нетерминальный символ A называется непродуктивным (непроизводящим), если он не порождает ни одной терминальной цепочки, т.е. не существует вывода $A \to^* x$, где $x \in V^*_T$. Поэтому представляет интерес удаление из грамматики всех непродуктивных нетерминальных символов. Рассмотрим, как для произвольной КС-грамматики можно построить эквивалентную КС-грамматику, все нетерминальные символы которой продуктивны. С этой целью выделяется множество $W_1=\{A \mid A \to \varphi \in P, \varphi \in V^*_T\}$. Затем строится множество $W_1, W_2, \ldots, W_{n+1}$ по следующим правилам:

$W_{n+1} = W_n \cup \{B \mid B \to x \in P, x \in (V_T \cup W_n)^*\}$.

Пусть задана грамматика G со следующими правилами вывода: $S \to SA \mid BSb \mid bAb$; $A \to aSa \mid bb$; $B \to bBb \mid BaA$. Построим множество продуктивных нетерминалов:

$W_1 = \{A\}$, т.к. в множестве P есть правило $A \to bb$;

$W_2 = \{A, S\}$, т.к. имеется правило $S \to bAb$ и $A \in W_1$;

$W_3 = W_2$.

Все символы множества $V_N \backslash W_n$ являются непродуктивными, не используются в выводе никакой терминальной цепочки и их можно удалить из грамматики вместе со всеми правилами, в которые они входят. Грамматика G_1, эквивалентная исходной грамматике, будет включать следующее множество правил:

$S \to SA \mid bAb$; $A \to aSa \mid bb$.

В грамматике G_1 все нетерминалы продуктивны.

4.5.4. Построение грамматики, аксиома которой зависит от всех нетерминалов

Существует еще один тип нетерминальных символов, которые можно удалять из грамматики вместе с правилами, в которые они входят. Например, в грамматике G, заданной множеством правил P: $S \to aSb \mid ba$; $A \to aAa \mid bba$, нетерминал A не участвует ни в каком выводе, т.к. из аксиомы нельзя вывести цепочку, содержащую этот нетерминал. Поэтому из заданной грамматики можно удалить нетерминал A. Рассмотрим, как для произвольной КС-грамматики можно построить эквивалентную КС-грамматику, аксиома которой зависит от всех нетерминальных символов.

Для этого построим множество нетерминалов, от которых зависит аксиома. С этой целью выделим множества $W_1, W_2, \ldots, W_{n+1}$ по следующим правилам:

$W_1 = \{S\}$,

$W_{n+1} = W_n \cup \{B \mid A \to xBy \in P, A \in W_n\}$.

Нетерминал $B \in W_n$ тогда и только тогда, когда аксиома S зависит от B. Все нетерминалы, не содержащиеся в W_n, можно удалить из грамматики вместе с правилами, в которые они входят.

Пример. Пусть дана КС-грамматика G:

$S \to abS \mid ASa \mid ab$; $A \to abAa \mid ab$; $B \to bAab \mid bB$.

Найдем нетерминалы, от которых зависит аксиома:

$W_1 = \{S\}$,

$W_2 = \{S, A\}$, т.к. имеется правило $S \to Asa$ и $S \in W_1$;

$W_3 = W_2$.

Эквивалентная грамматика G_1, аксиома которой зависит от всех нетерминальных символов:

$S \to abS \mid ASa \mid ab$; $A \to abAa \mid ab$.

4.5.5. Удаление правил с терминальной правой частью

Пусть в грамматике G имеется с терминальной правой частью $A \to \beta$, где $\beta \in V^*_T$. Тогда любой вывод с использованием этого правила имеет вид: $S \to^* xAy \to x\beta y$. Здесь нетерминал A в сентенциальной форме xAy появился на предыдущем шаге вывода $B \to uAv$. Если в это правило вместо A подставить β, получим правило $B \to u\beta v$. Тогда длина вывода некоторой цепочки сократится на один шаг. Следовательно, для того чтобы удалить терминальное правило грамматики $A \to \beta$, необходимо учесть следующие правила:

а) если для нетерминала A больше нет правил, тогда во всех правых частях A заменяется на β;

б) если для нетерминала A есть другие правила, тогда добавляются новые правила, в которых A заменяется на β.

Пример. Пусть дана КС-грамматика G:

$S \to aSb \mid bAa \mid B$; $A \to ABa \mid b \mid \varepsilon$; $B \to ab \mid ba$.

Удалим правила для нетерминала B. Тогда эквивалентная грамматика G_1 будет включать следующие правила:

$S \to aSb \mid bAa \mid ab \mid ba$; $A \to Aaba \mid Abaa \mid b \mid \varepsilon$.

4.5.6. Построение эквивалентной праворекурсивной КС-грамматики

Некоторые специальные методы грамматического разбора неприменимы к леворекурсивным или праворекурсивным грамматикам, поэтому рассмотрим устранение левой или правой рекурсии. В общем случае избавиться от рекурсии в правилах грамматики невозможно, т.к. бесконечные языки порождаются грамматиками с конечным числом правил только благодаря рекурсии. Поэтому можно говорить лишь о преобразовании одного вида рекурсии в другой. Рассмотрим, как для любой леворекурсивной КС-грамматики построить эквивалентную праворекурсивную КС-грамматику.

Пусть нетерминал A - леворекурсивен, т.е. для него имеются правила следующего вида:

$A \to Ax_1 \mid Ax_2 \mid \ldots \mid Ax_p \mid w_1 \mid \ldots \mid w_k$, где x_i и w_j - цепочки над множеством $V_T \cup V_N$.

Введем дополнительные нетерминалы B и D и указанные правила заменим на эквивалентные им:

$A \to AB \mid D$;

$B \to x_1 \mid x_2 \mid \ldots \mid x_p$;

$D \to w_1 \mid w_2 \mid \ldots \mid w_k$.

В результате замены получим вывод, который имеет вид:

$A \to AB \to ABB \to ABBB \to \ldots \to AB^* \to DB^*$.

Таким образом, для нетерминала A можно определить эквивалентные правила:

$A \to DK$;

$K \to BK \mid \varepsilon$.

Выполняя подстановку D и B в эти правила, получим следующие праворекурсивные правила:

$A \to w_1K \mid w_2K \mid \ldots \mid w_kK$;

$K \to x_1K \mid x_2K \mid \ldots \mid x_pK \mid \varepsilon$.

Пример. Пусть задана грамматика G со следующими правилами вывода: $S \to Sa \mid ba$. Требуется построить эквивалентную ей праворекурсивную грамматику.

Для решения задачи воспользуемся рассмотренным выше алгоритмом. В исходной грамматике один леворекурсивный нетерминал S. Построим для него вывод:

$S \to SB \mid D$;

$B \to a$; $D \to ba$. Вывод из S имеет вид: $S \to SB \to SBB \to \ldots \to DB^*$.

Запишем эквивалентные правила для S:

$S \to DK$; $K \to BK \mid \varepsilon$.

Подставим в эти правила B и D и получим эквивалентную праворекурсивную грамматику G_1:

$S \to baK$; $K \to aK \mid \varepsilon$.

Рассмотрим вывод цепочки baaaa в исходной леворекурсивной грамматике G:

$S \to Sa \to Saa \to Saaa \to baaaa$.

Эту же цепочку можно вывести в эквивалентной праворекурсивной грамматике G_1:

$S \to baK \to baaK \to baaaK \to baaaaK \to baaaa$.

4.6. Задания для самостоятельной работы

1. Построить КС-грамматику, порождающую язык a^n для n>0.
2. Построить КС-грамматику, порождающую язык a^nb^n для n>0.
3. Построить КС-грамматику, порождающую язык $(ab)^nc^*a^{n+m}$ для n>0 и m>0.
4. Построить КС-грамматику, порождающую язык $(ab)^*ca^*b^{n+m}$ для n>0 и m>0.
5. Построить КС-грамматику, порождающую язык $a^nb^{3m}c^ma^{2n} = a^n(bbb)^mc^m(aa)^n$ для n>1 и m>1.
6. Построить КС-грамматику, порождающую язык $(ab)^n(ca)^*b^{n+2}(abc)^*$ для n>0.
7. Для леворекурсивной грамматики G: $S \to SabA \mid ba$; $A \to AbA \mid bAa \mid c$ построить эквивалентную праворекурсивную грамматику.
8. Построить алгоритм устранения правой рекурсии и доказать эквивалентность соответствующего преобразования.
9. Построить КС-грамматику, порождающую идентификаторы, при этом обозначить символом t любую букву, а символом f - любую цифру.
10. Дана КС-грамматика G:

$S \to Aab \mid bSb \mid abB$;

$A \to abS \mid ba \mid bbA$;

$B \to bB \mid Da \mid Aa$;

$D \to Dbb \mid aDa$.

Построить эквивалентную грамматику, все нетерминалы которой продуктивны.

5. ТЕОРИЯ АВТОМАТОВ

5.1. Понятие автомата. Типы автоматов

Автомат - это алгоритм, определяющий некоторое множество и, возможно, преобразующий его в другое множество. Неформальное описание автоматов выглядит следующим образом: автомат имеет входную ленту, управляющее устройство с конечной памятью для хранения номера состояния, а также может иметь вспомогательную (рабочую) и выходную ленты.

Существует два типа автоматов:

1) распознаватели - автоматы без выхода, которые распознают, принадлежит ли входная цепочка заданному множеству L;

2) преобразователи - автоматы с выходом, которые преобразуют входную цепочку x в цепочку y при условии, что $x \in L$.

Входную ленту можно рассматривать как линейную последовательность ячеек, причем каждая ячейка может хранить один символ из некоторого конечного входного алфавита. Лента автомата бесконечна, но занято на ней в каждый момент времени только конечное число ячеек. Граничные слева и справа от занятой области ячейки могут занимать

специальные концевые маркеры. Маркер может стоять только на одном конце ленты или может отсутствовать вообще.

Входная головка в каждый момент времени читает (обозревает) одну ячейку входной ленты. За один такт работы автомата входная головка может сдвинуться на одну ячейку вправо или остаться на месте, при этом она выполняет только чтение, т.е. в ходе работы автомата символы в ячейках входной ленты не меняются.

Рабочая лента - это вспомогательное хранилище информации. Данные с нее могут читаться автоматом, могут и записываться на нее.

Управляющее устройство - это программа, управляющая поведением автомата. Оно представляет собой конечное множество состояний вместе с отображением, описывающим, как меняются состояния в соответствии с текущим входным символом, читаемым входной головкой, и текущей информацией, извлеченной с рабочей ленты. Управляющее устройство также определяет направление сдвига рабочей головки и то, какую информацию записать на рабочую ленту.

Автомат работает, выполняя некоторую последовательность рабочих тактов. В начале такта читается входной символ и исследуется информация на рабочей ленте. Затем, в зависимости от прочитанной информации и текущего состояния, определяются действия автомата:

1) входная головка сдвигается вправо или стоит на месте;
2) на рабочую ленту записывается некоторая информация;
3) изменяется состояние управляющего устройства;
4) на выходную ленту (если она есть) записывается символ.

Поведение автомата удобно описывать в терминах конфигурации автомата, которая включает в себя:

а) состояние управляющего устройства;

б) содержимое входной ленты с положением входной головки;

в) содержимое рабочей ленты вместе с положением рабочей головки;

г) содержимое выходной ленты, если она есть.

Управляющее устройство может быть недетерминированным. В таком случае для каждой конфигурации существует конечное множество возможных следующих тактов, любой из которых автомат может сделать, исходя из этой конфигурации. Управляющее устройство будет детерминированным, если для каждой конфигурации будет возможен только один следующий такт.

Существуют следующие типы автоматов:

1) машина Тьюринга (МТ);
2) линейно-ограниченный автомат (ЛОА);
3) автомат с магазинной памятью (МП-автомат);
4) конечный автомат (КА).

Сложность рабочей ленты определяет сложность автомата. Так, например:

1) машина Тьюринга имеет неограниченную в обе стороны ленту;
2) у линейно-ограниченного автомата длина рабочей ленты представляет собой линейную функцию длины входной цепочки;
3) у МП-автомата рабочая лента работает по принципу магазина LIFO;
4) у конечного автомата рабочая лента отсутствует.

5.2. Формальное определение автомата

Неинициальный автомат - это пятерка вида $A = (K, X, Y, \delta, \gamma)$, где

K - множество состояний (алфавит состояний);

X - входной алфавит;

Y - выходной алфавит;

δ - функция переходов, задающая отображение $K \cdot X \rightarrow K$;

γ - функция выходов, задающая отображение К·Х→У.

Функционирование автомата можно задать множеством команд вида qx → py, где q и p ∈ K, x ∈ X, y ∈ Y.

Пусть на некотором такте t_i управляющее устройство находится в состоянии q, а из входной ленты читается символ x. Если в множестве команд есть команда qx → py, то на такте t_i на выходную ленту записывается символ y, а к следующему такту t_{i+1} управляющее устройство перейдет в состояние p, т.е.:

$y(t) = \gamma(q(t), x(t)),\ q(t+1) = \delta(q(t), x(t))$.

Если же команда qx → py отсутствует, то автомат оказывается блокированным и не реагирует на символ, принятый в момент t_i, а также перестает воспринимать символы в последующие моменты времени.

В соответствии с определением неинициального автомата в начальный момент состояние автомата может быть произвольным.

Если зафиксировано некоторое начальное состояние, то такой автомат называют инициальным, т.е. $q(0) = q_0$.

Инициальный автомат - это шестерка вида $A = (K, X, Y, \delta, \gamma, q_0)$, где

K - множество состояний (алфавит состояний);

X - входной алфавит;

Y - выходной алфавит;

δ - функция переходов (отображение $K \cdot X \rightarrow K$);

γ - функция выходов (отображение $K \cdot X \rightarrow У$);

q_0 - начальное состояние.

5.3. Распознаватели

5.3.1. Языки и автоматы

Задача грамматического разбора заключается в нахождении вывода цепочки в заданной грамматике и определения дерева вывода этой цепочки.

Языки могут быть заданы двумя способами:

1) грамматиками (порождающее средство языка);

2) автоматами (распознающее средство языка).

Различным по сложности автоматам соответствуют разные типы языков. Простейшим типом автоматов являются конечные автоматы.

Конечный автомат имеет входную ленту, с которой за один такт может быть считан один входной символ. Возврат по входной ленте не допускается.

Конечным автоматом называется пятерка вида $A = (K, \Sigma, \delta, p_0, F)$, где

K - конечное множество состояний;

Σ - алфавит;

δ - функция переходов;

p_0 - начальное состояние;

F - множество заключительных состояний.

Автомат можно определить как формальную систему через состояния, через символы, которые пишутся (читаются) с ленты или с нескольких лент, и через набор команд.

Конечный автомат можно представить графом, таблицей переходов, командами, а также матрицей переходов.

5.3.2 Регулярные множества

Регулярные множества образуют класс языков, имеющих важное значение для теории формальных языков. Рассмотрим несколько методов задания языков, каждый из которых

определяет регулярные множества. Среди них - регулярные выражения, праволинейные грамматики, детерминированные и недетерминированные конечные автоматы.

Пусть Σ - некоторый алфавит. Регулярное множество в алфавите Σ определяется рекурсивно следующим образом:

1) 0 - пустое множество;
2) $\{\varepsilon\}$ - множество из пустой цепочки;
3) $\{a\}$ - регулярное множество для каждого элемента $a \in \Sigma$;
4) если P и Q - регулярные множества в алфавите Σ, то регулярными являются множества:
   а) $P \cup Q$
   б) PQ
   в) P^*.

Других регулярных множеств в алфавите Σ нет. Таким образом, некоторое множество цепочек в заданном алфавите Σ называется регулярным тогда и только тогда, когда либо оно является одним из множеств: 0, $\{\varepsilon\}$ или $\{a\}$ для некоторого $a \in \Sigma$, либо его можно получить из этих множеств применением конечного числа операций объединения, конкатенации и итерации.

Для каждого регулярного множества существует по крайней мере одно регулярное выражение, обозначающее это множество.

Язык, распознаваемый конечным автоматом, - это множество цепочек, читаемых автоматом при переходе из начального состояния в одно из заключительных состояний:

$$L(A)=\{a_1 a_2 \ldots a_n \mid p_0a_1 \to p_1, p_1a_2 \to p_2, \ldots, p_{n-1}a_n \to p_n;\ p_n \in F\}.$$

Множество называется регулярным, если существует конечный детерминированный автомат, распознающий его.

5.3.3. Операции над регулярными языками

Так как произвольному конечному автомату однозначно соответствует детерминированный конечный автомат, операции над конечными автоматами эквивалентны операциям над регулярными множествами, или регулярными языками.

Известно, что для произвольного конечного автомата можно построить эквивалентный автомат без циклов в начальных и (или) конечных состояниях.

Теорема. Для произвольного конечного автомата существует конечный автомат без циклов в начальном состоянии.

Доказательство. Пусть $A = (K, \Sigma, \delta, p_0, F)$ - произвольный конечный автомат. Построим автомат:

$$A_1 = (K \cup \{q_0\}, \Sigma, \delta \cup \{q_0a \to p_i \mid p_0a \to p_i \in \delta\}, q_0, F \cup \{q_0 \mid p_0 \in F\}).$$

Любая цепочка $x = a_1 a_2 \ldots a_k$ принадлежит языку L(A) тогда и только тогда, когда существует следующая последовательность команд автомата A:

$p_0a_1 \to p_1;\ p_1a_2 \to p_2;\ \ldots;\ p_{h-1}a_h \to p_h,\ p_h \in F$

и соответствующая ей последовательность команд автомата A_1:

$q_0a_1 \to p_1;\quad p_1a_2 \to p_2;\ \ldots;\quad p_{k-1}a_k \to p_h.$

Таким образом, имеем: $A = A_1$.

Теорема. Для произвольного конечного автомата существует эквивалентный автомат без циклов в заключительном состоянии.

Доказательство. Будем считать, что автомат не имеет циклов в начальном состоянии. Сопоставим заданному произвольному конечному автомату $A = (K, \Sigma, \delta, p_0, F)$ новый автомат A_1:

$$A_1 = (K \cup \{f\}, \Sigma, \delta \cup \{q_ja \to f \mid p_ja \to p_i \in \delta\ \&\ p_i \in F\}, p_0, \{f\} \cup \{p_0 \mid p_0 \in F\}).$$

В результате имеем: $A = A_1$.

Теорема. Множество регулярных языков замкнуто относительно операций итерации, усеченной итерации, объединения, произведения, пересечения, дополнения и разности.

Доказательство. Для доказательства необходимо выполнить операции над соответствующими конечными автоматами и показать, что в результате таких преобразований построенный автомат допускает требуемый язык. Предполагается, что в автомате удалены циклы из начальных и заключительных состояний. Для выполнения перечисленных в теореме операций необходимо выполнить соответствующие преобразования над заданными автоматами.

1. Операция итерации реализуется удалением циклов из начальных и заключительных состояний и объединения полученных состояний. Объединение начального и заключительного состояний означает, что построенный автомат допускает пустую цепочку. Однократный переход из начального в заключительное состояние исходного автомата соответствует допуску цепочек языка L. Поскольку эти состояния объединены, автомат допускает цепочки языков LL, LLL и т.д., т.е. он распознает язык $\{\varepsilon\} \cup L \cup L^2 \cup \ldots = L^*$.

2. Операция произведения над $L(A_1)$ и $L(A_2)$ выполняется с помощью двух преобразований: а) удаляются циклы из начального состояния A_2 и заключительного состояния A_1; б) каждому заключительному состоянию A_1 ставим в соответствие свой экземпляр A_2 и объединяем заключительные состояния A_1 с начальным состоянием соответствующего экземпляра A_2.

3. Объединение $L(A_1)$ и $L(A_2)$ строится с помощью удаления циклов в начальных состояниях A_1 и A_2 и объединения полученных начальных состояний.

4. Усеченная итерация может быть построена двумя способами:

а) $L(A_1)^+ = L(A_1)^* L(A_1)$,
б) $L(A_1)^+ = L(A_1) L(A_1)^*$.

5. Рассмотрим дополнение $L(A_1)$ до Σ^*. Пусть автомат A_1 детерминированный, тогда любая цепочка $x=a_1a_2 \ldots a_n$ распознается по единственному маршруту:

$p_0a_1 \to p_1$
$p_1a_2 \to p_2$
. . .
$p_{n-1}a_n \to p_n$, $p_n \in F$.

Автомат не распознает только те цепочки, которые:

1) либо представляют собой начальную часть цепочки $a_1a_2 \ldots a_j$, при чтении которой автомат переходит в состояние, не являющееся заключительным;

2) либо имеют вид $y = a_1a_2 \ldots a_kbc_1c_2 \ldots c_m$ $(k < n)$, где начало цепочки $a_1a_2 \ldots a_k$ совпадает с началом цепочки $x \in L(A_1)$, но за символом a_k стоит такой символ b, что автомат A_1 его прочитать не может.

Поэтому для того чтобы построить автомат, распознающий дополнение языка, необходимо:

а) все заключительные состояния сделать незаключительными, а незаключительные состояния - заключительными;

б) ввести дополнительное состояние, сделать его заключительным и из каждого состояния провести в это состояние такие дуги, каждая из которых соответствует символам алфавита, не читаемым в этом состоянии;

в) в построенном дополнительном состоянии построить петли для всех символов алфавита, чтобы обеспечить чтение произвольного окончания цепочки $c_1c_2 \ldots c_m$.

6. Разность $L(A_1)$ и $L(A_2)$ строится в соответствии со следующим преобразованием:

$$L(A_1) \setminus L(A_2) = L(A_1) \cap \overline{L(A_2)} .$$

7. Операция пересечения строится в соответствии со следующим преобразованием:

$$L(A_1) \cap L(A_2) = \overline{\overline{L(A_1)} \cup \overline{L(A_2)}}\ .$$

На основании этой теоремы можно строить конечные автоматы, последовательно синтезируя их на основе уже построенных автоматов.

5.3.4. Автоматные грамматики

Линейные грамматики (праворекурсивные и леворекурсивные) называются автоматными грамматиками, так как языки, порождаемые ими, совпадают с языками, распознаваемыми конечными автоматами.

Рассмотрим ряд теорем.

Теорема. Для каждой праволинейной грамматики существует эквивалентный конечный автомат.

Доказательство. Каждому нетерминалу произвольной праволинейной грамматики G поставим в соответствие одно состояние конечного автомата A. Добавим еще одно состояние - единственное конечное состояние. Состояние, соответствующее аксиоме, назовем начальным состоянием.

Каждому правилу $A \to aB$ поставим в соответствие команду $Aa \to B$, а каждому терминальному правилу $A \to a$ поставим в соответствие команду $Aa \to F$.

Таким образом, выводу цепочки в грамматике

$S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow ... \Rightarrow a_1a_2...a_{k-1}A_{k-1} \Rightarrow a_1a_2...a_k$ взаимно однозначно соответствует последовательность команд построенного автомата A:

$Aa_1 \to A_1$; $A_1a_2 \to A_2$; . . . ; $A_{k-1}a_k \to F$. Таким образом, язык L(G) = L(A).

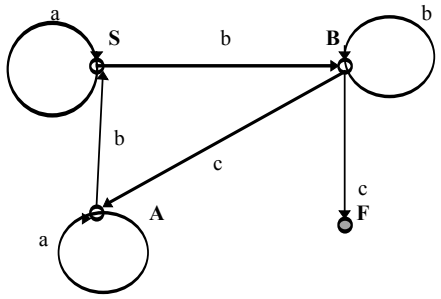
Пример. Пусть для заданной грамматики G требуется построить конечный автомат.

$S \to aS \mid bB$;

$A \to aA \mid bS$;

$B \to bB \mid c \mid cA$.

Решение. Граф автомата будет иметь четыре вершины, три из них помечены нетерминальными символами грамматики S, A, B, четвертая вершина, помеченная символом F, является единственным заключительным состоянием. Начальным состоянием является вершина, соответствующая аксиоме S.



Каждому правилу грамматики поставим в соответствие команду конечного автомата:

$Sa \to S$ - в начальном состоянии при поступлении на вход терминального символа *a* автомат остается в том же состоянии S;

$Sb \to B$ - из начального состояния при поступлении на вход терминала b автомат переходит в состояние B;

$Bb \to B$ - в состоянии B при поступлении на вход терминала *b* автомат остается в том же состоянии B;

Bc $\rightarrow$ F - из состояния B при поступлении на вход терминала *c* автомат переходит в заключительное состояние F;

Bc $\rightarrow$ A - из состояния B при поступлении на вход терминала *c* автомат переходит в состояние A;

Aa $\rightarrow$ A - в состоянии A при поступлении на вход терминала a автомат остается в этом же состоянии A;

Ab $\rightarrow$ S - из состояния A при поступлении на вход терминала *b* автомат переходит в состояние S.

Полученный недетерминированный конечный автомат распознает цепочки языка, порождаемые праворекурсивной грамматикой G.

Теорема. Для произвольного конечного автомата существует эквивалентная праволинейная грамматика.

Доказательство. Каждому состоянию произвольного автомата поставлен в соответствие нетерминал грамматики, причем начальному состоянию будет соответствовать аксиома.

Тогда для каждой команды Ac $\rightarrow$ B в множество правил грамматики включим правило A $\rightarrow$ cB, причем в случае, если B - заключительное состояние, добавим правило A $\rightarrow$ c. Эквивалентность исходного конечного автомата и построенной грамматики очевидна.

Теорема. Для каждой леворекурсивной грамматики существует эквивалентный конечный автомат.

Доказательство. Каждому нетерминальному символу произвольной леворекурсивной грамматики поставим в соответствие состояние конечного автомата, причем состояние, соответствующее аксиоме S, сделаем заключительным. Добавим еще одно состояние N и сделаем его начальным состоянием.

Каждому правилу грамматики A $\rightarrow$ Ba поставим в соответствие команду автомата Ba $\rightarrow$ A. Тогда каждому выводу в грамматике:

$S \Rightarrow A_1a_k \Rightarrow A_2a_{k-1}a_k \Rightarrow \ ... \ \Rightarrow A_{k-1}a_2a_3...a_k \Rightarrow a_1a_2 \ . \ . \ . \ a_k$ однозначно соответствует следующая последовательность команд построенного автомата A:

$Na_1 \rightarrow A_{k-1}$; . . .; $A_2a_{k-1} \rightarrow A_1$; $A_1a_k \rightarrow S$.

Таким образом, язык L(G) = L(A).

Теорема. Для произвольного конечного автомата существует эквивалентная леворекурсивная грамматика.

Доказательство. Каждому состоянию произвольного автомата поставим в соответствие нетерминальный символ грамматики, добавим нетерминал S и сделаем его аксиомой.

Каждой команде Aa $\rightarrow$ B в множество правил включим соответствующее правило B $\rightarrow$ Aa, причем в том случае, если B - заключительное состояние, дополнительно введем правило S $\rightarrow$ Aa, а если A - начальное состояние, то дополнительно введем правило B $\rightarrow$ a.

Тогда последовательности команд:

$A_0a_1 \rightarrow A_1$; $A_1a_2 \rightarrow A_2$; . . . ; $A_{k-1}a_k \rightarrow F$

соответствует следующий вывод:

$S \Rightarrow A_{k-1}a_k \Rightarrow ... \Rightarrow A_1a_2a_3...a_k \Rightarrow a_1a_2a_3...a_k$.

Важной особенностью автоматных грамматик является возможность представления их с помощью конечных графов. По графу грамматики легко отыскивается вывод нужной цепочки.

Любой вывод цепочки в автоматной грамматике соответствует пути в графе этой грамматики, который начинается из вершины S (вершины, помеченной аксиомой) и заканчивается в конечной вершине.

Пример. Построить конечный автомат, распознающий язык $L(A) = \{(ab)^*\}$.

Сначала построим некоторую грамматику G, которая бы порождала язык L(A):

$S \rightarrow aA$;

$A \rightarrow bS \mid b$.

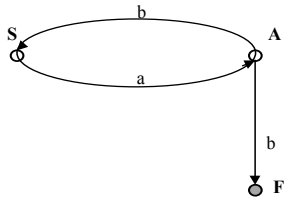
Проверим, действительно ли эта грамматика порождает язык L(A). Для этого построим несколько выводов возможных вариантов цепочек:

1) $S \Rightarrow aA \Rightarrow ab$;

2) $S \Rightarrow aA \Rightarrow abS \Rightarrow abaA \Rightarrow abab$;

3) $S \Rightarrow aA \Rightarrow abS \Rightarrow abaA \Rightarrow ababS \Rightarrow ababaA \Rightarrow ababab$; и т.д.

Таким образом, грамматика G действительно порождает язык L(A), следовательно, можно построить соответствующий этой грамматике конечный автомат. Для этого введем заключительное состояние F, начальное состояние соответствует аксиоме S.



Запишем преобразование правил вывода в команды:

$Sa \rightarrow A$ - из состояния S при поступлении на вход терминала *a* автомат переходит в состояние A;

$Ab \rightarrow S$ - из состояния A при поступлении на вход терминала *a* автомат переходит в состояние S;

$Ab \rightarrow F$ - из состояния A при поступлении на вход терминала *b* автомат переходит в заключительное состояние F.

Таким образом, построен недетерминированный конечный автомат, распознающий заданный язык L(G).

5.4. Автоматы с магазинной памятью

Автомат с магазинной памятью (МП-автомат) имеет рабочую ленту, которая организована в виде магазина.

МП-автомат - это семерка вида:

$M = (K, \Sigma, \Gamma, \delta, p_0, F, B_0)$, где

K - конечное множество состояний;

Σ - алфавит;

Γ - алфавит магазина;

δ - функция переходов;

p_0 - начальное состояние;

F - множество заключительных состояний;

B_0 - символ из множества Γ для обозначения маркера дна магазина.

В общем случае данное определение соответствует недетерминированному автомату.

В отличие от конечного автомата для произвольного МП-автомата нельзя построить эквивалентный детерминированный автомат.

Основное использование распознавательных средств задания языков состоит в построении алгоритмов грамматического разбора. Поэтому необходимо для произвольной КС-грамматики уметь строить эквивалентный МП-автомат.

МП-автомат представляет интерес как средство разбора в КС-грамматиках произвольного вида. Этот факт сформулирован в следующей теореме.

Теорема. Языки, порождаемые КС-грамматиками, совпадают с языками, распознаваемыми МП-автоматами.

Доказательство. Существуют две стратегии разбора: восходящий и нисходящий разбор. Рассмотрим обе стратегии разбора.

5.4.1. Восходящий разбор в МП-автомате

При восходящей стратегии необходимо найти основу и редуцировать ее к какому-нибудь нетерминалу в соответствии с правилами данной грамматики. Это можно сделать в том случае, если реализовать следующий алгоритм функционирования МП-автомата:

1) любой входной символ записывается в магазин;

2) если в верхушке магазина сформирована основа, совпадающая с правой частью правила, то она заменяется на нетерминал в левой части этого правила;

3) разбор заканчивается, если в магазине остается аксиома, а входная цепочка рассмотрена полностью.

В соответствии с этим алгоритмом для КС-грамматики $G = (V_T, V_N, P, S)$ построим МП-автомат:

$M = (K, V_T, \Gamma, \delta, p_0, F, B_0)$, где $\Gamma = V_T \cup V_N \cup \{B_0\}$,

$K = \{p_0, F\}$, $F = \{f\}$.

Функция переходов δ будет содержать следующие команды:

а) $p_0, a, \varepsilon \to p_0, a$ - для любых $a \in V_T$;

б) $p_0, \varepsilon, \varphi' \to p_0, A$ - для всех правил $A \to \varphi \in P$, где φ' - зеркальное отображение φ;

в) $p_0, \varepsilon, SB_0 \to f, B_0$.

В общем случае команда выглядит так:

$p_i, \sigma, \gamma \to p_j, \lambda$, где $p_i \in K$ – состояние автомата до выполнения команды, $\sigma \in V_t$ – символ на входной ленте, $\gamma \in \Gamma$ - символ верхушки магазина, $p_j \in K$ – состояние автомата после выполнения команды, $\lambda \in \Gamma$ - символ, который записывается в магазин.

Таким образом, любому выводу в грамматике G взаимно однозначно соответствует последовательность команд построенного МП-автомата. Обратное построение КС-грамматики по произвольному МП-автомату также возможно, но не представляет практического интереса.

Рассмотрим пример восходящей стратегии разбора.

Пусть дана грамматика G:

$S \to S+A \mid S/A \mid A$

$A \to a \mid (S)$;

$V_N=\{S, A\}$, $V_T=\{a, (,), +, /\}$.

Для заданной КС-грамматики G необходимо построить МП-автомат.

Эквивалентный МП-автомат должен содержать следующие команды:

1. Команды переноса терминальных символов в магазин:

$p_0, a, \varepsilon \to p_0, a$;

$p_0, +, \varepsilon \to p_0, +$;

$p_0, /, \varepsilon \to p_0, /$;

$p_0, (, \varepsilon \to p_0, (;$

$p_0,), \varepsilon \to p_0,).$

Эти команды обеспечивают занесение терминального символа из входной ленты в магазин.

2. Команды редукции по правилам грамматики:

$$p_0, \varepsilon, A+S \to p_0, S;$$
$$p_0, \varepsilon, A/S \to p_0, S;$$
$$p_0, \varepsilon, A \to p_0, S;$$
$$p_0, \varepsilon,)S(\to p_0, A;$$
$$p_0, \varepsilon, a \to p_0, A.$$

Эти команды заменяют зеркальное отображение правила, полученного в верхушке магазина, на нетерминал в левой части данного правила грамматики.

3. Команды проверки на завершение разбора:

$$p_0, \varepsilon, SB_0 \to f, B_0.$$

Разбор завершается, если в магазине остались аксиома и маркер дна магазина, а входная цепочка полностью рассмотрена.

Подадим на вход автомата цепочку a/(a+a) и выполним разбор. Процесс разбора представлен в таблице 1.

5.4.2. Нисходящий разбор в МП-автомате

На любом шаге нисходящего разбора должно применяться какое-либо правило. В начальный момент таким нетерминалом является аксиома. МП-автомат, выполняющий нисходящий разбор, работает по следующему алгоритму:

1) в начальный момент времени в магазин заносится аксиома: $p_0, \varepsilon, \varepsilon \to p_1, S;$

2) для каждого правила $A \to \varphi \in P$ нетерминал в верхушке магазина заменяется на правую часть правила с помощью команды: $p_1, \varepsilon, A \to p_1, \varphi;$

3) для каждого терминала $a \in V_T$ выполняется сравнение символа на входной ленте с символом в верхушке магазина и его поглощение: $p_1, a, a \to p_1, \varepsilon$

4) разбор заканчивается по команде: $p_1, \varepsilon, B_0 \to f, B_0.$

Для грамматики, рассмотренной в предыдущем примере, разбор той же входной цепочки по нисходящей стратегии будет выполняться посредством следующего множества команд:

1) команда занесения аксиомы в магазин:

$$p_0, \varepsilon, \varepsilon \to p_0, S;$$

2) команды замены нетерминала правой частью правила:

$$p_1, \varepsilon, S \to p_1, S+A$$
$$p_1, \varepsilon, S \to p_1, S/A$$
$$p_1, \varepsilon, S \to p_1, A$$
$$p_1, \varepsilon, A \to p_1, a$$
$$p_1, \varepsilon, A \to p_1, (S);$$

3) команды сравнения и поглощения символа с входной ленты и символа в верхушке магазина:

$$p_1, a, a \rightarrow p_1, \varepsilon$$
$$p_1, +, + \rightarrow p_1, \varepsilon$$
$$p_1, /, / \rightarrow p_1, \varepsilon$$
$$p_1, (, (\rightarrow p_1, \varepsilon$$
$$p_1,),) \rightarrow p_1, \varepsilon;$$

4) команда завершения разбора:

$$p_1, \varepsilon, B_0 \rightarrow f, B_0.$$

Процесс разбора цепочки представлен в таблице 2.

5.5. Выводы

Рассмотренные выше МП-автоматы работают недетерминированно, то есть если цепочка принадлежит языку, порождаемому заданной грамматикой, то какой-то из вариантов функционирования автомата осуществит правильный разбор. Если же цепочка не принадлежит языку, то никакой из вариантов разбора не приведет к цели.

Отсутствие детерминированного эквивалентного автомата для произвольной КС-грамматики означает невозможность построения универсальной простой однопроходной программы синтаксического анализа. Поэтому для эффективного разбора необходимо выделять специальные классы КС-грамматик, удовлетворяющие требованиям конкретных типов анализаторов.

Если требуется выполнить разбор для произвольной КС-грамматики, то придется использовать детерминированную программную модель недетерминированного МП-автомата.

5.6. Понятие преобразователей

Автоматы с выходом называются преобразователями. В зависимости от вида функции, отображающей множество состояний и входных символов в множество выходных символов и новых состояний, а также от типа рабочей ленты различают разные виды преобразователей. Рассмотрим конечные автоматы-преобразователи.

Конечным преобразователем называется шестерка вида

$P = (K, X, Y, f, g, q_0)$, где

K - конечное множество состояний;

X - входной алфавит;

Y - выходной алфавит;

f - функция переходов;

g - функция выходов;

q_0 - начальное состояние.

Типы отображений f и g определяют различные виды автоматов. Если g - отображение $K \cdot X$ в Y, то конечный преобразователь называется синхронным. В общем случае это отображение имеет вид $K \cdot X \rightarrow Y^*$.

Пусть $P = (K, X, Y, f, g, q_0)$ - конечный преобразователь. Тогда отображение $S(x) = g(q_0, x)$, определенное для любой цепочки $x \in X^*$, называется конечным преобразователем.

Заметим, что для того чтобы выходную цепочку y можно было считать переводом входной цепочки x, цепочка x должна перевести преобразователь из начального состояния в заключительное.

5.7. Автоматы Мили и Мура

Автоматы Мили и Мура являются неинициальными автоматами. В отображении $S(x) = g(q_0, x)$ зафиксируем начальное состояние q_0, в котором автомат находится в начальный момент времени. Оно существенно влияет на процесс конечного преобразования, т.к. определяет не только результирующую цепочку, но и множество входных цепочек.

Рассмотрим поведение инициальных автоматов, которые могут начинать работать из любого указанного состояния. Такой автомат получает на вход одну цепочку бесконечной длины и перерабатывает её. Реакция такого преобразователя на определенные воздействия непредсказуема, если неизвестно его начальное состояние. Поэтому необходимо решить две задачи, имеющие важное практическое значение:

1) определение того состояния автомата, в котором он находится в момент, начиная с которого исследуется его поведение;

2) распознавание конечного состояния, в которое перешел автомат после завершения испытательной операции. Это состояние будет начальным для следующей серии испытаний.

Эти задачи анализа получили название экспериментов по распознаванию состояния.

5.7.1. Автомат Мили

Автомат Мили - это пятерка вида $M = (K, X, Y, f, g)$, где:

K - множество состояний автомата;

X - входной алфавит;

Y - выходной алфавит;

f - функция переходов (отображение $K \cdot X \to K$);

g - функция выходов (отображение $K \cdot X \to Y$).

Как и любой другой автомат, автомат Мили можно представить в виде таблицы или графа. В графе переходов автомата Мили на дугах указываются через символ ‘/’ входные и выходные символы. Таблица переходов состоит из двух частей: в левой части записываются значения функции выходов, в правой части - значения функции переходов.

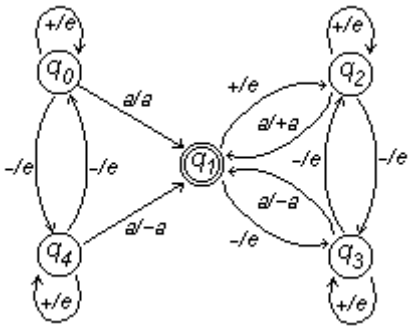
Пример. Построим преобразователь, который распознает арифметические выражения, порождаемые грамматикой:

$$S \to a+S \mid a-S \mid +S \mid -S \mid a$$

и устраняет из этих выражений избыточные унарные операции. Например, выражение –a+–a–+–a он переведет в –a–a+a. Во входном языке символ a представляет идентификатор и перед идентификатором допускается произвольная последовательность знаков унарных операции + и –. Заметим, что входной язык является регулярным множеством.

Пусть $M = (K, X, Y, f, g)$, где

1. $K = \{q_0, q_1, q_2, q_3, q_4\}$,
2. $X = \{a,+, -\}$,
3. $Y = X$.



Преобразователь М начинает работу в состоянии q_0 и, чередуя состояния q_0 и q_4 на входном символе «–», определяет, четное или нечетное число знаков – предшествует первому символу a. Когда появляется a, преобразователь М переходит в состояние q_1, допуская вход, и выдает a или –a в зависимости от того, четно или нечетно число появившихся минусов. Для следующих символов a он подсчитывает, четно или нечетно число предшествующих минусов, с помощью состояний q_2 и q_3. Единственное различие между парами q_2, q_3 и q_0, q_4 состоит в том, что если символу a предшествует четное число минусов, то первая из них выдает +a, а не только a.

Таблица переходов выглядит следующим образом:

	Y			**K**		
K\X	**a**	**+**	**–**	**a**	**+**	**–**
q_0	a	ε	ε	q_1	q_0	q_4
q_1	-	ε	ε	-	q_2	q_3
q_2	+a	ε	ε	q_1	q_2	q_3
q_3	–a	ε	ε	q_1	q_3	q_2
q_4	–a	ε	ε	q_1	q_4	q_0

5.7.2. Автомат Мура

Автомат Мура - это «пятерка» вида $U = (K_1, X, Y, f_1, h)$, где:

K_1 - множество состояний автомата;
X - входной алфавит;
Y - выходной алфавит;
f_1 - функция переходов (отображение $K \cdot X \to K$);
h - функция выходов (отображение $K \cdot X \to Y$).

При представлении автомата Мура графом дуги помечаются символами входного алфавита, а каждая вершина графа - состоянием и символом выходного алфавита.

При формальном сравнении определений автоматов Мили и Мура может показаться, что автомат Мура может быть задан как входонезависимый автомат Мили, т.е. такой автомат Мили, выходная функция которого удовлетворяет следующим условиям: $\forall a \in X, \forall b \in X, \forall z \in Z (g(z, a) = g(z, b))$. Однако это не соответствует способу функционирования автоматов Мура в соответствии с введенным определением.

В автомате Мура реализована иная временная связь между переходами из одного состояния в другое и выходом, по сравнению с автоматом Мили, у которого выход, соответствующий некоторому входу и определенному состоянию, порождается во время перехода автомата в следующее состояние. У автомата Мура сначала порождается выход, а потом - переход в следующее состояние, причем выход определяется только состоянием автомата.

5.7.3. Равносильность автоматов Мили и Мура

Равносильность заключается в том, что множество реакций этих автоматов совпадает:

$L(M) = \{q_z \mid q_Z \in K\}$;

$L(U) = \{h_t \mid h_t \in K_1\}$;

$L(M) = L(U)$.

Теорема. Для каждого автомата Мура можно построить равносильный автомат Мили.

Доказательство. Граф равносильного автомата Мили M можно получить в том случае, если каждому ребру автомата Мура U сопоставить ребро автомата M.

Пусть $w = x_1 x_2 \ldots x_n$ - входная цепочка, тогда множества реакций для автоматов M и U будут соответственно представлены следующим образом:

$q_0 / y_1, x_1 \to q_1 / y_2, x_2 \to \ldots \to q_{n-1} / y_n, x_n$;

$q_0, x_1 / y_1 \to q_1, x_2 / y_2 \to \ldots \to q_{n-1}, x_n / y_n$.

Теорема. Для любого автомата Мили можно построить эквивалентный автомат Мура.

Доказательство. В качестве множества K_1 автомата Мура возьмем $K_1 = K \cdot Y$. Для обеспечения равносильности автоматов M = U функции переходов и выходов определим следующим образом:

$f_1(p \cdot y, a) = \{qb \mid f(p, a) = q, b \in X\}$;

$h(p \cdot y) = y$.

Если реакция автомата M на входную цепочку вида $w = x_1 x_2 \ldots x_n$ из состояния q_0 имеет вид

$$q_0, x_1 / y_1 \to q_1, x_2 / y_2 \to \ldots \to q_{k-1}, x_k / y_k \qquad (1),$$

то существует такое состояние $q_0 \cdot x_1$ недетерминированного автомата U, что, начиная работу из этого состояния, автомат U выполняет следующие действия:

$$q_0 \cdot x_1 / y_1 \to q_1 \cdot x_2 / y_2 \to \ldots \to q_{k-1} \cdot x_k / y_k, x_k \qquad (2).$$

Аналогично можно доказать и обратную теорему о том, что из существования реакции (2) следует существование реакции (1), что подтверждает равносильность автоматов Мили и Мура.

5.7.4. Задания для самостоятельной работы.

1. Построить конечный преобразователь, моделирующий работу светофора. Рассмотреть различные алгоритмы переключения светофора.
2. Построить автомат Мили, который читает текст, написанный на русском языке. Автомат должен считать все слова, начинающиеся с символа "б" и оканчивающиеся символом "т", т.е. такие, как "бит", "байт", "батут" и т.д. При построении автомата все буквы кроме "б" и "т" можно обозначить каким-либо символом, например, "|".
3. Построить автомат Мили, который читает программу, написанную на языке программирования высокого уровня. Автомат должен считать все целые константы.

ЗАКЛЮЧЕНИЕ

Всякий алгоритм можно рассматривать как некоторое универсальное средство для решения целого класса задач. Но существуют такие классы задач, для решения которых нет общего универсального алгоритма. Проблемы решения такого рода задач называются алгоритмически неразрешимыми проблемами. Однако алгоритмическая неразрешимость проблемы решения задач того или иного класса не означает невозможность решения любой частной задачи из этого класса. Переход от интуитивного понятия алгоритма к формальному определению алгоритма (рекурсивные функции, машины Тьюринга) позволяет доказать алгоритмическую неразрешимость ряда проблем.

Понятия, алгоритмы и методы теории формальных языков, грамматик и автоматов являются теоретической основой современной теории программирования, построения алгоритмических языков, проектирования языковых процессоров, в частности, компиляторов, ассемблеров, макрогенераторов и т.д.

Рекомендуемая литература

1. *Алферова З.В.* Теория алгоритмов. - М.: Статистика, 1973.
2. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода, компиляции. В 2 т. Т. 1, 2. - М.: Мир, 1980.
3. *Брауэр В*. Введение в теорию конечных автоматов. -М.: Радио и связь, 1987.
4. *Гинзбург С*. Математическая теория контекстно-свободных языков. - М.: Мир, 1970.
5. *Гросс М., Лантен А*. Теория формальных грамматик.- М.: Мир, 1971.
6. *Крючкова Е.Н*. Теория алгоритмов. - Барнаул; 1995.
7. *Крючкова Е.Н*. Теория формальных языков и автоматов. - Барнаул; 1996.
8. *Кузнецов О.П., Адельсон-Вельский Г.М*. Дискретная математика для инженера. - М.: Энергоатомиздат, 1988.
9. *Любимский Э.З., Мартынюк В.В., Трифонов Н.П*. Программирование. - М.: Наука, 1980.
10. *Мелихов А.Н., Кодачигов В.И*. Теория алгоритмов и формальных языков. - Таганрог; 1983.
11. *Рейуорд - Смит В. Дж*. Теория формальных языков. Вводный курс. - М.: Мир, 1988.
12. *Саломаа А*. Жемчужины теории формальных языков. - М.: Мир, 1987.

ОГЛАВЛЕНИЕ

Таблица 1

вход	a		/	(	a		+	A		)						
магазин	B_0	a B_0	A B_0	S B_0	/ S B_0	(/ S B_0	a (/ S B_0	A (/ S B_0	S (/ S B_0	+ S (/ S B_0	a + S (/ S B_0	A + S (/ S B_0	S (/ S B_0 |) S (/ S B_0	A / S B_0	S B_0	B_0
p	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	f

Таблица 2

вход			a	/		(			a	+		a	)		
магазин	B_0	S B_0	S / A B_0	a / A B_0	a / A B_0	/ A B_0	A B_0	(S) B_0	S) B_0	S + A) B_0	A + A) B_0	a + A) B_0	+ A) B_0	A) B_0	a) B_0	) B_0	B_0
p	p_0	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	f