Содержание

[Лекция 1. Основные понятия теории формальных языков и грамматик 2](#_Toc406055381)

[Распознавание типов формальных языков и грамматик 2](#_Toc406055382)

[Классификация грамматик по Хомскому 3](#_Toc406055383)

[Лекция № 2.Теория контекстно-свободных языков 6](#_Toc406055384)

[Алгоритм 1. Проверка существования языка грамматики 6](#_Toc406055385)

[Алгоритм 2. Устранение нетерминалов, не порождающих терминальных строк 7](#_Toc406055386)

[Алгоритм 3. Устранение недостижимых символов 7](#_Toc406055387)

[Алгоритм 4. Устранение ε-правил 8](#_Toc406055388)

[Алгоритм 5. Устранение цепных правил 9](#_Toc406055389)

[Алгоритм 6. Устранение левой факторизации правил 10](#_Toc406055390)

[Алгоритм 7. Устранение прямой левой рекурсии 11](#_Toc406055391)

[Лекция № 3 . Виды грамматик и языков 13](#_Toc406055392)

[Регулярные языки. 13](#_Toc406055393)

[Лекция № 4. Источники и языки. 16](#_Toc406055394)

[Грамматики 19](#_Toc406055395)

[Упрощение грамматик. 20](#_Toc406055396)

[Детерминированные источники. 21](#_Toc406055397)

[Лекция № 5. Автоматы. 24](#_Toc406055398)

[Определение и способы задания. 24](#_Toc406055399)

[Пример различных представлений автомата. 25](#_Toc406055400)

[Отображение языков, задаваемое автоматом. 25](#_Toc406055401)

[Автомат Мура. 26](#_Toc406055402)

[Язык, создаваемый муровским автоматом 27](#_Toc406055403)

[Лекция № 6. Минимизация автоматов 28](#_Toc406055404)

[Лекция № 7. Машины Тьюринга 32](#_Toc406055405)

[Примеры машин Тьюринга для конкретных вычислений. 35](#_Toc406055406)

# Лекция 1. Основные понятия теории формальных языков и грамматик

## Распознавание типов формальных языков и грамматик

**Определение 1.1.** Алфавитом *V* называется конечное множество символов.

**Определение 1.2.** Цепочкой α в алфавите *V* называется любая конечная

последовательность символов этого алфавита.

**Определение 1.3.** Цепочка, которая не содержит ни одного символа, называется пустой цепочкой и обозначается ε.

**Определение 1.4.** Формальное определение цепочки символов в алфавите

*V*:

1) ε - цепочка в алфавите *V*;

2) если α - цепочка в алфавите *V* и *а* – символ этого алфавита, то α*а* – цепочка в алфавите *V*;

3) β - цепочка в алфавите *V* тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

**Определение 1.5.** Длиной цепочки α называется число составляющих ее

символов (обозначается *|* α *|*).

Обозначим через *V\** множество, содержащее все цепочки в алфавите *V*,

включая пустую цепочку ε, а через *V+* - множество, содержащее все цепочки в алфавите *V*, исключая пустую цепочку ε.

**Пример 1.1.** Пусть *V* = {1, 0}, тогда *V*\* = {ε, 0,1, 00, 01,10,11, 000, K}, а

*V* + = {0,1, 00, 01,10,11, 000, K}.

**Определение 1.6.** Формальной грамматикой называется четверка вида:

*G* = (*VT, VN* , *P*, *S*), (1.1)

где *VN* - конечное множество нетерминальных символов грамматики

(обычно прописные латинские буквы);

*VT* - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры, и т.п.), *VT* ∩*VN =*∅;

*Р* – множество правил вывода грамматики, являющееся конечным

подмножеством множества (*VT* ∪ *VN*)*+* × (*VT* ∪ *VN*)*\**;

элемент (α*,* β) множества *Р* называется правилом вывода и записывается в виде α→β (читается: «из цепочки α выводится цепочкаβ»);

*S -* начальный символ грамматики, *S* ∈*VN*.

Для записи правил вывода с одинаковыми левыми частями видаα →β1,

α→β 2, K, α →β *n* используется сокращенная форма записи

α→β1 | β 2 |K| β *n.*

**Пример 1.2.** Грамматика *G*1*=* ({0, 1}, {*A*, *S*}, *P*1, *S*), где множество *Р*1 состоит из правил вида: 1) *S*→ 0*A*1; 2) 0*A*→ 00*A*1; 3) *A*→ε*.*

**Определение 1.7.** Цепочка β ∈ (*VT* ∪ *VN*)*\** непосредственно выводима из

цепочки α ∈ (*VT* ∪*VN* )+ в грамматике *G* = (*VT* ,*VN* , *P*, *S*) (обозначается: α⇒β), если α =ξ1γξ 2 и β =ξ1δξ 2, где \*

ξ1, ξ 2, δ ∈ (*VT* ∪*VN* ), ∈( ∪ )+ γ *VT VN* и правило

вывода γ →δ содержится во множестве *Р*.

**Определение 1.8.** Цепочка β ∈ (*VT* ∪ *VN*)*\** выводима из цепочки α ∈(*VT* ∪*VN* )+ в грамматике *G* = (*VT* ,*VN* , *P*, *S*) (обозначается α⇒\*β), если существует последовательность цепочек γ 0 ,γ 1 ,K,γ *n* (*n*≥*0*) такая, что α =γ 0 ⇒γ 1⇒K⇒γ *n* = β .

**Пример 1.3.** В грамматике *G*1 *S*⇒\*000111, т.к. существует вывод

*S* ⇒0*A*1⇒00*A*11⇒000*A*111⇒000111.

**Определение 1.9.** Языком, порожденным грамматикой *G* = (*VT, VN,* *P*, *S*),

называется множество всех цепочек в алфавите *VT*, которые выводимы из начального символа грамматики *S* c помощью правил множества *Р*, т.е. множество *L*(*G*) = {α ∈*VT*\* | *S* ⇒\*α}.

**Пример 1.4.** Для грамматики *G*1 *L*(*G*1) *=* {0*n*1*n* | *n>*0}.

Определение 1.10. Цепочка α ∈ (*VT* ∪*VN)* \*, для которой существует вывод

*S*⇒\*α, называется сентенциальной формой в грамматике *G* = (*VT, VN,* *P*, *S*).

**Определение 1.11.** Грамматики *G*1 и *G*2 называются эквивалентными, если *L*(*G*1) = *L* (*G*2).

**Пример 1.5.** Для грамматики *G*1 эквивалентной будет грамматика

*G*2 = ({0, 1}, {*S*}, *P*2, *S*), где множество правил вывода *P*2 содержит правила вида *S* → 0*S*1 | 01.

## Классификация грамматик по Хомскому

**Тип 0.** Грамматика *G* = (*VT, VN,* *P*, *S*) называется грамматикой типа 0, если на ее правила вывода не наложено никаких ограничений, кроме тех, которые указаны в определении грамматики.

**Тип 1.** Грамматика *G* = (*VT* ,*VN* , *P*, *S*) называется контекстно-зависимой

грамматикой (КЗ-грамматикой), если каждое правило вывода из множества *Р*

имеет вид α→β, где α ∈ (*VT* ∪ *VN*) +, β ∈ (*VT* ∪ *VN*) \* и |α| ≤ |β|.

**Тип 2.** Грамматика *G* = (*VT* ,*VN* , *P*, *S*) называется контекстно-свободной

грамматикой (КС-грамматикой), если ее правила вывода имеют вид: *A*→β,

где *A*∈*VN* и β ∈*V* \*.

**Тип 3.** Грамматика *G* = (*VT* ,*VN* , *P*, *S*) называется регулярной грамматикой (Р-грамматикой) выровненной вправо, если ее правила вывода имеют вид

*A*→*aB* | *a* , где *a*∈*VT* ; *A*, *B*∈*VN* .

Грамматика *G* = (*VT* ,*VN* , *P*, *S*) называется регулярной грамматикой (Р-

грамматикой) выровненной влево, если ее правила вывода имеют вид

*A*→*Ba* | *a* , где *a*∈*VT* ; *A*, *B*∈*VN* .

**Определение 1.12.** Язык *L*(*G*) называется языком типа *k*, если его можно описать грамматикой типа *k*, где *k* – максимально возможный номер типа грамматики.

Соотношение типов грамматик и языков представлено на рисунке 1.1.

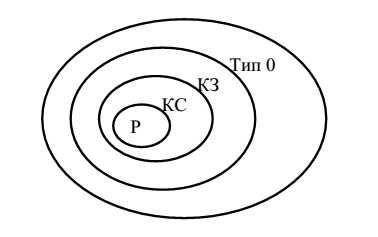


Рисунок 1.1 – Соотношение типов формальных языков и грамматик

Р – регулярная грамматика;

КС – контекстно-свободная грамматика;

КЗ – контекстно-зависимая грамматика;

Тип 0 – грамматика типа 0.

Пример 1.6. Примеры различных типов формальных языков и грамматик по

классификации Хомского. Терминалы будем обозначать строчными cимволами, нетерминалы – прописными буквами, начальный символ грамматики – *S*.

а) Язык типа 0 *L*(*G*)={ 2 1 | 1 2*a bn* − *n* ≥ } определяется грамматикой с прави-

лами вывода:

1) *S* → *aaCFD*;

2) *AD* → *D*;

3) *F* → *AFB | AB*;

4) *Cb* → *bC*;

5) *AB* → *bBA*;

6) *CB* → *C*;

7) *Ab* → *bA*;

8) *bCD* → ε.

б) Контекстно-зависимый язык *L*(*G*)={*anbncn* | *n*≥1} определяется грамматикой с правилами вывода:

1) *S* → *aSBC | abc* ;

2) *bC* → *bc*;

3) CB → BC;

4) cC → cc;

5) *BB* → *bb*.

в) Контекстно-свободный язык *L*(*G*)={(*ab*)*n*(*cb*)*n* | *n*>0 } определяется

грамматикой с правилами вывода:

1) *S* → *aQb | accb*;

2) *Q* → *cSc*.

г) Регулярный язык *L*(*G*)= {ω⊥ | ω∈ {*a, b*} +, где нет двух рядом стоящих, *а*} определяется грамматикой с правилами вывода:

1) *S* → *A*⊥ *| B*⊥;

2) *A* → *a | Ba*;

3) *B* → *b | Bb | Ab*.

# Лекция № 2. Теория контекстно-свободных языков

**Определение 1**. КС-грамматика называется приведенной, если она не

имеет циклов, ε-правил и бесполезных символов.

Рассмотрим основные алгоритмы приведения КС-грамматик.

Перед всеми другими исследованиями и преобразованиями КС-грамматик выполняется проверка существования языка грамматики.

## Алгоритм 1. Проверка существования языка грамматики

Вход: КС-грамматика G= (*VT, VN* , *P*, *S*).

Выход: заключение о существовании или отсутствии языка грамматики.

Определим множество нетерминалов, порождающих терминальные строки .

*Шаг 1*. Положить N0=Ø.

*Шаг 2*. Вычислить



*Шаг3*. Если , то положить i=i+1 и перейти к пункту 2, иначе

считать .

Если , то выдать сообщение о том, что язык грамматики существует, иначе сообщить об отсутствии языка.

Пример 2.1. Дана грамматика , где множество

правил P:

.

Построим последовательность приближений множества N:

N0 = Ø;

N1 = {A, B};

N2 = {S, A, B};

N3 = {S, A, B}.

Т.к. N2=N3, то N = {S, A, B}, следовательно, язык грамматики существует,

потому что начальный символ .

**Определение 2**. Бесполезными символами грамматики называют:

а) нетерминалы, не порождающие терминальных строк, т.е. множество

символов



б) недостижимые нетерминалы, порождающие терминальные строки, т.е.

множество символов



в) недостижимые терминалы, т.е. множество символов



## Алгоритм 2. Устранение нетерминалов, не порождающих терминальных строк

Вход: КС-грамматика G= (*VT, VN* , *P*, *S*).

Выход: КС-грамматика , такая, что  ) (и для всех  существуют выводы , где .

*Шаг 1*. Определить множество нетерминалов, порождающих терминальные строки, с помощью алгоритма 2.1.

*Шаг 2*. Вычислить

,

где  - это множество правил, содержащих бесполезные нетерминалы .

Пример 2.2. Дана грамматика с правилами P : Преобразуем ее в эквивалентную грамматику G′ по алгоритму 2.2:

N0= Ø;

N1= {S, B, C};

N2= {S, B, C}.

Т.к. N1 =N2, то N= {S, B, C}. После удаления бесполезных нетерминалов

и правил вывода, получим грамматику  с правилами P′ : 

## Алгоритм 3. Устранение недостижимых символов

Вход: КС-грамматика G= (*VT, VN* , *P*, *S*).

Выход: КС-грамматика , такая, что  и для всех  существует вывод , где .

Определим множество достижимых символов Z грамматики G, т.е. множество

.

*Шаг 1*. Положить 

*Шаг 2*. Вычислить очередное приближение следующим образом:

.

*Шаг 3*. Если, , то положить i:=i+1 и перейти к шагу 2, иначе считать .

*Шаг 4*. Вычислить



где  - это множество правил, содержащих недостижимые символы .

Пример 2.3. Дана грамматика  с правилами P′: .

Преобразуем ее в эквивалентную грамматику G′ по алгоритму 2.3:

W0= {S};

W1= {S, a, b};

W2= {S, a, b}.

Т.к. W1=W2, то W={S, a, b}. Множество недостижимых символов

.

Тогда после удаления недостижимых символов, получим

грамматику  с правилом P′: .

## Алгоритм 4. Устранение ε-правил

Вход: КС-грамматика .

Выход: Эквивалентная КС-грамматика  без ε-правил для всех нетерминальных символов, кроме начального, который не должен встречаться в правых частях правил грамматики.

*Шаг 1*. В исходной грамматике G найти ε-порождающие нетерминальные

символы , такие, что  .

1.1 Положить 

1.2 Вычислить 

1.3 Если , то положить i:=i+1 и перейти к пункту 1.2, иначе считать .

*Шаг 2*. Из множества P правил исходной грамматики G перенести во

множество P′ все правила, за исключением ε-правил, т.е.

.

*Шаг 3*. Пополнить множество P′ правилами, которые получаются из каждого правила этого множества путем исключения всевозможных комбинаций

ε-порождающих нетерминалов в правой части. Полученные при этом ε-правила во множество P′ не включать.

*Шаг 4*. Если , то , где  иначе .

Пример4.4. Дана грамматика) G=({0,1},{ S, A,B},P,S) и правилами P :

. Преобразуем ее в эквивалентную

грамматику по алгоритму 4.

*Шаг 1*. N0 = {A, B};

N1 = {S, A, B};

N2 = {S, A, B}.

Т.к. N1 = N2, то искомое множество построено и N = {S, A, B}.

*Шаг 2, 3*. Множество P′: 1) S → AB | A| B; 2) A → 0A| 0; 3) B →1B |1.

*Шаг 4*. Т.к. S ∈ N , то введем новый нетерминал С и пополним множество

P′ правилом вида C → S |ε .

Результирующая грамматика будет иметь вид:

 с правилами P′:

1) C→S|ε;

2) S→AB|A|B ;



## Алгоритм 5. Устранение цепных правил

## 

Вход: КС-грамматика G=(VT, VN,P,S . )

Выход: Эквивалентная КС-грамматика G′ = (VT ,V′N , P′, S′) без цепных

правил, т.е. правил вида A→ B, где A B∈,VN .

*Шаг 1*. Для каждого нетерминала A вычислить множество выводимых из

него нетерминалов, т.е. множество NA ={B | A⇒\*B где B∈ ,VN}.

1.1 Положить N0A ={A}.

1.2 Вычислить NiA =NAi−1∪{C|(B→ C)∈P, B∈ NAi−1,C∈VN}.

1.3 Если NiA ≠ NAi−1, то положить i:=i+1 и перейти к пункту 1.2, иначе

считать NA = NiA .

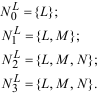
*Шаг 2*. Построить множество P′ так: если (B →α)∈P не является цепным правилом (α ∉VN , то) включить в P′ правило A→α для каждого A, такого, что B∈ N A.

Пример 2.5. Грамматика G=({+,n},{ L,M,N},P,L ) с правилами P :

.

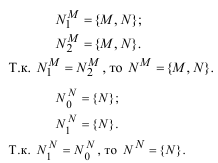
Преобразуем ее в эквивалентную грамматику G′ по алгоритму 5.

Шаг 1.



Т.к. N2L = N3L, то NL ={ L, M, N}.





Шаг 2. Преобразовав правила вывода грамматики, получим грамматику

с правилами P′ :



## Алгоритм 6. Устранение левой факторизации правил

Вход: КС-грамматика G=(VT, VN,P,S).

Выход: Эквивалентная КС-грамматика G′ = (VT ,V′N , P′, S′) без одинаковых префиксов в правых частях правил, определяющих нетерминалы.

*Шаг 1*. Записать все правила для нетерминала X , имеющие одинаковые

префиксы α ∈V\*, в виде одного правила с альтернативами:



*Шаг 2.* Вынести за скобки влево префикс α в каждой строке-альтернативе: X →α(β1 |β2 |...|βn).

*Шаг 3.* Обозначить новым нетерминалом Y выражение, оставшееся в

скобках: X →αY,Y → β1 |β2 |...|βn.

*Шаг 4.* Пополнить множество нетерминалов новым нетерминалом Y и

заменить правила, подвергшиеся факторизации, новыми правилами для X и Y .

*Шаг 5*. Повторить шаги 1-4 для всех нетерминалов грамматики, для которых это возможно и необходимо.

Пример 2.6. Дана грамматика G=({k,l,m ,n},{S},P,S) с правилами P : 

1)Преобразуем ее в эквивалентную грамматику G′ по алгоритму 6:

*Шаг 1.* S → kSl | kSm | n .

*Шаг 2*. S → kS(l | m) | n.

*Шаг 3,4*. Пополнив множество нетерминалов новым нетерминалом С и

заменив правила, подвергшиеся факторизации, получим грамматику



с правилами P′:

1)S→kSC;

2) S→n;

3)C→l ;

4) C→m .

## Алгоритм 7. Устранение прямой левой рекурсии

Вход: КС-грамматика G=(VT, VN,P,S).

Выход: Эквивалентная КС-грамматика G′ = (VT ,V′N , P′, S′) без прямой левой рекурсии, т.е. без правил вида A→ Aα, A ∈VN, α∈V\* .

*Шаг 1*. Вывести из грамматики все правила для рекурсивного нетерминала X :



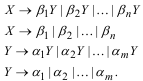
*Шаг 2*. Внести новый нетерминал Y так, чтобы он описывал любой

«хвост» строки, порождаемой рекурсивным нетерминалом X :



*Шаг 3*. Заменить в рекурсивном правиле для X правую часть, используя

новый нетерминал и все не рекурсивные правила для X так, чтобы генерируемый язык не изменился:



*Шаг 4.* Пополнить множество нетерминалов грамматики новым нетерминалом Y . Пополнить множество правил грамматики правилами, полученными на шаге 3.

*Шаг 5*. Повторить действия шагов 1-4 для всех рекурсивных нетерминалов грамматики, после чего полученные множества нетерминалов и правил принять в качестве V′N и P′ .

Пример 4.7. Дана грамматика G=({a,b,c,d,z} ,{S,A,B,C},P,S ) с правилами P:

1) S → Aa;

2) A→ Bb;

3) B →Cc|d;

4) C →Ccbz |dbz.

После устранения прямой левой рекурсии получим эквивалентную грамматику

G′=({a,b,c,d,z} ,{S,A,B,C, Z},P′,S ) с правилами P′:



# 

# Лекция № 3 . Виды грамматик и языков

## Регулярные языки.

Возьмем некоторое конечное множество символов A, назовем его *алфавитом*, а его элементы - *буквами*.

*Словом* в данном алфавите называется конечная цепочка букв этого алфавита.

Буквы будем обозначать a, a1, a2, ..., b, b1, ..., а слова - α, β, ..., причем α = a(1)a(2)...a(n), где α(i) - i-тая буква слова α.

*Длиной слова* α называется число букв в данном слове: |α| = n. Например, |abbbc|= 5.

Введем также пустое слово Λ как слово нулевой длины: |Λ| = 0.

Слово β называется подсловом слова α, если найдутся слова α1 и α2,необязательно непустые, что α = α1βα2. Например, подсловами слова abc является само abc, а также a, b, c, ab и bc.

Множество всех возможных слов в алфавите A обозначим A∗.

*Языком* в данном алфавите A называется любое подмножество L множества всех слов A∗, L ⊂ A∗.

Пример. A = {a,b,c}

L = {Λ, aa, abc, cb, bc}. A∗ - все слова, которые можно составить из букв a, b, c: Λ, a, b, c, aa, ab, ac, ba,...

**Операции над языками. Регулярные языки.**

Рассмотрим произвольный алфавит A и всевозможные языки в нем.

Определим следующие операции.

1) *Объединением* языков L1 и L2 называется множество слов, входящих хотя бы в один из этих языков: L = L1 ∨ L2 = {α|α ∈ L1 или α ∈ L2}.

2) *Конкатенацией* (произведением) языков L1 и L2 называется множество слов вида L1 · L2 = {αβ, где α ∈ L1, β ∈ L2}. Таким образом, это слова, получающиеся приписыванием к каждому слову из L1 слова из L2. Конкатенация слов α и β есть слово αβ.

Например, пусть L1 = {a, ab, b}, L2 = {b, ca}. ТогдаL1 ∨ L2 = {a, ab, b, ca}, L1·L2= {ab, abb, bb, aca, abca, bca}

В частности,

 обозначается как Lk и есть {α1...αk|αi ∈ L, i =1...k}.

Например, L = {a, bb}, L2 = {aa, abb, bba, bbbb}

Рассмотрим произвольный язык L и пустое слово Λ. По определению

Λ · L = L, L · Λ = L.

В качестве языка можно рассматривать и пустое множество слов.

Выполнено:

L · ∅ = ∅ = ∅ · L

L ∨ ∅ = L

3) *Итерацией языка L* называется язык вида Λ∨L∨L2 ∨...∨Li ∨..., он обозначается L∗.

Например, в алфавите A = {a,b} итерация языка L = {a2,ab} будет L∗ = {Λ, a2, ab, a4, abab, a3b, ab2a, a6, a5b, aba4, ...}.

Для L = {a2} итерация такова: L∗ = {Λ, a2, a4, a6, ...}.

Множество всех слов в алфавите A = {a1, ..., ar} получается итерацией объединения его букв: A∗ = (a1 ∨ ... ∨ ar) ∗.

Язык называется *регулярным*, если его можно получить из простейших языков {Λ},{a},a ∈ A, с помощью этих трех операций за конечное число шагов.

***Формальное определение***.

1) ∅ - регулярный язык, {Λ} - регулярный язык, {a},a ∈ A, - регулярный язык.

2) Пусть L1, L2 - регулярные языки. Тогда языки L1 · L2, L1 ∨ L2 и L1∗ также регулярны.

3) Других регулярных языков нет.

Выражение, задающее регулярный язык, называется *регулярным выражением.* Для простейших языков эти выражения - ∅,Λ,a, остальные составляются из простейших с помощью ∨,·,∗ и скобок.

*Задача*. Составить регулярное выражение для языка в алфавите {a, b, c}, состоящее из всех слов, начинающихся на *ab*, но не заканчивающихся на c.

Как уже было сказано, множество всех слов в алфавите A = {a,b,c} есть A∗=(a∨b∨c) ∗.

Все слова, начинающиеся на *ab* - конкатенация *ab* со множеством всех слов. Выражение для такого языка есть ab(a ∨ b ∨ c) ∗. Слово не заканчивается на букву c, значит, оно заканчивается на a или на b. Поэтому регулярное выражение для данного языка имеет вид ab(a ∨ b ∨ c) ∗ (a ∨ b).

*Задача*. Составить регулярное выражение для языка в алфавите{a, b, c} из всех слов, где буква *b* встречается только в виде массива *bn*,где n - четное число.

Сначала зададим массив *bn*. Это *(bb) ∗*, кстати, сюда входит и пустое слово (случай n = 0). Слова языка Ð всевозможные последовательности букв *a, c* и таких массивов.

Искомое регулярное выражение - (a ∨ (bb) ∗ ∨ c) ∗.

Но не все формальные языки являются регулярными.

***Пример нерегулярного языка***. Рассмотрим алфавит A = {a}. Тогда язык, состоящий из слов, длина которых - квадрат некоторого натурального числа, будет нерегулярным. L = {ak|k = n2,n ∈ N}

# Лекция № 4. Источники и языки.

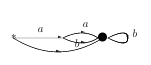
Пусть зафиксирован некоторый алфавит A. Возьмем ориентированный псевдограф, некоторым ребрам которого приписаны буквы из алфавита A. Ребра без букв назовем пустыми. Выделим некоторое множество вершин, называемых начальными и множество вершин, называемых заключительными. Такая конструкция называется *источником.*

Начальные вершины обозначаются ∗, а заключительные - •.

Рассмотрим путь e1, ..., ek в источнике. Выпишем последовательно буквы, приписанные ребрам e1, ..., ek. Получившееся слово назовем словом, порожденным данным путем. Если все ребра пути пустые, то такой путь порождает пустое слово.

Каждому источнику ставится в соответствие язык L ⊂ A∗ следующим образом. Для каждого пути из некоторой начальной вершины в некоторую заключительную выписывается порожденное им слово. Все такие слова, и только они составляют язык L. Говорят, что источник *порождает* язык L.

Пример.



L = a2b∗ ∨ abb∗ ∨ b∗

Чтобы проверить, что данный источник порождает именно этот язык, нужно рассмотреть все пути, ведущие из начальной вершины в заключительную.

***Первая теорема Клини.***

*Каждый язык, порождаемый источником, является регулярным.*

Для доказательства будет полезна лемма об источниках.

Пусть вершины источника пронумерованы. И пусть Rkij обозначает

множество всех слов, порожденных путями в данном источнике из вер-

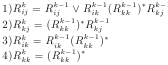
шины с номером i в вершину с номером j, не проходящими вершину с

номером больше k. Следующая лемма иллюстрирует "понижение степе-

ни" Rkij и позволяет таким образом перейти к простейшим языкам Ri0j.

Лемма.

Выполнены следующие равенства.



Доказательство леммы.

Докажем первое равенство. Для этого рассмотрим множество путей, ведущих из вершины i в вершину j и не проходящих вершину с номером большим, чем k ( но саму вершину k проходить можно).

Таким образом, все пути, слова на которых составляют Rkij, могут либо вовсе не проходить вершину k, либо проходить ее некоторое количество раз. В первом случае слова на всех таких путях по определению составляют язык Rkij−1.

Во втором случае каждый такой путь можно поделить на части: путь до вершины k, несколько путей, выходящих из k и возвращающихся в нее, и путь из k в j, причем во всех этих путях вершина k может встречаться только в начале и в конце. Таким образом, множество слов, порожденное этими путями, можно описать выражением Rikk−1(Rkkk−1)∗Rkkj−1. Поэтому Rkij = Rkij −1 ∨ Rikk −1(Rkkk −1)∗ Rkkj −1. Первое равенство доказано.

Пункты 2) − 4) следуют из 1). Выведем пункт 2).

Пусть i = k. Тогда Rkkj = Rkkj −1 ∨ Rkkk−1(Rkkk −1)∗ Rkkj −1 = (Λ ∨ Rkkk −1(Rkkk −1))∗ Rkkj −1 = (Rkkk −1)∗) Rkkj −1.

Остальные пункты выводятся аналогично.

**Доказательство теоремы.**

Составим регулярное выражение для языка, порожденного произвольным источником.

Итак, рассмотрим источник И с n вершинами. Некоторым образом

перенумеруем его вершины. Множество начальных вершин обозначим I,

а множество заключительных ꂈ F.

Очевидно, что вырабатываемое им множество слов есть и , это слова, порожденные путями от начальных вершин к заключительным, которые не проходят вершины с номерами, большими, чем количество вершин. Далее, для каждого Rnlk применяем лемму до тех пор, пока в ней не будут участвовать лишь R0ij, то есть слова, соответствующие множествам путей из вершины i в вершину j, не заходящих ни в какую другую вершину. Можно выписать конкретные выражения для каждого R0ij следующим образом.

R0ij = ∅, если нет ребер, ведущих из вершины i в вершину j, причем i ꁘ= j.

R0ij = a1 ∨ ... ∨ ak, если из i в j ведут ребра с буквами a1, ..., ak. Если от i к j ведет еще и пустое ребро, то в объединение добавляется пустое слово Λ.

R0ij = Λ, если есть только пустое ребро. Множество Ri0i также всегда содержит пустое слово Λ.

Ясно, что языки R0ij регулярны. Видно, что все языки Rkij получаются из них с помощью операций объединения, конкатенации и итерации. Следовательно, все языки Rkij регулярны, поэтому регулярен и язык  .

Теорема доказана.

Применяя лемму, можно также формализовать процесс определения

языка, вырабатываемого источником.

Пример.



 Источник называется двухполюсником, если в нем ровно одна начальная вершина и ровно одна заключительная, причем они не совпадают, ни одно ребро не входит в начальную вершину, а из заключительной вершины не выходит ни одно ребро.

Утверждение. Для любого источника существует эквивалентный ему двухполюсник.

Доказательство.

В данном источнике все начальные и заключительные вершины сделаем обыкновенными и введем дополнительные вершины q0 и qf. Из q0 проведем пустые ребра в бывшие начальные, а из бывших заключительных проведем пустые ребра в qf. Получившийся источник - двухполюсник, эквивалентный данному.

***Вторая теорема Клини***.

*Любой регулярный язык порождается некоторым источником*.

Доказательство.

Построим для регулярного языка источник.

Для простейших ∅,Λ,a соответствующие источники выглядят так:



Λ ∗y

a ∗ E y

a ∅ ∗ ' y

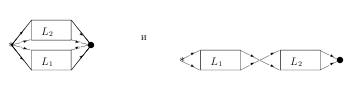
a

Далее, пусть построены источники для регулярных языков L1 и L2.

Можно считать их двухполюсниками.



Тогда для языков L1 ∨ L2 и L1L2 источники будут:



А для языка L1∗ 

Так можно построить источник для любого регулярного языка.

Теорема доказана.

## Грамматики

*Утверждение.*

Задаваемый грамматикой язык порождается источником, и наоборот.

*Доказательство.*

1. Переход от грамматики к источнику.

Пусть дана грамматика G:

T = {a, b, ...}, N = {A, B, ...}, I ∈ N и некоторый набор правил вида

*A → aB, A → a, I → b.*

Построим источник, порождающий тот же язык, что и грамматика.

Каждой вспомогательной букве ставится в соответствие вершина источника, причем I соответствует начальной вершине *q0*. Отдельно вводится заключительная вершина *qf*.

Каждому правилу вида *A → aB* ставится в соответствие ребро с буквой a из вершины A в вершину B, а правилу *A → a* - ребро с *a* из *A* в заключительную вершину *qf*. Если правило имеет вид *A → B* или *A → Λ,* то ребра соответственно в B и в *qf* проводятся пустые.

2. Переход от источника к грамматике.

Пусть L - язык, порожденный источником И. Построим грамматику, порождающую L.

Начальной вершине поставим в соответствие аксиому I, остальным вершинам *v1, v2, ...* - различные буквы *A1, A2*, ..., которые будут составлять нетерминальный алфавит.

Каждому ребру вида  ставится в соответствие правило вывода *Ai → aAj (Ai → Aj*, если это пустое ребро*)*. Если vj - заключительная вершина, то добавляем еще и правило *Ai → a (Ai → Λ*, если ребро пустое*)*. Если начальная вершина является заключительной, то добавляем правило *I → Λ*.

Утверждение доказано.

## Упрощение грамматик.

Рассмотрим грамматику с алфавитами T = {a,b},N = {I,A,B}, аксиомой I и следующими правилами вывода:

*I → a, I → aA, I → bI, A → aA, A → bA, B → a.*

Видно, что в данной грамматике нетерминальный символ B никогда не появится в слове, выведенном из аксиомы, а из символа A нельзя вывести слово только из терминальных символов. Значит, эти символы и содержащие их правила не влияют на язык, порожденный грамматикой. Поэтому данная грамматика порождает тот же язык, что и грамматика со вспомогательным алфавитом N = {I} и двумя правилами вывода:

*I → a,I → bI.*

Таким образом, можно упростить грамматику путем удаления недостижимых и бесполезных символов. Формализуем вышесказанное.

Символ *B ∈ N* назовем недостижимым, если не существует такого выводимого из аксиомы I слова β, что *β = α1Bα2*, где *α1,α2 ∈ (T ∪ N)∗.*

Символ *A ∈ N* назовем бесполезным, если не существует такого слова *α ∈ T∗* из основного алфавита, что α выводится из A.

Рассмотрим аксиому I и правила вида *I → C* и *I → aC*. Символы, встретившиеся в правой части таких правил, отнесем к достижимым символам. Включим в это множество и аксиому. Проделаем то же самое для достижимых символов. Будем рассматривать правила с достижимыми символами в левой части и добавлять символы из правой части ко множеству достижимых символов. На каком-то шаге множество достижимых символов перестанет увеличиваться. Символы вспомогательного алфавита, не вошедшие в это множество - *недостижимые.* Удалим их из грамматики, а также все правила, в которых они встречаются.

Если изобразить источник для грамматики, то недостижимые символы грамматики будут соответствовать вершинам, недостижимым из начальной вершины.

Теперь рассмотрим символы вспомогательного алфавита *A ∈ N*, для которых в грамматике есть правила вида *A → a, a ∈ T* или *A → Λ.*

Отнесем их ко множеству D символов, не являющихся бесполезными. Рассмотрим правила *C → A, C → aA,* где *A ∈ D, a ∈ T*. Добавим символ *C* ко множеству *D*. Будем проделывать этот шаг до тех пор, пока множество *D* не перестанет увеличиваться. Символы вспомогательного алфавита, не вошедшие в *D*, - *бесполезные*. Удалим их из грамматики вместе с содержащими их правилами.

В источнике бесполезным символам грамматики соответствуют вер-

шины, из которых недостижимы никакие заключительные вершины.

**Пример.**

Рассмотрим грамматику с основным алфавитом *T = {a,b},* вспомогательным алфавитом *N = {I,A,B,C,E},* аксиомой *I* и следующими правилами вывода:

*I → aI|bA,*

*A → aI|aB|b,*

*B → aB|bC,*

*C → bC,*

*E → a.*

Упростим эту грамматику. Символы *A, B, C* достижимы, так как из аксиомы выводимы слова *bA, baB, babC*. Других достижимых нетерминальных символов нет, так как ни одно правило, начинающееся с *I, A, B* или *C*, не содержит других символов из N. Оставшийся символ E - недостижимый, удалим его из грамматики.

Теперь найдем бесполезные символы. Образуем множество D из A, так как есть правило A → b, выводящее терминальное слово. Найдем правила, в которых элемент D встречается в правой части. Это правило *I → bA*, поэтому *D = {I,A}*. Дальше это множество не расширяется. Оставшиеся нетерминальные символы *B,C* являются бесполезными.

В итоге в упрощенной грамматике есть лишь два символа вспомогательного алфавита (I и A) и следующие правила:

*I → aI|bA,*

*A → aI|b.*

## Детерминированные источники.

Рассмотрим источник со следующими свойствами:

1) ровно одна начальная вершина;

2) из каждой вершины выходит ровно |A| ребер, и всем этим ребрам приписаны разные буквы алфавита A.

Такой источник называется *детерминированным*.

Для любого источника можно построить детерминированный источник, порождающий тот же язык (эквивалентный детерминированный источник).

Для этого применяется процесс детерминизации источника.

Возьмем произвольный источник И и построим эквивалентный ему детерминированный источник Д следующим способом.

Перенумеруем вершины И; таким образом, множество вершин можно

отождествить с {1, 2, ..., n}.

Вершинам источника Д будут поставлены в соответствие некоторые подмножества (будем называть их массивами) множества вершин исходного источника И. Построим эти массивы и исходящие из них ребра с буквами алфавита A.

В качестве начальной вершины источника Д возьмем массив, состоящий из начальных вершин источника И и вершин, которые достижимы из этих начальных по путям, составленных из пустых ребер (эти пути порождают пустое слово Λ); обозначим этот массив через 10 и поставим ему в соответствие точку 10 на плоскости.

Далее, для любой буквы a ∈ A рассмотрим всевозможные пути, выходящие из некоторой вершины массива 10 и порождающие слово a (если такие существуют). Множество концов таких путей обозначим через 20 и поставим ему в соответствие точку 20 на плоскости. Из вершины 10 проведем ребро с буквой *a* в вершину 20. Если же путей с указанным свойством не существует, то введем вершину (и точку на плоскости) f0, соответствующую пустому множеству вершин источника И, и направим ребро с буквой *a* из 10 в f0.

Это построение проведем для каждой буквы алфавита *A*.

И так далее. Из вершины k0 для каждой *a ∈ A* проводится ребро *a* в вершину l0, образованную номерами вершин, достижимых в И из вершин множества k0 по путям, порождающим слово *a*. Если таких вершин нет, то из k0 проводим ребро с буквой *a* к упомянутой вершине f0. Если k0 - пустое множество, то проводятся петли с каждой буквой входного алфавита.

Заключительными вершинами объявляются массивы {i1, ..., it}, содержащие хотя бы одну заключительную вершину источника И.

Из построения видно, что источник детерминированный. Также он порождает тот же язык, что и И, так как любому слову, приписанному пути в И из некоторой начальной вершины в некоторую заключительную, соответствует путь в Д и наоборот.

**Пример.**

Слева - источник И, справа - процесс его детерминизации.



Детерминированный источник выглядит так:



Детерминированный источник без выделенных заключительных вершин и такой, что каждому ребру сопоставлена еще и буква некоторого "выходного" алфавита, задает по определению инициальный автомат.

# Лекция № 5. Автоматы.

## Определение и способы задания.

Рассмотрим алфавит A - входной алфавит, его элементы – входные буквы, алфавит V - выходной алфавит, его элементы - выходные буквы. Составленные из этих букв слова называются соответственно входными и выходными. Также вводится Q - множество (алфавит) состояний. В дальнейшем буквы из A, V, Q будем обозначать, соответственно, через *a, b, ...;* *x, y, ..* и q0, q1, q2....

Чтобы определить автомат, нужно задать, в каких состояниях автомат будет находиться в зависимости от поступившей последовательности входных букв, и какие выходные буквы будет выдавать.

*Конечным неинициальным* автоматом называется пятерка

= (A, Q, V, φ, f),

где Q×A → Q - функция переходов, f: Q×A → V - функция выходов.

Если выделено начальное состояние q0, автомат (A, Q, V, φ, f, q0) называется *инициальным*.

Если в момент времени t на вход автомату A поступает буква x(t), на выходе получается некоторая буква y(t), и qt означает состояние автомата в момент времени t, то функционирование автомата может быть задано следующей системой уравнений:

Также распространены способы задания в виде диаграммы Мура или таблицы. Диаграмма Мура изображается как ориентированный псевдограф, где вершины - состояния, из каждого состояния выходят ровно |A| ребер, и на каждом из них написана одна из входных букв (разным ребрам - разные буквы). Через запятую пишется выходная буква (выход при данных входной букве и состоянии, из которого выходит ребро). То есть диаграмма Мура выглядит как детерминированный источник без заключительных вершин, но с выходными значениями.

Начальное состояние (для инициального автомата) обозначается звездой.



В таблице отображаются новое состояние, и выходная буква при заданных входной букве и текущем состоянии.



## Пример различных представлений автомата.

Рассмотрим автомат , для которого все три алфавита A, V, Q совпадают с {0,1}. Таким образом, A = ({0, 1}, {0, 1}, {0, 1}, , f).

Функции f, зададим формулами f(q,a) = q, (q,a) = q ⊕ a, где ⊕ есть сложение по модулю 2.

Система уравнений для данного автомата такова:

Изобразим тот же автомат в виде диаграммы и в виде таблицы.



Этот автомат осуществляет задержку входной буквы на один такт.

## Отображение языков, задаваемое автоматом.

Пусть дан инициальный автомат. Возьмем произвольное слово α из A∗, α=a(1)a(2)...a(r). Рассмотрим в диаграмме Мура путь, начинающийся в начальной вершине q0, дальше ведущий по ребру a(1), потом по a(2) и так далее до a(r). Выпишем последовательно буквы выходного алфавита, написанные на этих ребрах: v(1)...v(r) ∈ V ∗.

Получится отображение Φ: A∗ → V ∗, Φ(a(1) ...a(r)) = v(1) ...v(r), в частности, Φ(Λ) = Λ.

**Пример.** 

Два автомата с одними и теми же входными и выходными алфавитами называются *эквивалентными*, если задаваемые ими отображения совпадают.

## Автомат Мура.

Для распознавания языков удобно использовать автомат следующего вида: для каждого состояния qi ∈ Q на всех ребрах, входящих в qi, написана одна и та же буква выходного алфавита V (V -метка).

Диаграмму в этом случае можно изображать проще: состояния изображаются в виде круга, внутрь которого вписана V -метка. На ребрах выходные метки не отображаются.

Например, для автомата из предыдущего примера муровский автомат выглядит так:



***Теорема.***

*Для любого автомата можно построить эквивалентный ему муровский автомат.*

*Доказательство.*

Рассмотрим состояние qi и ребра, в него входящие. Возможны два случая:

1) На всех входящих ребрах V -метки одинаковые. Назовем такое состояние простым. Тогда стираем метки на ребрах и пишем это выходное значение в состоянии. Таким образом, простому состоянию обычного автомата соответствует одно состояние муровского автомата.

2) V -метки разные. Назовем такое состояние сложным. Пусть число различных V -меток, написанных на ребрах, входящих в вершину qi, равно k. Тогда в муровском автомате состоянию qi будет соответствовать ровно k состояний. В каждом из k состояний своя V -метка, и в него ведут ребра, входившие в qi в исходном автомате с этой V -меткой. qi стираем, а ребра, выходившие из него, выводим из каждой новой вершины.

Так преобразуется каждое сложное состояние qi:



В частности, если в сложном состоянии есть петля, расщепление происходит так:



Процедура проделывается для каждого состояния. В итоге получается муровский автомат, эквивалентный данному.

Язык, создаваемый муровским автоматом**.**

Рассмотрим инициальный автомат с выходным алфавитом B = {0,1} и произвольным входным алфавитом A. Построим для него детерминированный источник, взяв тот же ориентированный псевдограф. Состояния муровского автомата, в которых написано 1, назовем *заключительными* вершинами. Начальная вершина - q0 (начальное состояние автомата). Этот источник порождает некоторый язык L.

Итак, любой автомат с выходным алфавитом {0,1} задает детерминированный источник, и ясно, что это соответствие взаимно-однозначно.

Таким образом, указанный автомат задает язык L ⊆ A∗. (Говорят также, что автомат распознает или допускает язык.) Язык, для которого существует распознающий его автомат, называется *автоматным*.

Таким образом, автомат распознает язык L с помощью выделенного символа выходного алфавита: слова из L он преобразует в выходные слова, которые заканчиваются на 1, а слова не из L - в слова, заканчивающиеся на 0.

Язык, распознаваемый рассмотренным выше муровским автоматом - все слова, в которое a входит нечетное число раз.

Переформулируем теоремы Клини следующим образом.

***Теорема Клини*** *(для автоматов)*

*1) Любой регулярный язык является автоматным.*

*2) Любой автоматный язык является регулярным.*

Таким образом, совпадают следующие классы языков:

1) регулярные,

2) автоматные,

3) задаваемые грамматикой.

Причем в каждом случае представление языка не единственно. Но для конечных автоматов существует алгоритм, находящий оптимальный по числу состояний автомат.

# Лекция № 6. Минимизация автоматов

Так как существуют различные автоматы, задающие одинаковые отображения Φ, возникает вопрос, как построить автомат, эквивалентный данному, но с меньшим числом состояний. Процесс минимизации позволяет получить автомат с наименьшим числом состояний.

Состояния qi и qj некоторого автомата называются эквивалентными, если инициальные автоматы с начальными состояниями qi и qj эквивалентны.

Разобьем множество состояний автомата Q на классы эквивалентности (эквивалентные состояния находятся в одном классе, состояния из разных классов неэквивалентны). Каждый класс объявляется состоянием нового автомата . От состояния qi к состоянию qj проводится ребро с буквами a,v, если такие ребра были между состояниями исходного автомата из соответствующих классов.

q ꂘ начальная вершина, если в соответствующем классе содержится

начальная вершина.

Полученный автомат называется минимальным или приведенным.

Разбиение на классы происходит следующим образом.

**Алгоритм минимизации.**

1) qi, qj отнесем к одному классу, если f(qi, a) = f(qj, a) для каждой буквы входного алфавита. Полученный класс назовем классом 1-эквивалентности.

2) Каждый полученный класс разобьем следующим образом:

qs, qt из класса 1-эквивалентности находятся в одном классе 2-эквивалентности, если для каждой буквы входного алфавита a состояния ϕ(qs,a) и ϕ(qt, a) находятся в одном классе 1-эквивалентности для каждого a.

i) qs, qt из одного класса (i − 1)-эквивалентности находятся в одном

классе i-эквивалентности, если для каждого a состояния ϕ(qs, a) и ϕ(qt, a)

находятся в одном классе (i − 1)-эквивалентности.

Когда этот процесс остановится (классы прекращают делиться), получится требуемое разбиение. Легко видеть, что максимальное число шагов данного алгоритма на единицу меньше числа состояний исходного

автомата.

**Пример.**



На первом шаге алгоритма (проверка совпадения выходных символов) выделяются два класса: I = {q0}, II = {q1, q2}. Рассмотрим ϕ(qi, aj)

для второго класса. 

Функции переходов переводят состояния в одинаковые классы, дальше разбиения не происходит. Значит, состояния q1 и q2 эквивалентны.

Минимальный автомат имеет вид:

****

**Пример.**

Задание: по грамматике построить источник, детерминировать его, получить автомат, распознающий тот же язык, минимизировать полученный автомат и записать грамматику для минимального автомата.

I → aI|bI|bA,

A → aB|bC,

B → aB|bI|aI,

C → bC|a|b.

Источник, порождающий тот же язык, что и грамматика.

****

Детерминируем этот источник.



Детерминированный источник имеет вид



Такому источнику можно поставить в соответствие муровский авто-

мат, допускающий тот же язык с помощью выходного символа 1. 

На первом шаге выделяются классы 1-эквивалентности:

I = {1, 2, 3, 5}, II = {4, 6} 

Классы 2-эквивалентности: I = {1, 3, 5}, II = {2}, III = {4, 6}



Минимизация закончена. Получившийся автомат таков:



Порождающая грамматика имеет вид

I → aI|bA,A → aI|bB,B → bB|aI|a|b.

# Лекция № 7. Машины Тьюринга

**Определение 1.** Машина Тьюринга — это семёрка M = ‹Q, Σ, Γ, b0, Δ, I, F›, где Q, Γ и Δ — конечные множества, Σ ⊂ Γ, b0 ∈ Γ − Σ, I ⊆ Q, F ⊆ Q и

Δ ⊆ ((Q − F) × Γ) × (Q × Γ × {−1, 0, 1}).

Здесь Q — множество состояний, Σ — входной алфавит (внешний алфавит),

Γ — ленточный алфавит (tape alphabet), b0 — бланк (пробел, пустой символ) (blank symbol), Δ — множество переходов, I — множество начальных состояний, F — множество заключительных состояний.

**Определение 2.** Конфигурацией машины Тьюринга называется любая четвёрка ‹x, q, a, y›, где x ∈ Γ∗, q ∈ Q, a ∈ Γ, y ∈ Γ∗.

**Определение 3.** Определим на множестве всех конфигураций машины Тьюринга M бинарное отношение ├ M (такт работы) следующим образом.

Если ((p, a), (q, c, 0)) ∈ Δ, то (x, p, a, y) ├ (x, q, c, y) для всех x ∈ Γ∗ и y ∈ Γ∗.

Если ((p, a), (q, c, 1)) ∈ Δ, то (x, p, a, dy) ├ (xc, q, d, y) и (x, p, a, ε) ├ (xc, q, b0, ε для всех x ∈ Γ∗, y ∈ Γ∗ и d ∈ Γ.

Если ((p, a), (q, c,−1)) ∈ Δ, то (xd, p, a, y) ├ (x, q, d, cy) и (ε, p, a, y) ├ (ε, q, b0, cy) для всех x ∈ Γ∗, y ∈ Γ∗ и d ∈ Γ.

**Замечание 4.** Если из контекста ясно, о какой машине Тьюринга идёт речь, вместо ├M будем писать просто ├.

**Определение 5.** Как и для МП-автомата, для машины Тьюринга бинарное отношение ├∗ определяется как рефлексивное, транзитивное замыкание отношения ├.

**Замечание 6.** Если (x1, q1, a1, y1) ├ ∗ (x2, q2, a2, y2), то для любых k ∈ N и l ∈ N найдутся такие m ∈ N и n ∈ N, что (b0kx1, q1, a1, y1b0l ) ├∗ (b0mx2, q2, a2, y2b0n ).

**Замечание 7.** Конфигурацию (b0mx, q, a, yb0n) иногда изображают сокращённо .

**Пример.** Рассмотрим машину Тьюринга M =(Q,Σ, Γ, b0,Δ, I, F), где

Q = 0, 1, 2, 3}, Σ = {a},

Γ = {a, b}, b0 = b, I = {0}, F = {3},

Δ =.

Можно проверить, что для любого k ∈ N выполняется следующее:



**Определение 9.** Машина Тьюринга (Q,Σ, Γ, b0,Δ, I, F) называется детерминированной, если множество I содержит ровно один элемент и для каждой пары \_p, a ∈ (Q − F) × Γ существует не более одной тройки \_q, c, k ∈ Q × Γ × {−1, 0, 1} со свойством \_\_p, a, \_q, c, k ∈ Δ.

**Пример 10 .** Машина Тьюринга из примера 13.8 является детерминированной.

**Определение 11.** Пусть f — частичная функция из Σ∗ в Σ∗. Машина Тьюринга M = (Q,Σ, Γ, b0,Δ, I, F) вычисляет (computes) функцию f тогда и только тогда, когда для каждого слова w ∈ Σ∗

1) если f(w) не определено, то не существует таких p ∈ I,q ∈ F, a ∈ Γ, x ∈ Γ∗, y ∈ Γ∗, что \_ε, p, b0, w ∗\_ \_x, q, a, y,2) если f(w) = z, то для некоторых p ∈ I, q ∈ F, m ∈ N и n ∈ N \_ε, p, b0, w ∗\_ \_bm0

, q, b0, zbn0

.

**Пример 12.** Машина Тьюринга из примера 13.8 вычисляет следующую частичную функцию:



**Определение 14.** Говорят, что детерминированная маши-

на Тьюринга (Q,Σ, Γ, b0,Δ, {qs}, {qa, qr}) с выделенным состоянием qa разрешает (decides) язык L ⊆ Σ∗, если

1. для каждого слова w ∈ L найдутся такие m ∈ N и n ∈ N, что



1. для каждого слова w ∈ Σ∗ − L найдутся такие m ∈ N и n ∈ N, что



Состояние qa называется допускающим (accept state), состояние qr называется отвергающим (reject state).

Эта машина Тьюринга разрешает язык {a3n | n ∈ N}.

**Определение 16.** Язык L над алфавитом Σ называется разрешимым или рекурсивным (decidable, recursive),если существует детерминированная машина Тьюринга (Q,Σ, Γ, b0,Δ, {qs}, {qa, qr}) с выделенным состоянием qa, которая разрешает язык L.

**Определение 17.** Говорят, что машина Тьюринга M == (Q,Σ, Γ, b0,Δ, {qs}, {qa}) допускает (accepts) слово w ∈ Σ∗, если для некоторых 

**Определение 18.** Язык, допускаемый машиной Тьюринга M, — это язык, состоящий из всех допускаемых данной машиной Тьюринга слов.

**Определение 19.** Язык называется перечислимым (или рекурсивно перечислимым, или полуразрешимым) (recursively enumerable), если существует детерминированная машина Тьюринга, допускающая этот язык.

**Замечание 20.** В определении 13.19 можно отбросить требование детерминированности машины Тьюринга.

**Теорема 21.** Каждый разрешимый язык является перечислимым.

***Доказательство***. Пусть дана машина Тьюринга M =(Q,Σ, Γ, b0,Δ, {qs}, {qa, qr}) с выделенным состоянием qa, которая разрешает язык L ⊆ Σ∗. Тогда машина Тьюринга M\_ = (Q,Σ, Γ, b0,Δ, {qs}, {qa} ) допускает язык L.

**Тема: Машина Тьюринга, её состав.**

В данном разделе рассматриваются более мощные (по сравнению с конечными автоматами) модели устройств, выполняющих вычисления, – машины Тьюринга (МТ). Анализ показал, что с помощью машин Тьюринга можно реализовать любой алгоритм. Одновременно МТ служит для уточнения самого понятия алгоритма и его формализации, поскольку, как установлено в теории вычислений, широко применяемые словесные определения алгоритма не являются точными и исчерпывающими.

Рассмотрим, например, следующее определение: Алгоритм – это определенное на некотором языке конечное предписание (способ, рецепт), задающее дискретную  
последовательность исполнимых элементарных операций для решения проблемы. Процесс выполнения предписания состоит из отдельных шагов, на каждом из которых выполняется одна очередная операция.

Как отмечено в, это определение, понятное в интуитивном смысле, не является формальным. Употребленные в нем термины "предписание", "элементарная операция", а также объекты, к которым применяется алгоритм, требуют уточнения, если мы хотим говорить об алгоритмах строго. Алгоритмы в интуитивном смысле не являются математическими объектами, к ним не применимы формальные исследования и доказательства. Так, сравнение двух алгоритмов по эффективности, проверка их эквивалентности и т. д., возможны только на основе их формального представления.

Машина Тьюринга представляет собой *бесконечный* автомат, благодаря бесконечной (потенциально) ленте, разбитой на ячейки. В ячейках записываются символы некоторого алфавита МТ. Имеется также конечный автомат с головкой записи и считывания (ГЗЧ). ГЗЧ *обозревает* одну ячейку ленты в текущий момент дискретного времени.

Функции ГЗЧ: считывание символа из обозреваемой ячейки; запись символа   
в обозреваемую ячейку; передвижение влево или вправо на одну ячейку.

В каждый момент времени МТ описывается следующей пятеркой:

*(qi, sj, δ(qi, sj), λ(qi, sj), d(qi, sj))*,

где

*qi* – состояние МТ в текущий момент времени;

*sj* – обозреваемый символ в текущий момент времени;

*δ* – функция переходов, которая определяет следующее состояние;

*λ* – функция выходов, определяющая запись нового символа в обозреваемую ячейку;

*d* – функция, определяющая передвижение головки влево (L) или вправо (R) на один шаг.

Более краткое обозначение элементов пятерки: (*qi , sj , qij , sij , dij*).

**Тезис Тьюринга:** Любой процесс, который было бы естественно назвать эффективной процедуройможет быть реализован МТ.

Следует подчеркнуть, что *тезис* – это, в общем случае, правдоподобное утверждение, которое не обязательно математически строго доказано. Однако многие научные тезисы действенны, так как прошли проверку временем и практикой.

## Примеры машин Тьюринга для конкретных вычислений.

1. Счетчик четности единиц.



Рис.. МТ для счетчика четности

На рис. показана МТ в начальном состоянии *q0* , при этом обозревается первый символ двоичной последовательности, которая заканчивается ограничителем *B*. Далее на каждом такте ГЗЧ продвигается вправо, заменяя все символы символом "0", причем смена   
состояний (*q0* на *q1* и обратно) происходит всякий раз, когда ГЗЧ обнаружит единицу. Таким образом, состояние *q0* связано с четным числом обнаруженных единиц, а состояние *q1* – с нечетным. Останов МТ (переход в заключительное состояние *H* ) происходит по достижении символа *B*, при этом на его место записывается "0", если число единиц   
в исходной последовательности было четным, и "1", если это число было нечетным.   
После завершения процесса вычислений ГЗЧ будет указывать на эту ячейку с заключительной информацией, а во всех остальных ячейках ленты будут записаны нули.

Представим работу МТ двумя способами: таблицей и графом.

Строки таблицы содержат все "пятерки", описывающие функционирование МТ.

* + - 1. **Таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **qi** | **sj** | **qij** | **sij** | **dij** |
| q0 | 0 | q0 | 0 | R |
| q0 | 1 | q1 | 0 | R |
| q0 | В | Н | 0 | – |
| q1 | 0 | q1 | 0 | R |
| q1 | 1 | q0 | 0 | R |
| q1 | В | Н | 1 | – |



Рис. Граф счетчика четности единиц

В вершинах графа явно указан символ, обозначающий движение ГЗЧ вправо в каждом из двух состояний. Заключительное состояние обозначено на рисунке буквой *H* – "halt" (останов).

2. Машина Тьюринга для проверки правильности скобочных выражений.

Рекурсивное определение правильного скобочного выражения (ПСВ) было дано   
в разделе 1.3. Заметим, что конечный автомат не может решить поставленную задачу для скобочного выражения произвольной длины. МТ решает задачу в общем случае, так как наличие неограниченной ленты эквивалентно неограниченному объему памяти.



Рис. 17. МТ для проверки скобочных выражений

Скобочное выражение заключено между левым и правым ограничителями, обозначенными символом *А*. В начальном состоянии автомата ГЗЧ обозревает первый символ скобочного выражения (рис. 17). Реализация вычислений представлена графом (рис. 18).



Рис. 18. Граф МТ для проверки скобочных выражений

*Работа машины:* сначала ГЗЧ движется вправо до первой правой скобки, заменяет ее символом *X*, переходит в состояние *q1* и движется влево до ближайшей левой скобки, заменяет ее символом *X*, переходит в состояние *q0* , и челночное движение повторяется.

Если машина, находясь в состоянии *q1*, достигает левого символа *А*, то печатает "0" и останавливается – скобочное выражение неправильно.

Если машина, находясь в состоянии *q0*, достигает правого символа *А*, не обнаружив больше правой скобки, то переходит в заключительное состояние *q2*, связанное с движением влево, и в последний раз просматривает последовательность: не осталась ли непарная левая скобка. Если по пути встретится левая скобка, то машина печатает "0" и останавливается. При достижении в состоянии *q2* левого символа *А*, машина печатает "1" и останавливается – скобочное выражение правильно.

3.Машина Тьюринга для умножения двух чисел в унарном коде.

Действие машины представлено рисунками 19 – 21.



Рис. 19. МТ для умножения – исходное состояние

Рис. 20. МТ для умножения – результат операции



Рис. 21. Граф МТ – множительного устройства

На рис. 21 граф представлен в сокращенном виде – не показаны петли с одинаковыми числителем и знаменателем.

МТ представляет собой простой базис для описания *эффективных процедур*,   
поскольку все шаги, регламентирующие поведение автомата, определяются четкими правилами. В современной терминологии, МТ – простой базис для описания алгоритмов и уточнения самого понятия алгоритма.

В приведенных выше примерах для каждого вычисления использовался свой специальный конечный автомат – так называемая *конкретная* машина Тьюринга. Конечный автомат в конкретных МТ играет роль алгоритма вычислений.

Можно считать, что любая МТ вычисляет некоторую функцию (заключительное содержимое ленты) от заданного аргумента (исходное содержимое ленты). В связи   
с этим возникло понятие: *функция, вычислимая по Тьюрингу*.

Тьюринг показал, как построить *универсальную* машину Тьюринга (УМТ), которая интерпретирует поведение любой конкретной МТ и, следовательно, может вычислить любую функцию, которую вычисляет конкретная МТ.

УМТ имеет структуру, показанную на рис. 22.



Рис. 22. Структура УМТ

Левая часть ленты (до символа *qi*) имитирует ленту конкретной МТ, правая – содержит описание автомата конкретной МТ в виде пятерок.

Символ *М* показывает расположение ГЗЧ конкретной МТ; будем считать, что обозреваемый символ находится справа от символа *М*.

Пара ячеек *q*i, *s*j – зона режима, указывающая текущее состояние и текущий обозреваемый символ (на рисунке это символ "0").

*Работа УМТ:* УМТ начинает работу с запоминания символов *qi, sj* зоны режима, затем ГЗЧ движется вправо до тех пор, пока не найдет пятерку, в которой первые два символа совпадают с символами зоны режима. Найдя такую пятерку, УМТ запоминает *qij, sij*, *dij* , и ГЗЧ движется влево. Далее выполняются следующие действия:

1. *qi* : = *qij*, т. е. в зону режима записывается символ, обозначающий новое состояние автомата конкретной МТ;
2. справа от *М* записывается *sij* ;
3. выполняется сдвиг *М* согласно значению *dij* (*L* или *R*), что соответствует передвижению ГЗЧ имитируемой машины;
4. запоминается новый обозреваемый символ (справа от *М*) и переносится   
   в зону режима в качестве *s*j .

Таким образом, зона режима оказывается обновленной, и указанные действия повторяются до останова машины.

Вариант формулировки тезиса Тьюринга**:** *Вычислимы те, и только те, объекты, которые могут быть вычислены УМТ*.

Согласно Тьюрингу, алгоритмом можно считать только ту процедуру, закодированную для МТ, которая приводит к останову машины.

В связи с этим возникает вопрос: существует ли универсальный распознаватель алгоритмов, т. е. МТ, которая для любой другой МТ определит, остановится последняя или нет. Этот вопрос назван в теории машин Тьюринга *проблемой останова*. Доказано, что проблема останова неразрешима и, следовательно, универсального распознавателя алгоритмов не существует [1]. Существуют и другие *алгоритмически неразрешимые* проблемы, к доказательству неразрешимости которых привлекается механизм вычислений, введенный Тьюрингом.

Простые системы (базисы) для решения проблем вычислимости создали независимо друг от друга и другие выдающиеся исследователи в области теории алгоритмов: А. Черч (математический аппарат рекурсивных функций), А. А. Марков (*нормальные алгоритмы*, сводящиеся к преобразованию слов в некотором алфавите), Э. Пост (механизм преобразования двоичных последовательностей, подобный МТ). Все названные системы равноценны с точки зрения их принципиальных возможностей и эквивалентны как формализмы для определения алгоритма.