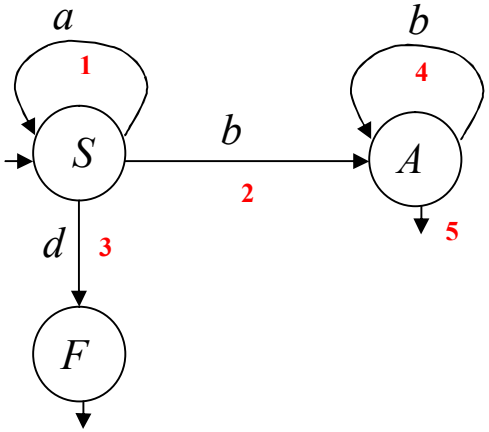
Алгоритм построения НКА по праволинейной автоматной грамматике

1. Множество вершин НКА состоит из нетерминалов грамматики и, возможно, еще одной новой вершины F, которая объявляется заключительной.
2. Каждому правилу вида $A \to aB$ в автомате соответствует дуга из вершины A в вершину B, помеченная символом a: $A \xrightarrow{a} B$. Каждому правилу вида $A \to a$ соответствует дуга $A \xrightarrow{a} F$. Других дуг нет.
3. Начальной вершиной автомата является вершина, соответствующая начальному символу грамматики. Заключительными являются новая вершина F, если она использовалась на шаге 2, и каждая вершина A, такая что для нетерминала A в грамматике есть правило $A \to \varepsilon$.

Пример (НКА по праволинейной гр-ке)

1. $S \to aS$
2. $S \to bA$
3. $S \to d$
4. $A \to bA$
5. $A \to \varepsilon$



$S \to_1 aS \to_2 abA \to_4 abbA \to_5 abb$ – вывод в грамматике для abb

$\to S \overset{a}{\to}_1 S \overset{a}{\to}_2 A \overset{b}{\to}_4 A \to_5$ – соответствующий путь в автомате для abb

Алгоритм построения праволинейной автоматной грамматики по НКА с единственной начальной вершиной

1. Нетерминалами грамматики будут вершины автомата, терминалами — пометки дуг.
2. Для каждой дуги $A \xrightarrow{a} B$ в грамматику добавляется правило $A \to aB$. Для каждой заключительной вершины B в грамматику добавляется правило $B \to \varepsilon$.
3. Начальным символом будет нетерминал, соответствующий начальной вершине.

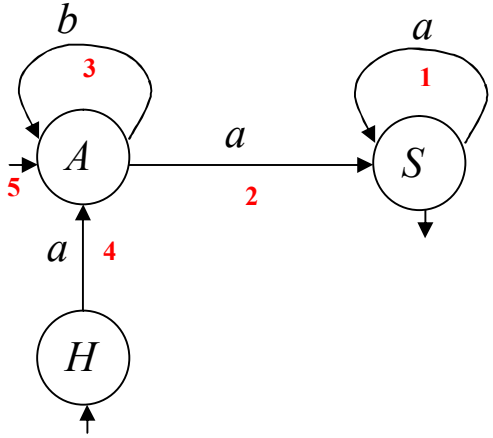
(4.) К построенной по пунктам 1—3 грамматике можно применить алгоритм устранения ε-правил.

Алгоритм построения НКА по леволинейной автоматной грамматике

1. Множество вершин НКА состоит из нетерминалов грамматики и, возможно, еще одной новой вершины H, которая объявляется начальной.
2. Каждому правилу вида $A \to Ba$ в автомате соответствует дуга из вершины B в вершину A, помеченная символом a: $B \xrightarrow{a} A$. Каждому правилу вида $A \to a$ соответствует дуга $H \xrightarrow{a} A$. Других дуг нет.
3. Заключительной вершиной автомата является вершина, соответствующая начальному символу грамматики. Начальными являются вершина H, если она использовалась на шаге 2, и каждая вершина A, такая что для нетерминала A в грамматике есть правило $A \to \varepsilon$.

Пример (НКА по леволинейной гр-ке)

1. $S \to Sa$
2. $S \to Aa$
3. $A \to Ab$
4. $A \to a$
5. $A \to \varepsilon$



$abb \leftarrow_4 Abb \leftarrow_3 Ab \leftarrow_2 S$ – обращение вывода в грамматике для abb

(свертки)

$\to H \xrightarrow{a}_4 A \xrightarrow{b}_3 A \xrightarrow{b}_2 S \to$ – соответствующий путь в автомате для abb

(моделирует свертки)

Алгоритм построения леволинейной автоматной грамматики по НКА с единственным заключительным состоянием

1. Нетерминалами грамматики будут вершины автомата, терминалами — пометки дуг.
2. Для каждой дуги $A \xrightarrow{a} B$ в грамматику добавляется правило $B \to Aa$. Для каждой начальной вершины B в грамматику добавляется правило $B \to \varepsilon$.
3. Начальным символом будет нетерминал, соответствующий заключительной вершине.

(4.) К построенной по пунктам 1—3 грамматике можно применить алгоритм устранения ε-правил.

$G = (\{a, b, \perp\}, \{S, A, B, C\}, P, S)$, где

P: $S \to C\perp$

$C \to Ab \mid Ba$

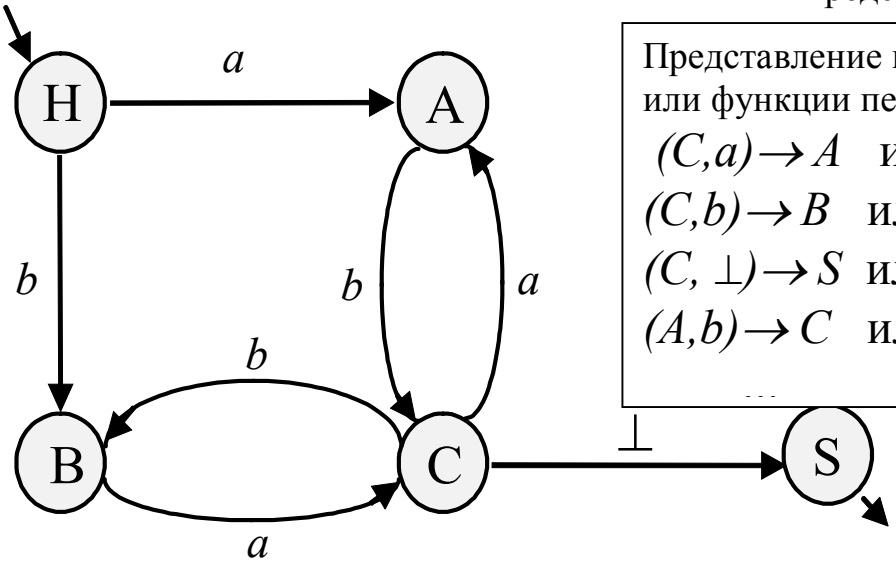
$A \to a \mid Ca$

$B \to b \mid Cb$

Диаграмма состояний для грамматики G – это граф, представляющий конечный автомат, построенный нашим алгоритмом по грамматике G.

	a	b	$\perp$
H	A	B	-
C	A	B	S
A	-	C	-
B	C	-	-
S	-	-	-

Представление в виде таблицы



Представление в виде набора команд или функции переходов :

$(C,a) \to A$ или $\delta(C,a) = \{A\}$

$(C,b) \to B$ или $\delta(C,b) = \{B\}$

$(C, \perp) \to S$ или $\delta(C, \perp) = \{S\}$

$(A,b) \to C$ или $\delta(A,B) = \{C\}$

...

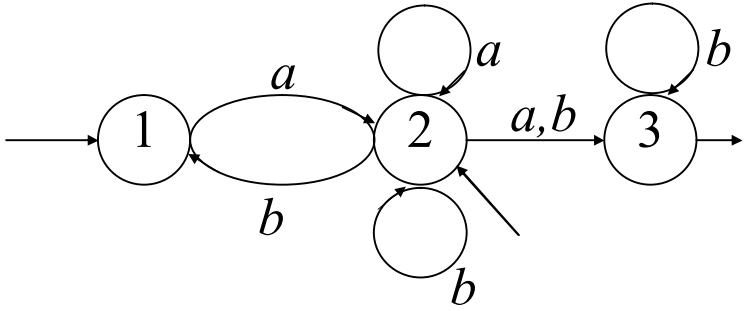
Конечный автомат называется **детерминированным** конечным автоматом (ДКА), если он имеет единственное начальное состояние, и любые две дуги, исходящие из одной и той же вершины имеют различные пометки.

Множество δ в ДКА можно интерпретировать как отображение $K \times \Sigma$ в множество K.

Тогда конечный автомат ***допускает цепочку*** $a_1a_2...a_n$, если $\delta(H,a_1) = A_1$; $\delta(A_1,a_2) = A_2$; ... ; $\delta(A_{n-2},a_{n-1}) = A_{n-1}$; $\delta(A_{n-1},a_n) = S$, где $a_i \in \Sigma$, $A_j \in K$, $j = 1, 2,..., n-1$; $i = 1,2,..., n$; H – начальное состояние, S – одно из заключительных состояний.

Язык, допускаемый ДКА — это множество всех допускаемых им цепочек.

Пример. $\mathcal{A}'$:



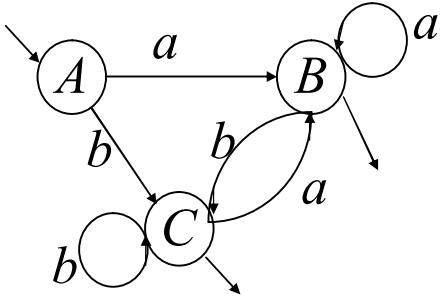
Процесс построения ДКА по заданному НКА удобно изобразить в виде таблицы, начав с состояния {1, 2}. Затем заполняем строки для вновь появляющихся состояний.

символ / состояние	a	b
{1, 2}	{2, 3}	{1, 2, 3}
{2, 3}	{2, 3}	{1, 2, 3}
{1, 2, 3}	{2, 3}	{1, 2, 3}

Обозначим состояние $\{1, 2\}$ через A, $\{2, 3\}$ — B, $\{1, 2, 3\}$ — C.

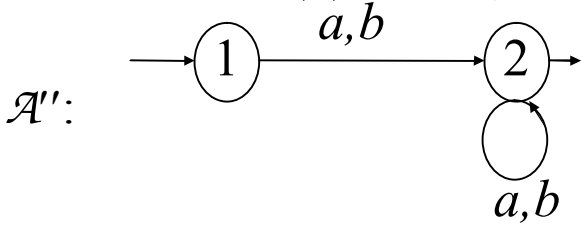
символ / состояние	a	b
A	B	C
B	B	C
C	B	C

С учетом переобозначений построим по таблице ДКА $\mathcal{A}$:



$$L(\mathcal{A})=\{a, b\}^{+}$$

Можно заметить, что язык $L = \{a, b\}^+$, допускаемый автоматом $\mathcal{A}$, допускается также ДКА $\mathcal{A}''$, имеющим только два состояния.



Существует алгоритм, позволяющий по любому ДКА построить эквивалентный ДКА с минимальным числом состояний.

Алгоритм построения ДКА по НКА

Вход: $\mathcal{A}' = (K', \Sigma, \delta', I, F)$ — НКА.

Выход: $\mathcal{A} = (K, \Sigma, \delta, InitState, FinalStates)$ — ДКА.

Метод: Вершинами (состояниями) автомата $\mathcal{A}$ будут подмножества множества K' автомата $\mathcal{A}'$. *CurState* и *NewState* — вспомогательные переменные для хранения таких подмножеств. Сам алгоритм запишем в паскалеподобном стиле. Фигурные скобки означают конструкторы множеств.

begin $InitState := \{s \mid s \in I\}$; $K := \{InitState\}$; $\delta := \varnothing$;
  while (*в K есть нерассмотренный элемент*)
  begin
    CurState := *нерассмотренный элемент из K*;
    for (*каждого* $a \in \Sigma$)
    begin
      $NewState := \{q \mid (p \xrightarrow{a} q) \in \delta, p \in CurState\}$;
      $K := K \cup \{NewState\}$;
      $\delta := \delta \cup \{(CurState \xrightarrow{a} NewState)\}$;
    end
  end;
  $FinalStates := \{P \in K \mid$ *существует* $q \in P: q \in F\}$
end.

Для более удобной работы с диаграммами состояний введем несколько соглашений:

a) если из одного состояния в другое выходит несколько дуг, помеченных разными символами, то будем изображать одну дугу, помеченную всеми этими символами;

b) непомеченная дуга будет соответствовать переходу при любом символе, кроме тех, которыми помечены другие дуги, выходящие из этого состояния.

c) введем состояние ошибки (ERR); переход в это состояние будет означать, что исходная цепочка языку не принадлежит.

Алгоритм моделирования работы ДКА

Вход: ДКА $\mathcal{A} = (K, \Sigma, \delta, I, F)$ и цепочка $x\perp$, где $x \in \Sigma^*$, $\perp \notin \Sigma$ — маркер конца цепочки.

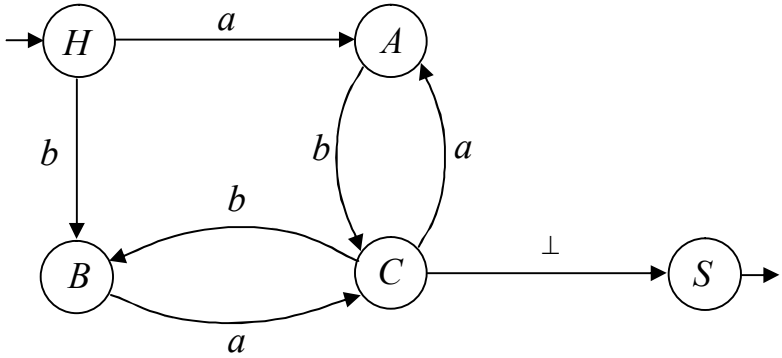
Выход: «Да», если $x \in L(\mathcal{A})$, иначе — «Нет».

Метод: Введем переменные St для хранения текущего состояния автомата и c для хранения очередного считанного символа входной цепочки x.

```
begin
    c := первый символ цепочки x;
    St := I; {начальное состояние}
    while ( St ≠ ERR and c ≠ '⊥' )
      begin
          St := δ (St, c);
          c := очередной символ
      end;
    if St ∈ F then
          write ('Да')
    else
          write ('Нет')
end;
```

Пример анализатора для грамматики
$G = \langle \{a, b, \perp\}, \{S, A, B, C\}, P, S \rangle$, где

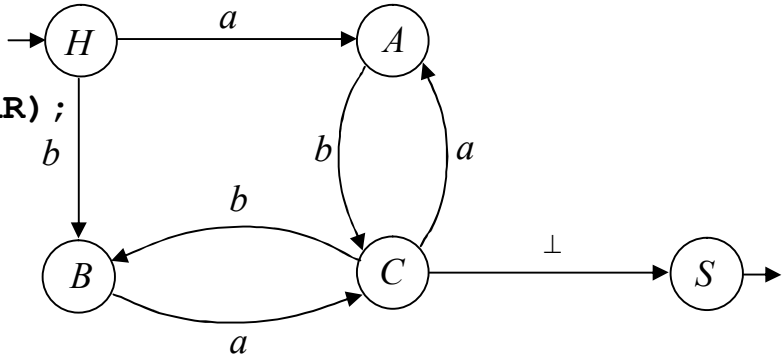
P: $S \to C\perp$
$C \to Ab \mid Ba$
$A \to a \mid Ca$
$B \to b \mid Cb$



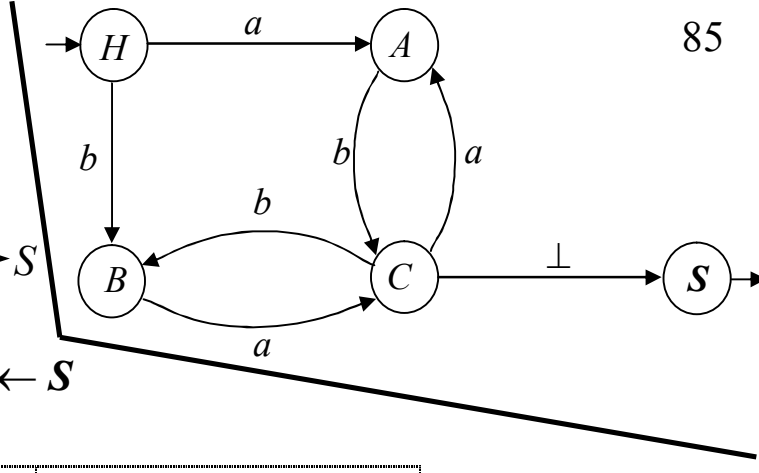
Программа-анализатор на C++ :

```
#include <iostream>
char c; //текущий символ

void gc () {std::cin >> c;} // считать очередной символ со входа

bool scan_G(){

enum state {H, A, B, C, S, ERR};  //множество состояний

state CS;  // CS — текущее состояние

CS=H;  // начинаем с начального состояния H

gc(); // считываем первый символ
```

```
do {switch (CS) {
   case H: if (c == 'a') { gc(); CS = A;}
           else if (c == 'b') { gc(); CS = B;}
                else CS = ERR;
           break;
   case A: if (c == 'b') { gc(); CS = C;}
           else CS = ERR;
           break;
   case B: if (c == 'a') { gc(); CS = C;}
           else CS = ERR;
           break;
   case C: i (c =='a') { gc(); CS = A;}
           else if (c == 'b') { gc(); CS = B;}
                else if (c == '⊥') CS = S;
                     else CS = ERR;
           break;
                }
  } while (CS != S && CS != ERR);
if (CS == ERR)
  return false;
else
  return true;

}
```

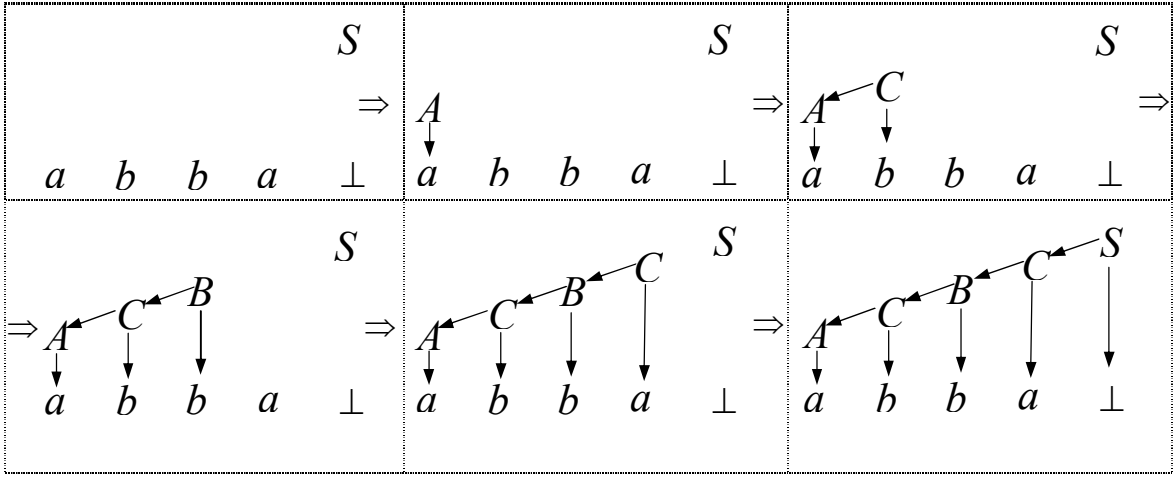


Пример разбора цепочки $abba\bot$



$H \xrightarrow{a} A \xrightarrow{b} C \xrightarrow{b} B \xrightarrow{a} C \xrightarrow{\bot} S$

$abba\bot \leftarrow Abba\bot \leftarrow Cba\bot \leftarrow Ba\bot \leftarrow \underline{C\bot} \leftarrow \boldsymbol{S}$



Недетерминированный разбор

Если конечный автомат, построенный по грамматике, недетерминированный, то нужно перебирать все возможные варианты переходов. Можно также преобразовать его в эквивалентный ДКА и проводить детерминированный разбор.

Пример использования автоматов в решении теоретических задач

Утверждение. *Контекстно-свободный язык*

$$L = \{a^n b^n \mid n \geq 1\}$$

нерегулярен

Вопросы и задачи

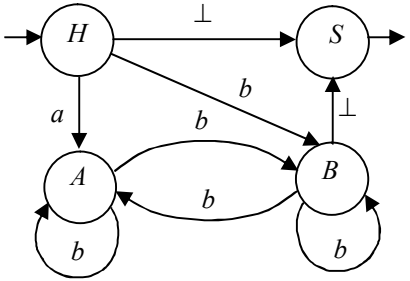
1. Построить праволинейную автоматную грамматику для языка
   $\{c^k \mid k \geq 0\} \cup \{cca(bcb)^m c \mid m \geq 0\} \cup \{ccb\}$

2. Построить НКА по праволинейной грамматике из задачи 1.

3. Построить леволинейную автоматную грамматику для языка $\{a^{2m}b^{3k} \mid m \geq 0\ k \geq 1\}$.

4. Построить НКА по леволинейной грамматике из задачи 3. Удовлетворяет ли построенный автомат определению ДКА?

5. Построить леволинейную и праволинейную грамматики по автомату:



6. Построить ДКА, эквивалентный НКА из задачи 5.

Еще задачи на стр. 101-102 методического пособия
http://cmcmsu.no-ip.info/download formal.grammars.and.languages.2009.pdf