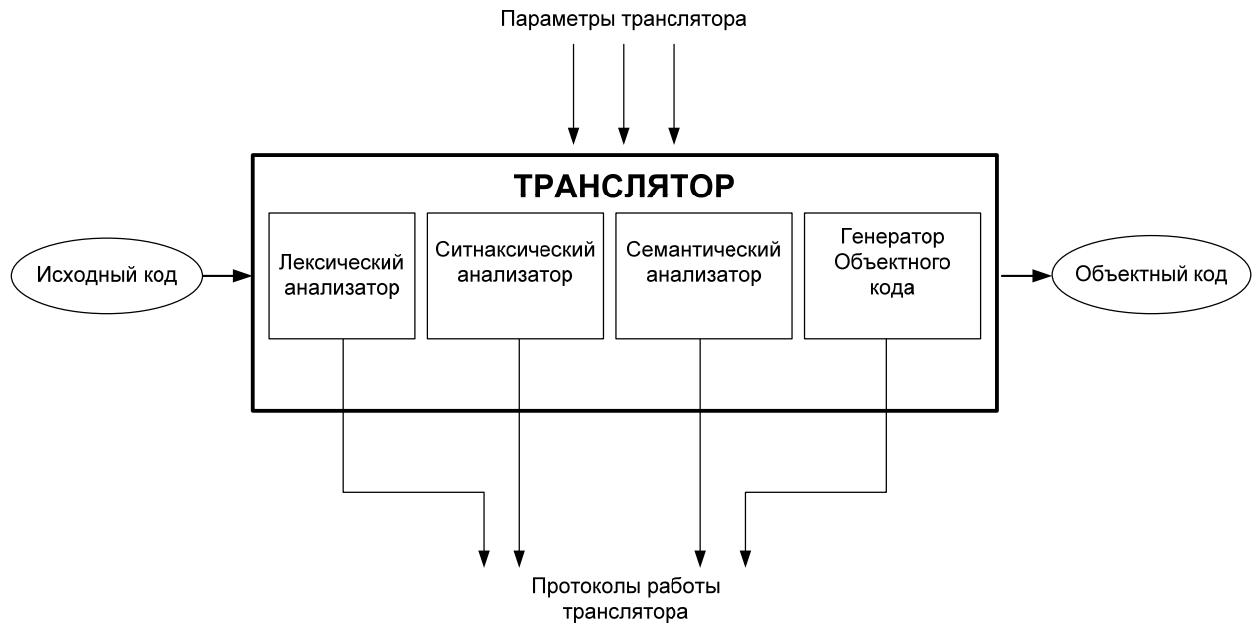
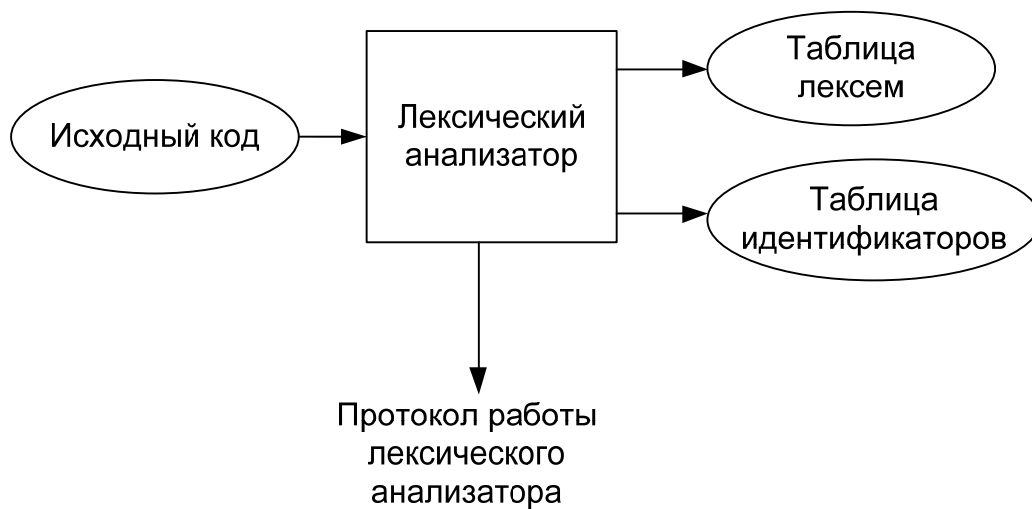


Лексический анализ

1. Структура транслятора:



2. Лексический анализ – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).

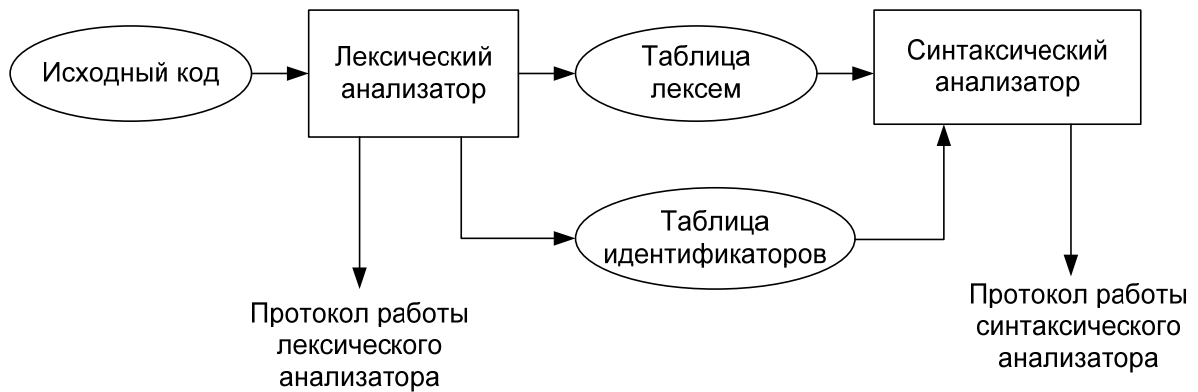


3. Взаимодействие лексического и синтаксического анализаторов:

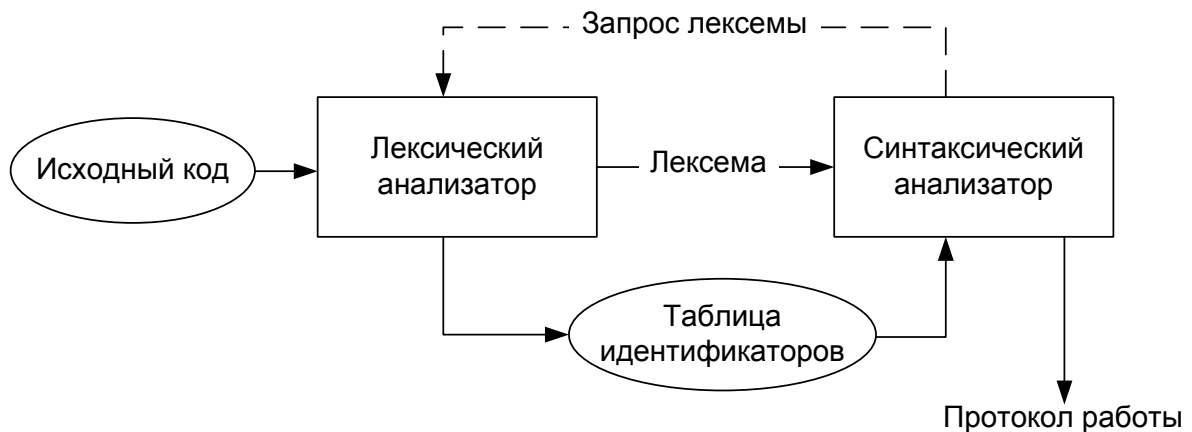
последовательное

параллельное.

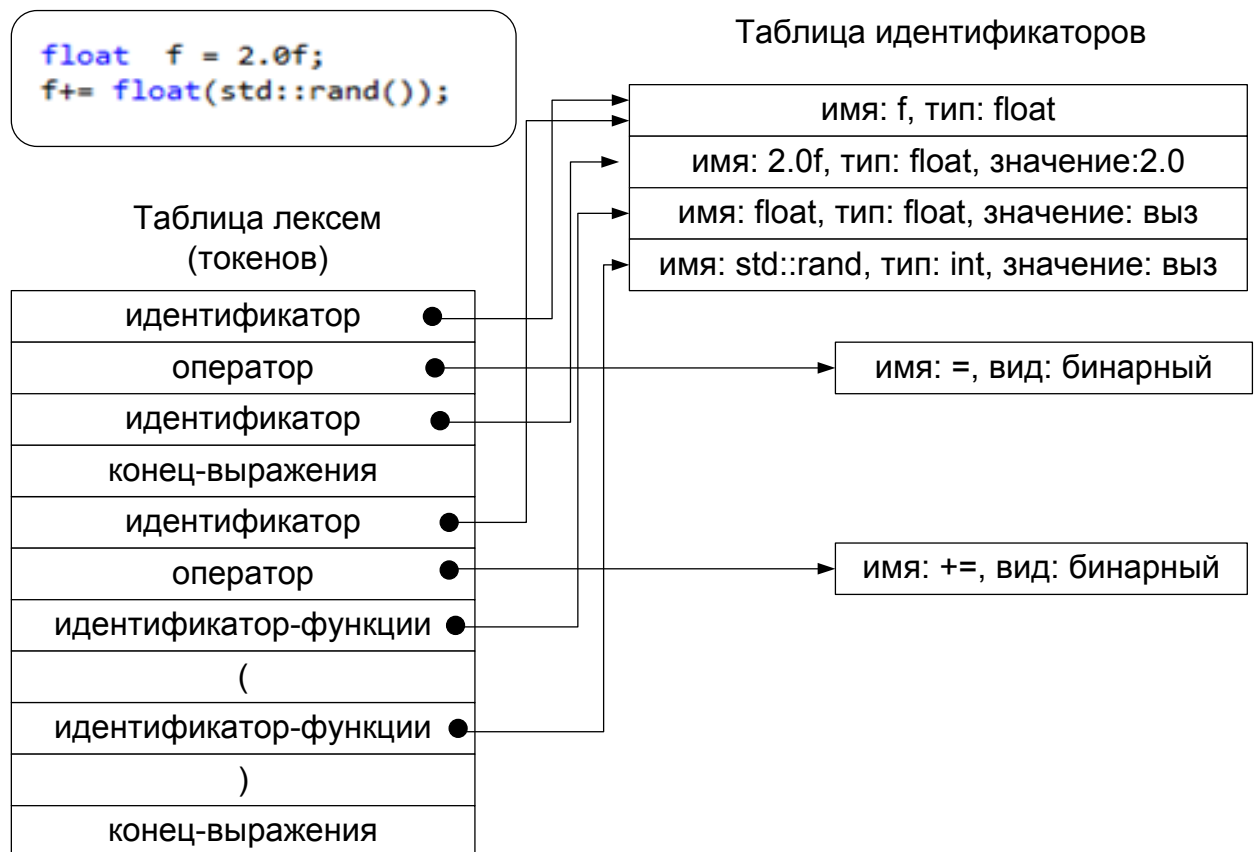
а) Последовательное взаимодействие лексического и синтаксического анализаторов.



б) Параллельное взаимодействие лексического и синтаксического анализаторов.



4. Пример:



Взаимодействие лексического и синтаксического анализаторов C, C++, Java последовательное.

5. Примеры:

а) грамматика $G = (T, N, P, S')$ для цепочки **float id**

$T = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9, \omega, ;, =\}, N = \{S', A', B', C', D'\},$

$S' \rightarrow floatA'$

$A' \rightarrow \omega B'$

$B' \rightarrow \omega B'$

$B' \rightarrow A \mid \dots \mid Z \mid a \mid \dots \mid z \mid AC' \mid \dots \mid ZC' \mid aC' \mid \dots \mid zC'$

$B' \rightarrow A \mid \dots \mid Z \mid a \mid \dots \mid z \mid AD' \mid \dots \mid ZD' \mid aD' \mid \dots \mid zD'$

$C' \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9 \mid 0C' \mid 1C' \mid 2C' \mid \dots \mid 9C'$

$C' \rightarrow A \mid \dots \mid Z \mid a \mid \dots \mid z \mid AC' \mid \dots \mid ZC' \mid aC' \mid \dots \mid zC'$

$C' \rightarrow 0D' \mid 1D' \mid 2D' \mid \dots \mid 9D'$

$C' \rightarrow AD' \mid \dots \mid ZD' \mid aD' \mid \dots \mid zD'$

$D' \rightarrow \omega D'$

$D' \rightarrow = \mid ;$

ω – пробел

$G = (T, N, P, S')$ - регулярная **праволинейная** грамматика

б) **Пример:** вывод (распознавание) цепочки **float f;**

Вывод цепочки	Дерево нисходящего разбора цепочки
$S' \rightarrow floatA'$ $A' \rightarrow \omega B'$ $B' \rightarrow fD'$ $D' \rightarrow ;$	<pre> graph TD S_prime((S')) --> float((float)) S_prime --> A_prime((A')) A_prime --> omega((ω)) A_prime --> B_prime((B')) B_prime --> f((f)) B_prime --> D_prime((D')) D_prime --> semicolon((;)) </pre>

6. Примеры: правосторонний вывод.

Вывод называется **правосторонним**, если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему правому нетерминальному символу в цепочке.

а) Пример: вывод (распознавание) цепочки **float f2x =**

Вывод цепочки	Дерево нисходящего разбора цепочки
$S' \rightarrow float A'$ $A' \rightarrow \omega B'$ $B' \rightarrow f C'$ $C' \rightarrow 2 C'$ $C' \rightarrow x D'$ $D' \rightarrow \omega D'$ $D' \rightarrow =$	<pre> graph TD S'((S')) --> float((float)) S' --> A'((A')) A' --> omega1((ω)) A' --> B'((B')) B' --> f((f)) B' --> C1((C')) C1 --> 2((2)) C1 --> C2((C')) C2 --> x((x)) C2 --> D1((D')) D1 --> omega2((ω)) D1 --> D2((D')) D2 --> eq((=)) </pre>

б) Пример: вывод (распознавание) цепочки **float 22x =**

$S' \rightarrow float A'$

$A' \rightarrow \omega B'$

! нет правила вывода – цепочка не распозналась

с) Пример: грамматика $G = (T, N, P, S')$ для **float f**;

$T = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9, \omega, ;, =\}, N = \{S', A', B', C', D'\},$

$S' \rightarrow A'$

$A' \rightarrow B'; | B' \omega | B' =$

$B' \rightarrow B'A | \dots | B'Z | B'a | \dots | B'z$

$B' \rightarrow B'0 | \dots | B'9$

$B' \rightarrow C'A | \dots | C'Z | C'a | \dots | C'z$

$C' \rightarrow C' \omega$

$C' \rightarrow float \omega$

ω – пробел.

$G = (T, N, P, S')$ - регулярная левосторонняя грамматика.

8. Примеры: левосторонний вывод.

Вывод называется *левосторонним*, если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему левому нетерминальному символу в цепочке. Другими словами – на каждом шаге вывода происходит подстановка цепочки символов на основании правила грамматики, т.е. вместо крайнего левого нетерминального символа в исходной цепочке.

a) Пример: восходящий вывод (распознавание) цепочки *float f;*

Вывод цепочки	Дерево восходящего разбора цепочки
$C' \rightarrow float\omega$ $B' \rightarrow C'f$ $A' \rightarrow B';$ $S' \rightarrow A'$	

b) Пример: вывод (распознавание) цепочки *float f2x =*

$C' \rightarrow float\omega$
 $B' \rightarrow C'f$
 $B' \rightarrow B'2$
 $B' \rightarrow B'x$
 $A' \rightarrow B'\omega;$
 $S' \rightarrow A'$

Восходящий вывод цепочки

9. Для описания лексики языка программирования, обычно применяются регулярные грамматики.

С точки зрения лексического анализатора – язык программирования набор лексем (токенов), которые распознаются (классифицируются) лексическим анализатором.

Язык программирования на уровне лексического анализа представляет собой **регулярный язык** (язык типа 3 иерархии Хомского).

10. Грамматика языка описывает множество правильных цепочек символов над заданным алфавитом.

Для описания регулярных языков используют другую форму описания – регулярные выражения.

11. Регулярное выражение

Регулярное выражение описывает множество цепочек – формальный язык.

Для записи регулярного выражения используются метасимволы.

Множество цепочек описанных регулярным выражением называется **регулярным множеством** (или регулярным языком).

12. Определение.

Пусть I – алфавит.

Регулярные выражения над алфавитом I и языки, представляемые ими, рекурсивно определяются следующим образом:

- 1) \emptyset – регулярное выражение, представляет пустое множество;
- 2) λ – регулярное выражение, представляет множество $\{\lambda\}$;
- 3) для каждого $a \in I$ символ a является регулярным выражением и представляет множество $\{a\}$;
- 4) если p – регулярное выражение, представляющее множество P и q – регулярное выражение, представляющее множество Q , то $p + q$, pq , q^* являются регулярными выражениями и представляют множества:
 - a) $P \cup Q$ (объединение),
 - b) PQ (конкатенация множеств),
 - c) P^* (итерация) соответственно.
- 5) $pp^* = p^+$

- а) Символы, применяемые для описания регулярных выражений, называются **метасимволами** или **символами-джокерами**.

Джокерами являются символы: $*$, $^+$, $+$, $(,)$, \emptyset .

13. Примеры

- а) регулярные выражения и множества описанные ими:

Регулярное выражение	Множество
a	a
$a+b$	a, b
$a+b+c$	a, b, c
a^+	$a, aa, aaa, aaaa, \dots$
a^*	$\lambda, a, aa, aaa, aaaa, \dots$
ab	ab
$ab+cd$	ab, cd
$(ab+cd)^+$	$ab, cd, abab, abcd, cdcd, cdab, ababab, cdcdcd, abcdab, cdabcd, abcdcdcdcd, abababab, \dots$
$(ab+cd)^*$	$\lambda, (ab+cd)^+$
$a(bc+de)$	abc, ade
$a(bc+de)f$	$abcf, adef$
ab^+c	$abc, abbc, abbbc, abbbbc, \dots$
ab^*c	$ac, abc, abbc, abbbc, abbbbc, \dots$
$a(bc+de)^+f$	$abcf, adef, abcbcf, abcdef, adebcf, adedef, abcdedcdef, \dots$
$a(bc+de)^*f$	$af, a(bc+de)^+f$
$(ab+cd)(ef+gh)$	$abef, abgh, cdef, cdgh$
$(ab+cd)e^+$	$abe, cde, abee, cdee, abeee, cdeee, \dots$
$(ab+cd)e^*$	$ab, cd, (ab+cd)e^+$

- б) Пример для *float f*

$$float(\omega)^+(A + B + \dots + Z + a + b + \dots + z)^+$$

$$(A + B + \dots + Z + a + b + \dots + z + 0 + 1 + \dots + 9)^*$$

14. Теория регулярных языков была разработана в 1940-х годах.

Нейрофизиологи **Уоррен Мак-Каллох** и **Уолтер Питс** моделировали нервную систему на нейронном уровне. Математик **Стивен Клин** формально описал модели нейрофизиологов с помощью алгебры, которую назвал регулярными множествами. Для формального описания этих множеств он разработал простую математическую запись, которую назвал регулярным языком.

Запись использует специальные символы, которые в настоящее время называют метасимволами или символами-джокерами.

15. На сегодняшний день существует несколько диалектов регулярных языков (наборов метасимволов):

grep (global regular expression print) – команда в unix/linux;

egrep (extended grep) – разработал Альфред Ахо;

BRE (Basic Regular Expression) и **ERE** (Extended regular expression) – BRE + POSIX (Portable Operating System Interface) – попытка стандартизировать;

Perl – встроенные в язык Perl регулярные выражения;

ECMA-стандарт регулярных выражений в JavaScript;

SED (Stream Editor) – Bell Labs (1973-74 Lee E. McMahon).

16. Чаще всего используется Perl-нотация (набор метасимволов) регулярных выражений (в том числе в стандартных библиотеках C++ и C#).

17. В стандартной библиотеке C++ набор функций **<regex>**.

18. Пример применения функции `regex_match`:

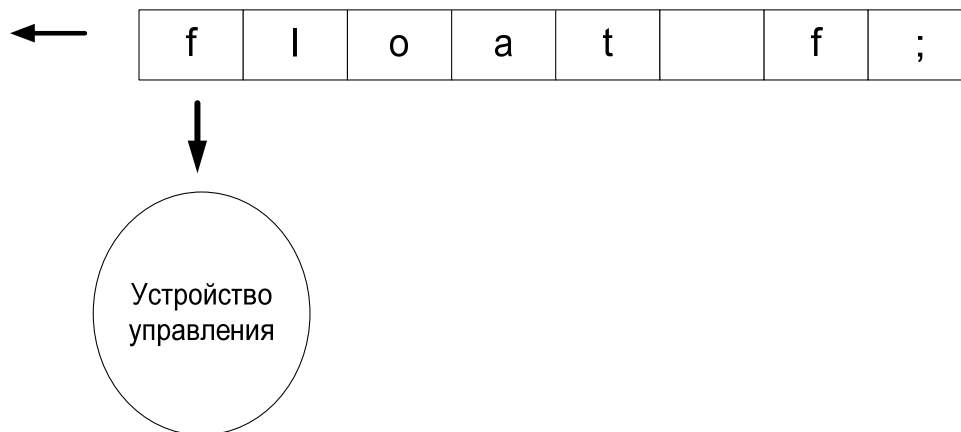
```
#include "stdafx.h"
#include <regex>
//std::regex_constants, std::regex_error
//std::regex_replace, std::regex_iterator
//std::regex_match, std::regex_search, std::regex_token_iterator
//std::regex_traits

int _tmain(int argc, _TCHAR* argv[])
{
    char    ch1[]    = "1234567899";
    wchar_t ch1w1[] = L"1234567899", wch2[] = L"12345X6799";
    char    ch2[]    = "ABCDEFGHJR",  ch2r[] = "АБВГДЕЖЗИК";
    char    ch3[]    = "abcdefghir",  ch4[]  = "a1b2c3de3f";
    char    ch5[]    = "11345.2234",  ch6[]  = "11345. 223";
    char    ch7[]    = "27.01.1960";
    bool b1  = std::regex_match(ch1,   ch1+10,   std::regex("[0-9]*")); // true
    bool b1w = std::regex_match(ch1w1, ch1w1+10, std::regex("[0-9]*")); // true
    bool b1w1 = std::regex_match(wch2, wch2+10,  std::regex("[0-9]*")); // false
    bool b2  = std::regex_match(ch1,   ch1+10,   std::regex("[A-Z]*")); // false
    bool b2r = std::regex_match(ch2r,  ch2r+10,  std::regex("[A-Я]*")); // true
    bool b3  = std::regex_match(ch2,   ch2+10,   std::regex("[A-Z]*")); // true
    bool b4  = std::regex_match(ch2,   ch2+10,   std::regex("[a-z]*")); // false
    bool b5  = std::regex_match(ch3,   ch3+10,   std::regex("[a-z]*")); // true
    bool b5r = std::regex_match(ch3,   ch3+10,   std::regex("[a-z]*")); // true
    bool b6  = std::regex_match(ch4,   ch4+10,   std::regex("[a-z|0-9]*")); // true
    bool b7  = std::regex_match(ch3,   ch3+10,   std::regex("[a-z|0-9]*")); // true
    bool b8  = std::regex_match(ch2,   ch2+10,   std::regex("[a-z|0-9]*")); // false
    bool b9  = std::regex_match(ch2,   ch2+10,   std::regex("[a-z|A-Z]*")); // true
    bool b10 = std::regex_match(ch5,   ch5+10,   std::regex("[0-9]+.[0-9]*")); // true
    bool b11 = std::regex_match(ch6,   ch6+10,   std::regex("[0-9]+.[0-9]*")); // false

    #define DD "(0[1-9]+|1[0-9]+|2[0-9]+|3[0|1]+)"
    #define MM "(0[1-9]+|1[0-2]+)"
    #define YYYY "([0-9]{4,4})"
    bool b12 = std::regex_match(ch7, ch7+10, std::regex(DD "." MM "." YYYY)); // true

    return 0;
}
```

19. Схема работы лексического анализатора



Класс алгоритмов, соответствующих приведенной выше схеме, может быть записан в форме конечного автомата (КА).

20. Определение КА:

КА это пятерка $M = (S, I, \delta, s_0, F)$,

где

S – конечное множество состояний устройства управления;

I – алфавит входных символов;

δ – функция переходов, отображающая $S \times (I \cup \{\lambda\})$ во множество подмножеств S : $\delta(s, i) \subset S, s \in S, i \in I$;

$s_0 \in S$ - начальное состояние устройства управления;

$F \subseteq S$ - множество заключительных (допускающих) состояний устройства управления.

Если $\delta(s, \lambda) = \emptyset$ и $|\delta(s, a)| \leq 1$, то конечный автомат – **детерминированный (ДКА)**.

Т.е. отсутствуют состояния, имеющие λ -переходы и для каждого состояния s и входного символа a существует не более одной дуги, выходящей из s и помеченной как a . ДКА – это автомат, который переходит из любого состояния по любому символу точно в одно состояние.

Иначе - конечный автомат является **недетерминированным (НКА)**.

21. Мгновенное описание КА – пара (s, w) ,

где $s \in S$ – состояние КА,

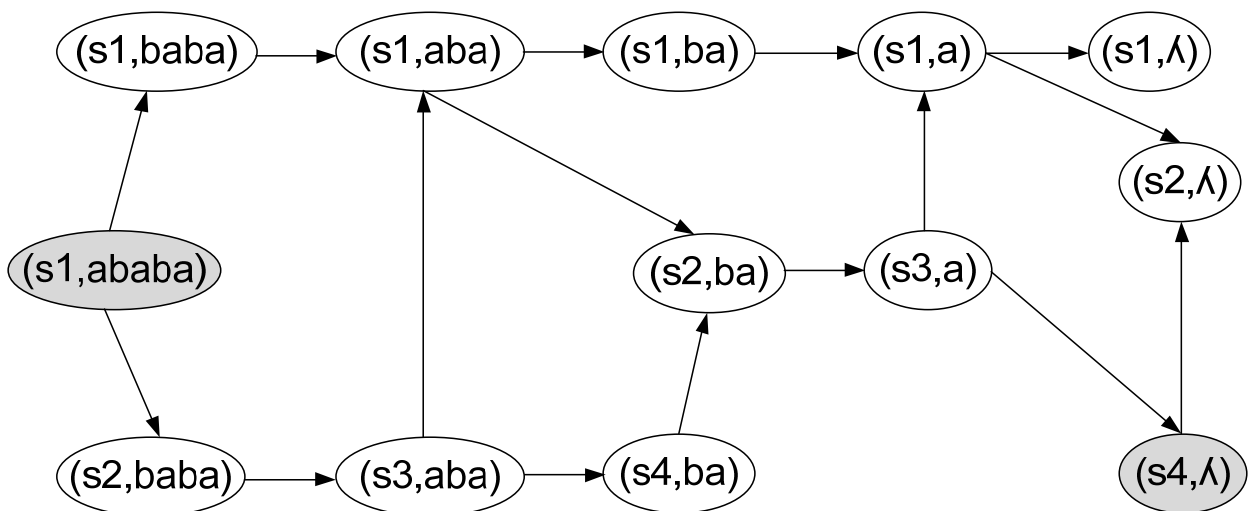
$w \in I^*$ – неиспользованная часть входной цепочки.

22. (s_0, w_0) – начальное мгновенное описание КА,
где w_0 – анализируемая цепочка.
23. (s_f, λ) , $s_f \in S$ – допускающее мгновенное описание КА.
24. Если (s, aw) и $s' \in \delta(s, a)$, где $s', s \in S$, $a \in I \cup \lambda$, $w \in I^*$, то
 $(s, aw) \succ (s', w)$ – читается: непосредственно следует.
25. Если $(s_i, w_i) \succ (s_{i+1}, w_{i+1}) \succ (s_{i+2}, w_{i+2}) \succ \dots \succ (s_k, w_k)$, то
 $(s_i, w_i) \succ^* (s_k, w_k)$ – следует.
26. Если $(s_0, w) \succ^* (s_f, \lambda)$, $s_0 \in S$ – начальное состояние,
 $s_f \in F$ – конечное состояние,
то цепочка $w \in I^*$ допускается (или распознается) КА.

27. Пример: пусть $w \in (a + b)^* aba$ входная цепочка,
КА $M = (\{s_1, s_2, s_3, s_4\}, \{a, b\}, \delta, s_1, \{s_4\})$, где функция δ задана
следующей таблицей:

	a	b	λ
s_1	$\{s_1, s_2\}$	$\{s_1\}$	\emptyset
s_2	\emptyset	$\{s_3\}$	\emptyset
s_3	$\{s_4\}$	\emptyset	$\{s_1\}$
s_4	\emptyset	\emptyset	$\{s_2\}$

28. Последовательность мгновенных описаний автомата



29. $(s_1, abaaba) \succ^*(s_4, \lambda)$ – значит, что автомат M допускает (распознает) цепочку $abaaba$.

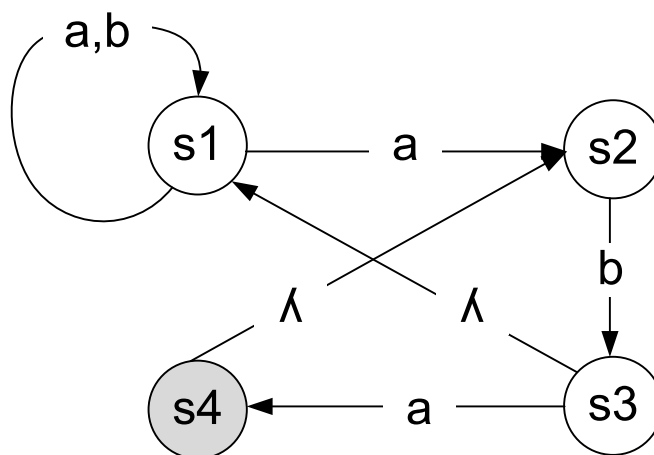
30. **Определение.**

Графом переходов конечного автомата $M = (S, I, \delta, s_0, F)$ называется ориентированный граф $G = (S, E)$,

где S – множество вершин графа совпадает с множеством состояний конечного автомата,

E – множество ребер (направленных линий, соединяющих вершины), ребро $(s_i, s_j) \in E$, если $s_j \in \delta(s_i, a), a \in I \cup \lambda$.

Метка ребра (s_i, s_j) – все a , для которых $s_j \in \delta(s_i, a)$.



31. Конечный автомат может быть однозначно задан своим графом переходов.

32. Доказаны 4 утверждения:

- 1) язык является регулярным множеством тогда и только тогда, когда он задан регулярной грамматикой;
- 2) язык может быть задан регулярной грамматикой (левосторонней или правосторонней) тогда и только тогда, когда язык является регулярным множеством;
- 3) язык является регулярным множеством тогда и только тогда, когда он задан конечным автоматом;
- 4) язык распознается с помощью конечного автомата тогда и только тогда, когда он является регулярным множеством.

Другими словами:

любой регулярный язык может быть задан регулярной грамматикой, регулярным выражением или конечным автоматом.

Или:

любой конечный автомат задает регулярный язык, а значит грамматику или регулярное выражение.

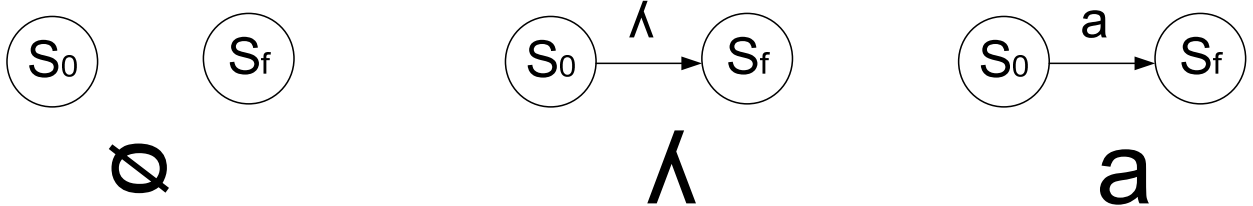
33. Доказана теорема (А. Ахо, Дж. Хопкрофт, Дж. Ульман):

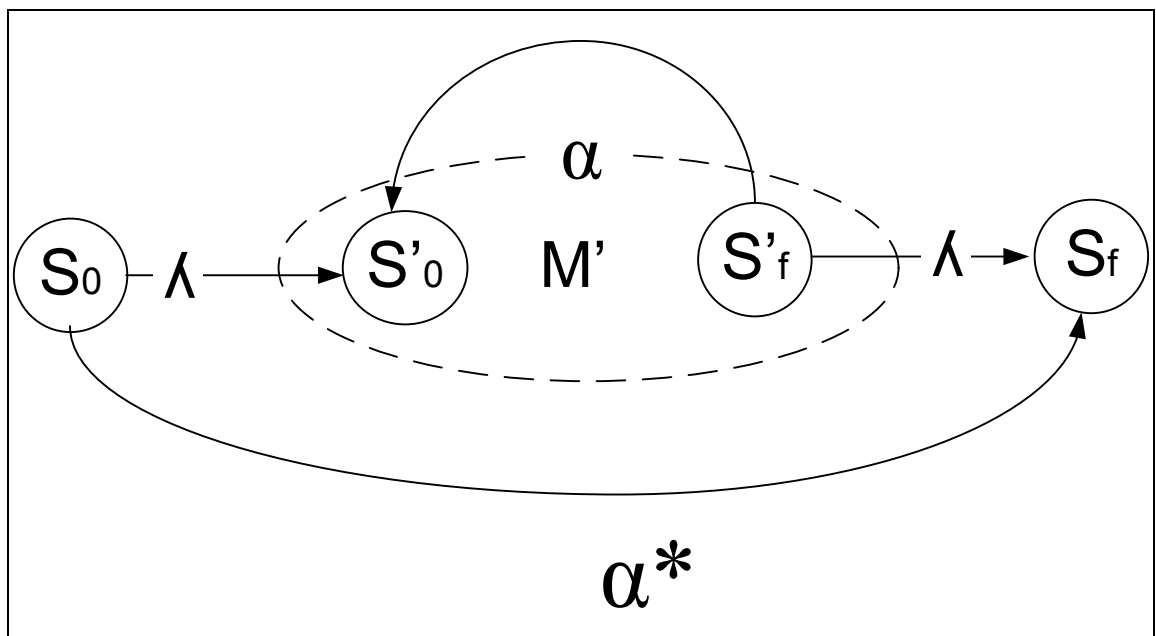
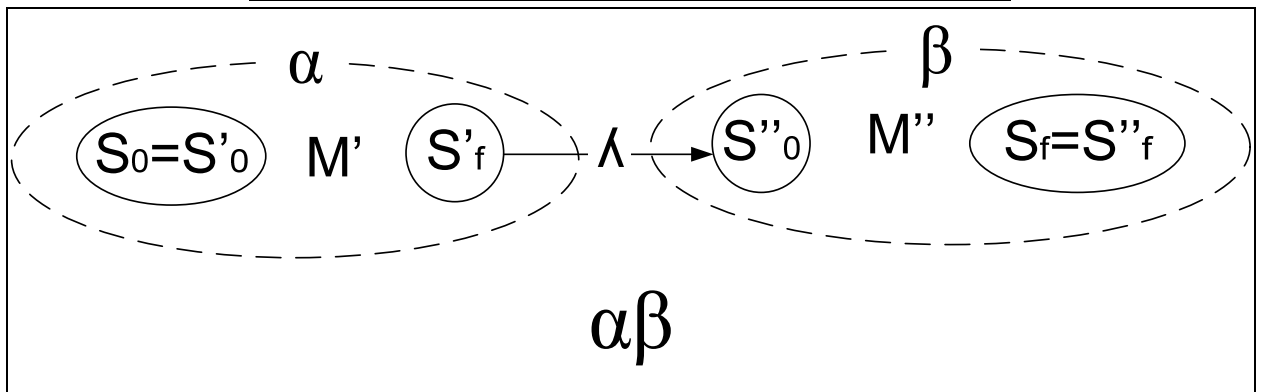
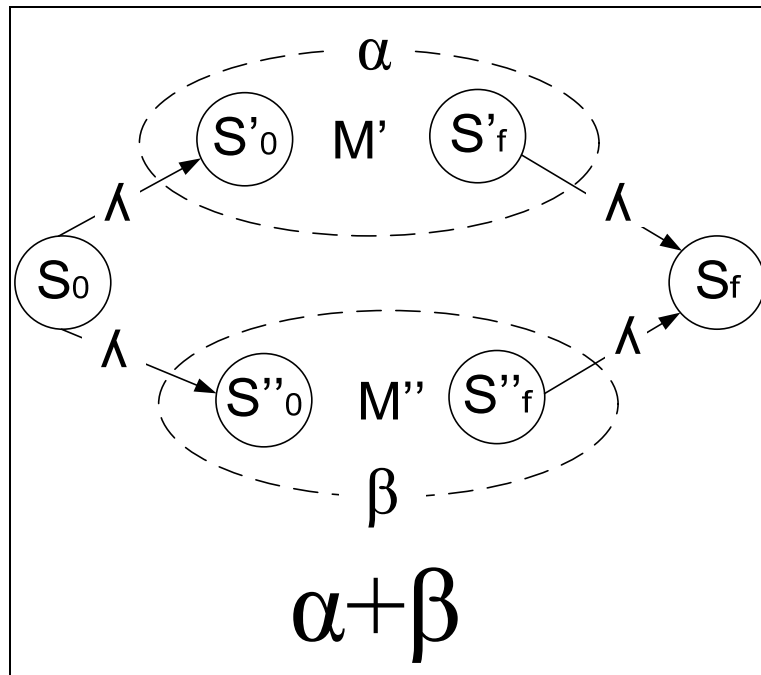
Пусть α - регулярное выражение, тогда найдется недетерминированный конечный автомат $M = (S, I, \delta, s_0, \{s_f\})$, допускающий автомат, представленный α , и обладающий следующими свойствами:

- 1) $|S| \leq 2|\alpha|$;
- 2) $\forall a \in I \cup \{\lambda\} : \delta(s_f, a) = \emptyset$;
- 3) $\forall s \in S : \sum_{a \in I \cup \{\lambda\}} |\delta(s, a)| \leq 2$.

34. Построение графа конечного автомата по регулярному выражению:

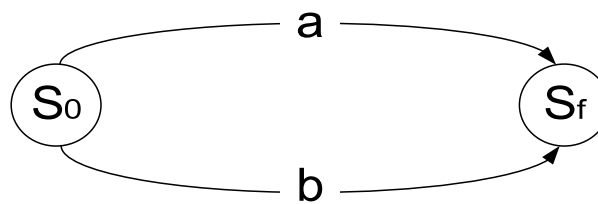
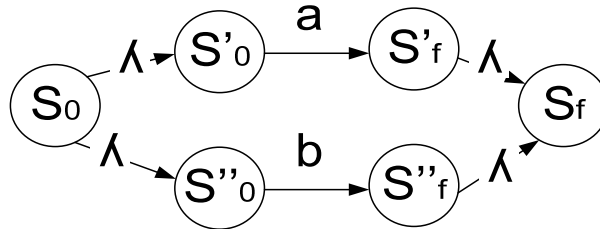
Метод построения. Автомат для выражения строится композицией из автоматов, соответствующих подвыражениям. На каждом шаге построения строящийся автомат имеет в точности одно заключительное состояние, в начальное состояние нет переходов из других состояний и нет переходов из заключительного состояния в другие.



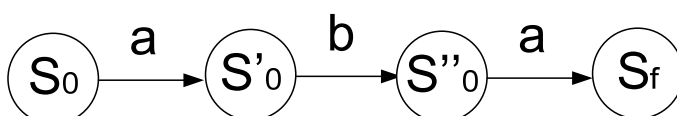
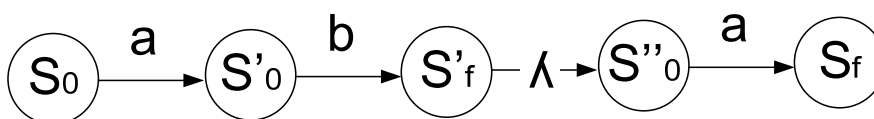
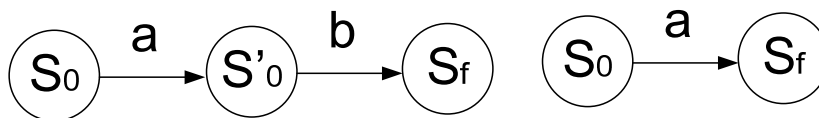
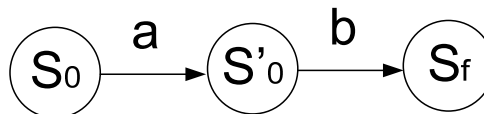
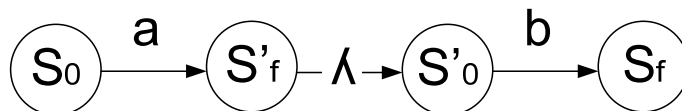


35. **Пример:** пусть язык L задан регулярным выражением $(a + b)^* aba$

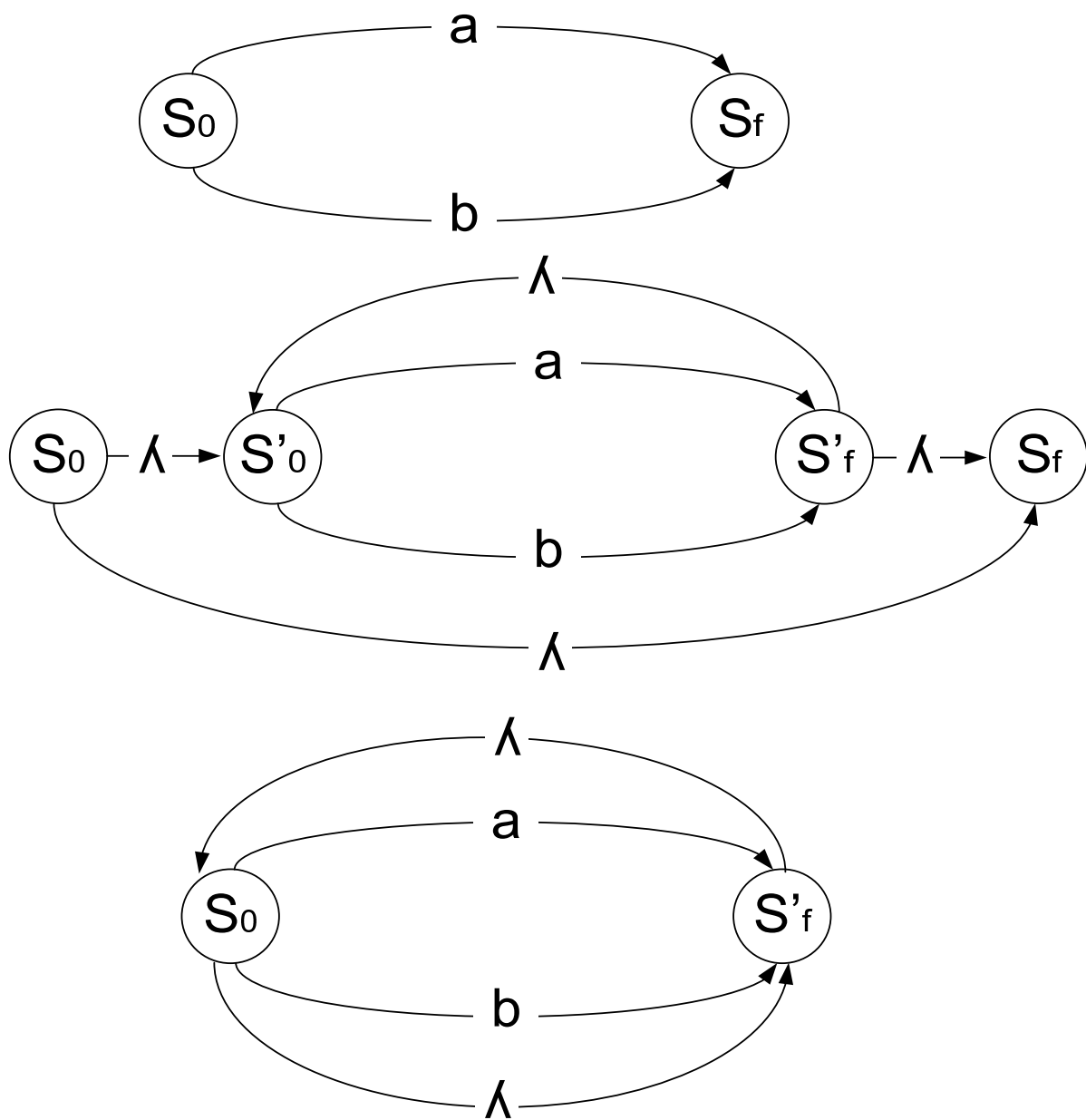
а) Подвыражение: $a + b$



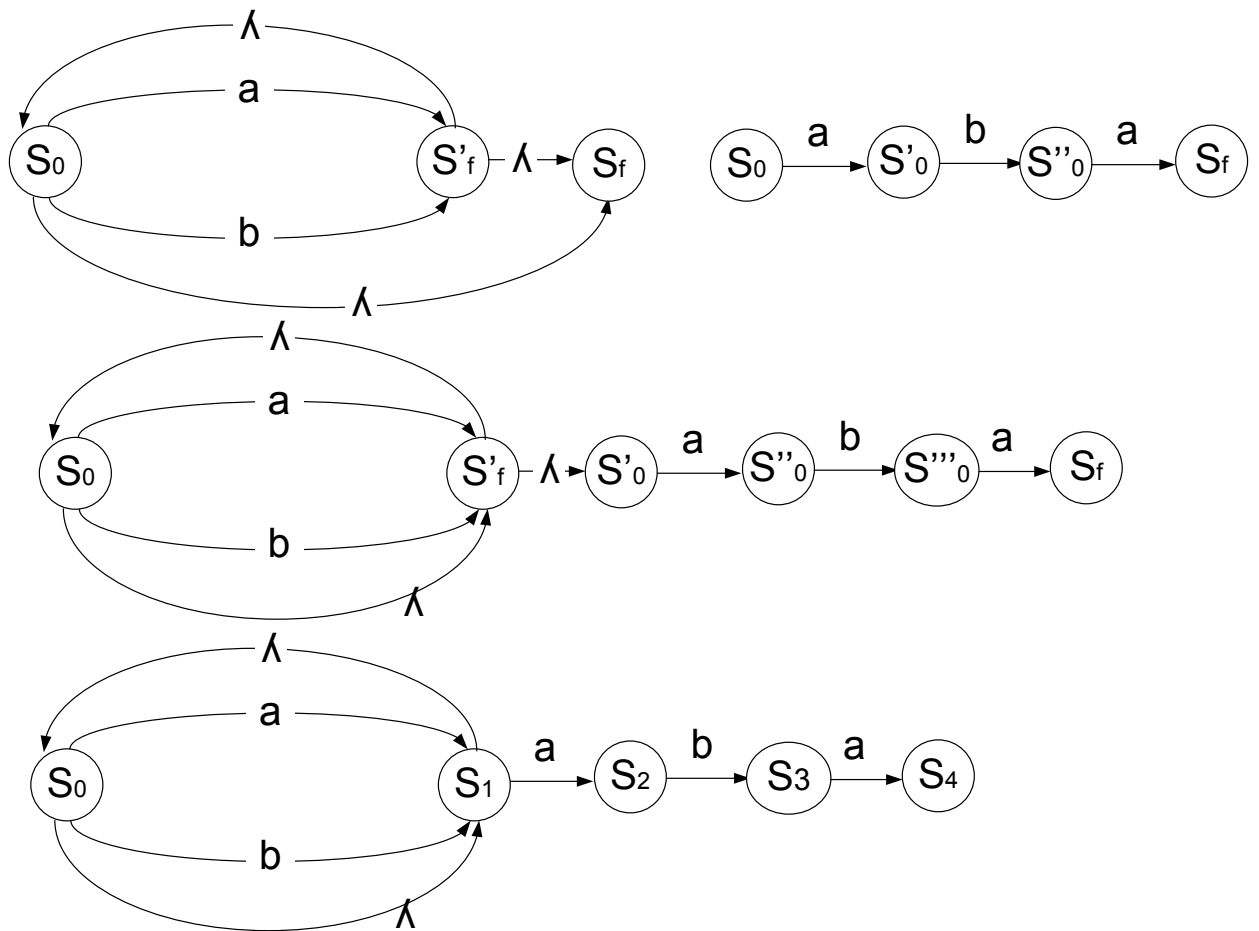
б) Подвыражение: aba



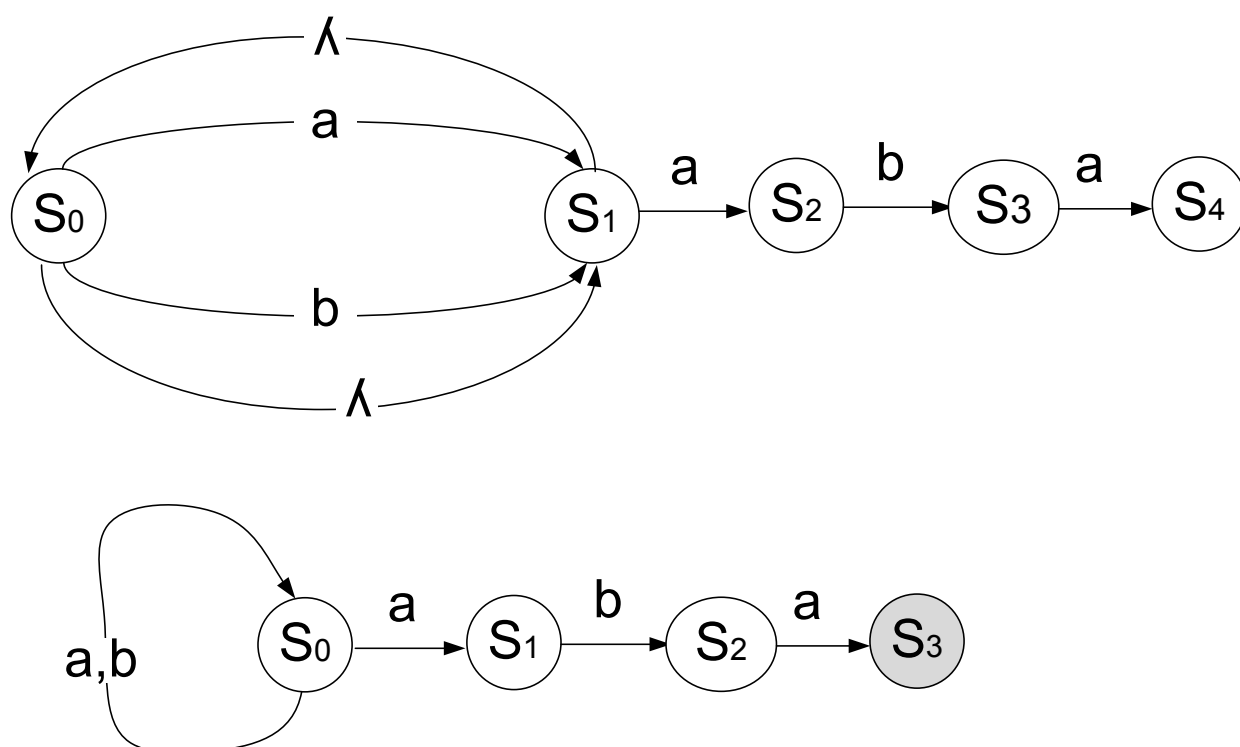
с) Продолжение. Подвыражение: $(a + b)^*$



d) Выражение: $(a + b) * aba$



е) Пример (продолжение): $(a + b) * aba$

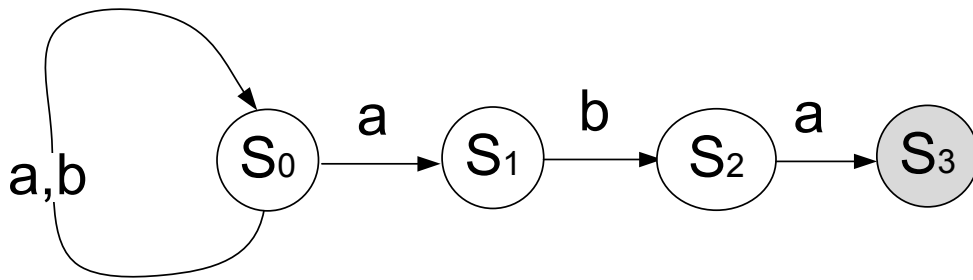


ф) Пример (продолжение): $(a + b) * aba$

$$M = (\{s_0, s_0, s_2, s_3\}, \{a, b\}, \delta, s_0, \{s_3\})$$

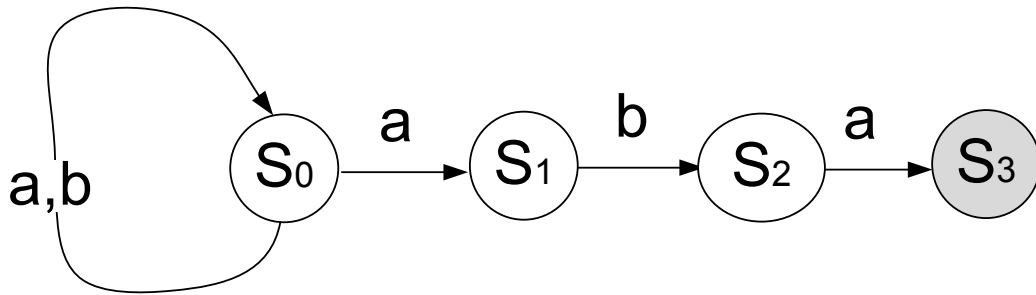
	a	b	λ
s_0	$\{s_0, s_1\}$	$\{s_0\}$	\emptyset
s_1	\emptyset	$\{s_2\}$	\emptyset
s_2	$\{s_3\}$	\emptyset	\emptyset
s_3	\emptyset	\emptyset	\emptyset

36. Пример (продолжение): алгоритм разбора



aabbabaaba	{S0}	
abbabaaba	{S0,S1}	
bbabaaba	{S0,S1}	
babaaba	{S0,S2}	
abaaba	{S0}	
baaba	{S0,S1}	
aaba	{S0,S2}	
aba	{S0,S1,S3}	
ba	{S0,S1}	
a	{S0,S2}	
	{S0, S3 }	Успешный разбор

37. Пример (продолжение): алгоритм разбора



aabbababba

{S₀}

abbababba

{S₀, S₁}

bbababba

{S₀, S₁}

bababba

{S₀, S₂}

ababba

{S₀}

babba

{S₀, S₁}

abba

{S₀, S₂}

bba

{S₀, S₁, S₃}

ba

{S₀, S₂}

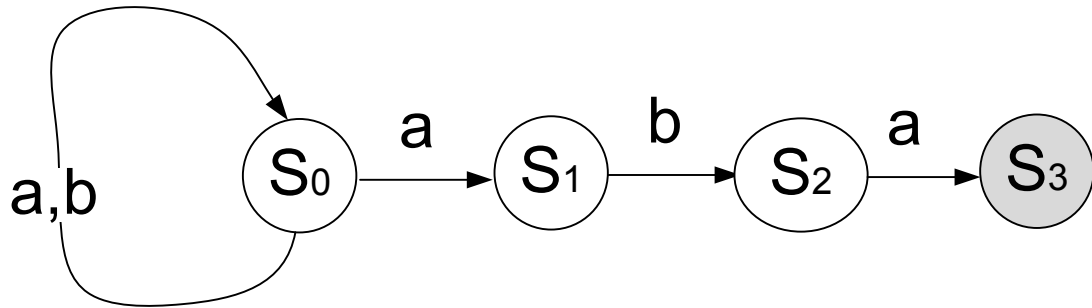
a

{S₀}

{S₀, S₁}

Ошибка разбора

38. Пример (продолжение): алгоритм разбора



aabbxbabba

{S₀}

abbxbabba

{S₀, S₁}

bbxbabba

{S₀, S₁}

bxbabba

{S₀, S₂}

xbabba

{

Ошибка разбора

39. Реализация алгоритма для разбора

aabbaba

