Титульный лист материалов по дисциплине

ДИСЦИПЛИНА *Теория формальных языков*

ИНСТИТУТ *Информационных технологий*

КАФЕДРА *Вычислительной техники*

ВИД УЧЕБНОГО МАТЕРИАЛА *Лекция*

ПРЕПОДАВАТЕЛЬ *Унгер Антон Юрьевич*

СЕМЕСТР *3 семестр*

6. РАЗБОР ПО РЕГУЛЯРНОЙ ГРАММАТИКЕ

Как уже было сказано, в основе лексического анализа формальных языков лежат регулярные грамматики. Напомним, что к регулярным грамматикам относятся леволинейные и праволинейные грамматики. Одним из основных положений регулярных грамматик является то, что для любой леволинейной грамматки всегда существует эквивалентная праволинейная грамматика.

Правила вывода для леволинейной регулярной грамматики имеют вид

$$A \to aB|a. \tag{3.5}$$

Здесь, напомним, $A, B \in N; a \in T$. Данная грамматика не содержит ε-правила, т.е. правил с пустой правой частью вида $A \to \varepsilon$. Это ограничение не существенно, поскольку мы рассматриваем разбор по регулярным грамматикам применительно к лексическому анализу языка программирования. Лексемы языка не могут быть пустыми.

Регулярные выражения являются основой лексического анализа языка программирования.

Пример 25. Идентификатор языка определяется регулярным выражением вида

$$letter(letter|digit)^*. \tag{3.6}$$

Реализация лексического анализатора – простейшая программа на любом языке высокого уровня. В листинге 2 приведен фрагмент программы на языке С, реализующий проверку введенного идентификатора на соответствие шаблону (3.6).

Листинг 2

```
#include <stdio.h>
#include <ctype.h>
int main() {
     char in;
     in = getchar();
     if (isalpha(in)) in = getchar();
     else return 1;
     while (isalpha(in) || isdigit(in)) in = getchar();
     return 0;
}
```

Приведенная программа лишь проверяют вводимые данные на соответствие шаблону, но не извлекают из них никакой информации.

Приведенный пример показывает насколько просто реализовать разбор выражения по регулярной грамматике, однако в общем случае применяют аппарат *конечных автоматов*. Известно (см. выше), что регулярные выражения и конечные автоматы тесно связаны друг с другом.

Конечный автомат может находиться в одном их конечного множества состояний и может переходить из одного состояния в другое в результате считывания символа из входного потока.

6.2.1. Конечный автомат

Определение 28. *Детерминированный конечный автомат (ДКА)* – это пятерка элементов $M = (K, \Sigma, \delta, S, F)$, где

K – множество состояний;

Σ – входной алфавит;

δ – множество переходов между состояниями;

S – начальное состояние ($S \in K$);

F – одно или несколько конечных состояний ($F \in K$).

Стоит сделать несколько уточняющих замечаний. Во-первых, может быть только одно конечное состояние, поскольку все цепочки регулярного языка должны заканчиваться на символ конца $\perp$. Это замечание относится исключительно к лексическому анализу языков программирования и не связано с теорией автоматов, которая допускает у конечного автомата несколько конечных состояний.

Во-вторых, множество переходов $\delta(A, t) = B$ есть функция, которая ставит в соответствие некоторому состоянию A и текущему символу t на входе другое состояние B. Если для текущего состояния A_i и текущего прочитанного символа t_i значение $\delta(A_i, t_i)$ не определено, автомат останавливается с ошибкой.

Определение 29. Говорят, что ДКА *допускает* цепочку символов $a_1 a_2 \dots a_n$, если определены $\delta(H, a_1) = A_1, \delta(A_1, a_2) = A_2, \dots, \delta(A_{n-1}, a_n) = S$, где $a_i \in T, A_j \in K$, H – начальное состояние, S – конечное состояние.

Пример 26. На рис. 9 приведен конечный автомат для распознавания идентификатора.

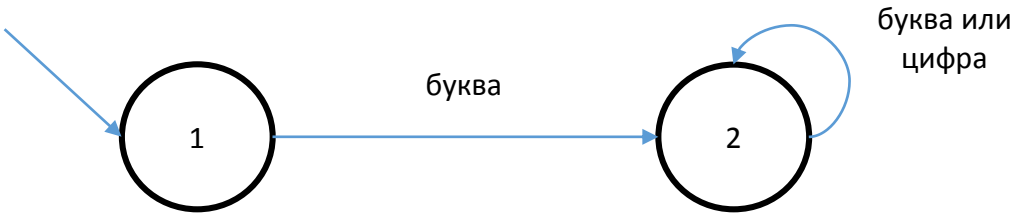

Рисунок 9. Детерминированный конечный автомат

Здесь при считывании первой буквы происходит переход из *начального* состояния 1 в состояние 2 (*конечное*). Далее, любая буква или цифра переводят автомат снова в состояние 2.

Пример 27. Рассмотрим регулярное выражение для распознавания действительного числа

$$(+|-|\varepsilon)digit^*.digit\ digit^*. \qquad (3.7)$$

Необходимо отметить, что данное выражение не вполне соответствует лексике языка *C*, которая допускает отсутствие дробной части (после десятичной точки).

Конечный автомат, соответствующий регулярному выражению (3.7), приведен на рис. 10.

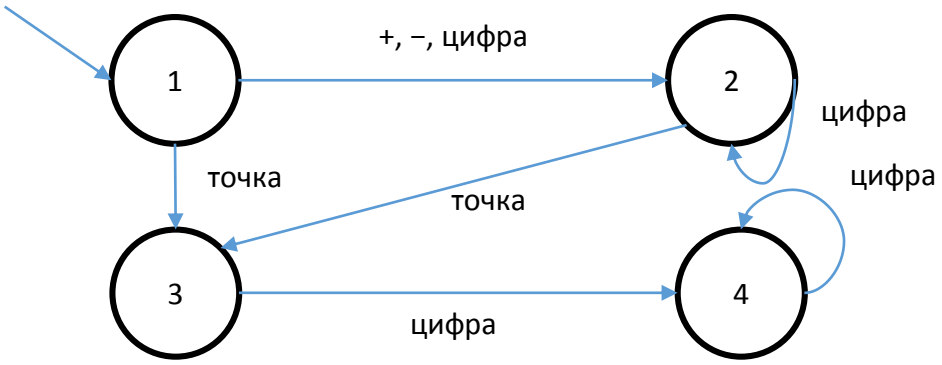

Рисунок 10. Конечный автомат для распознавания действительного числа

В листинге 3 приведен фрагмент программы на языке C, который реализует автоматный разбор входной строки на соответствие регулярному выражению.

Листинг 3

```
#include <stdio.h>
#include <ctype.h>
int issign(char in) {return (in == '+' || in == '-');}
int main() {
    int state = 1;
    char in = getchar();
    while (isdigit(in) || issign(in) || in == '.') {
        switch (state) {
        case 1:
            if (isdigit(in) || issign(in)) state = 2;
            else if (in == '.') state = 3;
```

```
            break;
        case 2:
            if (isdigit(in)) state = 2;
            else if (in == '.') state = 3;
            else return 1;
            break;
        case 3:
            if (isdigit(in)) state = 4;
            else return 1; // Error!
            break;
        case 4:
            if (isdigit(in)) state = 4;
            else return 1;
            break;
        }
        in = getchar();
    }
    return (state == 4);
```

Обращает на себя внимание тот факт, что в блоке ***while*** мы фактически указываем алфавит всех допустимых символов.

6.2.2. Недетерминированный конечный автомат

Недетерминизм конечного автомата состоит в том, что из текущего символа, находясь в некотором состоянии, автомат может перейти в несколько возможных состояний.

Определение 30. *Недетерминированный конечный автомат (НКА)* – это пятерка $< K, T, D, H, S >$, в которой

K – конечное множество состояний;

T – конечное множество возможных символов на входе;

D – функция разрешенных переходов между состояниями;

H – начальное состояние;

S – множество заключительных состояний.

Графически работу НКА можно представить так (рис. 11).

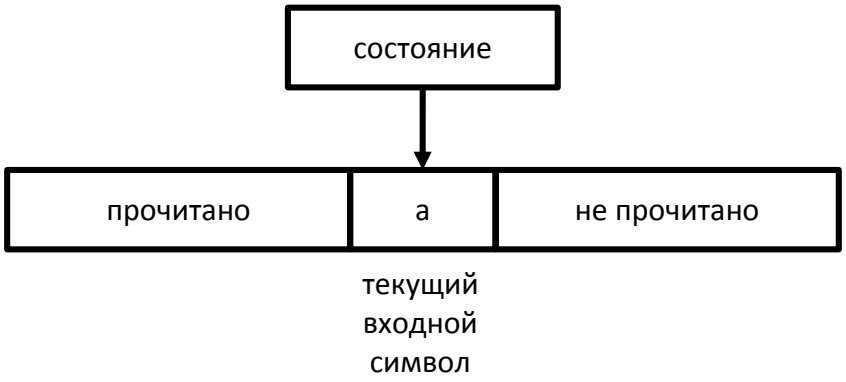

Рисунок 11. Недетерминированный конечный автомат

6.2.3. Детерминированный конечный автомат

В отличие от НКА из каждого состояния детерминированный конечный автомат (ДКА) может перейти только в одно допустимое состояние.

ДКА является частным случаем НКА. Другими словами, класс языков, определяемых НКА, совпадает с классом языков, определяемых ДКА. Из этого следует важный вывод о том, что НКА может быть преобразован в ДКА и наоборот. Покажем это.

Пример 28. Построим конечный автомат для следующего регулярного выражения

$$(a|b)^*a(a|b)(a|b). \tag{3.8}$$

Недетерминированный конечный автомат содержит 4 состояния и определяется так

$$M = \{\{1,2,3,4\},\{a,b\},D,1,\{4\}\}. \tag{3.9}$$

Здесь D – правила переходов, который удобно разместить в табл. 1.

Таблица 1

$D(1,a)=\{1,2\}$	$D(3,a)=\{4\}$
$D(1,b)=\{1\}$	$D(2,b)=\{3\}$
$D(2,a)=\{3\}$	$D(3,b)=\{4\}$

Данному конечному автомату соответствует следующая диаграмма состояний (рис. 12).

Эту же задачу можно решить с помощью детерминированного конечного автомата. ДКА в данном случае будет иметь 8 состояний и будет определяться так

$$M = \{\{1,2,3,4,5,6,7,8\},\{a,b\},D,1,\{3,5,6,8\}\}. \tag{3.10}$$

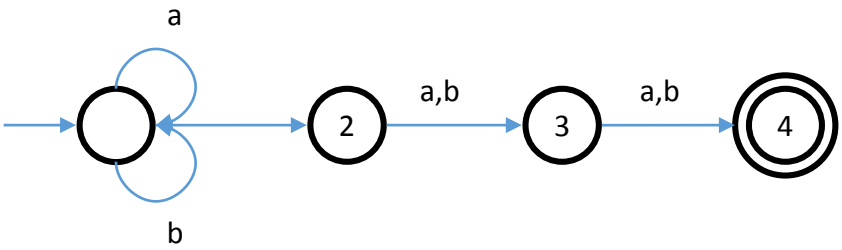

Рисунок 12. НКА для примера 28

Автомату будет соответствовать следующая диаграмма состояний (рис. 13).

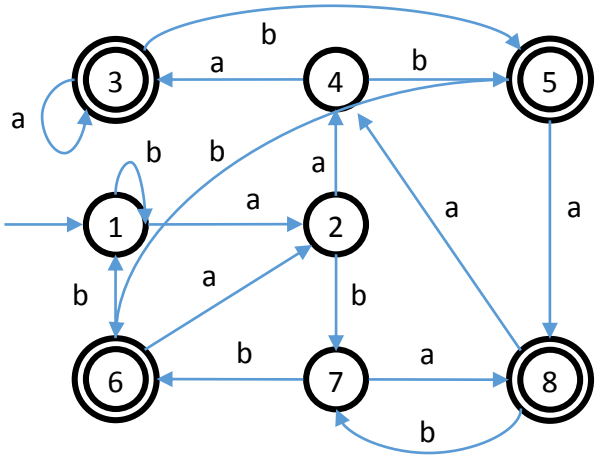

Рисунок 13. ДКА для примера 28

Заключительными будут состояния 3, 5, 6 и 8 (они показаны кружками с двойной границей).

6.2.4. Построение НКА по регулярному выражению

Напомним, что в регулярном выражении допустимы следующие подвыражения:

1. Пустое множество $\emptyset$, которому соответствует диаграмма (рис. 14а).
2. Пустая строка ε (рис. 14б).
3. Любой символ из алфавита a (рис. 14в).
4. Объединение $s|t$, где s и t – регулярные выражения (рис. 14г).
5. Конкатенация двух подвыражений st (рис. 14д).
6. Итерация (звездочка Клини) s^* (рис. 14е).

Рассмотрим случай леволинейной грамматики $G =< T, N, P, S >$ с правилами вывода $A \to Bt|t$, где $A, B \in N; t \in T$.

6.2.5. Алгоритм разбора

Разбор по регулярной грамматике заключается в том, чтобы определить принадлежит ли строка $a_1 a_2 \ldots a_n$ $\perp$ языку, задаваемому данной грамматикой.

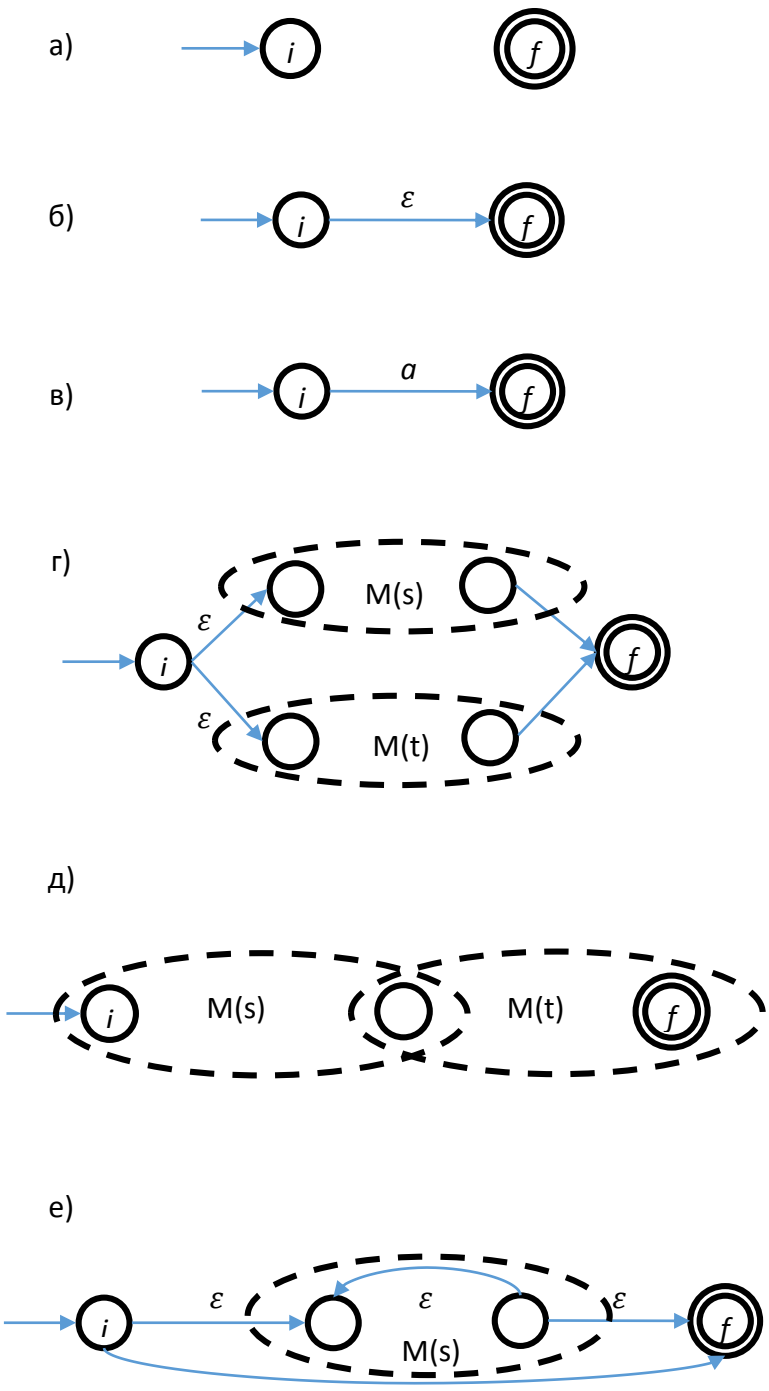

Рисунок 14. Автоматы, соответствующие простейшим регулярным выражениям

Разбор состоит из следующей последовательности шагов:

1. Первый символ исходной строки заменить нетерминалом, для которого есть правило $A \to a_1$. Свертка терминала a_1 к нетерминалу A.

2. Повторять следующие действия до считывания символа конца строки: свернуть очередной входной символ с a_i с нетерминалом A и получить нетерминал B, для которого есть правило вывода $B \to Aa_i$ $(i = 2 \ldots n)$.

Разбор по данному алгоритму может привести к следующим возможным ситуациям:

1. Прочитана вся цепочка. На каждом шаге имелась нужная свертка. На последнем шаге свертка привела к символу *S*. Следовательно, цепочка принадлежит языку $a_1 a_2 \dots a_n \in L(G)$.

2. Прочитана вся цепочка. На каждом шаге находилась нужная свертка. Последняя свертка не привела к символу *S*. Следовательно, цепочка не принадлежит языку $a_1 a_2 \dots a_n \notin L(G)$.

3. На некотором шаге не нашлось нужной свертки. Следовательно, $a_1 a_2 \dots a_n \notin L(G)$.

4. На некотором шаге нашлось более одной нужной свертки. Эта *недетерминированная* ситуация должна рассматриваться отдельно.

Для того, чтобы автоматизировать процесс разбора, построим таблицу возможных переходов. Это позволит облегчить процесс нахождения нужного правила вывода с подходящей правой частью. Пусть столбцы таблицы содержат алфавит допустимых терминалов. Каждая строка таблицы разбора соответствует нетерминалу в левой части грамматических правил. При этом первая строка содержит начальный (стартовый) символ *H*. На пересечение каждой строки и столбца расположен нетерминал, к которому можно свернуть соответствующий терминал, помечающий столбец, и соответствующий нетерминал, помечающий строку.

Пример 29. Рассмотрим грамматику $G_l =< \{a, b, \perp\}, \{S, A, B, C\}, P, S >$, где

$$\begin{aligned} &S \to C \perp; \\ &C \to Ab|Ba; \\ &A \to a|Ca; \\ &B \to b|Cb. \end{aligned} \qquad (3.11)$$

Таблица разбора, которая определяется правилами (3.11) и не зависит от вида анализируемой цепочки приведена ниже. Здесь символ «-» означает, что для соответствующей пары «терминал-нетерминал» нет подходящей свертки.

Таблицу разбора удобно комбинировать с *диаграммой состояний* (ДС). Диаграмма представляет собой *ориентированный граф*, такой что:

1. Каждому нетерминалу грамматики соответствует вершина (состояние). Еще одна вершина, помеченная символом *H*, служит для обозначения начального состояния.

2. Для правил вывода $W \to t$ проводим дугу, помеченную *t*, ведущую из состояния *H* в состояние *W*.

3. Для правил вывода $W \to Vt$ соединяем дугой, помеченной t, состояния V и W.

Таблица 2

	$\boldsymbol{a}$	$\boldsymbol{b}$	$\perp$
$\boldsymbol{H}$	A	B	-
$\boldsymbol{C}$	A	B	S
$\boldsymbol{A}$	-	C	-
$\boldsymbol{B}$	C	-	-
$\boldsymbol{S}$	-	-	-

Полученная таким образом диаграмма состояний является конечным автоматом для заданной регулярной грамматики.

Пример 30. На рис. 15 представлена диаграмма состояний для грамматики (3.11).

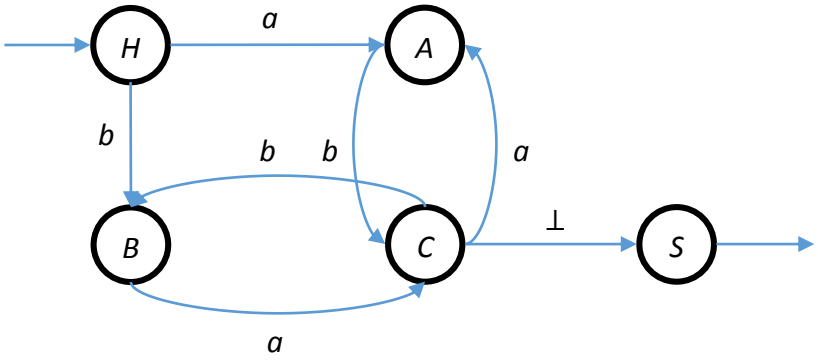

Рисунок 15. Диаграмма состояний для примера 29

Приведем алгоритм разбора входной строки $a_1 a_2 \ldots a_n$ по диаграмме состояний. Алгоритм включает следующие шаги:

1. Сделать текущее состояние начальным H.
2. Считать первый символ входной строки a_1.
3. Перейти в состояние W, ведущее из H по символу a_1.
4. Сделать состояние W текущим.
5. Повторять шаги 2-4 до считывания маркера конца строки или ошибки ввода.

Если все строки языка заканчиваются маркером $\perp$, до конечное состояние единственно S.

Результатом разбора могут быть следующие ситуации:

1. Последний считанный символ привел в состояние S, следовательно, строка принадлежит языку L.

2. Последний считанный символ не привел в состояние *S*, следовательно, строка не принадлежит языку *L*.

3. Для некоторого символа a_i не нашлось перехода в новое состояние, следовательно, строка не принадлежит языку *L*.

4. Для некоторого символа a_i нашлось более одной дуги ведущей в разные состояния. В этом случае разбор *недетерминированный*.

6.2.6. Анализатор регулярной грамматики

Для регулярной грамматики (2.11) нами была составлена таблица разбора, нарисована диаграмма состояний, однако конечной целью разбора является написание программы анализатора на выбранном языке программирования. Для этого введем следующее соглашение: дополним множество возможных состояний конечного автомата дополнительным состоянием *ER*, переход в которое соответствует ошибке ввода.

В листинге 4 приведен фрагмент программы на языке C++, который реализует разбор введенной строки согласно грамматике (3.11). В данном случае конец ввода определяется стандартным для C++ способом – по «нулевому» символу.

Листинг 4

```
#include <iostream>
using namespace std;
char c; // текущий символ
void gc () { cin >> c; }
bool scan_G () {
    enum state { H, A, B, C, S, ER };
    state CS; // текущее состояние
    CS = H;
    do {
        gc ();
        switch (CS) {
        case H:
            if ( c == 'a' ) CS = A;
            else if ( c == 'b' ) CS = B;
            else CS = ER;
            break;
```

Окончание листинга 4

```
        case A:
            if ( c == 'b' ) CS = C;
            else CS = ER;
            break;
```

```
            case B:
                if ( c == 'a' ) CS = C;
                else CS = ER;
                break;
            case C:
                if ( c =='a' ) CS = A;
                else if ( c == 'b' ) CS = B;
                else if ( c == '\0' ) CS = S;
                else CS = ER;
                break;
            }
        }
        while ( CS != S && CS != ER);
        return CS == S;
}
```

Можно видеть, что здесь применен стандартный «автоматный» подход, которому в C++ соответствует оператор ***switch-case***.

Рассмотрим работу алгоритма подробнее. Для примера построим дерево разбора для строки $abba\perp$. Данной строке соответствует следующая последовательность *сверток*

$$abba\perp \rightarrow Abba\perp \rightarrow Cba\perp \rightarrow Ba\perp \rightarrow C\perp \rightarrow S. \qquad (3.12)$$

Как видим, строка сворачивается в стартовый символ S, следовательно, она принадлежит описываемому языку. На рис. 16 показаны этапы построения дерева разбора.

Вспомним, что для леволинейной грамматики применим подход построения дерева снизу-вверх. При таком подходе правила вывода применяются в обратную сторону, т.е. каждому терминалу t и нетерминалу N, для который в грамматике есть правило $L \rightarrow Nt$, ставится в соответствие нетерминал L. Для рассматриваемой цепочки последовательность замен будет такой

$$abba\perp \leftarrow \boldsymbol{A}bba\perp \leftarrow \boldsymbol{C}ba\perp \leftarrow \boldsymbol{B}a\perp \leftarrow \boldsymbol{C}\perp \leftarrow \boldsymbol{S}. \qquad (3.13)$$

Эта последовательность соответствует перевернутому правому выводу для данной цепочки в грамматике (3.11). Выше мы рассмотрели леволинейную грамматику и осуществили по ней разбор цепочки. Леволинейной грамматике соответствует построение дерева разбора снизу-вверх.

Пример 31. Рассмотрим праволинейную грамматику $G_r =< \{a, b, \perp\}, \{H, A, B, C\}, P, H >$, у которой правила вывода таковы

$$H \rightarrow aA|bB; \qquad (3.14)$$

$A \to bC;$
$C \to bB|aA|\perp;$
$B \to aC.$

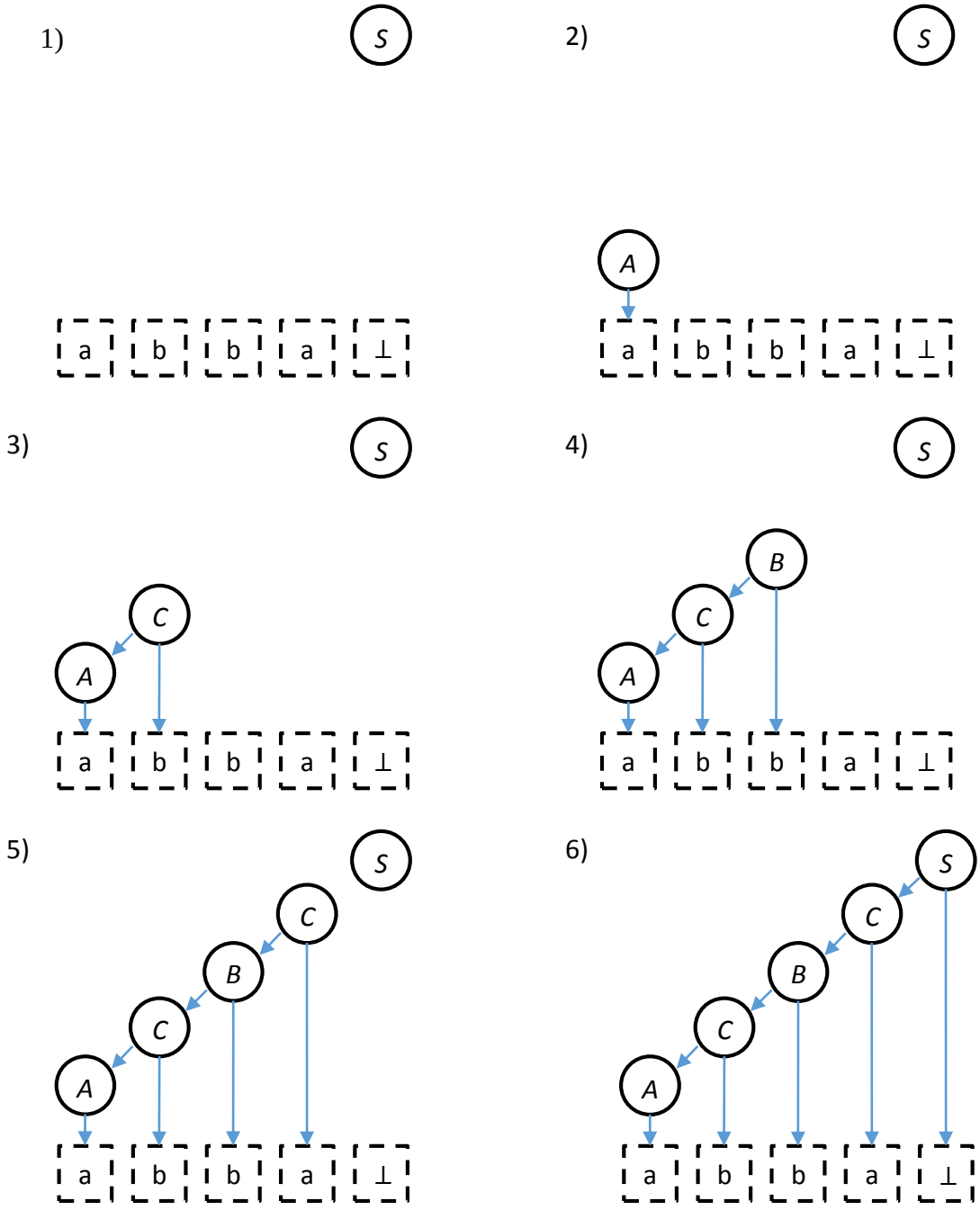

Рисунок 16. Дерево восходящего разбора

Цепочка $abba\ \perp$, которую мы уже рассматривали, здесь выводится путем последовательно замены каждого нетерминала сентенциальной формы на левую часть подходящего правила вывода. Таким образом, левый вывод указанной цепочки имеет вид

$$\boldsymbol{H} \to a\boldsymbol{A} \to ab\boldsymbol{C} \to abb\boldsymbol{B} \to abba\boldsymbol{C} \to abba\ \bot. \qquad (3.15)$$

Соответствующее данному выводу дерево разбора теперь строится сверху-вниз и показано на рис. 17.

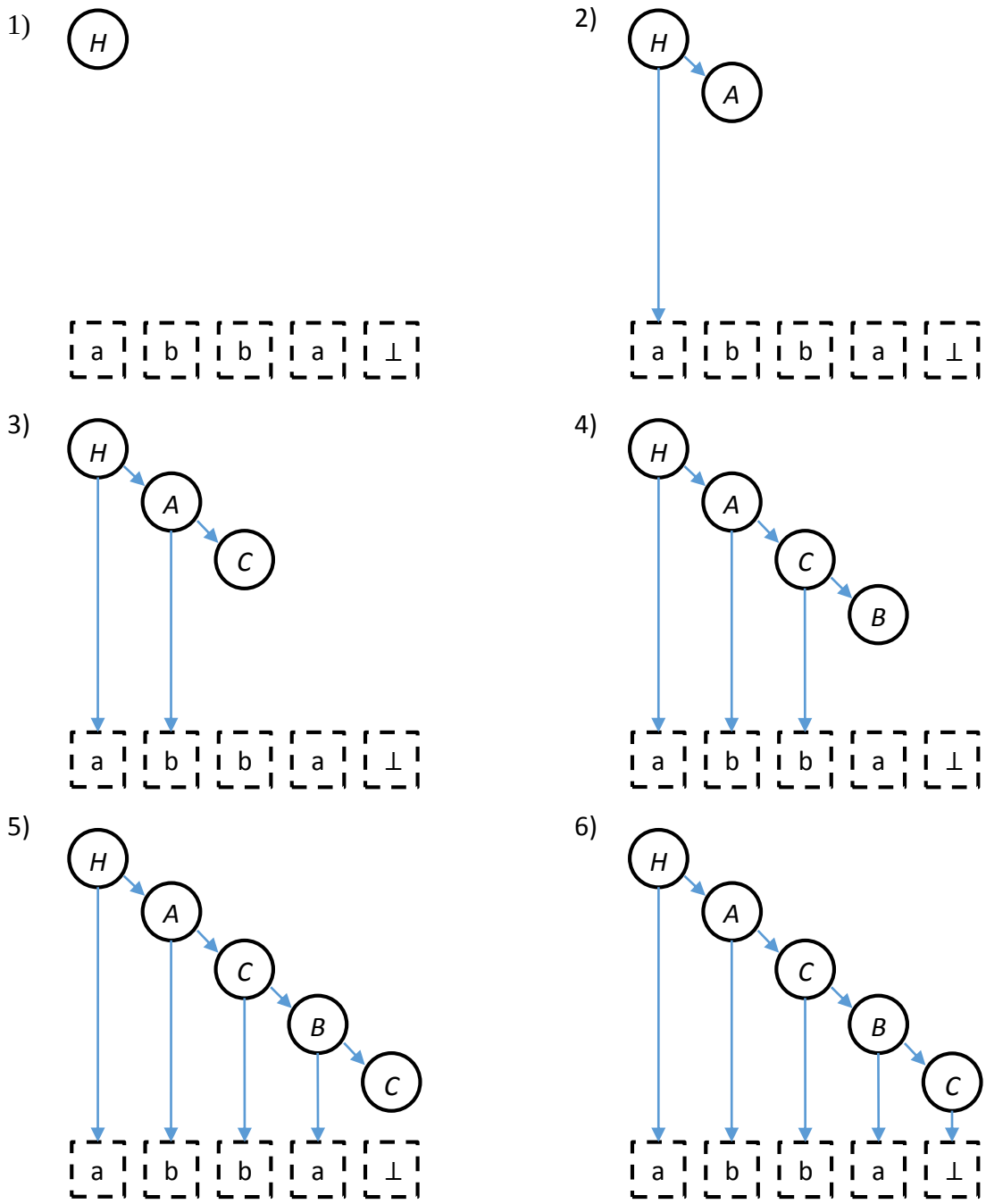

Рисунок 17. Дерево нисходящего разбора

Заметим, что грамматики G_l и G_r эквивалентны в том смысле, что описывают один и тот же язык.

Пример 32. Рассмотрим грамматику со следующими правилами вывода:

$$S \to A\ \bot;$$

$$A \to a|Bb;$$
$$B \to b|Bb.$$

Разбор строки по данной грамматике будет недетерминированным, поскольку разным нетерминалам слева (A и B) соответствуют одинаковые правые правила справа. Данной грамматике соответствует недетерминированный конечный автомат.

Разбор строки по НКА предполагает поиск всех возможных путей вплоть до нахождения успешного пути, где *успешный путь* – тот, который соответствует строке, принадлежащей языку. Однако, поиск всех успешных путей приводит к экспоненциальной сложности алгоритма. К счастью, как уже было сказано выше, НКА всегда можно преобразовать в ДКА.

Утверждение 6. Если язык L порожден некоторой регулярной грамматикой, то справедливо:

1) существует НКА, который допускает язык L;

2) существует ДКА, который допускает язык L.

3.3. Алгоритм преобразования НКА в ДКА

Вход: НКА $M = \{K, T, D, H, S\}$

Выход: ДКА $M_1 = \{K_1, T, D_1, H_1, S_1\}$

Как видим, четыре элемента в определении конечного автомата претерпели изменения. Неизменным остался только первичный алфавит T.

Принцип: объединять состояния НКА в одно состояние ДКА.

Алгоритм:

1. Все начальные состояния M объединить в одно начальное состояние M_1.

2. Для каждого терминала $t \in T$ добавить в множество состояний K_1 и множество переходов D_1 новое состояние, соответствующее переходу по символу t и начального состояния H_1 в множество возможных состояний M.

3. Выбрать заключительные состояния H_1 из множества заключительных состояний H.

Замечание. Следуя описанному алгоритму, может получиться несколько заключительных состояний. Это вызвано тем, что не всякий регулярный язык допускает детерминированный разбор с единственным заключительным состоянием. Поэтому во всех случаях в алфавите должен присутствовать маркер конца ввода, которым заканчивается каждая входная цепочка. Этим маркером может служить, например, признак конца файла (*EOF*, от англ. *end-of-file* – конец файла), символ конца строки и т.д. Обязательное наличие

маркера конца требует дополнительного состояния Q, в которое автомат должен может перейти из всех заключительных состояний S_1. Следовательно, все состояния из множества S_1 больше не считаются заключительными, а заключительным становится единственное состояние Q. Грамматика, равно как и перестроенный автомат, дополняются новым переходом

$$S_1 \to Q \perp, \tag{3.16}$$

и допускает детерминированный разбор.

Пример 33. Построить ДКА по НКА $M = \{\{H, A, B, S\}, \{0,1\}, D, \{H\}, \{S\}\}$, где

$$D(H, 1) = \{B\};$$
$$D(A, 1) = \{B, S\};$$
$$D(B, 0) = \{A\}.$$

Данный НКА допускает язык

$$L(M) = \{1(01)^n | n \geq 1\}, \tag{3.17}$$

и описывается регулярным выражением

$$101(01)^*. \tag{3.18}$$

Прежде всего выпишем правила вывода для данного НКА

$$S \to A1;$$
$$A \to B0;$$
$$B \to A1|1.$$

Получившаяся грамматика леволинейная. Очевидно, что разбор по данной грамматике будет недетерминированным, так как последовательности $A1$ соответствуют разные нетерминалы S и B. Диаграмма состояний для данного НКА представлена на рис. 18.

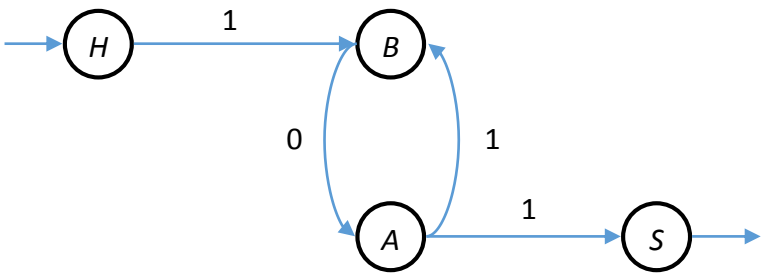

Рисунок 18. НКА для примера 33

Преобразуем недетерминированный конечный автомат в детерминированный по приведенному алгоритму.

Начальное состояние в НКА единственно, следовательно, оно становится начальным состоянием ДКА. Обозначим его H_1. Переход по единице (1) из состояния H в B детерминированный, следовательно, состояние B становится состоянием B_1. Переход по 0 из состояния B в состояние A также детерминированный. Из состояния A по символу 1 можно перейти, как в состояние B, так и в состояние S. Следовательно, введем новое состояние BS. Переход по 1 из состояния A ведет в BS. Переход по 0 из состояния BS ведет в A. Новая грамматика, соответствующая детерминированному автомату, такова:

$$S_1 \to A_1 1;$$
$$A_1 \to S_1 0 | B_1 0;$$
$$B_1 \to 1.$$

Здесь состояние BS переименовано в S_1. Получившейся грамматике соответствует конечный автомат $M_1 =< \{H_1, B_1, A_1, S_1\}, \{0,1\}, D_1, H_1, S_1 >$. Здесь функции переходов

$$D(H_1, 1) = B_1;$$
$$D(B_1, 0) = A_1;$$
$$D(A_1, 1) = S_1;$$
$$D(S_1, 0) = A_1.$$

Заключительным состоянием является S_1. Диаграмма состояний ДКА такова (рис. 19).

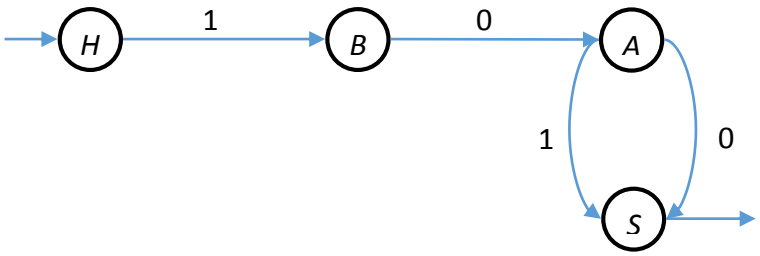

Рисунок 19. ДКА для примера 33

Пример 34. Пусть дана недетерминированная грамматика

$$S \to Sb | Aa | a;$$
$$A \to Aa | Sb | b.$$

Требуется построить эквивалентную детерминированную грамматику и соответствующий ей ДКА.

Выпишем все разрешенные переходы

$$D(H, a) = \{S\};$$
$$D(H, b) = \{A\};$$
$$D(A, a) = \{A, S\};$$
$$D(S, b) = \{A, S\}.$$

Воспользуемся таблицей для нахождения функции переходов ДКА. Построение таблицы начнем с состояния H. Начав с состояния H мы вводим два новых состояния S и A, в которые по символам a и b, соответственно, возможен переход.

Таблица 3

	$\boldsymbol{a}$	$\boldsymbol{b}$
$\boldsymbol{H}$	S	A
$\boldsymbol{S}$	$\emptyset$	AS
$\boldsymbol{A}$	AS	$\emptyset$
$\boldsymbol{AS}$	AS	AS

Введем новые обозначения состояний: $A \Rightarrow A, H \Rightarrow H, AS \Rightarrow Y, S \Rightarrow X$. Таким образом, получаем два заключительным состояния.

Добавим еще одно состояние S и введем маркер конца строки $\perp$. Это позволит свести два заключительных состояния в одно. Итоговая функция переходов ДКА такова:

$$D(H, a) = X;$$
$$D(H, b) = A;$$
$$D(X, b) = Y;$$
$$D(X, \perp) = S;$$
$$D(A, a) = Y;$$
$$D(Y, a) = Y;$$
$$D(Y, b) = Y;$$
$$D(Y, \perp) = S.$$

Данный конечный автомат допускает детерминированный разбор строк. Соответствующая ему детерминированная грамматика такова:

$$S \to X \perp | Y \perp;$$
$$Y \to Ya|Yb|Aa|Xb;$$
$$X \to a;$$
$$A \to b.$$