Лекція 4. Основы теории языков и формальных грамматик

Способы определения языков

Описание языков программирования во многом опирается на теорию формальных языков. Эта теория является фундаментом для организации синтаксического анализа и перевода.

Существует два основных способа определения языков:

- механизм порождения или генератор;
- механизм распознавания или распознаватель.

Они тесно связаны. Первый обычно используется для описания языков, а второй для их реализации. Оба способа позволяют описать языки конечным образом, несмотря на бесконечное число порождаемых ими цепочек.

Неформально язык **L** - это множество цепочек конечной длины в алфавите **T**. Механизм порождения позволяет описать языки с помощью системы правил, называемой грамматикой. Цепочки (предложения) языка строятся в соответствии с этими правилами. Достоинство определения языка с помощью грамматик в том, что операции, производимые в ходе синтаксического анализа и перевода, можно делать проще, если воспользоваться структурой, предписываемой цепочкам с помощью этих грамматик.

Механизм распознавания использует алгоритм, который для произвольной входной цепочки остановится и ответит "**да**" после конечного числа шагов, если эта цепочка принадлежит языку. Если цепочка не принадлежит языку, алгоритм ответит "**нет**". Распознаватели используются непосредственно при построении синтаксических анализаторов и являются как бы их формальной моделью. Распознаватели строятся на основе теорий конечных автоматов и автоматов с магазинной памятью.

Формальные грамматики

Грамматикой называется четверка **G = (N, T, P, S)**, где **N** - конечное множество нетерминальных символов (нетерминалов), **T** - множество терминалов (не пересекающихся с **N**), **S** - символ из **N**, называемый начальным, **P** - конечное подмножество множества:

$$(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*,$$

называемое множеством правил. Множество правил **P** описывает процесс порождения цепочек языка. Элемент $p_i = (\alpha, \beta)$ множества **P** называется правилом (продукцией) и записывается в виде α. Здесь αи β - цепочки, состоящие из терминалов и нетерминалов. Данная запись может читаться одним из следующих способов:

- цепочка α порождает цепочку β;

- из цепочки **α** выводится цепочка**β**.

Таким образом, правило **P** имеет две части: левую, определяемую, и правую, подставляемую. То есть правило $\mathbf{p_i}$ - это двойка ($\mathbf{p}_{i1}$, $\mathbf{p_{i2}}$), где p_{i1} = $\mathbf{(N \cup T)^* N (N \cup T)^*}$ - цепочка, содержащая хотя бы один нетерминал, $\mathbf{p_{i2}} = \mathbf{(N \cup T)^*}$ - произвольная, возможно пустая цепочка (**ε** - цепочка).

Если цепочка **α** содержит $\mathbf{p_{i1}}$, то, в соответствии с правилом $\mathbf{p_i}$, можно образовать новую цепочку **β**, заменив одно вхождение $\mathbf{p_{i1}}$ на $\mathbf{p_{i2}}$. Говорят также, что цепочка **β** выводится из **α** в данной грамматике.

Для описания абстрактных языков в определениях и примерах будем пользоваться следующими обозначениями:

- терминалы обозначим буквами a, b, c, d или цифрами 0, 1, ..., 9;
- нетерминалы будем обозначать буквами A, B, C, D, S (причем нетерминал S - начальный символ грамматики);
- буквы U, V, ..., Z используем для обозначения отдельных терминалов или нетерминалов;
- через **α**, **β**, **γ**... обозначим цепочки терминалов и нетерминалов;
- u, v, w, x, y, z - цепочки терминалов;
- для обозначения пустой цепочки (не содержащей ни одного символа) будем использовать знак **ε**;
- знак "→" будет отделять левую часть правила от правой и читаться как "порождает" или "есть по определению". Например, A→cd, читается как "A порождает cd".

Эти обозначения определяют некоторый язык, предназначенный для описания правил построения цепочек, а значит, для описания других языков. Язык, предназначенный для описания другого языка, называется **метаязыком**.

Пример грамматики G1:

G1 = ({A, S}, {0, 1}, P, S),

где P:

1. S → 0A1;

1. 0A → 00A1;
2. A → **ε**.

Выводимая цепочка грамматики **G**, не содержащая нетерминалов, **называется терминальной цепочкой**, порождаемой грамматикой **G**.

Язык **L(G)**, порождаемый грамматикой **G**, - это множество терминальных цепочек, порождаемых грамматикой **G**.

Введем отношение $\Rightarrow_G$ непосредственного вывода на множестве **(N ∪ T)***, которое будем

записывать следующим образом:

$$\mathbf{j} \Rightarrow_G \psi.$$

Данная запись читается: **ψ** непосредственно выводима из **φ** для грамматики **G = (N, T, P, S)** и означает: если **αβγ** - цепочка из множества **(N ∪ T)*** и **β → δ** - правило из **P** то **αβγ ⇒$_G$ αδγ**.

Через $\Rightarrow_G^+$ обозначим транзитивное замыкание (нетривиальный вывод за один и более шагов). Тогда $\mathbf{j} \Rightarrow_G^+ \psi$ читается как: **ψ** выводима из **φ** нетривиальным образом.

Через $\Rightarrow_G^*$ - обозначим рефлексивное и транзитивное замыкание (вывод за ноль и более шагов). Тогда $\mathbf{j} \Rightarrow_G^* \psi$ означает: **ψ** выводима из **φ**.

Пусть $\Rightarrow^k$ k - я степень отношения **⇒.** То есть, если $\mathbf{a} \Rightarrow^k \beta$, то существует последовательность $\alpha_0\alpha_1\alpha_2\alpha_3 \ldots \alpha_k$ из к+1 цепочек

$$\alpha = \alpha_0, \alpha_1, \ldots \alpha_{i-1} \Rightarrow a_i,\ 1 \le i \le k \text{ и } \alpha_k = \beta.$$

Пример выводов для грамматики G1:

S ⇒ 0A1 ⇒ 00A11 ч 0011;

S ⇒^{1} 0A1; S ⇒^{2} 00A11; S ⇒^{3} 0011;

S ⇒$^+$ 0A1; S ⇒$^+$ 00A11; S ⇒$^+$ 0011;

S ⇒* S; S ⇒* 0A1; S ⇒* 00A11; S ⇒* 0011;

где 0011 ∈ L(G1).

Грамматики с ограничениями на правила

Несмотря на большое разнообразие грамматик, при построении трансляторов нашли широкое применение только ряд из них, имеющих некоторые ограничения. Это связано с практической целесообразностью использования определенных типов правил, так как сложность их построения непосредственно влияет на сложность построения трансляторов. По виду правил выделяют несколько классов грамматик. В соответствии с классификацией Хомского грамматика **G** называется:

- **праволинейной**, если каждое правило из **P** имеет вид: **A→xB** или **A→x**, где **A**, **B** - нетерминалы, **x** - цепочка, состоящая из терминалов;
- **контекстно-свободной** (КС) или **бесконтекстной**, если каждое правило из **P** имеет вид: A→ **α**, где A ∈ N, а **α** ∈ (N ∪ T)*, то есть является цепочкой, состоящей из множества терминалов и нетерминалов, возможно пустой;
- **контекстно-зависимой** или **неукорачивающей**, если каждое правило из **P** имеет вид: **α → β**, где **|α| ≥ |β|.** То есть, вновь порождаемые цепочки не могут быть короче, чем исходные, а, значит, и пустыми (другие ограничения отсутствуют);
- **грамматикой свободного вида**, если в ней отсутствуют выше упомянутые

ограничения.

Пример праволинейной грамматики:

G2 = ({S,}, {0,1}, P, S), где

P:

1. S → 0S;
2. S → 1S;
3. S → **ε**,

определяет язык {0, 1}*.

Пример КС-грамматики:

G3 = ({E, T, F}, {a, +, *, (,)}, P, E) где

P:

1. E →T
2. E → E + T
3. T → F
4. T → T * F
5. F → (E)
6. F → a.

Данная грамматика порождает простейшие арифметические выражения.

Пример КЗ-грамматики:

G4 = ({B, C, S}, {a, b, c}, P, S) где

P:

1. S → aSBC;
2. S → abc;
3. CB → BC;
4. bB → bb;
5. bC → bc;
6. cC → cc,

порождает язык { $a^n b^n c^n$ }, n ≥ 1.

Примечание 1. Согласно определению каждая праволинейная грамматика является

контекстно- свободной.

Примечание 2. По определению КЗ-грамматика не допускает правил: A → **ε,** где **ε** - пустая цепочка. Т.е. КС-грамматика с пустыми цепочками в правой части правил не является контекстно-зависимой. Наличие пустых цепочек ведет к грамматике без ограничений.

Соглашение. Если язык L порождается грамматикой типа G, то L называется языком типа G.

Пример: L(G3) - КС язык типа G3.

Наиболее широкое применение при разработке трансляторов нашли КС-грамматики и порождаемые ими КС языки. В процессе изучения КС языков остановимся только на тех, которые будут полезны для нас с практической точки зрения (теория языков обширна и для детального ее изучения необходимо много времени). Те, кто желает приобрести более глубокие познания в данной области, могут обратиться к монографии Ахо и Ульмана [Axo78].

Способы записи синтаксиса языка

Существуют различные способы записи синтаксических правил, что в основном определяется условными обозначениям и ограничениями на структуру правил, принятыми в используемых метаязыках. Метаязыки используются для задания грамматики языков программирования со времен Алгола 60. Еще раньше они начали использоваться при описании небольших языков в в статьях, посвященных формальным грамматикам. Кратко рассмотрим основные вехи становления и развития метаязыков. Во всех случаях будем определять идентификатор.

Метаязык Хомского

Метаязык Хомского вышел из недр математической логики. Он имеет следующую систему обозначений:

- символ “→” отделяет левую часть правила от правой (читается как "порождает" и "это есть");
- нетерминалы обозначаются буквой **A** с индексом, указывающим на его номер;
- терминалы - это символы используемые в описываемом языке;
- каждое правило определяет порождение одной новой цепочки, причем один и тот же нетерминал может встречаться в нескольких правилах слева.

Описание идентификатора на метаязыке Хомского будет выглядеть следующим образом:

1. $A_1 \rightarrow$ A	**23. $A_1 \rightarrow$ W**	**45. $A_1 \rightarrow$ s**
2. $A_1 \rightarrow$ B	**24. $A_1 \rightarrow$ X**	**46. $A_1 \rightarrow$ t**
3. $A_1 \rightarrow$ C	**25. $A_1 \rightarrow$ Y**	**47. $A_1 \rightarrow$ u**
4. $A_1 \rightarrow$ D	**26. $A_1 \rightarrow$ Z**	**48. $A_1 \rightarrow$ v**
5. $A_1 \rightarrow$ E	**27. $A_1 \rightarrow$ a**	**49. $A_1 \rightarrow$ w**

6. $A_1 \rightarrow F$	**28. $A_1 \rightarrow b$**	**50. $A_1 \rightarrow x$**
7. $A_1 \rightarrow G$	**29. $A_1 \rightarrow c$**	**51. $A_1 \rightarrow y$**
8. $A_1 \rightarrow H$	**30. $A_1 \rightarrow d$**	**52. $A_1 \rightarrow z$**
9. $A_1 \rightarrow I$	**31. $A_1 \rightarrow e$**	**53. $A_2 \rightarrow 0$**
10. $A_1 \rightarrow J$	**32. $A_1 \rightarrow f$**	**54. $A_2 \rightarrow 1$**
11. $A_1 \rightarrow K$	**33. $A_1 \rightarrow g$**	**55. $A_2 \rightarrow 2$**
12. $A_1 \rightarrow L$	**34. $A_1 \rightarrow h$**	**56. $A_2 \rightarrow 3$**
13. $A_1 \rightarrow M$	**35. $A_1 \rightarrow i$**	**57. $A_2 \rightarrow 4$**
14. $A_1 \rightarrow N$	**36. $A_1 \rightarrow j$**	**58. $A_2 \rightarrow 5$**
15. $A_1 \rightarrow O$	**37. $A_1 \rightarrow k$**	**59. $A_2 \rightarrow 6$**
16. $A_1 \rightarrow P$	**38. $A_1 \rightarrow l$**	**60. $A_2 \rightarrow 7$**
17. $A_1 \rightarrow Q$	**39. $A_1 \rightarrow m$**	**61. $A_2 \rightarrow 8$**
18. $A_1 \rightarrow R$	**40. $A_1 \rightarrow n$**	**62. $A_2 \rightarrow 9$**
19. $A_1 \rightarrow S$	**41. $A_1 \rightarrow o$**	**63. $A_3 \rightarrow A_1$**
20. $A_1 \rightarrow T$	**42. $A_1 \rightarrow p$**	**64. $A_3 \rightarrow A_3A_1$**
21. $A_1 \rightarrow U$	**43. $A_1 \rightarrow q$**	**65. $A_3 \rightarrow A_3A_2$**
22. $A_1 \rightarrow V$	**44. $A_1 \rightarrow r$**	

Метаязык Хомского-Щутценберже

Приведенный в предыдущем разделе пример описания идентификатора показывает громоздкость метаязыка Хомского, что позволяет эффективно использовать его только для описания небольших абстрактных языков. Более компактное описание возможно с применением метаязыка Хомского-Щутценберже, использующего следующие обозначения метасимволов:

- символ **“=”** отделяет левую часть правила от правой (вместо символа “→”);
- нетерминалы обозначаются буквой A с индексом, указывающим на его номер;
- терминалы - это символы используемые в описываемом языке;
- каждое правило определяет порождение нескольких альтернативных цепочек, отделяемых друг от друга символом “+”, что позволяет, при желании, использовать в левой части только разные нетерминалы.

Введение возможности альтернативного перечисления позволило сократить описание языков. Описание идентификатора будет выглядеть следующим образом:

1. A_1=A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+ U+V+W+X+Y+Z+a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+ r+s+t+u+v+w+x+y+z
2. A_2=0+1+2+4+5+6+7+8+9
3. A_3=A_1+A_3A_1+A_3A_2

Бэкуса-Наура формы (БНФ)

Метаязыки Хомского и Хомского-Щутценберже использовались в математической литературе при описании простых абстрактных языков. Метаязык, предложенный Бэкусом и Науром, впервые использовался для описания синтаксиса реального языка программирования Алгол 60. Наряду с новыми обозначениями метасимволов, в нем использовались содержательные обозначения нетерминалов. Это сделало описание языка нагляднее и позволило в дальнейшем широко использовать данную нотацию для описания реальных языков программирования. Были использованы следующие обозначения:

- символ "::=" отделяет левую часть правила от правой;
- нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки "<" и ">";
- терминалы - это символы, используемые в описываемом языке;
- каждое правило определяет порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты "|".

Пример описания идентификатора с использованием БНФ:

1. <буква> :: = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V| W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
2. <цифра> :: = 0|1|2|3|4|5|6|7|8|9
3. <идентификатор> ::= <буква> | <идентификатор><буква> | <идентификатор><цифра>

Правила можно задавать и раздельно:

3. <идентификатор> :: = <буква>
4. <идентификатор> :: = <идентификатор> <буква>
5. <идентификатор> :: = <идентификатор> <цифра>

Расширенные Бэкуса-Наура формы (РБНФ)

Метаязыки, представленные выше, позволяют описывать любой синтаксис. Однако, для повышения удобства и компактности описания, целесообразно вести в язык дополнительные конструкции. В частности, специальные метасимволы были разработаны для описания необязательных цепочек, повторяющихся цепочек, обязательных альтернативных цепочек. Существуют различные расширенные формы метаязыков, незначительно отличающиеся друг от друга. Их разнообразие зачастую объясняется желанием разработчиков языков программирования по-своему описать создаваемый язык. К примерам таких широко известных метаязыков можно отнести: метаязык PL/I, метаязык Вирта, используемый при описании Модулы-2, метаязык Кернигана-Ритчи, описывающий Си. Зачастую такие языки называются

расширенными формами Бэкуса-Наура (РБНФ).

В частности, РБНФ, используемые Виртом, имеют следующие особенности:

- Квадратные скобки "**[**" и "**]**" означают, что заключенная в них синтаксическая конструкция может отсутствовать;
- фигурные скобки "**{**" и "**}**" означают ее повторение (возможно, 0 раз);
- круглые скобки "**(**" и "**)**" используются для ограничения альтернативных конструкций;
- сочетание фигурных скобок и косой черты "**{/**" и "**/}**" используется для обозначения повторения один и более раз. Нетерминальные символы изображаются словами, выражающими их интуитивный смысл и написанными на русском языке.

Если нетерминал состоит из нескольких смысловых слов, то они должны быть написаны слитно. В этом случае для повышения удобства в восприятии фразы целесообразно каждое ее слово начинать с заглавной буквы или разделять слова во фразах символом подчеркивания. Терминальные символы изображаются словами, написанными буквами латинского алфавита (зарезервированные слова) или цепочками знаков, заключенными в кавычки. Синтаксическим правилам предшествует знак "**$**" в начале строки. Каждое правило оканчивается знаком "**.**" (точка). Левая часть правила отделяется от правой знаком "**=**" (равно), а альтернативы - вертикальной чертой "|". Этот вариант РБНФ и будет использоваться для описания синтаксиса языков в лабораторной работе. В соответствии с данными правилами синтаксис идентификатора будет выглядеть следующим образом:

$ буква = "A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"|"O"|"P"|"Q"|"R"|
"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"|"a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"n"|
"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z".

$ цифра = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".

$ идентификатор = буква {буква | цифра}.

Диаграммы Вирта

Наряду с текстовыми способами описания синтаксиса языков широко используются и графические метаязыки, среди которых наиболее широкую известность получил язык диаграмм Вирта, впервые примененный для описания языка Паскаль. Метасимволы заменены следующими графическими обозначениями (рис. 2.1):

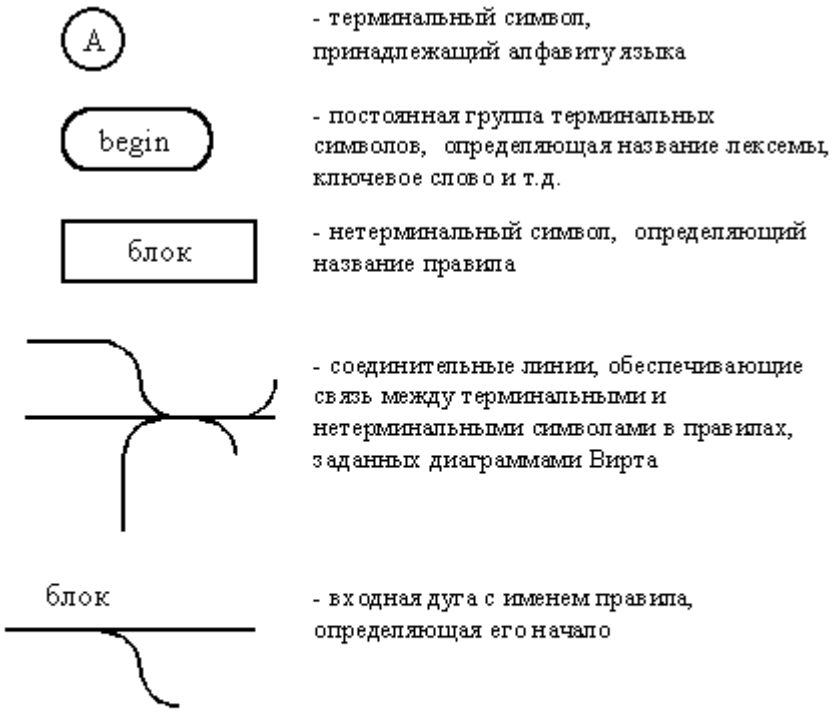

Рис. 2.1. Графические примитивы, используемые при построении диаграмм Вирта.

- терминальные символы и их постоянные группы располагаются в окружностях или прямоугольниках со скругленным вертикальными сторонами;
- нетерминальные символы заносятся внутрь прямоугольников;
- каждый графический элемент, соответствующий терминалу или нетерминалу, имеет по одному входу и выходу, которые обычно рисуются на противоположных сторонах;
- каждому правилу соответствует своя графическая диаграмма, на которой терминалы и нетерминалы соединяются посредством дуг;
- альтернативы в правилах задаются ветвлением дуг, а итерации - их слиянием;
- должна быть одна входная дуга (располагается обычно слева и сверху), задающая начало правила и помеченная именем определяемого нетерминала, и одна выходная, задающая его конец (обычно располагается справа и снизу).

Пример описания идентификатора с использованием диаграмм Вирта представлен на рис 2.2.

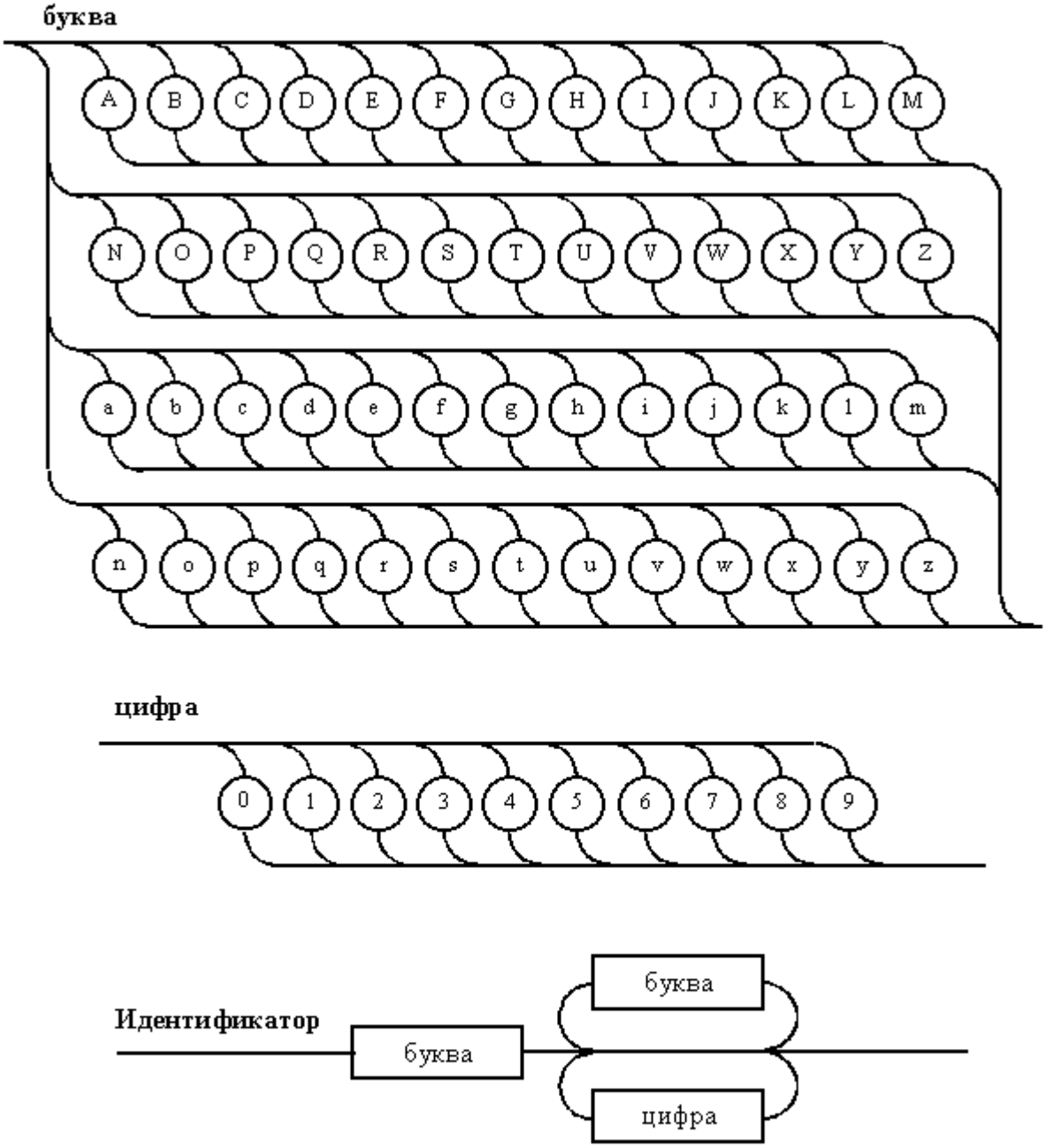

Рис. 2.2. Описание идентификатора с использованием диаграмм Вирта.

Обычно стрелки на дугах диаграмм не ставятся, а направления связей отслеживаются движением от начальной дуги в соответствии с плавными изгибами промежуточных дуг и ветвлений. Таким же образом определяются входы и выходы терминалов и нетерминалов. Специальных стандартов на диаграммы Вирта нет, поэтому графические обозначения могут меняться в зависимости от средств рисования. Можно, например, использовать псевдографику или просто текстовые символы, связи со стрелками. Однако такой вид правил менее удобен для восприятия и поэтому применяется крайне редко.

Диаграммы Вирта позволяют задавать альтернативы, рекурсии, итерации и по

изобразительной мощности эквивалентны РБНФ. Но графическое отображение правил более наглядно. Кроме этого допускается произвольное проведение дуг, что уменьшает количество элементов в правиле за счет его неструктурированности. Диаграммы Вирта являются удобным исходным документом для построения лексического и синтаксического анализаторов.

Распознаватели

Распознаватель – это очень схематизированный алгоритм, определяющий некоторое множество. Его можно представить в виде устройства (автомата). Этот автомат состоит из трех частей: входной ленты, устройства управления с конечной памятью и вспомогательной (рабочей) памяти (рис 2.3).

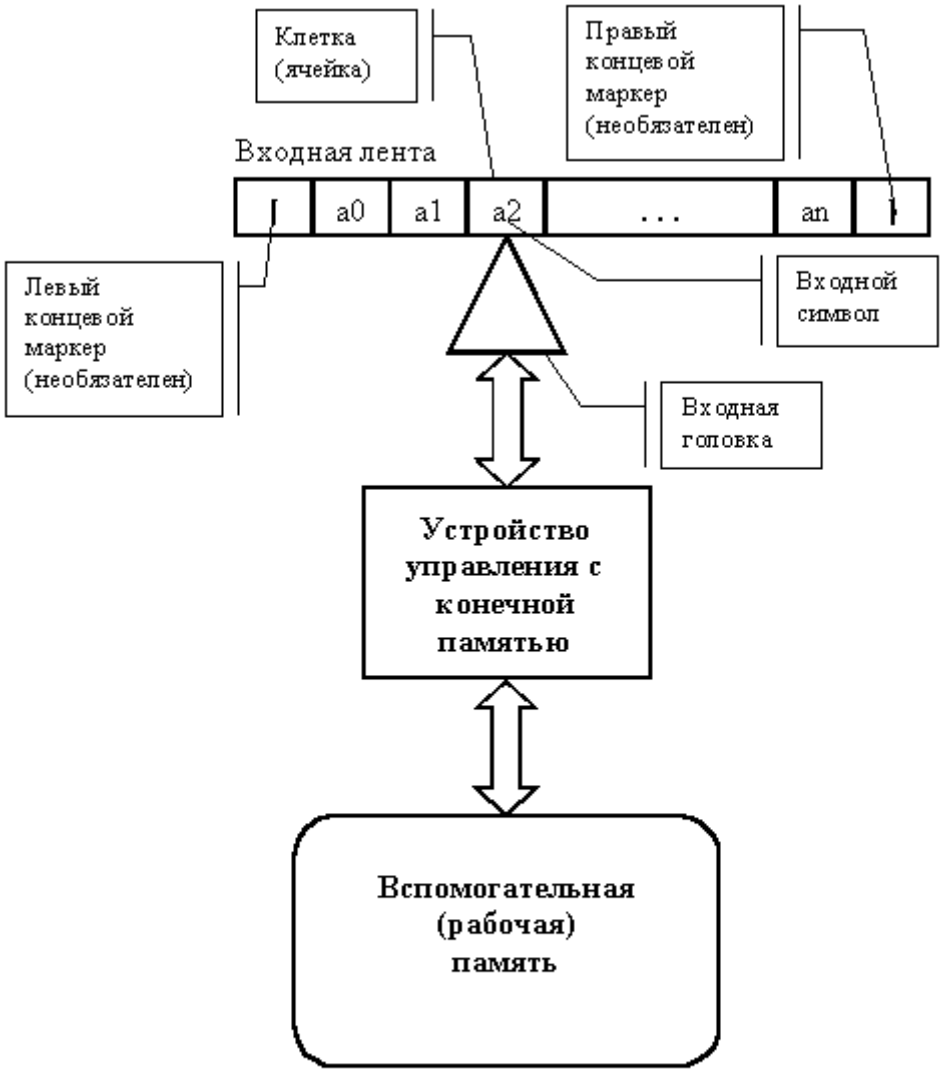

Рис. 2.3. Обобщенная структура распознавателя

Входная лента – линейная последовательность клеток (ячеек), каждая из которых содержит один входной символ из конечного входного алфавита. Могут присутствовать левый и правый

концевые маркеры, может присутствовать только один концевой маркер (левый или правый), могут отсутствовать оба маркера.

Входная головка – в каждый момент читает одну входную ячейку. За один шаг входная головка может сдвинуться на одну ячейку влево, вправо и остаться неподвижной.

Распознаватель, никогда не передвигающий входную головку влево, называется **односторонним**. Обычно предполагается, что входная головка только читает. Но могут быть такие распознаватели, у которых входная головка и читает, и пишет.

Память – хранит информацию, построенную только из символов конечного алфавита памяти. Может иметь различную структуру: очередь, стек (магазин) и т. д. Можно читать из вспомогательной памяти и писать в нее. Для стека и очереди используются специфические операции (вталкивание, выталкивание).

Устройство управления с конечной памятью – программа, управляющая поведением распознавателя. Может являться аналогом конечного автомата. Определяет перемещение входной головки и работу с памятью на каждом шаге (такте). Переходит за шаг из одного состояния в другое.

Конфигурация распознавателя – мгновенный снимок, на котором изображены:

1. состояние устройства управления;
2. содержимое входной ленты;
3. содержимое памяти.

Начальная конфигурация – устройство управления находится в заданном начальном состоянии, входная головка читает самый левый символ на входной ленте, память имеет заранее установленное начальное содержимое.

Заключительная конфигурация – устройство управления находится в одном из состояний, принадлежащем заранее выделенному множеству заключительных состояний, входная головка обозревает правый концевой маркер или, если маркер отсутствует, сошла с конца входной ленты. Иногда требуется, чтобы заключительная конфигурация памяти удовлетворяла некоторым условиям.

Распознаватель **допускает** входную цепочку **ω**, если, начиная с начальной конфигурации, в которой цепочка **ω** записана на входной ленте, распознаватель может проделать последовательность шагов, заканчивающуюся заключительной конфигурацией.

Язык, определяемый распознавателем – это множество цепочек, которые он допускает.

Для каждой из грамматик, приведенных выше в соответствии с иерархией Хомского, существуют распознаватели определяющие один и тот же класс языков.

1. Язык L **праволинейный** тогда и только тогда, когда он определяется конечным, односторонним детерминированным автоматом.
2. Язык L **контекстно-свободный** тогда и только тогда, когда он определяется односторонним недетерминированным автоматом с магазинной памятью.

3. Язык L **контекстно-зависимый** тогда и только тогда, когда он определяется двухсторонним недетерминированным линейно ограниченным автоматом.
4. Язык L **рекурсивно перечислимый** тогда и только тогда, когда он определяется машиной Тьюринга.

Данные определения показывают, что теория языков и формальных грамматик продвинулась достаточно далеко, чтобы служить самостоятельным предметом изучения. Не будем вдаваться в детали и выяснять смысл представленных понятий. Зафиксируем только сам факт эквивалентности между механизмами порождения и распознавания.