которой эквивалентно поведению всей системы. Декомпозиция особенно необходима при проектировании моделей, элементы носителей которых взвешены некоторыми функциями, например при проектировании логических схем из функциональных элементов. Введем несколько понятий.

Сетью называется граф $G = \langle V, U \rangle$ без циклов, в котором выделено два подмножества вершин V^+ и V^-, $V^+ \cap V^- = \emptyset$ таких, что дуги, инцидентные вершине v_α, $v_\alpha \in V^+$, только исходят из вершины v_α, дуги, инцидентные вершине v_β, $v_\beta \in V^-$, только входят в вершину v_β и для каждой вершины v_γ, $v_\gamma \in V - V^+ \cup V^-$) существуют дуги как исходящие, так и входящие в вершины v_γ.

Сеть называется *булевой*, если все ее дуги однозначно взвешены первичными термами x_i, $\bar{x}_i$.

Далее, последовательность вершин $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ $(k > 1)$ диаграммы Хассе, i-я $(i > 1)$ вершина которой покрывает $(i-1)$-ю и только $(i-1)$-ю вершину, называется π-поддиаграммой G^π (рис. 15-19).

Понятие покрытия вершин предполагается интуитивно очевидным. Совокупность вершин $v_{\gamma_1}, v_{\gamma_2}, \ldots, v_{\gamma_t}$ $(t > 1)$ диаграммы Хассе, покрываемых одними и только одними и теми же вершинами и покрывающих одни и только одни и те же вершины, называется σ-поддиаграммой G^δ (рис. 15-20).

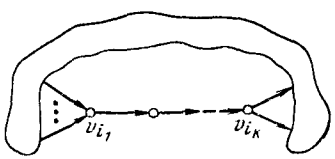


Рис. 15-19. π-поддиаграмма.

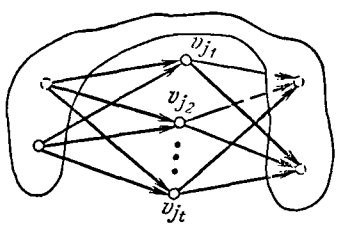


Рис 15-20. σ-поддиаграмма.

Очевидно, что при поиске декомпозиции естественно применять обратные преобразования, оформленные в виде какой-либо коалгебры, а выполнение преобразований в коалгебрах сводится к решению уравнений. *Коалгеброй графов* назовем

$$K = \langle M, \varkappa^\vee, \varkappa^- \rangle,$$

где носителем M является множество всевозможных диаграмм Хассе, а сигнатурой — кооперация дизъюнкции $\varkappa^\vee$ диаграмм графов и кооперация отрицания диаграмм. Для краткости в дальнейшем эти две кооперации будем называть соответственно операцией разложения $\varkappa^\vee$ и операцией инверсии $\varkappa^-$ диаграмм.

При *разложении* диаграмм указываются направления, по которым происходит разложение диаграммы H. Направление задается вершинами, являющимися максимальными элементами для подграфов диаграммы H. Операция разложения $\varkappa^\vee(H)$ графа H по направлениям V_i $(i = 1, 2, \ldots, k)$ есть выделение соответственно подграфов H_i, каждый из которых состоит из вершин $\{v_j, j = 1, 2, \ldots, l\}$ и соединяющих их дуг, для которых в графе H найдется путь, соединяющий

$$v_\alpha \in \{v_j, j = 1, 2, \ldots, l\} \text{ и } v_\beta = V_i.$$

Например, результат операции разложения диаграммы Хассе H по направлениям $\bar{x}_4$ и x_4 представлен на рис. 15-21.

Покажем, как выполняется *инверсия* диаграмм. Диаграмма, представляемая в виде суперпозиции диаграмм типа π и σ, называется диаграммой типа $\pi\sigma$.

Операция инверсии $\varkappa^-(H)$ диаграммы H есть приведение этого графа с помощью его разложения к графу типа $\pi\sigma$, замена каждого подграфа типа π подграфом типа σ и каждого подграфа типа σ подграфом типа π; при этом каждый вес

$x_i^{\sigma_i}$ первоначальной диаграммы H заменяется весом $x^{(\sigma_i+1) \bmod 2}$ на соответствующих вершинах полученной диаграммы.

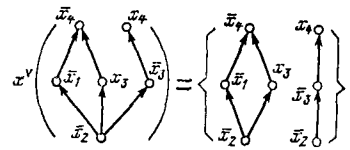


Рис. 15-21. Разложение диаграммы.

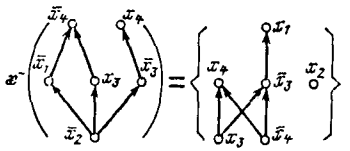


Рис. 15-22. Инверсия диаграммы.

Рисунок 15-22 является иллюстрацией операции $\varkappa^-$:

$$H_2 = \pi(\bar{x}_2, \sigma(\pi(x_3, x_4), \pi(\bar{x}_4, \sigma(x_1, x_3)))) \; \varkappa^-(H_2) =$$
$$= \sigma(x_2, \pi(\sigma(x_3, \bar{x}_4), \sigma(x_4, \pi(x_1, \bar{x}_3)))).$$

Введенные операции разложения $\varkappa^\vee$ и инверсии $\varkappa^-$ являются обратными соответственно операциям дизъюнкции и отрицания, выполняемым над функциями, соответствующими диаграммам Хассе.

ГЛАВА ШЕСТНАДЦАТАЯ

АВТОМАТНО-ЛИНГВИСТИЧЕСКИЕ МОДЕЛИ

При изучении процессов преобразования и переработки информации с целью управления в кибернетике появился большой интерес к естественным языкам, к их математическим описаниям, моделям, к методам традиционной и математической лингвистики.

Дело в том, что в процессе развития естественного языка, как некоторой информационной системы, было выработано большое количество рациональных правил, процедур, структур обработки лингвистической информации, которые при внимательном исследовании оказались очень плодотворными в кибернетике. При этом следует иметь в виду, что, как показали исследования в области моделирования естественного интеллекта или систем принятия решений, естественный язык является очень удобной средой для моделирования процессов мышления.

В широко используемых и ставших уже классическими автоматных кибернетических моделях стала развиваться лингвистическая концепция, согласно которой автомат представляется в виде некоторого устройства, определяющего допустимость подаваемых на его вход слов в соответствии с заложенными в него правилами. Иными словами, автоматы могут быть интерпретированы как распознающие устройства, которые определяют допустимость входных слов с позиций заложенных в них грамматик. Благодаря этому появились возможности рассматривать автоматные и лингвистические модели совместно и называть их автоматно-лингвистическими моделями. При построении этих моделей, как было изложено в пре-

дыдущей главе, используется алгебраический подход, в частности, алгебраический аппарат полусистем Туэ.

В начале данной главы будет подробно изложен лингвистический метод, затем будут описаны лингвистические модели и, наконец, с позиций лингвистики будут рассмотрены автоматные модели. При этом постоянно будет использоваться концепция распознавания в смысле кибернетических моделей распознавания образов, в особенности моделей структурного распознавания (см. гл. 12).

16-1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ФОРМАЛЬНЫХ ГРАММАТИК

Автоматные и лингвистические модели строятся на базе теории формальных грамматик, основы которой были заложены в фундаментальных работах Н. Хомского [32, 54].

Основными объектами, с которыми имеет дело эта теория, являются символы, представляющие собой базовые элементы какого-либо непустого множества A любой природы, а также цепочки, построенные из этих элементов. Символы будем обозначать строчными буквами латинского алфавита, а цепочки — в виде

$$ffghhh,$$

которые будем считать ориентированными слева направо. Цепочки будем обозначать также специальными символами — прописными буквами латинского алфавита или греческими буквами, например:

$$\gamma = ffg,$$
$$B = abba.$$

Введем в рассмотрение пустую цепочку ε, не содержащую ни одного символа.

Длиной цепочки будем называть число символов, входящих в эту цепочку, обозначаемых следующим образом:

$$|\gamma| = |ffg| = 3,$$
$$|B| = |abba| = 4,$$
$$|\varepsilon| = 0.$$

Конкатенацией двух цепочек X и Y называется такая цепочка Z, которая получается непосредственным слиянием цепочки X, стоящей слева, и цепочки Y, стоящей справа. Например, если $X = ffg$, $Y = ghh$, то конкатенация X и Y — это цепочка $Z = ffgghh$. Обозначим операцию конкатенации символом $\circ$. Свойства этой операции можно записать следующим образом:

1) свойство замкнутости:

$$\circ: A^* \times A^* \to A^*;$$

2) свойство ассоциативности:

$$(\forall X \in A^*,\ Y \in A^*,\ Z \in A^*)$$
$$[(X \circ Y) \circ Z = X \circ (Y \circ Z)],$$

где через A * обозначено множество всех возможных цепочек (разумеется, бесконечное) конечного множества A базовых элементов (символов), включая пустую цепочку ε; символ $\times$ обозначает операцию декартова произведения двух множеств; а X, Y, Z — произвольные цепочки, принадлежащие A *.

Рассмотрим пару $(A^*, \circ)$. С учетом перечисленных свойств операции $\circ$ эта пара представляет собой полугруппу с единичным элементом ε или моноид. Полугруппой в алгебре называют также только множество (в данном случае A*), снабженное всюду определенной ассоциативной операцией.

Множество A называют также *алфавитом*. Любое множество цепочек $L \subseteq A^*$, где A^* — моноид, называется *формальным языком*, определенным на алфавите A.

Пример 16-1. Пусть A — множество букв русского алфавита. Тогда множество цепочек, составленных из пяти букв, представляет собой формальный язык L_1. Другой пример языка, определенного на том же алфавите — множество L_2 пятибуквенных слов русского языка, которые можно разыскать в орфографическом словаре. Очевидно, что $L_2 \subset L_1$, так как многие цепочки языка L_1 не являются русскими словами.

Пусть B и C — некоторые подмножества множества A^*.

Произведением множеств[1] B и C называется множество D цепочек, являющихся конкатенацией цепочек из B и C, т. е.

$$D = \{X \circ Y \mid X \in B,\ Y \in C\}.$$

Обозначается произведение следующим образом:

$$D = BC.$$

Рассмотрим алфавит A. Обозначим множество, состоящее из ε, через A^0. Определим степень алфавита как

$$A^n = A^{n-1}A$$

для каждого $n \geqslant 1$.

Нетрудно показать, что множество всех возможных цепочек алфавита

$$A^* = \bigcup_{n=0}^{\infty} A^n.$$

Такое множество называют также *итерацией* алфавита A. *Усеченной итерацией* алфавита A называют

$$A^+ = \bigcup_{n=1}^{\infty} A^n.$$

Если X и Y — цепочки множества A^*, то цепочку X называют *подцепочкой* цепочки Y, когда существуют такие цепочки U и V из A^*, что

$$Y = U \circ X \circ V.$$

При этом, если U — пустая цепочка, то подцепочку X называют *головой* цепочки Y, а если V — пустая цепочка, то X называют *хвостом* цепочки Y.

Для обозначения конкатенации двух цепочек X и Y вместо $X \circ Y$ будем писать XY.

Рассмотрим пары цепочек (P_1, Q_1), (P_2, Q_2), ..., (P_n, Q_n) из $A^* \times A^*$.

---

[1] Не путать с декартовым произведением, элементами которого в данном случае были бы не цепочки, а упорядоченные пары цепочек (см. § 15-3).

Соотношениями Туэ (названными так в честь норвежского математика, впервые их исследовавшего) будем называть правила, согласно которым любой цепочке $X = UP_iV$ из множеств A* будет ставиться в соответствие цепочка $Y = UQ_iV$ из того же множества A* ($i = 1, 2, \ldots, n$) и наоборот. Эти соотношения приводят к так называемым *ассоциативным исчислениям.*

Если цепочка Y получается из цепочки X однократным применением одного соотношения Туэ (т. е. заменой подцепочки P_i на подцепочку Q_i), будем говорить, что X и Y являются *смежными цепочками.*

Цепочка X_n *соотносима* с цепочкой X_0, если существует последовательность цепочек

$$X_0,\ X_1,\ \ldots,\ X_n$$

такая, что X_{i-1} и X_i являются смежными цепочками ($i = 1, 2, \ldots, n$).

Пример 16-2. Пусть A — множество букв русского алфавита, на котором определим соотношение Туэ, заключающееся в праве замены любой одной буквы слова на любую другую.

Тогда в последовательности цепочек МУКА, МУЗА, ЛУЗА, ЛОЗА, ПОЗА, ПОРА, ПОРТ, ТОРТ две любые соседние цепочки являются смежными, а цепочки МУКА и ТОРТ являются соотносимыми в смысле заданных соотношений.

Выше было дано самое общее определение формального языка как любого подмножества A*, где A — алфавит. Введение соотношений Туэ позволяет выделить среди множества языков определенные их классы, которые используются при построении автоматно-лингвистических моделей самого различного типа.

Пример 16-3. В языках программирования, а также в различных математических исчислениях широко используется запись с помощью скобок. При этом выражение правильно только в том случае, если для каждой левой скобки в выражении присутствует соответствующая ей правая скобка. Рассмотрим язык, который получается, если из всех выражений данного языка удалить все символы за исключением символов скобок. Пусть имеется n различных сортов таких скобок (круглых, квадратных, фигурных и т. д.).

Цепочка X принадлежит *ограниченному языку Дика* [63], если она соотносима с цепочкой ε (пустой цепочкой) в смысле соотношений Туэ:

$$(\varepsilon,\ l_1p_1),\ (\varepsilon,\ l_2p_2),\ \ldots,\ (\varepsilon,\ l_np_n),$$

где l_i обозначает левую скобку i-го сорта, а p_i — соответствующую ей правую скобку. Легко видеть, что, например, цепочка

$$l_1l_2p_2p_1l_3p_3$$

принадлежит к ограниченному языку Дика, так как приводится к пустой цепочке последовательным применением данных соотношений:

$$l_1l_2p_2p_1l_3p_3 \to l_1l_2p_2p_1 \to l_1p_1 \to \varepsilon.$$

Соотношения Туэ являются двусторонними, если цепочка X является смежной по отношению к цепочке Y, и наоборот, цепочка Y является смежной по отношению к цепочке X. Более интересными с точки зрения теории формальных грамматик являются соотношения, в которых введено *направление.*

В этом случае их называют *полусоотношениями* Туэ или продукциями и обозначают следующим образом:

$$(P_1 \to Q_1),\ (P_2 \to Q_2),\ \ldots,\ (P_n \to Q_n).$$

В том случае, когда имеется набор продукций, говорят, что цепочка Y непосредственно порождается из цепочки X, и обозначается это как

$$X \Longrightarrow Y,$$

если существуют такие цепочки U и V, что

$$X = UP_iV,$$
$$Y = UQ_iV,$$

а $(P_i \to Q_i)$ — продукция из данного набора.

Говорят также, что X порождает Y.

Если существует последовательность цепочек $X_0, X_1, \ldots, X_n$ такая, что для каждого $i = 1, 2, \ldots, n$

$$X_{i-1} \Longrightarrow X_i,$$

говорят, что X_n порождается из X_0 (X_0 порождает X_n), и обозначают это как

$$X_0 \Longrightarrow {}^*X_n.$$

Грамматики Хомского, о которых идет речь в следующем параграфе, соответствуют формальным комбинаторным системам, являющимся полусистемами Туэ, в основу которых положены полусоотношения Туэ (продукции).

16-2. КЛАССИФИКАЦИЯ ГРАММАТИК ПО ХОМСКОМУ

Теория формальных языков (формальных грамматик) занимается описанием, распознаванием и переработкой языков (см. § 15-1). Описание любого языка должно быть конечным, хотя сам язык может содержать бесконечное множество цепочек. Полезно иметь возможность описания отдельных типов языков, имеющих те или иные свойства, т. е. иметь различные типы конечных описаний. Предположим, что имеется некоторый класс языков $\mathscr{L}$, который задается определенным типом описания. Теория формальных языков позволяет ответить на ряд вопросов, возникающих во многих прикладных задачах, в которых используются автоматно-лингвистические модели. Например, могут ли языки из класса $\mathscr{L}$ распознаваться быстро и просто; принадлежит ли данный язык классу $\mathscr{L}$ и т. д. Важной проблемой является построение алгоритмов, если они существуют, которые давали бы ответы на определенные вопросы о языках из класса $\mathscr{L}$, например: «Принадлежит или нет к языку L цепочка X?»

Существуют два основных способа описания отдельных классов языков. Первый из них основан на ограничениях, которые налагаются на систему полусоотношений Туэ (продукций), на базе которых определяются *грамматики* как механизмы, порождающие цепочки символов. Другим способом является определение языка в терминах множества цепочек, допускаемых некоторым распознающим устройством. Такие устройства будем называть *автоматами.* В настоящем параграфе будет рассматриваться первый из этих способов.

Н. Хомский [32, 54] определил четыре типа грамматик, служащих до сих пор образцом, по которому оцениваются возможности других способов описания.

Назовем алфавит символов (непустое конечное множество), из которых строятся цепочки языка L, алфавитом терминальных символов V_T. Очевидно, что $L \subseteq V_T^*$.

Определение формальной грамматики требует наличия еще одного алфавита V_N — непустого конечного множества нетерминальных символов ($V_N \cap V_T = \phi$). Объединение этих алфавитов назовем словарем формального языка L:

$$V = V_N \cup V_T.$$

Условимся обозначать элементы алфавита V_T строчными латинскими буквами, элементы множества V_N — прописными латинскими буквами, элементы словаря V^* (цепочки символов словаря) — греческими буквами.

Определим также множество упорядоченных пар (полутуэвских соотношений) следующим образом:

$$\Pi = \{(\alpha,\ \beta) \mid \alpha \in V^* V_N V^* \wedge \beta \in V^+\}.$$

Каждая пара (α, β) называется продукцией и обозначается как

$$\alpha \rightarrow \beta.$$

Заметим, что β является элементом усеченной итерации словаря, поэтому среди продукций нет пар вида $\alpha \rightarrow \varepsilon$, где ε — пустая цепочка.

Формальная грамматика G — это совокупность четырех объектов:

$$G = (V_T,\ V_N,\ P,\ S),$$

где P — непустое конечное подмножество Π, а $S \in V_N$ — начальный символ.

Типы грамматик Хомского определяются теми ограничениями, которые налагаются на продукции P. Если таких ограничений нет, грамматика принадлежит к типу 0 (по Хомскому). Единственное ограничение, налагаемое на длину цепочек α и β,

$$|\alpha| \leqslant |\beta|$$

относит грамматики к типу 1 (по Хомскому). Такие грамматики называют контекстно-зависимыми грамматиками, грамматиками непосредственных составляющих (НС-грамматиками).

В том случае, когда цепочка α состоит из одного символа, т. е. $\alpha \in V_N$, грамматики относят к типу 2 (по Хомскому). В этом случае их называют бесконтекстными (контекстно-свободными или КС-грамматиками).

Наконец, регулярными грамматиками (типа 3) называют такие, для которых $\alpha \in V_N$, а $\beta \in V_T V_N$, либо $\beta \in V_T$. Иными словами, правые части продукций регулярных грамматик состоят либо из одного терминального и одного нетерминального символов, либо из одного терминального символа.

Языком $L(G)$, порождаемым грамматикой G, будем называть множество цепочек $\alpha \in V_T$, каждая из которых порождается из

начального символа S в смысле полутуэвских соотношений P данной грамматики. Другими словами,

$$L(G) = \{\alpha \mid \alpha \in V_T^* \wedge S \Rightarrow * \alpha\}.$$

Нетрудно видеть, что каждая регулярная грамматика является бесконтекстной, каждая бесконтекстная грамматика является контекстно-зависимой, каждая контекстно-зависимая грамматика — грамматикой типа 0. Обратное, вообще говоря, неверно. Таким образом, имеется некоторая иерархия грамматик, которой соответствует иерархия формальных языков, каждый из них может быть порожден некоторой формальной грамматикой. Таким образом, тип языка может быть определен типом той грамматики, с помощью которой он может быть порожден.

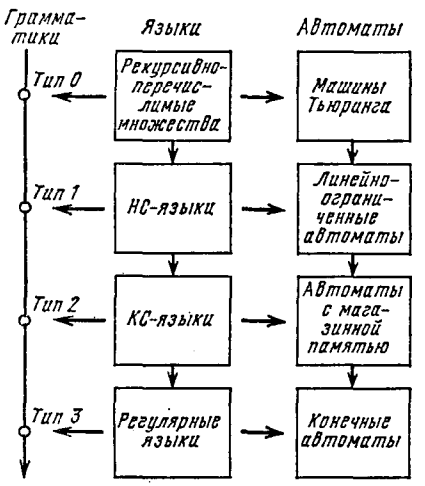


Рис. 16-1. Иерархия языков, грамматик и автоматов.

С другой стороны, типы языков могут быть определены типами абстрактных распознающих устройств (автоматов). При этом язык определяется как множество цепочек, допускаемых распознающим устройством определенного типа. На рис. 16-1 приведена иерархия языков и соответствующие ей иерархии грамматик и автоматов как распознающих устройств, которые будут подробно описаны в § 16-6.

В предыдущем параграфе формальный язык определялся как любое подмножество произвольного алфавита ($L \subseteq A^*$). Возникает вопрос, не ограничиваем ли мы себя, рассматривая лишь часть таких языков, вводя указанные типы грамматик. Однако доказано, что любое *рекурсивно-перечислимое множество*, т. е. множество, порождаемое автоматическим устройством произвольного вида, порождается некоторой грамматикой типа 0 по Хомскому [53]. Запомним, что для любого естественного языка в принципе возможно построить математическую модель, использующую такую грамматику.

Таким образом, грамматики типа 0 представляют собой порождающие устройства очень общего характера. А те формальные языки, с которыми имеют дело автоматно-лингвистические модели (языки программирования, ограниченные естественные языки и т. д.), как показывает практика, всегда описываются, по крайней мере, языками типа 1.

Языки типа 2 наиболее хорошо изучены [55]. К ним относятся многие языки программирования, например язык АЛГОЛ-60, синтаксис которого описывается с помощью так называемой нор-

мальной формы Бэкуса [56]. По существу эта форма представляет собой вариант записи продукций контекстно-свободной грамматики, например, запись

$$\langle A\rangle ::= \langle B\rangle\langle C\rangle \mid a\langle D\rangle$$

соответствует двум продукциям:

$$A \to BC,$$
$$A \to aD$$

где A, B, C, $D \in V_N$, $a \in V_T$.

Наконец, языки типа 3, введенные и изученные С. К. Клини в связи с исследованием модели нейрона, которые называют также языками Клини, автоматными языками, языками с конечным числом состояний и т. д., нашли широкое применение в исследовании электронных схем, а также в ряде других областей. Следует отметить, что впервые языки типа 3 были введены в рассмотрение еще А. А. Марковым. Имеются в виду *цепи Маркова*, играющие чрезвычайно важную роль в теории вероятностей.

16-3. ГРАММАТИКИ НЕПОСРЕДСТВЕННЫХ СОСТАВЛЯЮЩИХ И КС-ГРАММАТИКИ

Грамматики типа 0, порождающие языки, являющиеся рекурсивно-перечислимыми множествами, не нашли практического применения. Их использование для описания синтаксиса языков неудобно, так как для произвольной грамматики G данного типа очень трудно или невозможно построить приемлемый по трудоемкости алгоритм распознавания, который давал бы ответ на вопрос, принадлежит ли произвольная терминальная цепочка языку $L(G)$.

Контекстно-зависимые языки (языки непосредственных составляющих или НС-языки), порождаемые грамматиками типа 1, имеют большую практическую ценность, чем языки типа 0. Возникает вопрос, почему языки типа 1 называются контекстно-зависимыми. Можно доказать, что если G — контекстно-зависимая грамматика, то существует эквивалентная ей (т. е. порождающая тот же язык) грамматика G_1 такая, что каждая продукция из P имеет вид:

$$\alpha A\beta \to \alpha\gamma\beta,$$

где α, $\beta \in (V_T \cup V_N)^*$; $A \in V_N$; $\gamma \in (V_T \cup V_N)^+$.

Другими словами, символ A может быть заменен непустой цепочкой в определенном «контексте».

Говорят, что грамматика G_1 представляет собой так называемую нормальную форму контекстно-зависимой грамматики G.

Таким образом, нетерминальные символы можно интерпретировать как металингвистические переменные или, иными словами, конструкции языка, а вывод цепочек языка можно рассматривать как информацию о синтаксической структуре этих цепочек.

Пример 16-4. Пусть необходимо построить язык для описания равносторонних треугольников, каждая сторона которых состоит из конечного числа ориентированных отрезков определенной длины. На рис. 16-2, *а* приведены примеры таких треугольников. В зависимости от направления обозначим эти отрезки символами l, r или x так, как показано на рис. 16-2, *б*. Двигаясь по часовой стрелке от левого нижнего угла каждого треугольника, получим следующие описания этих треугольников:

$$lll\ rrr\ xxx,$$
$$ll\ rr\ xx,$$
$$llll\ rrrr\ xxxx.$$

Можно считать, что это цепочки некоторого языка описания

$$L = \{l^n r^n x^n \mid n \geqslant 1\}$$

равносторонних треугольников указанного типа.

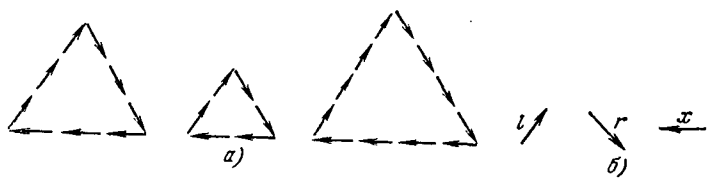


Рис. 16-2. Графическая интерпретация языка $\{l^n r^n x^n\}$ (*а*) и его терминальных символов (*б*).

Для данного языка можно предложить следующую грамматику типа 1, которую, однако, нельзя отнести к классу бесконтекстных грамматик:

$$G = (V_T, V_N, P, S),$$

где $V_T = \{l, r, x\}$, $V_N = \{S, R, X\}$, а множество продукций P состоит из элементов:

$$S \to lSRX,$$
$$S \to lRX,$$
$$XR \to RX,$$
$$lR \to lr,$$
$$rR \to rr,$$
$$rX \to rx,$$
$$xX \to xx.$$

Чтобы получить цепочку $l^n r^n x^n$, надо сделать следующее количество шагов:

$$N = \frac{n(n+5)}{2},$$

причем первую продукцию необходимо применить $n-1$ раз, затем один раз — вторую продукцию, $1/2\ n(n-1)$ раз — третью, один раз — четвертую, $n-1$ раз — пятую, один раз — шестую и $n-1$ раз седьмую.

Можно доказать, что с помощью данной грамматики нельзя построить терминальные цепочки, не являющиеся цепочками языка $\{l^n r^n x^n\}$. Более того, можно доказать, что данному языку не может соответствовать никакая бесконтекстная грамматика. Заметим, что данная грамматика не представляет собой нормальной формы, так как продукция

$$XR \to RX$$

имеет вид, отличный от вида

$$\alpha A\beta \to \alpha\gamma\beta.$$

Существует еще одна нормальная форма для грамматик непосредственных составляющих, называемая нормальной формой Куроды [34]. Грамматика $G = (V_T, V_N, P, S)$ задана в нормальной форме Куроды, если каждая ее продукция имеет одну из следующих форм:

$$A \to BC,$$
$$A \to AB,$$
$$AB \to CB,$$
$$A \to B,$$
$$A \to a,$$

где $A, B, C \in V_N$; $a \in V_T$.

Можно показать, что для каждой грамматики G типа 1 можно построить эквивалентную ей грамматику G_1 в нормальной форме Куроды.

Грамматики типа 2 (контекстно-свободные или КС-грамматики) используются наиболее широко в различных лингвистических моделях, находящих разнообразное применение. Продукции таких грамматик имеют вид:

$$A \to \beta,$$

где $A \in V_N$, $\beta \in (V_T \cup V_N)^+$.

Пример 16-5. Рассмотрим язык алгебраических выражений, словарь которого состоит из символов a, b, знаков $+$, $-$, $*$, $/$, а также символов ( ). Можно предложить грамматику типа 2 для порождения цепочек этого языка с начальным символом S и со следующими продукциями:

$$S \to W+S,$$
$$S \to W-S,$$
$$S \to W,$$
$$W \to W*W,$$
$$W \to W/W,$$
$$W \to (S),$$
$$W \to a,$$
$$W \to b.$$

Рассмотрим вывод цепочки $(a+b*b)-a/b$ из начального символа данной грамматики:

$$S \Rightarrow W-S \Rightarrow (S)-S \Rightarrow (W+S)-S \Rightarrow (a+S)-S \Rightarrow (a+W)-S \Rightarrow$$
$$\Rightarrow (a+W*W)-S \Rightarrow (a+b*W)-S \Rightarrow (a+b*b)-S \Rightarrow (a+b*b)-W \Rightarrow$$
$$\Rightarrow (a+b*b)-W/W \Rightarrow (a+b*b)-a/W \Rightarrow (a+b*b)-a/b.$$

Способ описания вывода цепочки из начального символа, приведенный в примере 16-5, не совсем удобен. Более наглядным является использование так называемых деревьев вывода. При этом корнем дерева будет являться вершина, имеющая метку начального символа S. Все подчиненные ей вершины, помеченные

слева направо $A_1, \ldots, A_n$ $(n \geqslant 1, A_i \in V_T \cup V_N)$, находятся в соответствии с символами продукции $S \to A_1, \ldots, A_n$, которая первой использовалась в выводе $S \Rightarrow * \alpha$, где $\alpha \in V_T^+$. Аналогично каждой вершине, помеченной нетерминальным символом, соответствует продукция, символы правой части которой помечают вершины, непосредственно подчиненные данной вершине.

На рис. 16-3 приведено дерево вывода для цепочки, рассмотренной в примере 16-5.

Как и для грамматик непосредственных составляющих, существуют нормальные формы и для бесконтекстных грамматик.

Нормальной формой Хомского любой бесконтекстной грамматики G называется такая грамматика G_1, эквивалентная G, все продукции которой имеют вид:

$$A \to BC,$$
$$A \to a,$$

где $A, B, C \in V_N$; $a \in V_T$.

Нормальной формой Грейбах любой бесконтекстной грамматики G называется такая грамматика G_2, эквивалентная G_1, все продукции которой имеют вид:

$$A \to a\alpha,$$

где $A \in V_N$; $a \in V_T$; $\alpha \in V_N^*$.

Тот факт, что для любой КС-грамматики можно построить ту и другую нормальные формы, строго доказан [34].

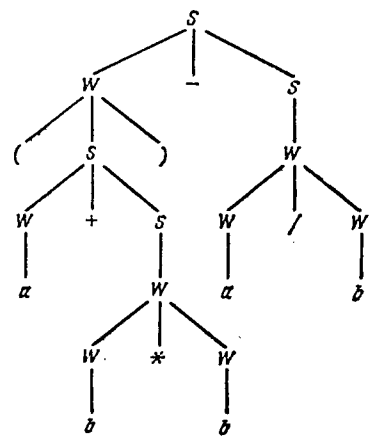


Рис. 16-3. Дерево вывода цепочки языка алгебраических выражений.

Кроме того, доказано и то, что любой бесконтекстный язык не является автоматным (принадлежит типу 2, но не принадлежит типу 3) тогда и только тогда, когда все задающие его грамматики обладают хотя бы одним таким нетерминальным символом A, для которого

$$A \Rightarrow * \alpha A\beta,$$

где $\alpha, \beta \in (V_T \cup V_N)^+$ (как α, так и β не пусты).

Такие символы называют *самовставляющимися*, а грамматики — *грамматиками с самовставлением*. Таким образом, отсутствие свойства самовставления может служить свидетельством того, что данная грамматика является регулярной (автоматной). Легко видеть, что грамматика из примера 16-5 не является автоматной в силу наличия продукций

$$S \to W,$$
$$W \to (S),$$

из чего следует, что $S \Rightarrow * (S)$, т. е. S — самовставляющийся символ.

16-4. РЕГУЛЯРНЫЕ ЯЗЫКИ И АВТОМАТНЫЕ ГРАММАТИКИ

Автоматные грамматики, порождающие класс регулярных (автоматных) языков, относятся к наиболее простому типу грамматик. Как было указано в § 16-2, продукции грамматик этого типа имеют вид:

$$A \to aB,$$
$$A \to a,$$

где $A,\ B \in V_N;\ a \in V_T$.

Иногда такие грамматики называют *автоматными грамматиками с правосторонними продукциями*. Можно ввести в рассмотрение также автоматные грамматики с левосторонними продукциями вида

$$A \to Ba,$$
$$A \to a.$$

Однако в каждой грамматике могут быть только правосторонние или только левосторонние продукции.

Класс регулярных языков чрезвычайно узок, однако исключительная простота алгоритмов распознавания этих языков делает их привлекательными для пользователей. В частности, автоматные грамматики используются для лексического анализа текстов программ в трансляторах с различных языков программирования. При этом с помощью автоматных грамматик, порождающих такие конструкции, как идентификатор и число, можно выделить их при чтении программы, записать в специальные таблицы, а в программу вставить на их места ссылки на соответствующие строки таблиц. Ссылки эти имеют стандартную длину, могут восприниматься как отдельные символы, что упрощает дальнейший анализ текста данной программы.

Пример 16-6. Пусть терминальный словарь V_T грамматики $G = (V_T, V_N, P, S)$ состоит из двух букв (a, b) и двух цифр (0, 1), нетерминальный словарь $V_N = \{S, L\}$, а продукции множества P имеют вид:

$$S \to aL,\ S \to bL,\ S \to a,\ S \to b,\ L \to 0,\ L \to 1,\ L \to a,$$
$$L \to b,\ L \to 0L,\ L \to 1L,\ L \to aL,\ L \to bL.$$

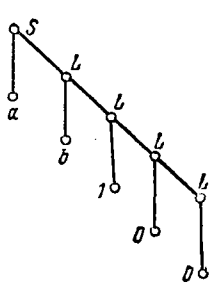


Рис. 16-4. Дерево вывода цепочки автоматного языка.

Такая грамматика порождает все возможные цепочки $\alpha \in V_T^*$, которые могут начинаться только с буквы. В языках программирования такие конструкции носят название *идентификаторов*. Очевидно, что если в словаре V_T содержится n букв и m цифр, множество P будет содержать $2(2n+m)$ продукций. На рис. 16-4 приведено дерево вывода для цепочки *ab100*, порождаемой данной грамматикой.

Нетрудно показать, что любой конечный язык, т. е. язык, содержащий конечное множество цепочек, $L = \{\alpha_1, \ldots, \alpha_n\}$ является автоматным языком. Действительно, учитывая, что любая цепочка языка $\alpha_i = a_{i_1} a_{i_2} \ldots a_{i_{ki}}$, можно задать множество продукций:

$$P = \{S \to a_{i_1} A_{i_1},\ A_{i_1} \to a_{i_2} A_{i_2},\ \ldots,\ A_{i_{k,i-1}} \to a_{i_{ki}} \mid i = 1, \ldots, n\}.$$

Грамматика, содержащая $k_1 + k_2 + \ldots + k_n$ таких продукций, очевидно, и будет порождать данный конечный язык.

Доказано, что *класс регулярных языков совпадает с наименьшим классом языков, содержащим все конечные языки и замкнутым относительно операций объединения, умножения и итерации (теорема Клини)*. Это позволяет определять любой автоматный язык не с помощью порождающей грамматики, а с помощью так называемого *представляющего выражения*. Определим представляющее выражение для цепочек из словаря V_T^* следующим образом:

1) любая цепочка из словаря V_T^* есть представляющее выражение;

2) если R_1 и R_2 — представляющие выражения, то R_1R_2 — тоже представляющее выражение;

3) если $R_1, \ldots, R_n$ — представляющие выражения, то $(R_{i_1}, \ldots, R_{i_m})^*$ — тоже представляющее выражение ($i_k \in \{1, \ldots, n\}$, $k = 1, \ldots, m$);

4) если $R_1, \ldots, R_n$ — представляющие выражения, то $R_1 \cup \ldots \cup R_n$ — тоже представляющее выражение;

5) других представляющих выражений нет.

Семантика (смысл) этих выражений такова:

1) цепочка из словаря V_T^* представляет собой язык, состоящий из единственной цепочки;

2) если R_1 представляет собой язык L_1, а R_2 — язык L_2, то R_1R_2 представляет собой L_1L_2, т. е. множество цепочек $\alpha_1\alpha_2$ таких, что $\alpha_1 \in L_1$, $\alpha_2 \in L_2$;

3) если $R_i\ (i = 1, \ldots, n)$ представляют собой языки $L_i\ (i = 1, \ldots, n)$, то $(R_{i_1}, \ldots, R_{i_m})^*$ представляет собой язык $\{L_{i_1} \cup \ldots \cup L_{i_m}\}^*$;

4) если $R_i\ (i = 1, \ldots, n)$ представляют собой языки $L_i\ (i = 1, \ldots, n)$, то $R = R_1 \cup \ldots \cup R_n$ представляет собой язык $L_1 \cup \ldots \cup L_n$.

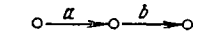
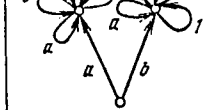
Представляющие выражения	*Соответствующие графы*
ab	
$(a)^*$	
$(a,b)^*$	
$a(a,b,1,0)^* \cup b(a,b,1,0)^*$	

Рис. 16-5. Примеры представляющих выражений и соответствующих им графов.

В силу теоремы Клини всякий регулярный язык может быть задан некоторым представляющим выражением. Каждому представляющему выражению можно поставить в соответствие граф. На рис. 16-5 приведены примеры представляющих выражений и соот-

ветствующих им графов. Первое представляющее выражение задает язык, состоящий из одной цепочки: $L_1 = \{ab\}$. Второе представляющее выражение задает бесконечный язык, являющийся итерацией алфавита $\{a\}$: $L_2 = \{\varepsilon, a, aa \ldots\}$. Третье представляющее выражение задает свободную полугруппу (моноид) с образующими $\{a, b\}$. Четвертое представляющее выражение задает язык, порождаемый автоматной грамматикой из примера 16-6 и состоящий из идентификаторов (цепочек символов, начинающихся с буквы).

16-5. ПРОМЕЖУТОЧНЫЕ КЛАССЫ ГРАММАТИК

В том случае, когда тот или иной язык (ограниченный естественный язык, язык программирования и т. д.) невозможно описать с помощью бесконтекстной грамматики, а использование контекстно-зависимых грамматик нежелательно из-за их чрезмерной сложности, необходим компромисс. При этом часто идут по пути обобщения КС-грамматик.

Другим прикладным аспектом формальных грамматик является их использование в качестве механизма для порождения решений в СИИ. При этом становится существенным тот факт, что в общем случае порождающие грамматики представляют собой исчисления, не обладающие свойством результативности. Это означает, что имея начальный символ грамматики и правила порождения (продукции), мы можем вывести любую терминальную цепочку данного языка, а не ту единственную цепочку, которая в данном случае представляет собой описание решения поставленной задачи. При этом говорят, что данный порождающий механизм (порождающая грамматика) описывает лишь *синтаксис* (структуру) данного языка, решений задачи, но не его *семантику* (смысл).

Пользуясь тем или иным методом для представления знаний (формализации семантики), можно существенно увеличить степень детерминированности порождающих грамматик. В пределе это может превратить грамматики в алгоритмы, которые работают с формализованными знаниями как с данными.

Однако наиболее интересным является компромиссное решение, при котором выводы терминальных цепочек зависят от тех или иных знаний (состояний модели предметной области, которая управляет порождающим механизмом), но при этом сохраняется некоторая недетерминированность вывода, позволяющая включать в контур управления человека.

Рассмотрим некоторые обобщения бесконтекстных грамматик, которые дают возможность вводить в грамматики средства, позволяющие управлять выводом и, тем самым, увеличивать степень их детерминированности.

А) БЕСКОНТЕКСТНЫЕ ПРОГРАММНЫЕ И ИНДЕКСНЫЕ ГРАММАТИКИ

Было предложено следующее обобщение бесконтекстных грамматик [57].

Бесконтекстной программной грамматикой (БП-грамматикой) называется следующая совокупность:

$$G = (V_T, V_N, P, I, S),$$

где помимо четырех компонентов КС-грамматики (V_T, V_N, P, S) введено конечное множество I так называемых меток продукций.

Каждая продукция из множества P состоит из следующих составляющих:

метки данной продукции $r \in I$;

ядра продукции, которым является продукция КС-грамматики вида

$$A \to \alpha,$$

где $A \in V_N$; $\alpha \in (V_N \cup V_T)^+$,

множества *переходов по успеху* $F_S \subseteq I$,

множества *переходов по неуспеху* $F_F \subseteq I$.

Вывод (генерация) терминальной цепочки $\beta \in V_T^+$ в грамматике G из начального символа S будет обозначаться следующим образом:

$$S \underset{(r_1, \ldots, r_n)}{\Longrightarrow} {}^{*} \beta \text{ или}$$

$$S \underset{r_1}{\Longrightarrow} \gamma_1 \underset{r_2}{\Longrightarrow} \gamma_2 \Longrightarrow \ldots \underset{r_n}{\Longrightarrow} \gamma_n = \beta.$$

Вывод осуществляется так: продукция с меткой r_1 применяется к символу S. Затем (рекурсивно), если применяется продукция с меткой r_i и ядром $A_i \to \alpha_i$ к цепочке $\gamma_{i-1} \in V^* A_i V^*$ ($V = V_T \cup V_N$), т. е. к цепочке, содержащей по крайней мере одно вхождение A_i, то самое левое вхождение A_i в этой цепочке заменяется на α_i и происходит переход к продукции, метка которой $r_{i+1} \in F_{S_i}$ (множеству переходов по успеху данной продукции). Если же продукция с меткой r_i не может быть применена, т. е. в γ_{i-1} не содержится ни одного A_i, то осуществляется переход к продукции, метка которой $r_{i+1} \in F_{F_i}$ (множеству переходов по неуспеху данной продукции). Если соответствующее множество (F_{S_i} или F_{F_i}) для данной продукции пусто, вывод прекращается.

Бесконтекстным программным языком (БП-языком) называется множество терминальных цепочек, порождаемых БП-грамматикой.

Доказано [57], что множество БП-языков строго содержит множество бесконтекстных языков и строго содержится внутри множества контекстно-зависимых языков.

Пример 16-7. В примере 16-4 приведена грамматика непосредственных составляющих для порождения описаний равносторонних треугольников в виде цепочек языка $L = \{l^n r^n x^n\}$. Этот язык не является бесконтекстным, но для него можно построить много вариантов БП-грамматик, например следующую:

$$G = (V_T, V_N, P, I, S),$$

где $V_T = \{l, r, x\}$; $V_N = \{S, L, W\}$;

$I = \{1, 2, 3, 4, 5\}$, а множество P состоит из следующих продукций:

(1) $(S \to LW)$ (2, 3) $(\emptyset)$,
(2) $(L \to lL)$ (4) $(\emptyset)$,
(3) $(L \to l)$ (5) $(\emptyset)$,
(4) $(W \to rWx)$ (2, 3) $(\emptyset)$,
(5) $(W \to rx)$ $(\emptyset)$ $(\emptyset)$.

Здесь каждая продукция состоит из четырех компонентов, каждый из которых заключен в скобки. Первый компонент — метка продукции, второй — ядро продукции, третий — множество переходов по успеху, четвертый — множество переходов по неуспеху. Символом $\emptyset$ обозначено пустое множество.

На рис. 16-6 приведено дерево вывода для цепочки $llrrxx$, на ветвях которого указан номер шага (первая цифра) вывода, а также метка соответствующей продукции (вторая цифра).

Чтобы получить цепочку $l^n r^n x^n$, надо сделать $2n+1$ шагов, причем первую, третью и пятую продукции необходимо применить по одному разу, а вторую и четвертую — по $n - 1$ раз.

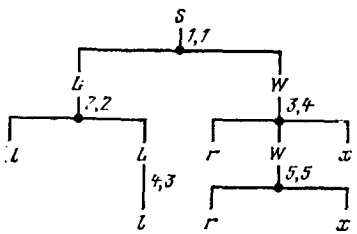


Рис. 16-6. Дерево вывода бесконтекстной программной грамматики.

Для порождения той же цепочки с помощью грамматики, приведенной в примере 16-4, необходимо сделать существенно большее число шагов ($1/2\ n\ (n + 5)$).

Другим обобщением бесконтекстных грамматик являются *индексные грамматики*, порождающие, как и БП-грамматики, языки, класс которых является промежуточным между классами КС-языков и языков непосредственных составляющих [53].

Индексной грамматикой называется следующая совокупность:

$$G=(V_T,\ V_N,\ F,\ P,\ S),$$

где V_T — терминальный словарь;

V_N — нетерминальный словарь;

F — множество индексов;

P — множество продукций, а

S — начальный символ.

Каждый индекс из F — это конечное множество так называемых индексных продукций вида

$$A\to\alpha,$$

где $A\in V_N$; $\alpha\in(V_T\cup V_N)^+$.

Продукции множества P имеют вид:

$$A\to A_1\psi_1\dots A_n\psi_n,$$

где $A_1,\ \dots,\ A_n\in V_T\cup V_N$, а $\psi_1,\ \dots,\ \psi_n$ лежат в множестве F^* (если $A_i\in V_T$, то $\psi_i=\varepsilon$).

Отношение непосредственного порождения $\Longrightarrow$ определяется на цепочках множества $(V_NF^*\cup V_T)^*$ следующим образом:

1. Если $\alpha A\theta\beta$ — цепочка, в которой

$$\alpha,\ \beta\in(V_NF^*\cup V_T)^*;\ A\in V_N;\ \theta\in F^*,$$

а среди продукций P есть продукция

$$A\to A_1\varphi_1\dots A_n\varphi_n,$$

то считаем, что

$$\alpha A\theta\beta\Longrightarrow\alpha A_1\psi_1\dots A_n\psi_n\beta,$$

где $\psi_i=\varepsilon$, если $A_i\in V_T$, и $\psi_i=\varphi_i\theta$, если $A_i\in V_N$.

2. Если $\alpha Af\theta\beta$ — цепочка, в которой $\alpha,\ \beta\in(V_NF^*\cup V_T)^*$; $A\in V_N$; $\theta\in F^*$; $f\in F$, а среди индексных продукций f есть продукция

$$A\to A_1\dots A_n,$$

то считаем, что

$$\alpha Af\theta\beta\Longrightarrow\alpha A_1\psi_1\dots A_n\psi_n\beta,$$

где $\psi_i=\varepsilon$, если $A_i\in V_T$, и $\psi_i=\theta$, если $A_i\in V_N$.

Пример 16-8. Рассмотрим тот же язык, что и в примере 16-4: $L=\{l^nr^nx^n\}$. Цепочки этого языка могут быть порождены с помощью индексной грамматики

$$G=(V_T,\ V_N,\ F,\ P,\ S),$$

где $V_T=\{l,\ r,\ x\}$; $V_N=\{S,\ W,\ L,\ R,\ X\}$; $F=\{f,\ g\}$, а множество P состоит из следующих продукций:

$$S\to Wg,$$
$$W\to Wf,$$
$$W\to LRX,$$

где $f=\{L\to lL,\ R\to rR,\ X\to xX\}$ — первое множество индексных продукций, а $g=\{L\to l,\ R\to r,\ X\to x\}$ — второе множество индексных продукций.

Для порождения цепочки $l^nr^nx^n$ необходимо один раз применить первую продукцию из множества P, $n-1$ раз — вторую продукцию и один раз — третью; затем необходимо $n-1$ раз применить первое множество индексных продукций и один раз — второе множество индексных продукций. При этом вывод цепочки $llrrxx$ будет иметь вид:

$$S\Longrightarrow Wg\Longrightarrow Wfg\Longrightarrow Lfg\ Rfg\ Xfg\Longrightarrow lLgrRgxXg\Longrightarrow llrrxx.$$

Б) МОДЕЛИ БЕСКОНТЕКСТНЫХ ЯЗЫКОВ С ФОРМАЛИЗОВАННОЙ СЕМАНТИКОЙ

Существует много важных приложений теории формальных грамматик. Об одном из них — моделировании механизма порождения решения задач в СИИ — шла речь в начале настоящего параграфа. Другое связано с так называемым грамматическим разбором или синтаксическим анализом цепочек языков, который широко используется в трансляторах с языков программирования, а также в системах общения человека с ЭВМ на естественном языке.

Под *грамматическим разбором* (синтаксическим анализом) цепочки некоторого языка понимают решение задачи о том, порождается или нет анализируемая цепочка символов с помощью данной грамматики. Задачу синтаксического анализа цепочек языка можно трактовать как задачу *распознавания*, а грамматику, порождающую синтаксически правильные цепочки, называют *распознающей грамматикой*.

Такая трактовка синтаксического анализа лежит в основе синтаксического подхода к распознаванию образов, который был изложен в гл. 12. Вопросы, связанные с грамматическим разбором (синтаксическим анализом), будут подробно рассмотрены в § 16-7, так как грамматический разбор играет наиболее важную роль в системах общения человека с ЭВМ, в которых используются формальные модели естественного языка (русского, английского и т. д.). При этом будут описаны различные методы грамматического разбора, а также классификация формальных грамматик, используемых в этих методах.

Грамматический разбор цепочек формальных языков, в особенности моделей естественных языков, является недостаточным для моделирования процесса *понимания* этих языков, под которым будем подразумевать преобразование этих цепочек в некоторое формальное представление их *смысла*, или *семантики*.

Существует несколько подходов к понятию семантики. Этот вопрос является одним из основных в проблеме искусственного интеллекта и будет подробно рассмотрен в гл. 17.

Понятие смысла (семантики) языка можно пояснить следующим образом. Будем говорить, что фраза языка (цепочка символов) имеет *смысл* для системы, целью которой является понимание языка, если эта система может преобразовать эту цепочку в некоторое ее *значение*, выраженное в заложенном в систему формализме. Этот формализм должен быть организован в структуру, позволяющую системе использовать свои знания совместно, например, с системой принятия решений, чтобы выполнять дедукцию, воспринимать новую информацию, отвечать на вопросы и интерпретировать команды.

При этом говорят, что в систему заложены возможности *семантической интерпретации* фразы естественного языка.

Порождающие грамматики Хомского, рассмотренные ранее, не могут служить целям понимания языка в силу того, что в них полностью отсутствует семантический аспект. Поэтому рассмотрим некоторые расширения моделей Хомского, в которых вводятся некоторые средства формализации семантики языков.

В простейшем случае смысл цепочки символов, которая подвергается грамматическому разбору, определяется через *семантические признаки* отдельных символов, входящих в эту цепочку. Впервые такая формализация семантики была предложена Д. Е. Кнутом [34] для бесконтекстных языков. Смысл цепочки определяется через признаки каждого нетерминального символа, возникающего при грамматическом разборе этой цепочки в соответствии с данными продукциями грамматики. Для каждой продукции бесконтекстной грамматики задаются так называемые семантические правила, которые определяют семантические признаки входящих в эту продукцию символов из нетерминального словаря V_N.

Семантические правила добавляются к бесконтекстной грамматике следующим образом. Каждому символу $A\in V_T\cup V_N$ приписано конечное множество $R(A)$ признаков, которое состоит из множества так называемых синтезированных признаков $R_0(A)$ и из множества так называемых унаследованных признаков $R_1(A)$, причем $R_0(A)\cap R_1(A)=\emptyset$.

Начальный символ S не должен иметь унаследованных признаков, а каждый терминальный символ — синтезированных. Каждый признак $r \in R(A)$ имеет множество значений D_r.

Пусть в множестве P содержится продукция

$$A_{i_0} \to A_{i_1} \ldots A_{i_{ki}}.$$

Семантическое правило — это функция f_{ijk}, переводящая определенные признаки символов A_{i_0}, A_{i_1}, ..., $A_{i_{ki}}$ в значение некоторого признака r символа A_{ij} $(0 \leqslant j \leqslant k_i)$.

Значения этой функции берутся из множества D_r, причем $r \in R_0(A_{ij})$, если $j = 0$, и $r \in R_1(A_{ij})$, если $j > 0$.

Рассмотрим на примере процесс оценки «смысла» цепочки терминальных символов по заданным значениям признаков этих символов.

Пример 16-9. Рассмотрим бесконтекстный язык описания чисел в двоичной системе счисления. Пусть задана грамматика, порождающая цепочки этого языка:

$$G = (V_T, V_N, P, S),$$

где $V_T = \{0, 1, .\}$ $V_N = \{S, L, B\}$; а множество P состоит из следующих продукций:

(1) $S \to L$,
(2) $S \to L . L$,
(3) $L \to B$,
(4) $L \to LB$,
(5) $B \to 0$,
(6) $B \to 1$.

Символы грамматики имеют следующие признаки:

$$A_0(B) = \{v\},$$
$$A_1(B) = \{s\},$$
$$A_0(L) = \{v, l\},$$
$$A_1(L) = \{s\},$$
$$A_0(S) = \{v\}.$$

Множества признаков терминальных символов пусты.
Множества величин признаков таковы:
D_v — множество рациональных чисел,
D_s — множество целых чисел,
D_l — множество целых чисел.
Семантика признаков следующая:
v — «величина» символа,
s — «позиция» символа,
l — «протяженность» символа.
Семантические правила имеют следующий вид:
для продукции (1)

$$v(S) = v(L), \quad s(L) = 0;$$

для продукции (2)

$$v(S) = v(L_1) + v(L_2), \quad s(L_1) = 0, \quad s(L_2) = -l(L_2);$$

для продукции (3)

$$v(L) = v(B), \quad s(B) = s(L), \quad l(L) = 1;$$

для продукции (4)

$$v(L_1) = v(L_2) + v(B), \quad s(B) = s(L_1); \quad s(L_2) = s(L_1) + 1, \quad l(L_1) = l(L_2) + 1;$$

для продукции (5)

$$v(B) = 0;$$

для продукции (6)

$$v(B) = 2^{s(B)}.$$

Дерево вывода, содержащее семантическую информацию цепочки 101.1 данного языка, представлено на рис. 16-7. *Смысл* этой цепочки заключается в том, что значением данного числа, представленного двоичной записью, является 5,5.

Другой подход к формализации семантики бесконтекстных языков, ориентированный не на грамматический разбор цепочек языка, а на порождение таких цепочек, зависящее от некоторых начальных условий, заключается в следующем. Задается семантика (смысл) начального символа БП-грамматики. Вывод терминальной цепочки, а также семантика входящих в нее терминальных символов зависят от семантики начального символа. При этом будем говорить, что к данной грамматике присоединен так называемый локально-параметрический механизм, а грамматику будем называть параметрической программной грамматикой.

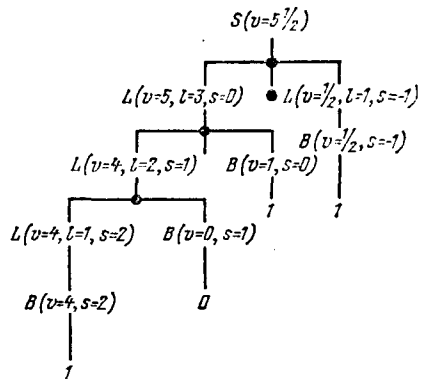


Рис. 16-7. Дерево вывода с семантическими признаками для цепочки языка описания чисел.

Формально параметрическая программная грамматика — это следующая совокупность:

$$G = (V_T, V_N, V_P, P, I, E, F, S),$$

где V_T, V_N, I, S определяются так же, как и для БП-грамматик;

V_P — алфавит некоторого исчисления предикатов первого порядка, построенного на базе семантических концепций (в смысле теории моделей Клини [59]);

E — область интерпретации для этого исчисления;

F — некоторое соответствие, относящее:

1) каждой предикатной букве данного исчисления предикатов из множества $Z = \{Q_1, Q_2, \ldots\}$ n-местное отношение в E;
2) каждой функциональной букве данного исчисления предикатов из множества $Y = \{f_1, f_2, \ldots\}$ n-местную функцию в E, отображающую E^n в E;
3) каждой константной букве данного исчисления предикатов из множества $W = \{a, b, c, \ldots\}$ некоторый элемент из E;

P — множество продукций, каждая из которых состоит из метки, ядра, множеств F_S и F_F (так же, как и в БП-грамматиках); при этом символ A_0, стоящий в левой части продукции $A_0 \to A_1 A_2 \ldots A_k$, снабжен некоторым множеством параметров

$$P_{A_0} \subseteq W \cup X,$$

где X — множество символов переменных из алфавита данного исчисления предикатов:

$$X = \{x, y, z, \ldots\},$$

а каждый символ правой части A_j $(1 \leqslant j \leqslant k)$ снабжен множеством

$$P_{A_j} \subseteq P_{A_0} \cup W \cup Y^*,$$

где Y — множество функций, каждая из которых имеет множество аргументов, являющееся подмножеством $P_{A_0} \cup W$.

Кроме того, каждая продукция содержит еще один элемент C, называемый условием применимости данной продукции и представляющий собой правильно построенную формулу (ППФ) данного исчисления предикатов, множество свободных переменных которой

$$P_Q \subseteq P_{A_0}.$$

Если указанная ППФ в момент попытки применения данной продукции имеет истинное значение для значений переменных, равных значениям соответствующих параметров левой части продукции, то продукция применяется и происходит обращение к полю F_S. При этом, естественно, должны соблюдаться и те условия, о которых шла речь в определении БП-грамматики. В противном случае осуществляется обращение к полю F_F.

Вывод в параметрической программной грамматике начинается с раскрытия начального символа S, все параметры которого имеют определенные значения из W. То же самое можно сказать и о любом символе цепочки $\gamma_{i-1}(S \Longrightarrow *\gamma_{i-1})$ перед тем, как к ней применяется продукция с меткой r_i и осуществляется непосредственное порождение цепочки γ_i:

$$\gamma_{i-1} \underset{r_i}{\Longrightarrow} \gamma_i,$$

при этом все параметры символов, стоящих в правой части продукции, принимают конкретные значения из W, равные значениям соответствующих переменных и функций. Все символы любой цепочки α языка

$$L(G) = \{\alpha \mid S \Longrightarrow * \alpha \wedge \alpha \in V_T^+\}$$

будут, таким образом, иметь «означенные» параметры, что соответствует выбранной семантической интерпретации данной цепочки языка.

Пример 16-10. Рассмотрим грамматику, порождающую решение известной «задачи о ханойской башне» [45] в системе редукций (т. е. в системе, использующей сведение задачи к совокупности элементарных подзадач). Задача заключается в следующем. Имеется три стержня a, b и c, на первом из которых находится n дисков различного диаметра (рис. 16-8). Необходимо перенести все диски, каждый раз перекладывая один из них, на третий стержень, причем не разрешается помещать диск на диск меньшего диаметра. Разрешается использовать промежуточный второй стержень. Элементарную подзадачу (перенести диск с номером i со стержня x на стержень y) обозначим терминальным символом с параметрами

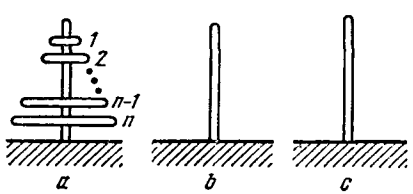


Рис. 16-8. Исходное состояние «задачи о ханойской башне».

$$p(i, x, y).$$

Решение задачи представим цепочкой языка, состоящей из этих символов, параметры которых имеют определенные значения.

На рис. 16-9, *а* приведены продукции грамматики, порождающей этот язык, а на рис. 16-9, *б* — дерево вывода для $n = 3$, $x = a$, $y = c$, т. е. для начального символа

$$S(3, a, c).$$

Компоненты грамматики следующие: нетерминальный словарь $V_N = \{S, A\}$; терминальный словарь $V_T = \{p\}$; алфавит исчисления предикатов

$$V_P = Z \cup Y \cup W \cup X,$$

где $Z = \{Q_1, Q_2, T\}$; $Y = \{f_1, f_2\}$; $W = \{a, b, c, 1, 2 \ldots\}$; $X = \{n, x, y\}$;
функции определяются следующим образом:

$$f_1(n) = n - 1, \quad n = 2, 3 \ldots;$$
$$f_2(x, y) \in \{a, b, c\}/\{x, y\}, \quad x, y \in \{a, b, c\};$$

предикаты, выступающие в роли условий применимости (ППФ), следующие:
$Q_1(n)$ — одноместный предикат, принимающий значение «истина» для $n > 1$;

$Q_2(n)$ — одноместный предикат, принимающий значение «истина» для $n = 1$;
T — 0-местный предикат, имеющий тождественно истинное значение.

Вывод терминальной цепочки в БП-грамматике можно сделать зависимым и от внешних условий. Для этой цели необходимо рассмотреть некоторую модель предметной области, которая будет служить областью интерпретации для продукций данной грамматики. Тогда состояние предметной области на каждом шаге вывода в грамматике будет управлять выбором продукции для дальнейшего вывода. Кроме того, применение тех или иных продукций может изменять состояние предметной области в рамках этой модели. В этом случае будем говорить о так называемых управляемых программных грамматиках, которые, так же как и параметрические программные грамматики, можно рассматривать как расширение КС-грамматик.

Метка	Ядро	F_S	F_F	C
1	$S(n,x,y) \rightarrow A(n,x,y)$	2	ϕ	T
2	$A(n,x,y) \rightarrow$ $A(f_1(n),x,f_2(x,y))$ $p(n,x,y)$ $A(f_1(n),f_2(x,y),y)$	2	3	$Q_1(n)$
3	$A(n,x,y) \rightarrow p(n,x,y)$	2	ϕ	$Q_2(n)$

Формально определим управляемую программную грамматику как совокупность из девяти компонентов:

$$G = (V_T, V_N, V_P, P, I, E, F, O, S),$$

где все компоненты, за исключением множества O, определяются так же, как и для параметрических программных грамматик.

Отличие состоит в иной структуре продукций из множества P. Проверка выполнимости правильно построенных формул исчисления предикатов в данном случае также служит иной цели. Каждый элемент множества операторов O включает в себя:

1) имя оператора из множества

$$N = \{A_1, A_2 \ldots\};$$

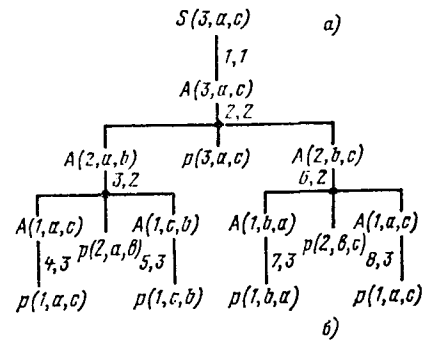


Рис. 16-9. Система продукции (*а*) и дерево вывода описания решения «задачи о ханойской башне» (*б*).

2) список параметров оператора из множества X, определенного для параметрических программных грамматик;

3) список изъятий, представляющих собой так называемые схемы ППФ, т. е. формулы исчисления предикатов, содержащие свободные переменные из множества $X \cup \{*\}$, где символу * может быть соотнесен любой элемент из множества W (в смысле заданной интерпретации);

4) список добавлений, также представляющих собой схемы ППФ, свободные переменные которых берутся из множества X.

С помощью таким образом определенных операторов должно осуществляться изменение описания предметной области, в ходе вывода, которое в данном случае представляет собой некоторый набор формул исчисления предикатов. Естественно, такой способ представления знаний о предметной области не единствен и можно предложить иные формализмы управления выводом в БП-грамматиках.

Каждая продукция из множества P состоит (так же как и в параметрических программных грамматиках) из метки, ядра (КС-продукции), множеств F_S и F_F, кроме того, вводятся новые компоненты: множества C_S и C_F (множества условий переходов) по мощности, равные соответственно множествам F_S и F_F.

Каждый элемент множества C_S ставится во взаимно-однозначное соответствие с элементом множества F_S, а элемент множества C_F — с элементом множества F_F. Элементы из множеств C_S и C_F представляют собой замкнутые ППФ, т. е. формулы,

не содержащие свободных переменных, и интерпретируются как условия перехода к продукциям, метки которых представляют собой соответствующие элементы из множеств F_S и F_F. Последним компонентом продукции является оператор из множества O.

Вывод в управляемой программной грамматике осуществляется следующим образом. Пусть к цепочке γ_{i-1} ($S \Longrightarrow *\gamma_{i-1}$) должна быть применена продукция с меткой $r_l \in I$, ядром $A_{i_0} \to A_{i_1} \dots A_{i_{kl}}$, множествами F_{S_l}, $F_{F_l} \subseteq I$, множествами условий переходов C_{S_l} и C_{F_l}, а также оператором $o_l \in O$. Пусть в цепочке γ_{l-1} содержится символ A_{i_0}, тогда из множества F_{S_l} выбираются именно те элементы, для которых соответствующие ППФ из множества C_{S_l} принимают истинные значения для данной семантической интерпретации. Если цепочка γ_{i-1} не содержит A_{i_0}, выбираются элементы из множества F_{F_l}, для которых соответствующие ППФ из множеств C_{F_l} принимают значение «истина».

Заметим, что если множества C_{S_l} и C_{F_l} будут подчиняться двум принципам: принципу истинности выбора, согласно которому дизъюнкция всех элементов из множества $C_{S_i}(C_{F_l})$ должна принимать тождественно истинное значение, а также принципу непротиворечивости выбора, согласно которому конъюнкция двух любых элементов $C_{S_l}(C_{F_l})$ должна принимать тождественно ложное значение, то вывод в грамматике будет строго детерминирован и грамматику можно рассматривать как алгоритм.

Непосредственный вывод $\gamma_{i-1} \underset{r_l}{\Longrightarrow} \gamma_l$ сопровождается изменением описания предметной области в том случае, когда данная продукция содержит оператор. Это изменение заключается в изъятии и добавлении к описанию предметной области информации в соответствии со списками изъятий и добавлений данного оператора.

Пример 16-11. Рассмотрим очень простую модель генерации описания процесса механической обработки детали. Для простоты описания условий переходов, а также операторов будем давать неформально.

На рис. 16-10 приведены продукции управляемой грамматики, а на рис. 16-11 представлено дерево вывода для цепочки языка, описывающей последовательность этапов обработки детали. Описания самой этой детали, а также технических условий ее изготовления в данном случае представляют собой модель предметной области, которая изменяется в ходе порождения указанной цепочки из начального символа грамматики.

В грамматику заложены следующие эвристические предпосылки.

Деталь сначала должна быть изготовлена, а затем упакована. Деталь производится либо с помощью штамповки, либо с помощью токарной обработки, которая предусматривает обточку детали (обработку наружных поверхностей) и сверление (обработку внутренних поверхностей). Кроме того, технология предусматривает первоочередность сверления перед обточкой. Если детали крупные, их упаковывают в отдельную коробку, если детали мелкие, их складывают в ящик.

Символы словарей грамматики имеют следующую мнемонику.

Нетерминальный словарь

S — начальный символ,
ИЗГД — изготовление детали,
УПАК — упаковка детали,
ТОКОБ — токарная обработка,
ТОЧ — обточка детали,
СВР — сверление детали.

Терминальный словарь

штам — штамповать,
точ — точить,
свр — сверлить,
уящ — упаковать в ящик,
укор — упаковать в коробку.

Неформально описанные ППФ и операторы для данного примера следующие:
Q_1 — деталь является телом вращения,
Q_2 — деталь содержит «фасонную» поверхность,
Q_3 — у детали нет ни одной внутренней «ступенчатой» поверхности,
Q_4 — у детали нет ни одной наружной «ступенчатой» поверхности,
Q_5 — у детали больше одной наружной «ступенчатой» поверхности,
Q_6 — у детали больше одной внутренней «ступенчатой» поверхности,
Q_7 — деталь большая,

Метка	Ядро	F_S	F_F	C_S	C_F	Оператор
1	S→ИЗГД УПАК	3,2	∅	Q_2, Q_1	T	∅
2	ИЗГД→ТОКОБ	4,5,6	∅	Q_3, Q_4^*	T	∅
3	ИЗГД→штам	11,12	∅	Q_7^*	T	A_3
4	ТОКОБ→ТОЧ	7,8	∅	Q_5^*	T	∅
5	ТОКОБ→СВР	9,10	∅	Q_6^*	T	∅
6	ТОКОБ→СВРТОЧ	7,8	∅	Q_5^*	T	∅
7	ТОЧ→ТОЧточ	7,8	∅	Q_5^*	T	A_1
8	ТОЧ→точ	9,10	∅	Q_6^*	T	A_1
9	СВР→СВРсвр	9,10	∅	Q_6^*	T	A_2
10	СВР→свр	11,12	11,12	Q_7^*	Q_7^*	A_2
11	УПАК→уящ	∅	∅	T	T	A_4
12	УПАК→укор	∅	∅	T	T	A_5

Рис. 16-10. Система продукций управляемой программной грамматики.

A_1 — из описания детали удаляется описание наружной «ступени»,
A_2 — из описания детали удаляется описание внутренней «ступени»,
A_3 — из описания детали удаляется описание «фасонной» поверхности,
A_4, A_5 — в описании предметной области появляются данные о том, что в ящике (в коробке) находится деталь.

Звездочкой на рис. 16-10 помечены те случаи, когда

$$|C_{S_l}| = |F_{S_l}| - 1 \quad \left(|C_{F_l}| = |F_{F_l}| - 1\right).$$

Это означает, что по умолчанию последний (n-й) элемент множества $C_{S_l}(C_{F_l})$ является конъюнкцией:

$$Q_{i_n} = \neg Q_{i_1} \wedge \dots \wedge \neg Q_{i_{n-1}},$$

что соответствует ранее введенным принципам истинности и непротиворечивости выбора. В данном случае это говорит о том, что для исходных данных (описания предметной области) из начального символа может порождаться единственная терминальная цепочка, а грамматика представляет собой алгоритм построения такой цепочки.

На рис. 16-11 помимо дерева вывода в грамматике изображены этапы обработки детали, соответствующие терминальным символам. Последовательность этапов обработки соответствует последовательному рассмотрению цепочки терминальных символов слева направо, как это показано на рис. 16-11.

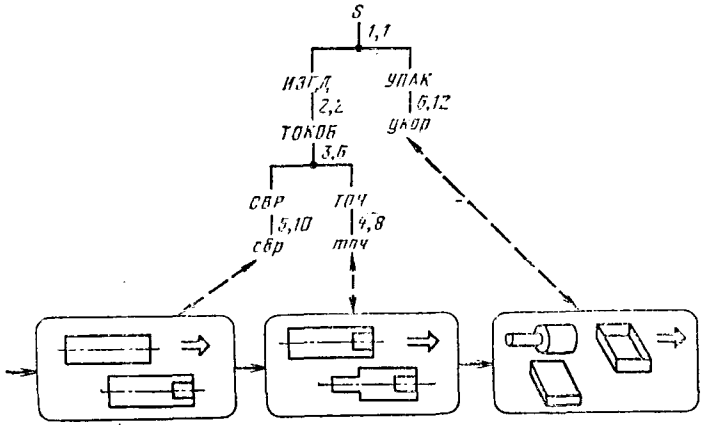


Рис. 16-11. Дерево вывода цепочки языка, описывающей последовательность этапов обработки детали.

16-6. РАСПОЗНАЮЩИЕ УСТРОЙСТВА И АВТОМАТЫ

В предыдущих параграфах различные типы языков определялись с точки зрения ограничений на порождение, т. е. с помощью грамматик различного типа. Другой метод определения языка основан на использовании множества цепочек, воспринимаемых некоторым абстрактным распознающим устройством. Такие устройства называются также *автоматами* [60]. Как показали многочисленные теоретические исследования, классам языков, соответствующим четырем типам грамматик по классификации Хомского, можно поставить во взаимно-однозначное соответствие четыре типа распознающих устройств. Простейшим из них является класс так называемых конечных автоматов, которые допускают (распознают) все языки, порождаемые автоматными (регулярными) грамматиками, и только их.

а) Конечные автоматы и регулярные языки

Различают детерминированные и недетерминированные конечные автоматы. *Детерминированным конечным автоматом* называют следующую пятерку:

$$A = (V,\ Q,\ \delta,\ q_0,\ F),$$

где $V = \{a_1, \ldots, a_m\}$ — входной алфавит (конечное множество символов);

$Q = \{q_0, q_1, \ldots, q_{n-1}\}$ — алфавит состояний автомата (конечное множество символов);

δ — функция, отображающая множество $Q \times V$ в Q и называемая функцией переходов;

$q_0 \in Q$ — начальное состояние автомата;

$F \subseteq Q$ — множество состояний, называемых заключительными.

На содержательном уровне функционирование конечного автомата можно представить себе следующим образом. Имеется бесконечная лента с ячейками, в каждой из которых может находиться один символ из V. На ленте находится цепочка символов $\alpha \in V^*$. Ячейки слева и справа от цепочки не заполнены. Имеется некоторое конечное управляющее устройство с читающей головкой, которое может последовательно считывать символы с ленты, передвигаясь слева направо. При этом устройство может находиться в каком-либо одном состоянии из Q. Каждый раз, переходя к новой ячейке, устройство переходит к новому состоянию в соответствии с функцией δ.

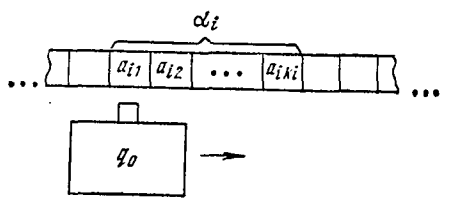


Рис. 16-12. Интерпретация конечного автомата.

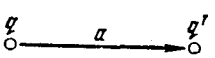


Рис. 16-13. Графическое представление команды автомата.

На рис. 16-12 изображен конечный автомат в начальном состоянии q_0, считывающий первый символ a_{i_1} входной цепочки α_i. Стрелкой показано направление движения читающей головки. Отображение δ можно представить в виде совокупности так называемых *команд*, которые обозначаются следующим образом:

$$(q,\ a) \to q',$$

где $q,\ q' \in Q;\ a \in V$.

Число команд конечно; левая часть команды (q, a) называется *ситуацией автомата,* а правая, q', есть состояние, в котором автомат будет находиться на следующем шаге своей работы.

Графически команду удобно представлять в виде дуги графа, идущей из вершины q в вершину q' и помеченную символом a входного алфавита (рис. 16-13). Полностью отображение δ изображают с помощью диаграммы состояний, т. е. ориентированного графа, вершинам которого поставлены в соответствие символы Q, а дугам — команды отображения δ.

Если автомат оказывается в ситуации (q_f, a_i), не являющейся левой частью какой-либо его команды, то он останавливается. Если управляющее устройство считает все символы цепочки α, записанной на ленту и при этом перейдет в состояние $q_f \in F$ (заключительное состояние), то говорят, что цепочка α *допускается авто-*

матом A (*автомат допускает* цепочку α). Множество цепочек, допускаемых данным автоматом, есть, по определению, *язык, допускаемый* этим *автоматом*.

Отображение δ можно представить и в виде функции

$$\delta(q,\ a) = q',$$

где $q,\ q' \in Q;\ a \in V$.

Эта функция интерпретируется так же, как и команда $(q,\ a) \to q'$. Ее можно распространить с одного входного символа на цепочку следующим образом:

$\delta(q,\ \varepsilon) = q$, где ε — пустая цепочка;

$\delta(q,\ \alpha a) = \delta(\delta(q,\ \alpha),\ a)$, где $a \in V,\ \alpha \in V^*$.

Таким образом, можно сказать, что α допускается автоматом A, если

$$\delta(q_0,\ \alpha) = q_f,\ \text{где}\ q_f \in F,$$

а язык, допускаемый автоматом A, это

$$L(A) = \{\alpha \mid \delta(q_0,\ \alpha) \in F\}.$$

Пример 16-12. Рассмотрим детерминированный конечный автомат

$$A = (V,\ Q,\ \delta,\ q_0,\ F),$$

где $V = \{a,\ b\}$; $Q = \{S,\ X,\ Y,\ T\}$; $q_0 = S$; $F = \{T\}$, а δ задается диаграммой состояний, представленной на рис. 16-14.

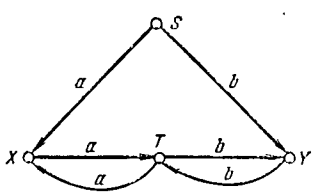


Рис. 16-14. Диаграмма состояний детерминированного конечного автомата, допускающего язык $\{aa,\ bb\}^+$.

	a	b	
q_0	1	2	S
q_1	3		X
q_2		3	Y
q_3	1	2	T

Рис. 16-15. Матрица переходов для автомата, допускающего язык $\{aa,\ bb\}^+$.

Легко видеть, что язык, допускаемый этим автоматом,

$$L(A) = \{M^n \mid n \geqslant 1\},$$

где $M = \{aa,\ bb\}$.

Цепочка $\alpha_1 = aabbaa$ допускается данным автоматом, так как после ее просмотра автомат окажется в состоянии $T \in F$.

Цепочка *aabba* не допускается, так как после ее просмотра автомат окажется в состоянии X, не являющемся заключительным.

Цепочка *abb* не допускается потому, что после считывания символа *a* автомат окажется в ситуации $(X,\ b)$, для которой нет команды.

Существует и другой способ определения детерминированного конечного автомата — с помощью так называемых матриц переходов.

Конечному автомату A ставится в соответствие матрица B, состоящая из $n \times m$ элементов. Элемент $B[i,\ j]$ содержит число k — номер состояния q_k, такого, что

$$\delta(q_i,\ a_j) = q_k.$$

При этом считается, что $Q = \{q_0,\ q_1,\ \ldots,\ q_{n-1}\}$, $V = \{a_1,\ \ldots,\ a_m\}$, а список заключительных состояний представлен вектором.

На рис. 16-15 представлена матрица переходов для автомата из примера 16-12.

Недетерминированный конечный автомат есть пятерка вида

$$A = (V,\ Q,\ \delta,\ q_0,\ F),$$

где $V = \{a_1,\ \ldots,\ a_m\}$ — входной алфавит;

$Q = \{q_0,\ q_1,\ \ldots,\ q_{n-1}\}$ — алфавит состояний автомата;

δ — функция переходов, отображающая множество $Q \times V$ в множество подмножеств Q;

$q_0 \in Q$ — начальное состояние автомата;

F — множество заключительных состояний.

Единственное отличие недетерминированного конечного автомата от детерминированного заключается в том, что значениями функции переходов являются не состояния, а множества состояний (или, в терминах команд, возможны различные команды с одинаковыми левыми частями). Это соответствует тому факту, что в диаграмме состояний из одной вершины может исходить несколько дуг с одинаковой пометкой.

Рассмотрим теперь произвольную автоматную грамматику с правосторонними продукциями:

$$G = (V_T,\ V_N,\ P,\ S),$$

продукции которой имеют вид:

$$A_i \to a_j A_k,\ \text{либо}\ A_i \to a_j.$$

Построим для нее недетерминированный конечный автомат

$$A = (V,\ Q,\ \delta,\ q_0,\ F),$$

где $V = V_T$; $Q = V_N \cup \{T\}$, причем символ T не должен содержаться в V_N; $q_0 = S$; $F = \{T\}$, а отображение δ строится следующим образом:

каждой продукции вида $A_i \to a_j$ ставится в соответствие команда $(A_i,\ a_j) \to T$; каждой продукции вида $A_i \to a_j A_k$ ставится в соответствие команда $(A_i,\ a_j) \to A_k$; других команд нет.

Построенный таким образом автомат обозначим A_G.

Ввиду того что любой регулярный язык может быть порожден автоматной грамматикой с правосторонними продукциями, можно ограничиться рассмотрением именно таких грамматик.

Теперь рассмотрим произвольный недетерминированный конечный автомат

$$A = (V,\ Q,\ \delta,\ q_0,\ F).$$

Такому автомату можно поставить в соответствие следующую автоматную грамматику:

$$G=(V_T,\ V_N,\ P,\ S),$$

где $V_T = V$; $V_N = Q$; $S = q_0$, а множество P строится следующим образом:

каждой команде автомата $(q_i,\ a_j) \to q_k$ ставится в соответствие продукция $q_i \to a_j\, q_k$, если $q_k \in Q$;

каждой команде $(q_i,\ a_j) \to q_k$ ставится в соответствие еще одна продукция $q_i \to a_j$, если $q_k \in F$; других продукций нет.

Построенную таким образом грамматику обозначим G_A.

Определим теперь язык, допускаемый недетерминированным конечным автоматом.

Как и в случае детерминированного конечного автомата можно расширить область определения функции δ до $Q \times V^*$ следующим образом:

$\delta(q,\ \varepsilon) = q$, где ε — пустая цепочка;

$$\delta(q,\ \alpha a) = \bigcup_{q_i \in \delta(q,\ \alpha)} \delta(q_i,\ a),\ \text{где}\ a \in V,\ \alpha \in V^*.$$

Цепочка α допускается автоматом A, если найдется такое состояние $q_f \in F$, что

$$q_f \in \delta(q_0,\ \alpha).$$

Язык, допускаемый недетерминированным конечным автоматом A, это

$$L(A) = \{\alpha \mid q_f \in \delta(q_0,\ \alpha) \wedge q_f \in F\}.$$

Описанные выше способы построения автоматов по заданным грамматикам и грамматик по заданным автоматам делают очевидными следующие утверждения:

1) $L(G_A) = L(A)$ для любого недетерминированного конечного автомата A;

2) $L(A_G) = L(G)$ для любой автоматной грамматики G.

Хотя недетерминированный конечный автомат располагает на первый взгляд бо́льшими возможностями, чем детерминированный, классы языков, которые они допускают, совпадают. Действительно, для любого недетерминированного конечного автомата A с алфавитом V, множеством состояний $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ и функцией переходов δ можно построить эквивалентный ему (т. е. допускающий тот же язык) детерминированный конечный автомат A' следующим образом. Состояниями автомата A' будут все подмножества Q, т. е.

$$Q' = 2^Q = \{\Phi,\ \{q_0\},\ \{q_1\},\ \ldots,\ \{q_0,\ \ldots,\ q_{n-1}\}\};$$

входной алфавит $V' = V$, начальное состояние $q'_0 = \{q_0\}$, заключительные состояния — все подмножества Q, содержащие хотя бы одно заключительное состояние автомата A.

Функцию переходов δ' автомата A' определим следующим образом:

$$\delta'(q',\ a) = \bigcup_{q \in q'} \{\delta(q,\ a)\}.$$

Пример 16-13. Рассмотрим регулярный язык, каждая цепочка $\alpha \in \{a,\ b\}^*$ которого содержит по крайней мере одну подцепочку aa. Можно рассмотреть автоматную грамматику G, порождающую этот язык, продукции которой следующие:

$$S \to aS,\quad S \to bS,\quad S \to aM,\quad M \to a,\quad M \to aN,\quad N \to aN,$$
$$N \to bN,\quad N \to a,\quad N \to b.$$

Построим недетерминированный конечный автомат $A_G = (V,\ Q,\ \delta,\ q_0,\ F)$, используя способ, описанный ранее. При этом компонентами автомата будут:

$$V = \{a,\ b\},\quad Q = \{S,\ M,\ N,\ T\},\quad q_0 = S,\quad F = \{T\},$$

а отображение δ представлено на рис. 16-16 в виде диаграммы состояний.

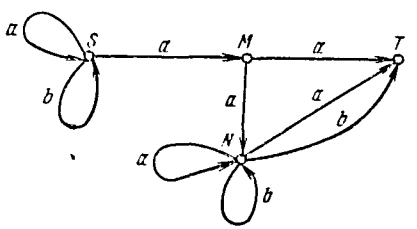


Рис. 16-16. Диаграмма состояний недетерминированного конечного автомата, допускающего язык $\{a,\ b\}^*\, aa\, \{a,\ b\}^*$.

Пример 16-14. Построим автоматную грамматику G_A для автомата A, описанного в примере 16-12. Компонентами этой грамматики будут:

$$V_T = \{a,\ b\},\quad V_N = \{S,\ X,\ Y,\ T\};$$

начальный символ S; продукции имеют вид:

$$S \to aX,\quad S \to bY,\quad X \to aT,\quad Y \to bT,\quad T \to aX,\quad T \to bY,\quad Y \to a,\quad Y \to b.$$

Пример 16-15. Рассмотрим недетерминированный конечный автомат A_G из примера 16-13. Построим эквивалентный ему детерминированный конечный автомат $A' = (V',\ Q',\ \delta',\ q'_0,\ F')$. Его компонентами будут:

$$V' = V = [a,\ b];$$
$$Q' = \{\{S\},\ \{S,\ M\},\ \{S,\ N,\ T\},\ \{S,\ M,\ N,\ T\}\}$$

(остальные 12 состояний можно отбросить, так как ни в одно из них автомат A' попасть не может);

$$q'_0 = \{S\};$$
$$F' = \{\{S,\ N,\ T\},\ \{S,\ M,\ N,\ T\}\};$$

функцию переходов δ' можно задать с помощью следующих команд:

$$(\{S\},\ a) \to \{S,\ M\};$$
$$(\{S\},\ b) \to \{S\};$$
$$(\{S,\ M\},\ a) \to \{S,\ M,\ N,\ T\};$$
$$(\{S,\ M\},\ b) \to \{S\};$$
$$(\{S,\ M,\ N,\ T\},\ a) \to \{S,\ M,\ N,\ T\};$$
$$(\{S,\ M,\ N,\ T\},\ b) \to \{S,\ N,\ T\};$$
$$(\{S,\ N,\ T\},\ a) \to \{S,\ M,\ N,\ T\};$$
$$(\{S,\ N,\ T\},\ b) \to \{S,\ N,\ T\}.$$

На рис. 16-17 изображена диаграмма состояний для автомата A'. Построим автоматную грамматику $G_{A'}$. Для этого дадим новые обозначения элементам множества Q': элемент $\{S\}$ обозначим символом S'; элемент $\{S, M\}$ — символом M';

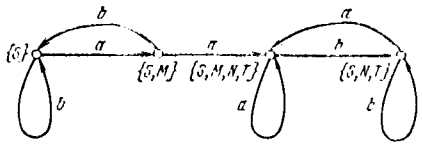


Рис. 16-17. Диаграмма состояний детерминированного конечного автомата, допускающего язык $\{a, b\}^*\ aa\ \{a, b\}^*$.

$\{S, N, T\}$ — символом T_1'; $\{S, M, N, T\}$ — символом T_2'. Продукциями грамматики $G_{A'}$ будут следующие:

$$S' \to aM', \quad S' \to bS', \quad M' \to bS', \quad M' \to aT_1', \quad T_1' \to aT_1', \quad T_1' \to bT_2',$$
$$T_2' \to aT_1', \quad T_2' \to bT_2', \quad T_1' \to a, \quad T_1' \to b, \quad T_2' \to a, \quad T_2' \to b.$$

На практике бывает удобным рассматривать расширение детерминированных и недетерминированных конечных автоматов — так называемые *конечные преобразователи*. Их отличие от описанных ранее автоматов состоит в том, что помимо входной ленты с анализируемой цепочкой $\alpha \in V_1^*$ в них имеется и выходная лента, на которую записывается цепочка β, символы которой берутся из особого словаря V_2. Такое расширение позволяет не просто решать вопрос о принадлежности цепочки α к данному языку, но и выдавать синтаксическую структуру этой цепочки в виде описания β.

Формально детерминированный конечный преобразователь — это шестерка вида

$$A = (V_1,\ V_2,\ Q,\ \mu,\ q_0,\ F),$$

где V_1, V_2 — соответственно входной и выходной алфавиты; Q, q_0 и F определяются так же, как и для конечных автоматов, а μ — функция, отображающая множество $Q \times V_1$ в $Q \times V_2^*$.

Функцию μ удобно представить в виде совокупности двух функций: функции переходов φ, отображающей $Q \times V_1$ в Q, и функции выходов ψ, отображающей $Q \times V_1$ в V_2^*.

Отображение μ в этом случае удобно представить совокупностью команд вида

$$(q,\ a) \to (q',\ b),$$

где $q,\ q' \in Q,\ a \in V_1,\ b \in V_2$.

Приведенные ранее определения конечного автомата и конечного преобразователя несколько отличаются от тех, которые используются в некоторых монографиях по теории конечных автоматов. Так, в [64, 65] под *конечным автоматом* понимают то, что здесь мы называем конечным преобразователем. А то, что здесь определено как конечный автомат, в [64, 65] носит название *автомата без выхода*. Кроме того, многие авторы вводят в рассмотрение ось времени, предполагая, что работа автомата протекает в дискретные такты времени $t = 0, 1, 2 \ldots$ [64] или $t = 1, 2, 3 \ldots$ [65].

Пусть конечный автомат [64, 65] в момент времени t_0 находится в состоянии $q_l\ (t_0)$. Если на его вход поступает символ $a_r\ (t_0) \in V_1$, то по команде

$$(q_i,\ a_r) \to (q_j,\ b_s)$$

автомат переходит в новое состояние $q_j\ (t_0 + 1)$ и выдает на выход символ $b_s\ (t_0) \in V_2$.

В рассмотрение вводятся [65] рекуррентные соотношения

$$q(t+1) = \varphi[q(t),\ a(t)];$$
$$b(t) = \psi[q(t),\ a(t)],$$

где $q(t),\ q(t+1) \in Q,\ a(t) \in V_1,\ b(t) \in V_2$, которые вместе с *начальным условием*

$$q(1) = q_0$$

задают *оператор*, обозначаемый через $T(A, q_0)$, переводящий всякую конечную последовательность входных символов

$$a(1)\,a(2)\ldots a(r)$$

в последовательность выходных символов

$$b(1)\,b(2)\ldots b(r).$$

Частными случаями детерминированных конечных автоматов в терминологии [64, 65] (в нашем случае детерминированных конечных преобразователей) являются следующие варианты автоматов:

автоматы без памяти, не имеющие алфавита Q; команды таких автоматов имеют вид:

$$a_r \to b_s;$$

автономные автоматы (автоматы без входа или генераторы), не имеющие входного алфавита V_1; команды таких автоматов имеют вид:

$$q_l \to (q_j,\ b_s);$$

автоматы без выхода, не имеющие выходного алфавита V_2, команды таких автоматов имеют вид:

$$(q_l,\ a_r) \to q_j.$$

Автоматы без выхода (здесь мы называем их просто автоматами) являются более сложными и интересными устройствами, чем автоматы без памяти и автономные автоматы. Поэтому обычно исследуется именно этот класс автоматов [65].

Иногда накладываются ограничения на функцию выходов. С этой точки зрения можно выделить следующие два типа автоматов: *автоматы Мили и автоматы Мура* [64].

Для автоматов Мили рекуррентные соотношения для оператора T имеют вид:

$$q(0) = q_0;$$
$$q(t) = \varphi[q(t-1),\ a(t)];$$
$$b(t) = \psi[q(t-1),\ a(t)],$$

где $t = 1, 2 \ldots$

Для автоматов Мура эти соотношения имеют вид:

$$q(0) = q_0;$$
$$q(t) = \varphi[q(t-1),\ a(t)];$$
$$b(t) = \lambda[q(t)],$$

где $t = 1, 2 \ldots$; λ — так называемая *сдвинутая функция выходов.*

Пример 16-16. Рассмотрим работу некоторого гипотетического устройства, определяющего распределение квантов времени работы центрального процессора вычислительного комплекса. В каждый такт времени t_j работы процессора к распределительному устройству поступает одна заявка, принадлежащая группе пользователей d_j, имеющих общий счет оплаты времени. Внутри каждой группы имеется n_k пользователей.

Пользователям удобно присвоить двойные индексы k, ν_k: сначала указывается номер группы k, а затем номер пользователя ν_k в группе ($k = 1, 2, \ldots, n$, $\nu_k = 1, 2, \ldots, n_k$). Все пользователи предполагаются однородными. В этом случае, как правило, используется циклическая дисциплина обслуживания, т. е. если в момент t_j квант времени процессора был предоставлен пользователю k, ν_k и в момент времени t_{j+1} поступила заявка той же группы, то процессор предоставляется пользователю с номером k, ν_k, когда $\nu_k < m_k$, или первому пользователю данной группы, если $\nu_k = m$.

Функционирование такого распределительного устройства можно описать в виде конечного преобразователя. Примем в качестве входного алфавита совокупность номеров групп пользователей 1, 2, ..., n, а в качестве выходного алфавита — совокупность номеров пользователей внутри группы k, ν_k. Состояниями автомата будем считать n-мерные векторы, составляющими которых являются номера ν_k пользователей в соответствующих группах, последними получившими квант времени.

Будем считать, что в момент времени $t = 0$ заявки не поступили, и все пользователи находятся в состоянии ожидания. Начальное состояние автомата можно установить в виде $q_0 = (0, 0, \ldots, 0)$, текущее состояние в виде $q = (\nu_1, \nu_2, \ldots, \nu_n)$, входные сигналы $x(t_j) = k_j$, ..., выходные сигналы $y = k$, ν_k.

Функция переходов в новое состояние $\varphi(q(t-1), x(t))$ определяется следующими соотношениями:

$$\nu_x(t) = \nu_x(t-1) + 1 \pmod{m_x} \quad \text{при} \quad k = x,$$
$$\nu_k(t) = \nu_k(t-1) \quad \text{при} \quad k \neq x.$$

Функция выходов $\psi(q(t-1), x(t))$ определяется соотношением

$$y(t) = (x(t),\ \nu_x(t)).$$

Б) АВТОМАТЫ С МАГАЗИННОЙ ПАМЯТЬЮ И БЕСКОНТЕКСТНЫЕ ЯЗЫКИ

Автоматы и преобразователи с магазинной памятью играют важную роль при построении автоматно-лингвистических моделей различного назначения, связанных с использованием бесконтекстных (контестно-свободных) языков. В частности, такие устройства используются в большинстве работающих программ для синтаксического анализа программ, написанных на различных языках программирования, которые во многих случаях можно рассматривать как бесконтекстные.

В отличие от конечных автоматов и преобразователей, рассмотренных в предыдущем параграфе, *автоматы и преобразователи с магазинной памятью* снабжены дополнительной магазинной памятью (рабочей лентой). На рис. 16-18 изображен такой преобразователь. Конечное управляющее устройство снабжается дополнительной управляющей головкой, всегда указывающей на верхнюю ячейку магазинной памяти; за один такт работы автомата (преобразователя) управляющая головка может произвести следующие движения:

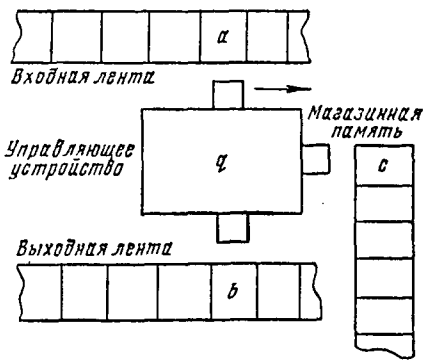


Рис. 16-18. Интерпретация преобразователя с магазинной памятью.

1) стереть символ из верхней ячейки (при этом все символы, находящиеся на рабочей ленте, перемещаются на одну ячейку вверх);

2) стереть символ из верхней ячейки и записать на рабочую ленту непустую цепочку символов (при этом содержимое рабочей ленты сдвигается вниз ровно настолько, какова длина записываемой цепочки).

Таким образом, устройство магазинной памяти можно сравнить с устройством магазина боевого автомата: когда в него вкладывается патрон, те, которые уже были внутри, проталкиваются вниз; достать можно только патрон, вложенный последним.

Формально *детерминированный магазинный автомат* определяется как следующая совокупность объектов:

$$M = (V,\ Q,\ V_M,\ \delta,\ q_0,\ z_0,\ F),$$

где V, Q, $q_0 \in Q$, F определяются так же, как и для конечного автомата;

$V_M = \{z_0, z_1, \ldots, z_{p-1}\}$ — алфавит магазинных символов автомата;

δ — функция, отображающая множество $Q \times (V \cup \{\varepsilon\}) \times V_M$ в множество $Q \times V_M$, где ε — пустая цепочка;

$z_0 \in V_M$ — так называемый граничный маркер, т. е. символ, первым появляющийся в магазинной памяти.

Недетерминированный магазинный автомат отличается от детерминированного только тем, что функция δ отображает множество $Q \times (V \cup \{\varepsilon\}) \times V_M$ в множество конечных подмножеств $Q \times V_M$.

Как и в случае конечных автоматов, *преобразователи с магазинной памятью* отличаются от автоматов с магазинной памятью наличием выходной ленты.

Далее будем рассматривать только недетерминированные магазинные автоматы.

Рассмотрим интерпретацию функции δ для такого автомата. Эту функцию можно представить совокупностью команд вида

$$(q,\ a,\ z) \to (q_1,\ \gamma_1),\ \ldots,\ (q_m,\ \gamma_m),$$

где $q,\ q_1,\ \ldots,\ q_m \in Q,\ a \in V,\ z \in V_M,\ \gamma_1,\ \ldots,\ \gamma_m \in V_M^*$.

При этом считается, что если на входе читающей головки автомата находится символ a, автомат находится в состоянии q, а верхний символ рабочей ленты z, то автомат может перейти к состоянию q_i, записав при этом на рабочую ленту цепочку γ_i $(1 \leqslant i \leqslant m)$ вместо символа z, передвинуть входную головку на один символ вправо так, как это показано на рис. 16-18, и перейти в состояние q_i. Крайний левый символ γ_i должен при этом оказаться в верхней ячейке магазина. Команда $(q,\ \varepsilon,\ z) \to (q_1,\ \gamma_1),\ \ldots,\ (q_m,\ \gamma_m)$ означает, что независимо от входного символа и не передвигая входной головки, автомат перейдет в состояние q_i, заменив символ z магазина на цепочку γ_i $(1 \leqslant i \leqslant m)$.

Ситуацией магазинного автомата называется пара $(q,\ \gamma)$, где $q \in Q,\ \gamma \in V_M^*$. Между ситуациями магазинного автомата $(q,\ \gamma)$ и $(q',\ \gamma')$ устанавливается отношение, обозначаемое символом $\vdash$, если среди команд найдется такая, что

$$(q,\ a,\ z) \to (q_1,\ \gamma_1),\ \ldots,\ (q_m,\ \gamma_m),$$

причем $\gamma = z\beta,\ \gamma' = \gamma_i\beta,\ q' = q_i$ для некоторого $1 \leqslant i \leqslant m$ $(z \in V_M,\ \beta \in V_M^*)$.

Говорят, что магазинный автомат переходит из состояния $(q,\ \gamma)$ в состояние $(q',\ \gamma')$ и обозначают это следующим образом:

$$a\colon (q,\ \gamma) \vdash (q',\ \gamma').$$

Вводится и такое обозначение:

$$a_1 \ldots a_n\colon (q,\ \gamma) \vdash * (q',\ \gamma'),$$

если справедливо, что

$$a_i\colon (q_i,\ \gamma_i) \vdash (q_{i+1},\ \gamma_{i+1}),\ 1 \leqslant i \leqslant n,$$

где

$$a_i \in V,\ \gamma_1 = \gamma,\ \gamma_2,\ \ldots,\ \gamma_{n+1} = \gamma' \in V_M^*;$$
$$q_1 = q,\ q_2,\ \ldots,\ q_{n+1} = q' \in Q.$$

Существует два способа определения языка, допускаемого магазинным автоматом. Согласно первому способу считается, что входная цепочка $\alpha \in V^*$ принадлежит языку $L_1(M)$ тогда, когда после просмотра последнего символа, входящего в эту цепочку, в магазине автомата M будет находиться пустая цепочка ε. Другими словами,

$$L_1(M) = \{\alpha \mid \alpha\colon (q_0,\ z_0) \vdash * (q,\ \varepsilon)\},$$

где $q \in Q$.

Согласно второму способу считается, что входная цепочка принадлежит языку $L_2(M)$ тогда, когда после просмотра последнего символа, входящего в эту цепочку, автомат M окажется в одном из своих заключительных состояний $q_f \in F$. Другими словами,

$$L_2(M) = \{\alpha \mid \alpha\colon (q_0,\ z_0) \vdash * (q_f,\ \gamma)\},$$

где $\gamma \in V_M^*,\ q_f \in F$.

Доказано, что множество языков, допускаемых произвольными магазинными автоматами согласно первому способу, совпадает с множеством языков, допускаемых согласно второму способу.

Доказано также, что если $L(G_2)$ — бесконтекстный язык, порождаемый грамматикой $G_2 = (V_N, V_T, P, S)$, являющейся нормальной формой Грейбах (см. § 16-3), произвольной бесконтекстной грамматики G, то существует недетерминированный магазинный автомат M такой, что $L_1(M) = L(G_2)$. При этом

$$M = (V,\ Q,\ V_M,\ \delta,\ q_0,\ z_0,\ \Phi),$$

где $V = V_T;\ Q = \{q_0\};\ V_M = V_N,\ z_0 = S$,
а для каждого правила G_2 вида

$$A \to a\alpha,\ a \in V_T,\ \alpha \in V_N^*$$

строится команда отображения δ:

$$(q_0,\ a,\ A) \to (q_0,\ \alpha).$$

Аналогично для любого недетерминированного магазинного автомата M, допускающего язык $L_1(M)$, можно построить бесконтекстную грамматику G такую, что $L(G) = L_1(M)$.

Если для конечных автоматов детерминированные и недетерминированные модели эквивалентны по отношению к классу допускаемых языков, то этого нельзя сказать для магазинных автоматов. Детерминированные автоматы с магазинной памятью допускают лишь некоторое подмножество бесконтекстных языков, которые называют детерминированными бесконтекстными языками.

Пример 16-17. Рассмотрим бесконтекстный язык $L = \{\alpha\beta\}$, словарь которого $V = \{ab\}$, а цепочка β является «зеркальным отражением» цепочки α, т. е.

$$\alpha = a_1 \ldots a_n,$$
$$\beta = b_1 \ldots b_n,$$

причем $a_i = b_{n-i+1},\ n \geqslant 1,\ 1 \leqslant i \leqslant n$.

Можно задать такой язык с помощью КС-грамматики в нормальной форме Грейбах:

$$G = (V_T,\ V_N,\ P,\ S),$$

где $V_T = \{a, b\},\ V_N = \{S, A, B\}$, а множество P состоит из следующих продукций:

$$S \to aSA,\quad S \to bSB,\quad S \to aA,\quad S \to bB,\quad A \to a,\quad B \to b.$$

Для этой грамматики построим соответствующий ей недетерминированный автомат с магазинной памятью

$$M=(V,\ Q,\ V_M,\ \delta,\ q_0,\ z_0,\ F),$$

для которого $V=V_T=\{a, b\}$; $Q=\{q_0\}$;

$$V_M=V_N=\{S,\ A,\ B\};\quad z_0=S;\quad F=\Phi,$$

а отображение δ соответствует множеству команд:

$$(q_0,\ a,\ S)\to(q_0,\ SA),\ (q_0,\ A),$$
$$(q_0,\ b,\ S)\to(q_0,\ SB),\ (q_0,\ B),$$
$$(q_0,\ a,\ A)\to(q_0,\ \varepsilon),$$
$$(q_0,\ b,\ B)\to(q_0,\ \varepsilon).$$

На рис. 16-19 изображены этапы просмотра цепочки *abba* и соответствующие этим этапам состояния рабочей ленты (магазинной памяти). Считывание последнего символа цепочки сопровождается переходом автомата в состояние, при

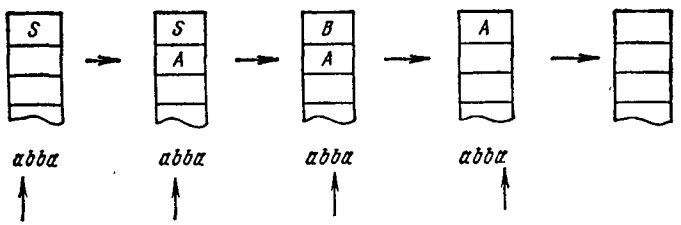


Рис. 16-19. Этапы просмотра цепочки *abba* магазинным автоматом.

котором рабочая лента содержит пустую цепочку. Таким образом, входная цепочка *abba* допускается данным автоматом [принадлежит языку $L_1(M)$].

в) МАШИНЫ ТЬЮРИНГА И ЛИНЕЙНО-ОГРАНИЧЕННЫЕ АВТОМАТЫ

Машины Тьюринга представляют собой абстрактные устройства самого общего типа и являются обобщением автоматов, рассмотренных в предыдущих параграфах. Этот класс абстрактных машин был рассмотрен еще в 1936 г., задолго до создания первых электронных вычислительных машин английским математиком и логиком А. Тьюрингом с целью формализации понятия вычислимости.

Если говорить об абстрактной схеме строения, то оказывается, что машина Тьюринга не так уж далека от реальных ЭВМ, т. е. она представляет собой хорошую математическую модель вычислительной машины.

С точки зрения лингвистики машины Тьюринга можно рассматривать как распознающие устройства, допускающие языки самого широкого из рассмотренных в предыдущих разделах классов — языки типа 0 или рекурсивно-перечислимые множества.

Машина Тьюринга состоит из конечного управляющего устройства, входной ленты и головки, которая в отличие от головки конечного автомата может не только считывать символы с ленты, но и записывать на нее новые символы. Лента считается бесконечной. Перед началом работы n ячеек ленты содержат символы входной цепочки $\alpha_i=a_{i_1}a_{i_2}\dots a_{i_n}$, все остальные ячейки считаются заполненными специальным символом B («пустое место»), который не является входным (рис. 16-20).

Формально *машина Тьюринга* определяется как следующая шестерка:

$$T=(V_1,\ V_2,\ Q,\ \delta,\ q_0,\ F),$$

где $V_1=\{a_1,\ \dots,\ a_m\}$ — входной алфавит (конечное множество символов);

$V_2=\{A_1,\ \dots,\ A_k,\ B\}$ — конечное множество ленточных символов, которое в качестве своего подмножества содержит входной алфавит;

$Q=\{q_0,\ q_1,\ \dots,\ q_{n-1}\}$ — конечное множество состояний;

$q_0\in Q$ — начальное состояние;

$F\subseteq Q$ — множество заключительных состояний, а

δ — функция, отображающая $Q\times V_2$ в $Q\times(V_2-\{B\})\times\{Л, П\}$ (Л и П — специальные символы, указывающие на направление движения головки).

Отображение (функцию) δ удобно задавать совокупностью команд вида $(q,\ A)\to(q',\ A',\ Л)$, либо $(q,\ A)\to(q',\ A',\ П)$.

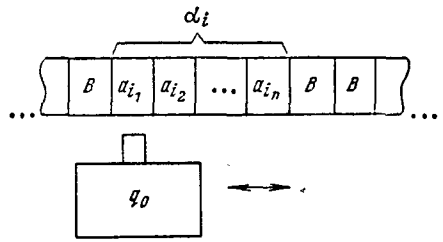


Рис. 16-20. Интерпретация машины Тьюринга.

Ситуация машины Тьюринга T — это тройка вида $(q,\ \beta,\ i)$, где $q\in Q$;

$\beta=A_1\ \dots\ A_n$ — часть ленты, не содержащая символов B (непустая часть ленты);

i $(0\leqslant i\leqslant n+1)$ — расстояние ленточной (пишущей — читающей) головки от левого конца β; при $i=0$ головка находится левее самого левого символа β, при $i=n+1$ правее самого правого.

Рассмотрим произвольную ситуацию машины T:

$$(q,\ A_1\dots A_i\dots A_n,\ i),\ 1\leqslant i\leqslant n.$$

Пусть среди команд отображения δ имеется следующая:

$$(q,\ A_i)\to(q',\ X,\ Л).$$

При этом возможно следующее движение (или элементарное действие) машины Тьюринга: головка стирает символ A_i, записывает вместо него символ X и перемещается на одну ячейку влево.

Между старой и вновь возникшей ситуациями в этом случае существует отношение, которое записывается следующим образом:

$$(q,\ A_1\dots A_i\dots A_n,\ i)\vdash(q',\ A_1\dots A_{i-1}XA_{i+1}\dots A_n,\ i-1).$$

Аналогично для команды $(q,\ A_i) \to (q',\ X,\ \text{П})$ движение машины Тьюринга T записывается как

$$(q,\ A_1 \dots A_i \dots A_n,\ i) \vdash (q',\ A_1 \dots A_{i-1} X A_{i+1} \dots A_n,\ i+1).$$

Кроме рассмотренной ситуации возможны и такие:

$$(q,\ A_1 \dots A_n,\ 0),$$
$$(q,\ A_1 \dots A_n,\ n+1).$$

К ним применимы команды вида $(q,\ B) \to (q',\ X,\ \text{Л})$, либо $(q,\ B) \to (q',\ X,\ \text{П})$.

Первая их этих команд меняет указанные ситуации соответственно следующим образом:

$$(q,\ A_1 \dots A_n,\ 0) \vdash (q',\ X A_1 \dots A_n,\ 0),$$
$$(q,\ A_1 \dots A_n,\ n+1) \vdash (q',\ A_1 \dots A_n X,\ n).$$

Вторая из этих команд меняет их так:

$$(q,\ A_1 \dots A_n,\ 0) \vdash (q',\ X A_1 \dots A_n,\ 2),$$
$$(q,\ A_1 \dots A_n,\ n+1) \vdash (q',\ A_1 \dots A_n X,\ n+2).$$

Если ситуации $(q_1,\ \beta_1,\ i_1)$ и $(q_2,\ \beta_2,\ i_2)$ связаны между собой некоторым числом элементарных действий, то между ними имеет место отношение

$$(q_1,\ \beta_1,\ i_1) \vdash * (q_2,\ \beta_2,\ i_2).$$

Язык, допускаемый машиной Тьюринга T, это

$$L(T) = \{\alpha \mid \alpha \in V_1^* \wedge (q_0,\ \alpha,\ 1) \vdash * (q_f,\ \beta,\ i)\},$$

где $q_f \in F$, $\beta \in V_2^*$, $i \geqslant 0$.

Другими словами, если, преобразуя входную цепочку α, машина T окажется в одном из своих заключительных состояний, эта цепочка допускается данной машиной.

Пример 16-18. Рассмотрим язык описания равносторонних треугольников из примера 16-4: $L = \{l^n r^n x^n \mid n \geqslant 1\}$.

Этот язык допускает следующая машина Тьюринга:

$$T = (V_1,\ V_2,\ Q,\ \delta,\ q_0,\ F),$$

где

$$V_1 = \{l,\ r,\ x\};\quad V_2 = \{l,\ r,\ x,\ L,\ R,\ X,\ Z,\ B\};$$
$$Q = \{q_0,\ q_1,\ \dots,\ q_{12}\};\quad F = \{q_{12}\},$$

а отображение δ задается следующими командами:

$(q_0,\ l) \to (q_1,\ l,\ \text{Л})$; $(q_7,\ R) \to (q_7,\ R,\ \text{Л})$;
$(q_1,\ B) \to (q_2,\ z,\ \text{П})$; $(q_7,\ l) \to (q_7,\ l,\ \text{Л})$;
$(q_2,\ l) \to (q_3,\ L,\ \text{П})$; $(q_7,\ L) \to (q_7,\ L,\ \text{Л})$;
$(q_3,\ l) \to (q_3,\ l,\ \text{П})$; $(q_7,\ z) \to (q_8,\ Z,\ \text{П})$;
$(q_3,\ r) \to (q_4,\ R,\ \text{П})$; $(q_8,\ L) \to (q_8,\ L,\ \text{П})$;
$(q_4,\ r) \to (q_4,\ r,\ \text{П})$; $(q_8,\ R) \to (q_8,\ R,\ \text{П})$;
$(q_4,\ x) \to (q_5,\ X,\ \text{П})$; $(q_8,\ X) \to (q_8,\ X,\ \text{П})$;
$(q_5,\ x) \to (q_5,\ x,\ \text{П})$; $(q_8,\ Z) \to (q_{12},\ Z,\ \text{Л})$;

$(q_5,\ B) \to (q_6,\ Z,\ \text{Л})$; $(q_9,\ l) \to (q_9,\ L,\ \text{П})$;
$(q_6,\ X) \to (q_6,\ X,\ \text{Л})$; $(q_9,\ l) \to (q_9,\ l,\ \text{П})$;
$(q_6,\ R) \to (q_6,\ R,\ \text{Л})$; $(q_9,\ R) \to (q_9,\ R,\ \text{П})$;
$(q_6,\ L) \to (q_6,\ L,\ \text{Л})$; $(q_9,\ r) \to (q_{10},\ R,\ \text{П})$;
$(q_6,\ Z) \to (q_{12},\ Z,\ \text{П})$; $(q_{10},\ r) \to (q_{10},\ r,\ \text{П})$;
$(q_6,\ x) \to (q_7,\ x,\ \text{Л})$; $(q_{10},\ X) \to (q_{10},\ X,\ \text{П})$;
$(q_7,\ x) \to (q_7,\ x,\ \text{Л})$; $(q_{10},\ x) \to (q_{11},\ X,\ \text{П})$;
$(q_7,\ X) \to (q_7,\ X,\ \text{Л})$; $(q_{11},\ x) \to (q_{11},\ x,\ \text{П})$;
$(q_7,\ r) \to (q_7,\ r,\ \text{Л})$; $(q_{11},\ Z) \to (q_8,\ Z,\ \text{Л})$.

Последовательность действий для входной цепочки $\alpha = lrx$ следующая:

$$(q_0,\ lrx,\ 1) \vdash (q_1,\ lrx,\ 0) \vdash (q_2,\ Zlrx,\ 2) \vdash (q_3,\ ZLrx,\ 3) \vdash (q_4,\ ZLRx,\ 4) \vdash$$
$$\vdash (q_5,\ ZLRX,\ 5) \vdash (q_6,\ ZLRXZ,\ 4) \vdash (q_6,\ ZLRXZ,\ 3) \vdash$$
$$\vdash (q_6,\ ZLRXZ,\ 2) \vdash (q_6,\ ZLRXZ,\ 1) \vdash (q_{12},\ ZLRXZ,\ 2).$$

Так же как для автоматов, введем понятие недетерминированной машины Тьюринга. Ее отличие от детерминированной машины заключается в том, что функция δ отображает множество $Q \times V_2$ в множество подмножеств $Q \times (V_2 - \{B\}) \times \{\text{Л},\ \text{П}\}$.

Если язык L порождается грамматикой типа 0, то L допускается некоторой машиной Тьюринга. Верно и обратное, если язык L допускается некоторой машиной Тьюринга, то L порождается грамматикой типа 0.

Ввиду того что рассмотренные типы автоматов и машин Тьюринга часто используются для построения автоматно-лингвистических моделей, предназначенных для распознавания языков, необходимо знать, разрешима для них так называемая проблема распознавания или нет.

Эта проблема заключается в следующем. Пусть есть некоторая цепочка α на входе устройства (машины Тьюринга), которое допускает язык L. Всегда ли можно установить принадлежность цепочки α к языку L за конечное число элементарных действий этой машины?

Однако не для всех языков типа 0 эта проблема разрешима. Другими словами, можно подобрать такой язык типа 0, что соответствующая ему машина Тьюринга для некоторой цепочки α за конечное число элементарных действий не сможет установить принадлежность ее к данному языку. Поэтому машина Тьюринга в общем виде не нашла применения в реальных кибернетических моделях; языки типа 0 также не используются. Наибольший интерес представляют различные специальные классы машин Тьюринга, к которым можно отнести автоматы, рассмотренные в предыдущих параграфах, а также так называемые линейно-ограниченные автоматы, допускающие языки типа 1 (НС-языки).

Линейно-ограниченным автоматом называется следующая шестерка:

$$M = (V_1,\ V_2,\ Q,\ \delta,\ q_0,\ F),$$

где $V_1 = \{a_1,\ \dots,\ a_m,\ Z_l,\ Z_p\}$ — конечный входной алфавит;

$V_2 = \{A_1, \ldots, A_k\}$ — конечное множество ленточных символов, причем $V_1 \subseteq V_2$;

$Q = \{q_0, q_1, \ldots, q_{n-1}\}$ — конечное множество состояний;

$q_0 \in Q$ — начальное состояние;

$F \subseteq Q$ — множество заключительных состояний;

δ — функция, отображающая $Q \times V_2$ в множество подмножеств $Q \times V_2 \times \{Л, П\}$.

Множество V_1 содержит два специальных символа Z_l и Z_p, называемых граничными маркерами, которые не позволяют головке управляющего устройства уйти с той части ленты, на которой задана входная цепочка.

Таким образом, линейно-ограниченный автомат является недетерминированной машиной Тьюринга с ограничением на длину цепочки просматриваемых символов. Ситуация линейно-ограниченного автомата и элементарное действие определяются так же, как и для машины Тьюринга. Язык, допускаемый линейно-ограниченным автоматом, определяется как множество

$$L(M) = \{\alpha \mid \alpha \in (V_1 - \{Z_l, Z_p\})^* \wedge (q_0, Z_l \alpha Z_p, 1) \vdash * (q_f, \beta, i)\},$$

где $q_f \in F$, $\beta \in V_2^*$, $1 \leqslant i \leqslant n$ (n — длина исходной цепочки).

Пример 16-19. Рассмотрим тот же язык, что и в примере 16-18. Этот язык относится к классу языков типа 1. Линейно-ограниченный автомат, допускающий этот язык, следующий:

$$M = (V_1, V_2, Q, \delta, q_0, F),$$

где

$$V_1 = \{l, r, x, Z_l, Z_p\}; \quad V_2 = \{l, r, x, L, R, X, Z_l, Z_p\};$$
$$Q = \{q_1, q_2, \ldots, q_{12}\}; \quad q_0 = q_1, \quad F = \{q_{12}\},$$

а отображение δ отличается от отображения машины Тьюринга из примера 16-18 следующим:

команда $(q_0, l) \to (q_1, l, Л)$ отсутствует; команды

$$(q_1, B) \to (q_2, Z, П); \qquad (q_5, B) \to (q_6, Z, Л);$$
$$(q_6, Z) \to (q_{12}, Z, П); \qquad (q_7, Z) \to (q_8, Z, П);$$
$$(q_8, Z) \to (q_{12}, Z, Л); \qquad (q_{11}, Z) \to (q_6, Z, Л)$$

заменяются соответственно на следующие команды:

$$(q_1, Z_l) \to (q_2, Z_l, П); \qquad (q_5, Z_p) \to (q_5, Z_p, Л);$$
$$(q_6, Z_l) \to (q_{12}, Z_l, П); \qquad (q_7, Z_l) \to (q_8, Z_l, П);$$
$$(q_8, Z_p) \to (q_{12}, Z_p, Л); \qquad (q_{11}, Z_p) \to (q_6, Z_p, Л).$$

Последовательность действий для входной цепочки $\alpha = Z_l l r x Z_p$ следующая:

$$(q_1, Z_l l r x Z_p, 1) \vdash (q_2, Z_l l r x Z_p, 2) \vdash (q_3, Z_l L r x Z_p, 3) \vdash (q_4, Z_l L R x Z_p, 4) \vdash$$
$$\vdash (q_5, Z_l L R X Z_p, 5) \vdash (q_6, Z_l L R X Z_p, 4) \vdash (q_6, Z_l L R X Z_p, 3) \vdash$$
$$\vdash (q_6, Z_l L R X Z_p, 2) \vdash (q_6, Z_l L R X Z_p, 1) \vdash (q_{12}, Z_l L R X Z_p, 2).$$

Детерминированный линейно-ограниченный автомат отличается от недетерминированного тем, что функция δ отображает множество $Q \times V_2$ в множество $Q \times V_2 \times \{Л, П\}$.

В настоящее время не известно, является ли класс языков, допускаемых детерминированными линейно-ограниченными автоматами, собственным подклассом класса языков, допускаемых недетерминированными линейно-ограниченными автоматами. Однако для недетерминированных линейно-ограниченных автоматов установлено, что допускаемые ими языки являются НС-языками. Более того, доказано, что для любого НС-языка можно построить недетерминированный линейно-ограниченный автомат.

16-7. ГРАММАТИКИ И ЕСТЕСТВЕННЫЕ ЯЗЫКИ

а) Грамматики, используемые в машинных лингвистических анализаторах

Теория формальных грамматик, элементы которой излагались в предыдущих параграфах, своему появлению во многом обязана многочисленным исследованиям естественных языков, таких как английский, французский, русский. Так, например, возникновение понятий автоматного языка, регулярного множества, марковской цепи было связано со статистическим исследованием последовательностей гласных и согласных в литературном тексте; введенное Н. Хомским понятие формальной грамматики возникло в результате усилий, направленных на создание строгих описаний грамматических закономерностей естественного языка.

В настоящее время естественный язык продолжает оставаться объектом пристального изучения специалистов в области кибернетики. Основная цель исследований естественного языка — это построение таких описаний этих языков, которые могли бы быть полностью формализованы с помощью абстрактного автомата, формальной грамматики или какой-либо алгебраической конструкции. Эти работы, помимо важного теоретического значения, имеют и практическую ценность. Они важны, например, для автоматизации анализа и синтеза текстов с помощью ЭВМ.

Естественный язык является наиболее универсальным средством для моделирования и описания творческого мышления человека в СИИ, в системах, использующих общение человека с ЭВМ, где актуально ставится вопрос о расширении семантической функции машины, для того чтобы она могла «понимать» естественно-языковую информацию и «разумно» отвечать на вопросы.

Как показали многочисленные исследования, описание механизмов, посредством которых люди строят и понимают высказывания, с помощью традиционных грамматик весьма несовершенно. Прежде всего было установлено, что естественные языки нельзя рассматривать как бесконтекстные. Однако КС-грамматики являются наиболее исследованной моделью для описания искусственных языков (например, языков программирования), а также ограниченных естественных языков. Глубина теоретических исследований и обширный опыт применения бесконтекстных грамматик

в практических приложениях говорят о том, что они все же могут быть использованы для синтаксического описания правильных фраз естественного языка, которое при этом не будет обладать достаточной объяснительной силой. *Объяснительной силой* описания фразы языка называют [54] способность синтаксической структуры этой фразы (например, дерева вывода) обеспечивать ее семантическую интерпретацию. А без семантической интерпретации, как было отмечено в § 16-5, не имеет смысла говорить о моделировании процесса понимания естественного языка.

Грамматики непосредственных составляющих (НС-грамматики) представляют собой класс грамматик, достаточный для описания естественных языков в их полном объеме.

Однако использование НС-языков на практике в значительной степени тормозится отсутствием приемлемых по эффективности и общности алгоритмов анализа таких языков и большим количеством алгоритмически неразрешимых проблем.

Кроме того, НС-грамматики, как и все порождающие (распознающие) механизмы, дают лишь правила генерации (анализа) цепочек языка. В них отсутствуют многоместные операции — правила преобразования правильно построенных выражений. Для естественного языка это приводит к тому, что существенно увеличивается сложность описания. Так, описание активной и пассивной формы предложений, утвердительных и вопросительных предложений и т. д. требует введения самостоятельных порождающих правил. С другой стороны, очевидно, что смысл фразы не зависит, по крайней мере, так жестко, от способа его выражения.

Ввиду того что синтаксический анализ играет важную роль в системах, предназначенных для понимания естественного языка, прежде всего рассмотрим методы грамматического разбора языков, порождаемых различными классами грамматик, а также более детальную классификацию формальных грамматик, наложенную на базовую классификацию (см. § 16-2). Вводя различные классы грамматик, мы не акцентировали внимание на том, что их определения дают, по существу, механизмы порождения цепочек. Вместе с тем очевидно, что те же устройства при инвертировании продукций, т. е. при изменении порядка следования левой и правой частей продукций, позволяют говорить об анализе (распознавании, допускании) цепочек языка.

Целью синтаксического анализа (грамматического разбора) является не только получение ответа на вопрос, принадлежит или нет к данному языку входная цепочка символов, но и построение структуры входной цепочки (дерева ее вывода). Именно этим вопросам уделяется наибольшее внимание при реализации языков программирования в работах по созданию трансляторов. При этом разработчиков помимо удобства описания интересует эффективность анализа языков, порождаемых различными классами грамматик.

В ходе исследований по методам анализа языковых текстов (цепочек языка) были предложены две *стратегии* грамматического разбора: *нисходящая* и *восходящая*. В соответствии с этим все *распознаватели* (*анализаторы* или *парсеры*, от английского parse — разбор) делятся на *нисходящие* и *восходящие*. Однако на практике часто используются и смешанные стратегии.

Работа нисходящего распознавателя теоретически основывается на идее использования порождающей грамматики при генерации всех возможных цепочек языка, пока не будет порождена цепочка, соответствующая входной. Для этого необходимо предусмотреть проверку альтернатив и способ возврата из тупиков. Такая ситуация возникает, когда среди продукций грамматики имеются продукции с одинаковыми левыми частями вида

$$A \to \alpha, \quad A \to \beta,$$

где $A \in V_N$; α, $\beta \in (V_T \cup V_N)^*$.

Стратегия восходящего разбора состоит в том, что во входной строке ищутся некоторые подстроки — правые части продукций. Они заменяются соответствующими левыми частями (сворачиваются), и процесс повторяется до получения начального символа грамматики. Основной проблемой для восходящего распознавателя является проблема эффективного поиска сворачиваемой части входной строки и выбора альтернатив, когда выделенная подстрока редуцируется (сворачивается) неоднозначно. Формально такая ситуация характеризуется наличием продукций с одинаковыми правыми частями вида

$$A \to \alpha, \quad B \to \alpha,$$

где A, $B \in V_N$; $\alpha \in (V_T \cup V_N)^*$.

Таким образом, и нисходящий, и восходящий распознаватели при разборе исходной цепочки могут делать неверные редукции (замены, свертки). Отсюда вытекает необходимость возвратов и, как следствие, усложнение анализа. В случае грамматического разбора фраз естественного языка эта проблема не является критической в силу небольшой длины исходных цепочек. Если же анализируемыми цепочками являются тексты программ на языках программирования, возвраты в большинстве случаев недопустимы. Поэтому все методы грамматического разбора, предложенные для эффективного анализа текстов на искусственных языках, используют полное исключение или существенное сокращение возвратов и повторного анализа входной строки.

Специфика этих методов привела к разработке специальных алгоритмов для специальных подклассов грамматик.

Простейшим типом анализаторов является такой, в котором реализуются регулярные или автоматные грамматики. Такие анализаторы могут быть использованы в системах общения на естественных языках на этапе *морфологического анализа*, т. е. анализа отдельных слов, из которых строятся предложения естественного языка, а также в процессе разбора простых конструкций естественного языка. Как было показано в § 16-6, для анализа цепочек, порож-

даемых такими грамматиками, с успехом применяются конечные автоматы.

Существует много различных алгоритмов грамматического разбора регулярных языков, однако общим для всех этих алгоритмов является наличие в том или ином представлении *диаграммы* (*сети*, *графа*) *переходов*, подробно рассмотренной в § 16-6. Узлы такой сети помечаются нетерминальными символами грамматики, а условием прохождения любой дуги является совпадение терминального символа, которым она помечена, с очередным просматриваемым символом из входной строки.

Такая диаграмма позволяет ответить на вопрос, принадлежит ли некоторая цепочка языку. Однако конечной целью анализатора является не только это, но и семантическая обработка выделенных понятий. Для осуществления такой обработки дуги помечаются не только терминальными символами, но и командами обработки просматриваемых символом. Таким образом, в данном случае мы имеем дело не с конечным автоматом, а с конечным преобразователем (см. § 16-6).

Рассмотрим теперь методы грамматического разбора бесконтекстных языков.

Одним из наиболее известных и широко используемых методов, позволяющих осуществить эффективный восходящий разбор, является метод Флойда, предложенный для анализа специального подкласса бесконтекстных грамматик, называемых *грамматиками предшествования* Флойда [66]. На множестве терминальных символов такой грамматики вводятся так называемые *отношения предшествования* $\doteq$, $\cdot>$, $<\cdot$, смысл которых заключается в следующем.

Рассмотрим пару (a, b), где a, b — терминальные символы словаря грамматики $G = (V_T, V_N, P, S)$.

Будем говорить, что для этой пары выполняется отношение $\doteq$, если среди множества продукций P есть продукция вида

$$A \to \alpha ab \beta,$$

где $A \in V_N$; α, $\beta \in V^*$ $(V = V_T \cup V_N)$.

Будем говорить, что для пары (a, b) выполняется отношение $<\cdot$, если среди множества продукции P есть продукция вида

$$A \to \alpha aB\beta,$$

а из B выводима цепочка, начинающаяся с символа b, т. е.

$$B \Rightarrow * b\gamma,$$

где A, $B \in V_N$; α, β, $\gamma \in V^*$.

Будем говорить, что для пары (a, b) выполняется отношение $\cdot>$, если справедливо хотя бы одно из следующих утверждений:

1) среди множества продукций P содержится продукция вида

$$A \to \alpha Bb\beta$$

и из B выводима цепочка, заканчивающаяся символом a, т. е.

$$B \Rightarrow * \gamma a,$$

где A, $B \in V_N$; α, β, $\gamma \in V^*$;

2) среди множества продукций P содержится продукция вида

$$A \to \alpha BC\beta,$$

причем $B \Rightarrow * \gamma a$, $C \Rightarrow * b\delta$,
где A, B, $C \in V_N$; a, $b \in V_T$; α, β, γ, $\delta \in V^*$.

Флойд предложил алгоритм [66], позволяющий без возвратов анализировать любую цепочку, порождаемую грамматикой предшествования, если на эту грамматику наложены следующие ограничения:

нет продукций вида $A \to \alpha$, $B \to \alpha$, где

$$A, B \in V_N; \alpha \in (V_N \cup V_T)^*;$$

нет продукций вида $A \to \alpha BC\beta$, где A, B, $C \in V_N$;

$$\alpha, \beta \in (V_N \cup V_T)^*;$$

определенные ранее отношения предшествования однозначны, т. е. для любых двух терминальных символов существует не более одного отношения предшествования.

В силу введенных ограничений грамматики предшествования допускают весьма ограниченный класс бесконтекстных языков. Поэтому В. Вирт и Х. Вебер, а в дальнейшем Дж. Маккиман [66] расширили этот класс последовательным ослаблением ограничений: Вирт и Вебер ввели отношения предшествования и для нетерминальных символов, а Маккиман допустил неоднозначность этих отношений. Однако при этом эффективность разбора уменьшилась за счет существенного увеличения необходимых для хранения специальных таблиц памяти.

Все перечисленные методы предшествования основаны на использовании контекста из одного символа (в методе Вирта) или двух символов (в методе Маккимана). Дальнейшим развитием такого подхода явились LR (k)-*грамматики* Кнута [60]. В них для управления разбором используется принцип считывания вперед на заданные k символов, что позволяет на каждом шаге разбора однозначно выбирать верный путь его продолжения. Однако, к сожалению, для реализации таких алгоритмов на каждом шаге приходится строить все множество возможных продолжений до тех пор, пока возможная неоднозначность не будет устранена. Поэтому, хотя и было показано [60], что для любой бесконтекстной грамматики можно построить эквивалентную ей LR (k)-грамматику для $k \geqslant 1$, на практике они используются в основном лишь для $k = 1$.

С точки зрения стратегии разбора LR (k)-распознаватели занимают положение, промежуточное между восходящей и нисходящей стратегиями. Однако условно их можно отнести к классу нисхо-

дящих распознавателей, так как в них все-таки преобладает нисходящая стратегия разбора.

Следует отметить, что применение восходящей стратегии при анализе фраз естественного языка представляется мало перспективным, так как поиск деревьев вывода в этом случае в той или иной степени «слепой» и связан с перебором. Поэтому все дальнейшее рассмотрение методов грамматического разбора будет касаться лишь нисходящих (или преимущественно нисходящих) распознавателей.

К наиболее известным распознавателям такого типа можно отнести так называемые *распознаватели с медленным и быстрым возвратом*, а также *глобальные анализаторы* [60]. В них используется следующий принцип работы.

Пусть грамматика $G = (V_T, V_N, P, S)$ содержит k продукций вида

$$S \to \alpha_{i1} A_{i1} \alpha_{i2} \ldots A_{in-1} \alpha_{in},$$

где $\alpha_{ij} \in V_T^*$, $A_{ij} \in V_N$ $(i = 1, 2, \ldots, k)$.

Пусть β — анализируемая цепочка. Тогда, если выбрать одну из k этих продукций, цепочка β должна содержать в качестве подцепочек цепочки α_{i1}, α_{i2}, ..., α_{in}, содержащиеся в правой части выбранной продукции. Если это не так, то данная продукция считается неприменимой и осуществляется переход к рассмотрению другой продукции. В свою очередь, для каждого нетерминального символа A_{ij} осуществляется точно такая же проверка, как и для символа S. Таким образом, анализ в данном случае представляется в виде рекурсивной процедуры, причем он должен заканчиваться за конечное число шагов, если в грамматике нет выводов с повторяющимися цепочками. Необходимость осуществлять *возвраты* в данном случае связана с недетерминированностью рассмотренного процесса анализа.

Процедура проверки может быть существенно ускорена за счет рассмотрения индивидуальных особенностей используемой в анализаторе грамматики. При этом применяют так называемые *ускоряющие тесты*. Пусть, например, анализируемая цепочка β имеет вид: *abcaba*, а продукция, для которой проверяется выводимость этой цепочки, такова:

$$C \to AbB.$$

Возможны два разбиения цепочки β, но при этом, например, известно, что из символа B выводятся только такие цепочки, которые начинаются с символа c. Поэтому из двух разбиений сразу же оставляем только одно, удовлетворяющее этому ограничению.

Анализаторы рассмотренного типа работают быстрее, чем LR (k)-анализаторы, особенно при удачном выборе ускоряющих тестов.

Использование *контекста* может еще более увеличить эффективность грамматического разбора. В связи с этим отметим, что рассмотренные в § 16-5 программные и индексные грамматики, порождающие языки из класса более широкого, чем класс бесконтекстных языков, могут служить основой для построения эффективных анализаторов, использующих контекстные условия.

Из современных методов грамматического разбора, непосредственно связанных с анализом предложений на естественном языке, отметим два наиболее известных. Во-первых, это так называемый *процедурный метод грамматического разбора* Винóграда, основанный на использовании *системных грамматик* Холидея [67]. Во-вторых, это так называемый *сетевой метод*, основанный на применении *расширенных сетей переходов* Вудса [61]. В обоих этих методах используется нисходящая стратегия распознавания. Системные грамматики Холидея, а также расширенные сети переходов Вудса представляют собой очень мощные механизмы, что позволяет использовать их для анализа достаточно богатых подмножеств естественного языка. В частности, расширенные сети переходов, которые будут подробно рассмотрены в п. «б» настоящего параграфа, имеют мощность машин Тьюринга и поэтому могут обрабатывать грамматику любого типа.

Рассмотрим процедурный метод грамматического разбора Винóграда. В отличие от традиционных методов разбора в этом методе синтаксис языка рассматривается в тесной связи с семантикой. Такой подход аргументируется тем, что естественный язык возник как средство коммуникации между людьми. При этом говорящий кодирует смысл (значение) произносимой фразы, закладывая его в ряд *синтаксических признаков*, которые он вносит в эту фразу в процессе ее порождения (генерации). Проблема для слушающего состоит в том, чтобы опознать присутствие таких признаков и использовать их для семантической интерпретации этой фразы. Для системных грамматик Холидея, лежащих в основе метода синтаксического анализа Винóграда, такое опознание синтаксических признаков имеет первостепенное значение, так как предполагается, что значение (смысл) фразы играет важнейшую роль в формировании ее структуры. При этом исследуются *системные сети*, описывающие способы, посредством которых различные признаки взаимодействуют и влияют друг на друга.

К синтаксическим признакам относятся, например, такие признаки категории ПРЕДЛОЖЕНИЕ (CLAUSE): MAJOR (главное), SEC (второстепенное), DECLARATIVE (утвердительное), IMPERATIVE (побудительное), QUESTION (вопросительное) и т. д.

Работа по соотнесению набора признаков с фактической фразой на естественном языке выполняется в системной грамматике *правилами реализации*. При использовании этой грамматики для синтаксического анализа предложений на естественном языке вместо правил реализации рассматривают обратные им *правила интерпретации*, которые воспринимают входную строку (фразу), идентифицируют ее структуру и распознают ее признаки.

Процедурный метод синтаксического анализа Винóграда в качестве формализма для описания грамматики использует специаль-

ный язык программирования PROGRAMMAR [67], а система грамматического разбора представляет собой интерпретатор с этого языка. Он производит разбор, используя нисходящую стратегию. При этом характерной особенностью и несомненным достоинством процедурного метода Винограда является то, что непосредственно в сам процесс грамматического разбора внесена семантическая интерпретация, которая активно управляет процессом разбора. Для этого в ходе разбора подключаются те или иные *семантические программы*, играющие здесь роль ускоряющих тестов, как это делается в глобальном анализаторе.

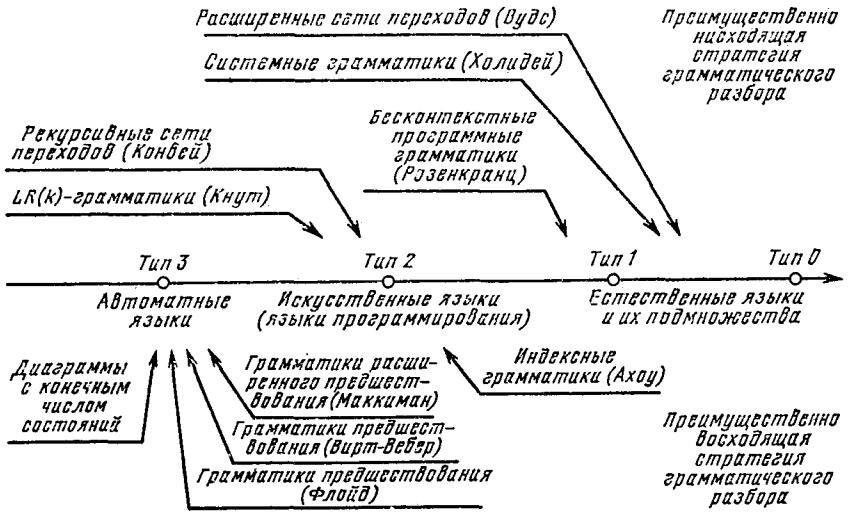


Рис. 16-21. Классификация грамматик с точки зрения их использования в синтаксических анализаторах.

Благодаря семантической интерпретации в ходе разбора PROGRAMMAR не имеет автоматического механизма возврата, потому что «слепой» возврат слишком неэффективен.

Вопросы семантической интерпретации, несомненно играющие определяющую роль при синтаксическом анализе естественного языка, базируются на проблеме представления знаний, которая будет подробно рассмотрена в гл. 17.

Другим перспективным подходом к построению машинных лингвистических анализаторов в системах общения человека с ЭВМ на естественном языке является сетевой метод Вудса. Этот метод синтаксического анализа основан на понятии расширенной сети переходов, которое в свою очередь является расширением понятия *рекурсивной сети переходов*, лежащей в основе анализатора бесконтекстных грамматик, впервые предложенного Конвеем [61]. В свою очередь рекурсивная сеть переходов представляет собой развитие

идеи распознающего автомата с конечным числом состояний (конечного автомата), о котором шла речь в § 16-6.

Рекурсивные и расширенные сети будут подробно рассмотрены далее.

Приведенный ранее обзор методов грамматического разбора, включающий два наиболее известных метода синтаксического анализа естественных языков, позволяет с этой точки зрения рассмотреть более детальную, чем в § 16-2, классификацию формальных грамматик как кибернетических моделей распознавания языков различных классов. На рис. 16-21 представлена эта классификация. Направление оси указывает на возрастание мощности языков. Отмечено также место искусственных языков (языков программирования) и языков, близких к естественному, в классической классификации Хомского.

Б) СЕТЕВЫЕ ГРАММАТИКИ ВУДСА

В. А. Вудс предложил модель, ориентированную на описание и анализ естественных языков, которую можно считать одной из наиболее удачных. Имеются в виду так называемые расширенные сети переходов Вудса, впервые введенные Конвеем [61], которые обобщают понятие конечного автомата (рассмотренного в § 16-6).

Конечные автоматы обладают тем свойством, что последовательность символов (в случае естественного языка в роли символов выступают слова), которые образуют цепочку (в данном случае предложение), считывается головкой конечного управляющего устройства слева направо; автомат при этом переходит из начального состояния в одно из заключительных состояний. Естественно, что конечные автоматы не могут служить адекватной моделью для представления грамматик естественного языка, поскольку они не способны охватить многие его закономерности. Порождающая сила такой модели эквивалентна автоматной грамматике, а естественные языки не могут быть адекватно описаны даже с помощью КС-грамматик. Поэтому к автоматной модели последовательно присоединяются механизмы, расширяющие ее возможности.

Прежде всего к автоматной модели добавляется механизм рекурсии. Устройство такого типа называется рекурсивной сетью переходов по названию модифицированной диаграммы переходов недетерминированного конечного автомата, к которой добавляется механизм рекурсии.

Такое устройство по своей конструкции напоминает автомат с магазинной памятью (§ 16-6). Однако работа данного устройства отличается от работы магазинного автомата прежде всего тем, что на рабочую ленту (в магазинную память) в данном случае могут заноситься лишь имена состояний автомата. При этом автомат может переходить к новому состоянию без продвижения вдоль входной ленты. Если автомат попадает в одно из своих заключительных состояний, но при этом входная цепочка еще полностью не проана-

лизирована, из магазина извлекается символ того состояния, которому передается управление. Цепочка считается допущенной данным автоматом, если одновременно после чтения последнего символа входной цепочки магазин пуст и автомат находится в одном из своих заключительных состояний.

Диаграмма состояний автомата (рекурсивная сеть переходов) в отличие от диаграммы состояний конечного автомата может содержать несколько не связанных между собой графов, вершины которых соответствуют состояниям автомата, а дуги могут быть помечены не только символами входного алфавита, но и символами состояний. Допускаются также непомеченные дуги. Дуга, помеченная именем состояния, имеет следующую интерпретацию: состояние, в которое ведет дуга, запоминается в магазинной памяти, а управление передается состоянию, которым помечена эта дуга. Если состояние заключительное, то управление передается состоянию, которое извлекается из магазинной памяти. Непомеченная дуга означает переход к новому состоянию без движения вдоль входной цепочки.

Построенное таким образом устройство допускает те же языки, что и автоматы с магазинной памятью, т. е. бесконтекстные языки. Преимуществом такого подхода является то, что он позволяет получить более компактное представление и более эффективные алгоритмы анализа КС-языков, а также то, что данную модель можно естественным образом «расширять» до более мощной модели, которая допускает НС-языки.

Пример 16-20. Рассмотрим пример рекурсивной сети переходов и работы автомата, соответствующего этой сети и допускающего язык, рассмотренный в примере 16-17. Цепочки этого языка имеют вид $\alpha\beta$, где цепочка β является «зеркальным отражением» цепочки $\alpha \in \{a, b\}^+$. Такой язык не является автоматным, так как можно построить грамматику, порождающую данный язык, в которой есть самовставляющиеся нетерминальные символы (см. пример 16-17).

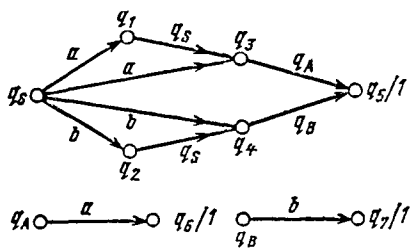


Рис. 16-22. Пример рекурсивной сети переходов.

Рекурсивная сеть переходов, соответствующая автомату, допускающему данный язык, изображена на рис. 16-22. Автомат может находиться в одном из десяти состояний:

$$q_S,\ q_A,\ q_B,\ q_1,\ q_2,\ q_3,\ q_4,\ q_5,\ q_6,\ q_7.$$

Заключительные состояния q_5, q_6, q_7 помечены специальным образом:

$$q_5/1,\quad q_6/1,\quad q_7/1.$$

Рекурсивная сеть переходов состоит из трех не связанных между собой графов, дуги которых помечены как символами алфавита $\{a, b\}$, так и символами состояний.

Работу автомата иллюстрирует рис. 16-23. Цепочка *abba* допускается данным автоматом, так как в конце его работы после считывания последнего символа цепочки автомат переходит в заключительное состояние q_5 при пустом магазине.

Пример 16-21. Рассмотрим маленький фрагмент русского языка, который может быть порожден КС-грамматикой, продукции которой зададим с помощью

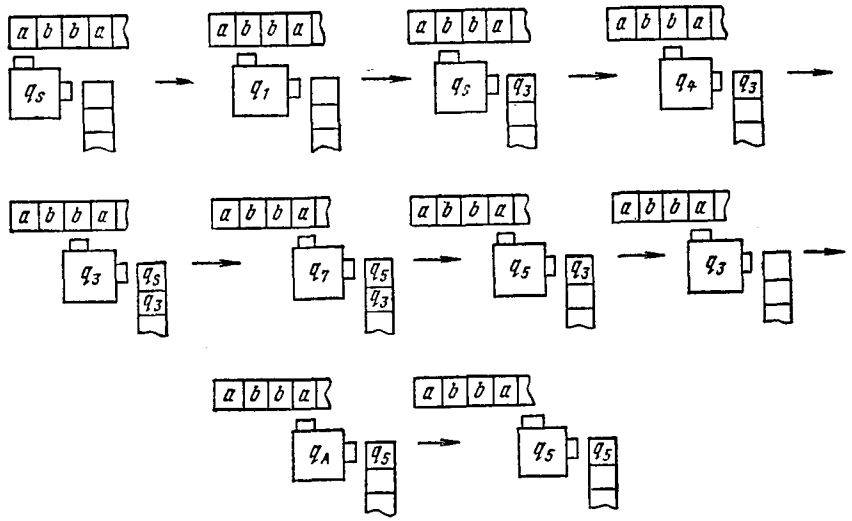


Рис. 16-23. Этапы анализа цепочки *abba* автоматом, соответствующим рекурсивной сети переходов.

нормальной формы Бэкуса, описанной в § 16-2:

⟨ПРЕДЛОЖЕНИЕ⟩ ::= ⟨ВОПРОСИТ.⟩ | ⟨УТВЕРДИТ.⟩
⟨УТВЕРДИТ.⟩ ::= ⟨ГРУППА СУЩ.⟩ ⟨ГЛАГОЛ⟩ | ⟨ГРУППА СУЩ.⟩ ⟨ГЛАГОЛ⟩ ⟨ОБСТ. МЕСТА⟩
⟨ВОПРОСИТ.⟩ ::= ⟨ГЛАГОЛ⟩ ⟨ЧАСТИЦА⟩ ⟨ГРУППА СУЩ.⟩ | ⟨ГЛАГОЛ⟩ ⟨ЧАСТИЦА⟩ ⟨ГРУППА СУЩ.⟩ ⟨ОБСТ. МЕСТА⟩
⟨ГРУППА СУЩ.⟩ ::= ⟨СУЩ.⟩ | ⟨ПРИЛАГ.⟩ ⟨ГРУППА СУЩ.⟩
⟨ОБСТ. МЕСТА⟩ ::= ⟨ПРЕДЛОГ⟩ ⟨ГРУППА СУЩ.⟩
⟨СУЩ.⟩ ::= (МИША) | (КОШКА) | (СТОЛ) | (ДИВАН)
⟨ГЛАГОЛ⟩ ::= (СИДЕТЬ) | (ЛЕЖАТЬ)
⟨ПРИЛАГ.⟩ ::= (ЧЕРНЫЙ) | (МЯГКИЙ)
⟨ПРЕДЛОГ⟩ ::= (ЗА) | (НА)
⟨ЧАСТИЦА⟩ ::= (ЛИ)

С помощью этой грамматики можно порождать предложения, соответствующие, таким, например, фразам русского языка:

МИША СИДИТ ЗА СТОЛОМ.
ЧЕРНАЯ КОШКА ЛЕЖИТ НА МЯГКОМ ДИВАНЕ.
СИДИТ ЛИ МИША ЗА СТОЛОМ?
ЛЕЖИТ ЛИ КОШКА?

На рис. 16-24 приведена рекурсивная сеть переходов для автомата, допускающего язык, порождаемый описанной выше грамматикой. Отметим, что символы цепочек, допускаемых автоматом (порождаемых грамматикой), в данном случае представляют собой так называемые «нормализованные» слова русского языка, заключенные в скобки. Так, фразе МИША СИДИТ ЗА СТОЛОМ соответствует цепочка (МИША) (СИДЕТЬ) (ЗА) (СТОЛ).

«Нормализация» обычно предшествует этапу синтаксического анализа фразы.

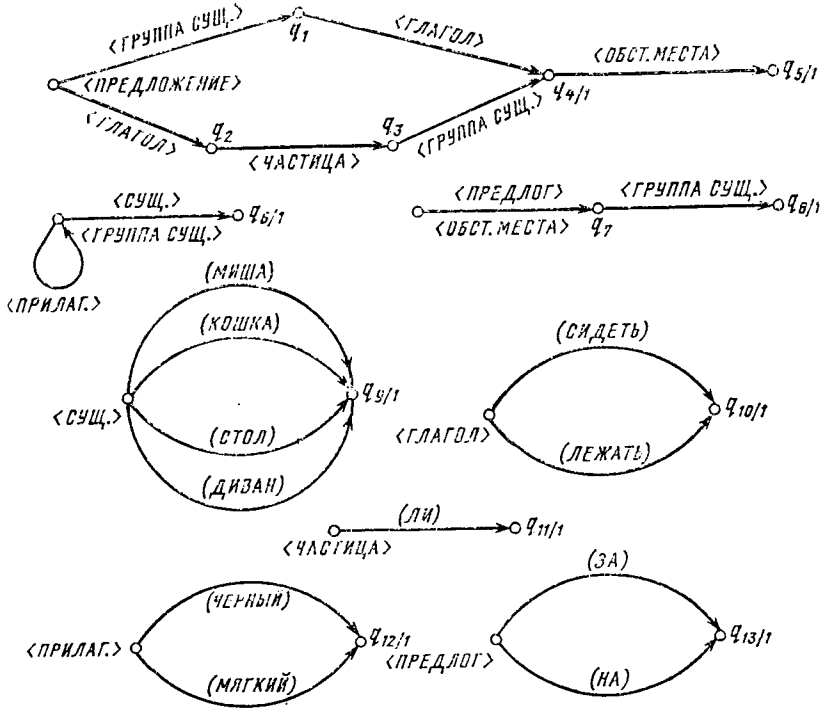


Рис. 16-24. Рекурсивная сеть переходов для фрагмента русского языка.

Следует отметить, что важным свойством любого естественного языка является то обстоятельство, что в нем всегда имеются предложения, для которых возможны различные пути анализа в сети переходов. Такие предложения называются неоднозначными. Поэтому любой алгоритм для сетевых грамматик, которые должны быть недетерминированными устройствами, должен уметь для любого заданного предложения проследить все возможные пути анализа. Дальнейшее расширение рассмотренной автоматной модели, позволяющее допускать языки более широкого класса, чем класс бесконтекстных языков, носит название расширенной сети переходов (РСП). Это расширение заключается в следующем. Каждой дуге сети переходов приписывается дополнительное условие, которое должно быть выполнено, чтобы переход по дуге был возможен, а также множество действий по построению синтаксической структуры входной цепочки, которые следует осуществить, если совершается переход по данной дуге.

Обобщение критерия прохождения дуг позволяет естественным образом выходить за рамки КС-языков. Действительно, контекст можно ввести в качестве значения переменных того предиката, истинность которого является одним из критериев прохождения дуги

расширенной сети переходов. Для хранения такого контекста необходима не магазинная память, а набор регистров, которые допускали бы многократное считывание информации, причем количество таких регистров, очевидно, должно зависеть от грамматики, которой соответствует данная РСП.

Достоинством РСП как устройств, допускающих контекстно-зависимые языки, является то, что контекст располагается не во входном тексте, а в регистрах и может быть эффективно проверен.

Таким образом вводятся следующие расширения в рекурсивные сети переходов:

разрешено использовать произвольные условия на дугах, истинность которых является критерием прохождения дуги;

введено необходимое количество регистров, работающих по принципу магазинной памяти, в которых хранятся фрагменты синтаксических подструктур и индикаторы, управляющие процессом анализа входной фразы;

введены в рассмотрение произвольные действия, которые могут изменять, объединять, удалять, проверять содержимое регистров.

Механизм интерпретации РСП называется сетевой грамматикой [61]. Более формально сетевая грамматика определяется как следующая тройка [62]:

$$F_S = (V,\ L,\ N),$$

где V — описание словарей, используемых при разборе входной цепочки;

L — описание нестандартных функций, необходимых для эффективного разбора;

N — описание РСП, представляющее собой описание множества так называемых кустов.

Кустом будем называть вершину РСП с множеством дуг, выходящих из этой вершины.

Под разбором входной цепочки (фразы языка) в данном случае понимается проверка ее допустимости данным устройством (РСП).

Описание сетевой грамматики удобно задать с помощью нормальной формы Бэкуса, расширенной за счет использования метасимволов { , } , специфицирующих возможность повторения заключенных в них конструкций любое (в том числе нулевое) число раз, а также метасимволов [,] , указывающих на необязательность конструкции в таких скобках.

Итак:

⟨сетевая грамматика⟩ : : = (⟨описания словарей⟩ [⟨описание нестандартных процедур⟩] ⟨расширенная сеть переходов⟩)

⟨описания словарей⟩ : : = (VOCAB {⟨описание словаря⟩})

⟨описание словаря⟩ : : = ⟨имя словаря⟩ (⟨функция ввода⟩) | ⟨имя словаря⟩ ({⟨словарная статья⟩})

Здесь намеренно опускается строгое определение словарной статьи, так как предполагается, что ее содержанием являются зна-

чения различных синтаксических и семантических признаков слов естественного языка.

Описание набора нестандартных функций также опускается, так как их конкретизация зависит от реализации.

Описание третьего компонента сетевой грамматики — расширенной сети переходов — это описание совокупности не связанных между собой графов. Вершины графов соответствуют нетерминальным символам некоторой грамматики, которые называют также категориями, а дуги помечены либо терминальными символами этой грамматики (конкретными словами входного языка, именами словарей), либо нетерминальными символами. Дуги могут и не иметь пометок.

В соответствии с указанными типами пометок вводятся понятия: CAT-дуги, TST-дуги и PUSH-дуги. Кроме того, для индикации заключительных состояний в сети и построения структур (форм) разобранных частей фразы вводится POP-дуга. Тогда ⟨дуга⟩ : : = = (CAT ⟨категория⟩ ⟨условие⟩ {⟨действие⟩} ⟨заключительное действие⟩) | (TST ⟨условие⟩ {⟨действие⟩} ⟨заключительное действие⟩) | (PUSH ⟨состояние⟩ ⟨условие⟩ {⟨действие⟩} ⟨заключительное действие⟩) | (POP ⟨форма⟩ ⟨ условие⟩)

Как следует из этого описания, для прохождения любой из дуг требуется выполнение необходимых и достаточных условий. Достаточные условия в явном виде присутствуют в синтаксических определениях и выражаются следующим образом:

⟨условие⟩ : : = ⟨И-условие⟩ | ⟨ИЛИ-условие | ⟨НЕ-условие⟩ | ⟨РАВНО-условие⟩

⟨И-условие⟩ : : = (AND⟨аргумент⟩ ⟨аргумент⟩ {⟨аргумент⟩})

⟨ИЛИ-условие⟩ : : = (OR ⟨аргумент⟩ ⟨аргумент⟩ {⟨аргумент⟩})

⟨НЕ-условие⟩ : : = (NOT ⟨аргумент⟩)

⟨РАВНО-условие⟩ : : = (EQUAL ⟨аргумент⟩ ⟨аргумент⟩)

В качестве аргументов используются T, NIL, ⟨имя⟩ (в этом случае берется значение заданного имени, а если его нет — NIL) или ⟨условие⟩.

Что же касается необходимых условий — они различны для разных типов дуг и обеспечиваются некоторыми встроенными механизмами. В частности, для дуг TST и POP необходимые условия выполняются всегда. Для дуги CAT необходимое условие принимает значение T в том случае, если текущее слово входной фразы, предварительно занесенное в некоторый регистр (будем называть его *), совпадает с одним из слов в словаре, имя которого задано в ⟨категории⟩, либо текстуально совпадает с пометкой дуги.

Обработка дуг типа PUSH базируется на использовании магазинной памяти и заключается в рекурсивном обращении к подграфу сети, начальная вершина которого помечена состоянием. Необходимое условие принимает значение T, если синтаксическая категория, соответствующая этому подграфу, найдена в анализируемой фразе. В случае невыполнения необходимого условия осуществляется возврат и выбор другой дуги. Вся необходимая информация для обеспечения возврата хранится в специальном системном регистре.

Таким образом, для проверки необходимых условий прохождения по дугам должны использоваться встроенные механизмы словарного поиска, перехода к подграфу и обеспечения возвратов.

При выполнении необходимых и достаточных условий разрешается проход по дуге, сопровождающийся выполнением действий. По определению

⟨действие⟩ : : = ⟨действие над содержимым⟩ | ⟨действие над указателем⟩

⟨действие над содержимым⟩ : : = (SETR ⟨имя регистра⟩ ⟨форма⟩) | (SENDR ⟨имя регистра⟩ ⟨форма⟩) | (LIFTR ⟨имя регистра⟩ ⟨форма⟩)

⟨действие над указателем⟩ : : = (UP ⟨имя регистра⟩)

Функция SETR специфицирует присваивание на текущем уровне; SENDR — замену информации в регистре, значение которой хранится в верхней ячейке системного регистра возвратов, а LIFTR — замену информации в позиции на единицу ниже верхней ячейки текущего регистра. Функция UP работает только на текущем уровне и выталкивает содержимое верхней ячейки соответствующего регистра.

Заключительные действия определяют переход к следующему кусту. Такой переход может сопровождаться считыванием следующего слова в регистр *. Таким образом,

⟨заключительное действие⟩ : : = ⟨переход со считыванием⟩ | ⟨переход без считывания⟩

⟨переход со считыванием⟩ : : = (TO⟨имя перехода⟩)

⟨переход без считывания⟩ : : = (JUMP⟨имя перехода⟩)

⟨имя перехода⟩ : : = ⟨состояние⟩ | ⟨имя регистра⟩

Осталось рассмотреть возможные формы, которые используются в определениях POP-дуги и действиях. В конечном счете формы нужны для построения некоторых структур и присваивания их в качестве значений определенным регистрам. Минимальный набор форм должен обеспечивать получение следующих значений:

текущего слова (*);

заданного свойства для текущего слова из регистра * (GETF);

произвольной структуры (QUOTE);

содержимого произвольного регистра (GETR);

списка, составленного из других форм (APPEND).

Пример 16-22. Рассмотрим описание фрагмента анализатора простых предложений русского языка, порождаемых в соответствии с грамматикой, представленной на рис. 16-25:

```
(ПРЕДЛ (PUSH ИМГР T (SETR R ИМГР*) (SETR R РДИМГР (GETR RРД))
       (SETR R ЧСИМГР (GETR R ЧС)) (SETR R СЕМИМГР (GETR
       RСЕМ)) (JUMP Q1)))
  (Q1 (PUSH ГЛГР T (SETR R ГЛГР*) (JUMP Q2)))
  (Q2 (POP (BUILDQ (ПРЕДЛ++) R ИМГР R ГЛГР) (EQUAL (GETR*)
       NIL))) (ИМГР (PUSH ГРПР T (SETR R ГРПР*) (JUMP Q3)))
```

(Q3 (CAT сущ (AND (EQUAL RP Д (GETF РОД)) (EQUAL R ЧС (GETF ЧИСЛО)) (EQUAL RП (GETF ПАДЕЖ)) (SETR RC*) (SETR RCEM (GETF ФИЗОБ)) TO Q4)))
(Q4 (POP (BUILDQ (ИМГР+(C (+))) R ГРПР RC) T)) (ГРПР (CAT ПРИЛ T (SETR R РД (GETF РОД)) (SETR R ЧС (GETF ЧИСЛО)) (SETR RП (GETF ПАДЕЖ)) (SETR RПР *) (TO Q5))
(TST (EQUAL R ПР NIL) (JUMP Q5)))
(Q5 (CAT прил (AND (EQUAL R РД (GETF РОД)) (EQUAL R ЧС GETF ЧИСЛО)) (EQUAL RП (GETF ПАДЕЖ))) APPEND R ПР*) (TO Q5))
(POP (BUILDQ (ГРПР (ПР (+))) R ПР) T)) (ГЛГР (CAT гл (AND (EQUAL R РДИМГР (GETF РОД))
(EQUAL R ЧС ИМГР (GETF ЧИСЛО))) (SETR R ГЛ*) (TO Q6))
(Q6 (... ...))

Пусть на вход анализатора поступает фраза:

БОЛЬШОЙ ЧЕРНЫЙ КОТ ВСКОЧИЛ НА СКОЛЬЗКУЮ ПОДНОЖКУ МОСКОВСКОГО ТРАМВАЯ.

Для разбора ее необходимо обратиться к встроенной функции (PARSE <фраза>). Основные действия функции PARSE сводятся к записи в регистр * первого слова фразы (БОЛЬШОЙ) и передаче управления в состояние ПРЕДЛ.

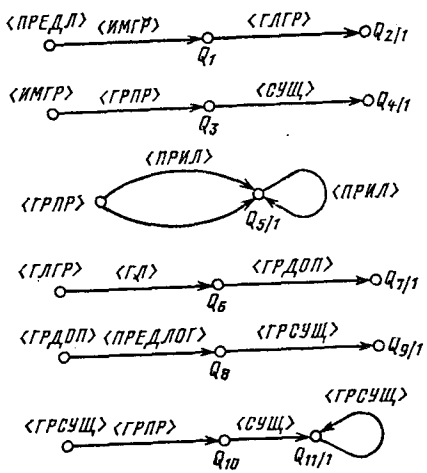


Рис. 16-25. Элемент расширенной сети переходов для фрагмента русского языка.

Описание куста, помеченного этим символом, содержит единственную дугу (PUSH ИМГР .....). Для прохода по этой дуге необходимо найти именную группу ИМГР. В свою очередь анализ именной группы предполагает последовательный поиск группы прилагательного (ГРПР) и существительного (СУЩ), которые должны быть согласованы по роду, числу, падежу и семантическим признакам, хранимым в словарях.

Для примера рассмотрим процесс анализа группы существительного состояния Q3. К этому моменту, как следует из фрагмента анализатора, состояния введенных регистров будут следующими:

(RГРПР) = (ГРПР(ПР(БОЛЬШОЙ ЧЕРНЫЙ)))
(RРД) = МУЖСКОЙ
(RЧС) = ЕДИНСТВЕННОЕ
(RП) = ИМЕНИТЕЛЬНЫЙ
(*) = КОТ

Единственная дуга, выходящая из состояния Q3 — это CAT-дуга, помеченная именем категории СУЩ. Поэтому анализатор обращается к словарю существительных для поиска слова (вернее, его основы) в регистре *. Предположим, что слово найдено в словаре. Тогда его характеристики переносятся со своими наименованиями в специальный регистр свойств, а в качестве результата этой процедуры выдается значение T (TRUE). Таким образом, необходимое условие перехода по CAT-дуге выполнено. В качестве достаточного условия в данном случае вычисляется значение предиката (AND....), где проверяется согласование свойств текущего слова (КОТ) со значениями регистров RРД, RЧС и RП. Значением этого предиката является T и, следовательно, проход по дуге возможен. Проход по дуге сопровождается выполнением присваивания значения текущего слова (КОТ) регистру RC и значения семантического

признака ФИЗОБ семантическому регистру RCEM. Далее в соответствии с заключительным действием (TO Q4) считывается очередное слово входной фразы (ВСКОЧИЛ) и осуществляется переход в состояние Q4.

Состояние Q4 заключительное и содержит единственную POP-дугу с тождественно истинным достаточным условием. В результате прохода по этой дуге в верхней ячейке регистра * формируется следующая структура:

(ИМГР(ГРПР(ПР(БОЛЬШОЙ ЧЕРНЫЙ))) (С(КОТ))).

На этом анализ именной группы заканчивается и анализатор возвращается к той точке, где разбор был приостановлен на дуге (PUSH ИМГР) проверкой необходимого условия. В этот момент содержимое регистров следующее:

(RГРПР) = $\emptyset$
(RРД) = МУЖСКОЙ
(RЧС) = ЕДИНСТВЕННОЕ
(RП) = ИМЕНИТЕЛЬНЫЙ
(RСЕМ) = ОДУШЕВЛЕННЫЙ
(*) = (ИМГР(ГРПР(ПР(БОЛЬШОЙ ЧЕРНЫЙ))) (С(КОТ))) (ВСКОЧИЛ)

Анализ оставшейся части предложения легко может быть продолжен.

16-8. ВЕРОЯТНОСТНЫЕ АВТОМАТНЫЕ МОДЕЛИ

Обобщением детерминированных автоматов и преобразователей, рассмотренных в § 16-6, являются вероятностные (стохастические) автоматы и преобразователи.

Отличие вероятностных автоматов и преобразователей от детерминированных автоматов и преобразователей (конечных, магазинных и т. д.) заключается в том, что на каждом шаге своей работы эти устройства могут переходить в любое из возможных состояний с определенной вероятностью. Стохастические автоматы допускают языки, которые называют стохастическими. Каждая цепочка таких языков является «взвешенной», т. е. имеет определенную вероятность. По аналогии с рассмотренными типами языков можно рассматривать стохастические автоматные, стохастические бесконтекстные, стохастические контекстно-зависимые языки, а также устройства, допускающие эти классы языков: стохастические конечные автоматы, стохастические магазинные автоматы и т. д.

Формально стохастический или вероятностный конечный автомат (или просто стохастический автомат) определяется как следующая пятерка:

$$A_S = (V_1,\ Q,\ M,\ \pi_0,\ F),$$

где $V_1 = \{a_1,\ \ldots,\ a_m\}$ — конечное множество входных символов (входной алфавит);

$Q = \{q_1,\ \ldots,\ q_n\}$ — конечное множество состояний автомата;

π_0 — n-мерный вектор, обозначающий начальное распределение состояний:

$$\pi_0 = \| p_{0_1},\ \ldots,\ p_{0_n} \|,\quad 0 \leqslant p_{0_i} \leqslant 1,$$

$$\sum_{i=1}^{n} p_{0_i} = 1;$$

$F \subseteq Q$ — множество заключительных состояний;

M — отображение множества V_1 в множество так называемых стохастических матриц перехода размерностью $n \times n$ (n — число

переходов):

$$M(a) = \|p_{ij}(a)\|, \quad 0 \leqslant p_{ij} \leqslant 1,$$

$$\sum_{j=1}^{n} p_{ij} = 1 \text{ для всех } i = 1, \ldots, n.$$

Число $p_{ij}(a)$ интерпретируется как вероятность перехода автомата из состояния q_i в состояние q_j при чтении входного символа $a \in V_1$.

Можно обобщить отображение M, расширив его область определения с V_1 до V_1^*:

$M(\varepsilon) = I$ (единичная матрица $n \times n$);

$$M(a_1 a_2 \ldots a_k) = M(a_1) M(a_2) \ldots M(a_k)$$

(произведение матриц $M(a_1)$, ..., $M(a_k)$),

$$a_i \in V_1, \quad i = 1, \ldots, k.$$

Взвешенной цепочкой, допускаемой автоматом A_S, называется такая пара $(\alpha, p(\alpha))$, что $\alpha \in V_1^*$,

$$p(\alpha) = \pi_0 M(\alpha) \pi_F > 0,$$

где π_F — n-мерный вектор-столбец, для которого i-й компонент равен 1, если $q_i \in F$, и 0 — в противном случае.

Язык, допускаемый стохастическим автоматом A_S — это множество взвешенных цепочек, допускаемых этим автоматом: $L_1(A_S) = \{(\alpha, p(\alpha)) \mid \alpha$ — взвешенная цепочка, допускаемая $A_S\}$.

Стохастические матрицы можно задавать в виде графов, вершины которых соответствуют состояниям автомата, а дуги, взвешенные вероятностями p_{ij} — переходам из состояния q_i в q_j.

Пример 16-23. Рассмотрим стохастический автомат $A_S = (V_1, Q, M, \pi_0, F)$, где $V_1 = \{a, b\}$; $Q = \{q_1, \ldots, q_6\}$; $\pi_0 = \|1\ 0\ 0\ 0\ 0\ 0\|$; $F = \{q_6\}$; $\pi_F = \|0\ 0\ 0\ 0\ 0\ 1\|^T$, а отображение M задается двумя графами, изображенными на рис. 16-26, *а* и *б* и соответствующими стохастическим матрицам $M(a)$ и $M(b)$.

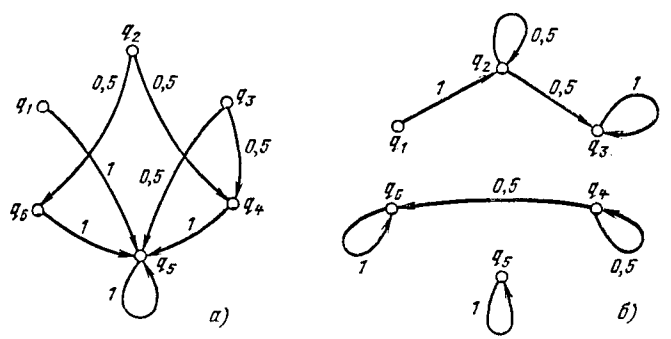


Рис. 16-26. Графы переходов для стохастического конечного автомата.

Язык, допускаемый автоматом A_S — это множество пар:

$$L_1(A_S) = \{(b^m a b^n), \quad p(b^m a b^n)\},$$

где $p(b^m a b^n) = 0{,}5 + (0{,}5)^m - (0{,}5)^{n+1}$; $m, n > 0$.

Цепочка bab допускается с вероятностью $p(bab) = 0{,}75$, цепочка $bbabb$ — с вероятностью $p(bbabb) = 0{,}625$, а цепочка ab не допускается данным автоматом.

Существует другой вариант определения языка, допускаемого стохастическим автоматом A_S.

Язык, допускаемый автоматом A_S с порогом λ ($0 \leqslant \lambda < 1$), есть следующее множество цепочек:

$$L_2(A_S, \lambda) = \{\alpha \mid \alpha \in V_1^* \wedge \pi_0 M(\alpha) \pi_F > \lambda\},$$

где π_F определяется так же, как и для $L_1(A_S)$.

Стохастическим конечным преобразователем называется следующая семерка объектов:

$$A_S = (V_1, V_2, Q, M, \pi_0, \psi, F),$$

где помимо входного словаря V_1, множеств Q и F, отображения M и вектора π_0, определяемых так же, как и для стохастического конечного автомата, вводится словарь выходных символов V_2, а также функция выходов ψ, которая в случае детерминированного преобразователя отображает множество $Q \times V_1$ в множество $Q \times V_2^*$, а в случае недетерминированного преобразователя — в множество подмножеств $Q \times V_2^*$.

Очевидно, что детерминированные конечные автоматы и преобразователи можно рассматривать как частный случай стохастических.

Аналогичным образом можно ввести понятия стохастического магазинного автомата, стохастического линейно-ограниченного автомата и т. д.

Рассмотрим теперь стохастические языки с точки зрения теории формальных грамматик. Ранее был рассмотрен один из способов такого расширения понятия формального языка, заключающийся в рандомизации выбора состояний в автоматах. Другой способ заключается в рандомизации выбора продукций в грамматиках.

Стохастической грамматикой называется следующая совокупность объектов:

$$G_S = (V_T, V_N, P_S, S_0, \rho_0),$$

где V_T — терминальный словарь;

V_N — нетерминальный словарь;

$\rho_0 = [p_{0_1}, \ldots, p_{0_n}]$ — вероятностное распределение начальных символов из упорядоченного множества $S_0 = \{S_1, \ldots, S_n\} \subseteq V_N$,

при этом $0 \leqslant p_{0_i} \leqslant 1$, $\sum_{i=1}^{n} p_{0_i} = 1$;

P_S — конечное множество вероятностных продукций, каждая из которых имеет вид:

$$\alpha_i \xrightarrow{p_{ij}} \beta_{ij},\ i=1,\ \ldots,\ k;\ j=1,\ \ldots,\ n_i,$$

где $\alpha_i \in V^*V_NV^*$, $\beta_{ij} \in V^*$, $V = V_T \cup V_N$, а p_{ij} — вероятность данной продукции, т. е. число, удовлетворяющее следующим условиям:

$$0 \leqslant p_{ij} \leqslant 1, \quad \sum_{j=1}^{n_i} p_{ij} = 1.$$

Если продукция $\alpha_i \xrightarrow{p_{ij}} \beta_{ij}$ принадлежит множеству P_S, то говорят, что цепочка $\gamma_1\ \alpha_i\ \gamma_2$ непосредственно порождает цепочку $\gamma_1\ \beta_{ij}\ \gamma_2$, и обозначают это

$$\gamma_1\alpha_i\gamma_2 \overset{p_{ij}}{\Longrightarrow} \gamma_1\beta_{ij}\gamma_2.$$

Цепочка γ_0 порождает цепочку γ_n, если существует такая последовательность цепочек $\gamma_0, \gamma_1, \ldots, \gamma_n$, что $\gamma_{i-1} \overset{p_i}{\Longrightarrow} \gamma_i$, $i=1, \ldots, n$.

Вероятностью такого порождения называют число $p = \prod_{i=1}^{n} p_i$.

Если γ_0 порождает γ_n с вероятностью p, то пишут

$$\gamma_0 \overset{p}{\Longrightarrow} *\,\gamma_n.$$

Стохастическим языком $L(G_S)$, порождаемым грамматикой G_S, называют множество пар:

$$L(G_S) = \left\{(\alpha,\ p(\alpha)) \mid \alpha \in V_T^*;\ S_i \overset{p_{ij}}{\Longrightarrow} *\,\alpha;\ i=1,\ \ldots,\ n;\ j=1,\ \ldots,\ k;\right.$$

$$\left. p(\alpha) = \sum_{i=1}^{n} \sum_{j=1}^{k} p_{0_i} p_{ij} \right\},$$

где k — число различных выводов цепочки α из символа S_i;
n — число начальных символов;
p_{0_i} — вероятность появления i-го начального символа;
p_{ij} — вероятность j-го вывода цепочки α из символа S_i.

Стохастический язык $L(G_S)$ определяется парой (L, p), где L — формальный язык, порождаемый грамматикой G'_S, которая получается исключением вероятностей из стохастических продукций, а p — распределение вероятностей на множестве L. Грамматика G'_S называется характеристической грамматикой для стохастической грамматики G_S.

По типу характеристических грамматик стохастические грамматики могут принадлежать к одному из четырех типов: 0, 1, 2 или 3.

Рассмотрим лишь один из этих типов — автоматные стохастические грамматики. Для любой такой грамматики G_S можно построить стохастический конечный автомат A_S, который допускает язык $L(A_S) = L(G_S)$.

Правила построения автомата $A_S = (V_1, Q_1, M, \pi_0, F)$ по грамматике $G_S = (V_T, V_N, P_S, S_0, \rho_0)$ таковы:

1) $V_1 = V_T$;

2) $Q = V_N \cup \{T, R\}$, если $V_N = \{A_1, \ldots, A_k\}$; обозначим $T = A_{k+1}$, $R = A_{k+2}$;

3) π_0 — вектор, у которого все элементы, соответствующие символам $S_1, \ldots, S_n$, соответствуют ρ_0, остальные равны нулю;

4) $F = \{T\}$;

5) отображение M формируется с помощью стохастических продукций: если в множестве P_S есть продукция $A_i \to a_l A_j$ с вероятностью p^l_{ij}, то элемент $m_{ij}(a_l)$ матрицы $M(a_l)$ будет равен p^l_{ij}, если в множестве P_S есть продукция $A_i \to a_l$ с вероятностью $p^l_{i_0}$, то элемент $m_{i,\,k+1}(a_l)$ матрицы $M(a_l)$ будет равен $p^l_{i_0}$. Все элементы $m_{i,\,k+2}(a_l)$ матрицы определяются так, чтобы

$$\sum_{j=1}^{k+2} m_{ij}(a_l) = 1.$$

Пример 16-24. Рассмотрим стохастическую грамматику $G_S = (V_T, V_N, P_S, S_0, \rho_0)$, где $V_T = \{a, b\}$; $V_N = \{S, A\}$, $\rho_0 = [1]$; $S_0 = \{S\}$, а стохастические продукции P_S имеют вид:

$$S \xrightarrow{0,1} aS;$$
$$S \xrightarrow{0,9} bA;$$
$$A \xrightarrow{0,2} aA;$$
$$A \xrightarrow{0,8} b.$$

Эта грамматика порождает язык

$$L(G_S) = (\{a^n b a^m b\},\quad 0{,}1^n \cdot 0{,}9 \cdot 0{,}2^m \cdot 0{,}8).$$

Вероятность вывода цепочки $\alpha = bb$ больше, чем вероятность вывода любой другой цепочки данного языка:

$$p(bb) = 0{,}9 \cdot 0{,}8 = 0{,}72.$$

Заметим, что

$$\sum_{\alpha \in L(G'_S)} p(\alpha) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} 0{,}1^n \cdot 0{,}9 \cdot 0{,}2^m \cdot 0{,}8 = 1.$$

Пример 16-25. Построим стохастический конечный автомат $A_S = (V_1, Q, M, \pi_0, F)$ для грамматики G_S, приведенной в предыдущем примере:

$$V_1 = V_T = \{a, b\};\quad Q = \{S, A, T, R\};$$

$\pi_0 = \|1\ 0\ 0\ 0\|$; $F = \{T\}$; отображение M зададим с помощью матриц

$$M(a) = \begin{array}{c} \\ S \\ A \\ T \\ R \end{array} \begin{array}{c} \begin{array}{cccc} S & A & T & R \end{array} \\ \left\| \begin{array}{cccc} 0{,}1 & 0 & 0 & 0{,}9 \\ 0{,}2 & 0 & 0 & 0{,}8 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right\| \end{array},$$

$$M(b) = \begin{array}{c} \\ S \\ A \\ T \\ R \end{array} \begin{array}{c} \begin{array}{cccc} S & A & T & R \end{array} \\ \left\| \begin{array}{cccc} 0 & 0{,}9 & 0 & 0{,}1 \\ 0 & 0 & 0{,}8 & 0{,}2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right\| \end{array}.$$

Вероятность порождения цепочки $\alpha = a^n b a^m b$

$$p(\alpha) = \|1\ 0\ 0\ 0\|\, M(a^n b a^m b)\, \|0\ 0\ 1\ 0\|^T.$$

Например, для цепочки $\alpha = bb$ вероятность порождения

$$p(bb) = \|1\ 0\ 0\ 0\| \left\| \begin{array}{cccc} 0 & 0{,}9 & 0 & 0{,}1 \\ 0 & 0 & 0{,}8 & 0{,}2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right\|^2 \left\| \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array} \right\| = 0{,}72.$$

Вероятностные (стохастические) языки, грамматики, автоматы и преобразователи играют важную роль при построении кибернетических моделей различного назначения.

Так, например, понятия стохастических автоматов, языков и грамматик находят применение при построении систем структурного (синтаксического) распознавания образов [34]. При этом языки, используемые для описания зашумленных или искаженных образов, часто оказываются неопределенными в том смысле, что одна и та же цепочка может быть порождена более чем одной из грамматик, используемых для описания отдельных классов объектов. С точки зрения теории распознавания образов — это случай частичного пересечения классов; объекты, принадлежащие различным классам, могут иметь одинаковые описания при разных вероятностях появления. Стохастические формальные языки и автоматы используются в этом случае для более адекватного моделирования таких ситуаций.

Другой важной областью применения вероятностных автоматов является моделирование вычислительных процессов и, в частности, матричных преобразований (умножения, возведения в степень, обращения матриц) [58]. Оно было продиктовано необходимостью борьбы с накоплением ошибок округления, которые очень часто приводят подобные преобразования к неверным результатам. С этой целью каждой из перечисленных операций над матрицами ставится в соответствие вероятностный преобразователь, на вход которого многократно подается одна и та же цепочка символов. Статистика выходных данных при этом служит приближенным решением задачи. Точность решения задачи зависит от числа испытаний данного вероятностного преобразователя (т. е. от числа полных просмотров входной цепочки).

Помимо перечисленных можно указать много других областей применения вероятностных автоматных моделей.

ГЛАВА СЕМНАДЦАТАЯ

МОДЕЛИ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

В большинстве современных систем управления очень остро встал вопрос о придании им свойства принятия решений, свойства интеллектуальности, т. е. реализации в них каких-то моделей принятия решений естественным интеллектом (человеком). Методы решения этой проблемы сформировались в научно-техническое направление, называемое искусственным интеллектом.

Искусственный интеллект (ИИ) представляет собой кибернетическую систему (или модель) по переработке информации. Для описания этих систем широко используется математический аппарат информационной алгебры, рассмотренный в гл. 15, а также лингвистические модели, рассмотренные в гл. 16.

Необходимость использования лингвистических методов определяется прежде всего человеко-машинным характером систем искусственного интеллекта (СИИ), в соответствии с которым осуществляется постоянное вмешательство человека в процесс функционирования СИИ при формировании решения. Кроме того, в естественном языке и его математических моделях имеется много примеров и средств для моделирования творческих процессов принятия решений человеком [80].

17-1. ОБЩИЕ СВЕДЕНИЯ О МОДЕЛЯХ СИИ

А) ПОНЯТИЯ ИИ, СИИ, ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ И МОДЕЛЕЙ СИИ

Любой искусственный интеллект представляет собой некоторую модель процесса принятия решений естественным интеллектом. В данном параграфе рассматриваются только кибернетические модели ИИ, т. е. такие, в которых осуществляется переработка информации с целью принятия решений.

Искусственный интеллект может претендовать на такое наименование при условии, что качество формируемых системой решений будет не хуже, а иногда и лучше по сравнению со средним естественным интеллектом. Несмотря на то, что существуют автоматизированные системы оценки качества принимаемых решений, наиболее распространенными являются экспертные системы оценок.

Очень важно отметить, что под СИИ подразумевается система, заменяющая естественный интеллект или помогающая ему в узкопрофессиональных условиях, а не создание общей модели естественного интеллекта, т. е. искусственный интеллект ориентируется на определенную предметную область (ПО), например: механическая сборка, аэропорт Внуково и т. д.

Системный принцип в кибернетике требует рассмотрения ИИ как некоторой кибернетической системы, называемой системой ис-