



**Современный
Гуманитарный
Университет**

Дистанционное образование

Рабочий учебник

Фамилия, имя, отчество _____

Факультет _____

Номер контракта _____

ЛИНГВИСТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

ЮНИТА 2

**ОСНОВЫ ТЕОРИИ ФОРМАЛЬНЫХ ЯЗЫКОВ
И ЯЗЫКИ ОБРАБОТКИ ДАННЫХ
ИНФОРМАЦИОННЫХ СИСТЕМ**

МОСКВА 2000

Разработано В.И. Киселевым, В.Н. Кузубовым

Рекомендовано Министерством
общего и профессионального
образования Российской Федерации
в качестве учебного пособия для
студентов высших учебных заведений

КУРС: ЛИНГВИСТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Юнита 1. Язык как знаковая система.

Юнита 2. Основы теории формальных языков и языки обработки
данных информационных систем.

ЮНИТА 2

Излагаются основы формальных грамматик и языков, их значение и место в теории языков программирования и информационных языков автоматизированных информационных систем. Подробно рассмотрены вопросы классификации формальных грамматик и использования автоматных грамматик в машинном лингвистическом анализе языков. Показана роль “машины Тьюринга” в создании основ теории современных ЭВМ. На основе выше перечисленных теоретических материалов излагаются методы и средства грамматического разбора и их использование при разработке трансляторов с языков программирования. Детально рассмотрены назначение, состав и функции современного компилятора (на примере языка C++).

Вторая глава посвящена вопросам создания и использования информационных языков автоматизированных информационных систем – локальных и сетевых. Основное внимание уделено проблемам информационного поиска и манипулирования данными, а также вопросам, которые возникают при общении пользователя с ЭВМ, в т.ч. в глобальных сетях.

Кроме того, уделено значительное внимание основам языка SQL и технологиям гипертекста.

Для студентов Современного Гуманитарного Университета

Юнита соответствует профессиональной образовательной программе № 1

(С) СОВРЕМЕННЫЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ, 2000

ОГЛАВЛЕНИЕ

ДИДАКТИЧЕСКИЙ ПЛАН	5
ЛИТЕРАТУРА	6
ТЕМАТИЧЕСКИЙ ОБЗОР	7
1. Формальные языки и программирование	7
1.1. Формальные грамматики и языки	7
1.1.1. Основные понятия теории формальных грамматик ..	7
1.1.2. Соотношения Туэ и ассоциативные исчисления ..	9
1.1.3. Классификация грамматик	11
1.1.4. Грамматики непосредственных составляющих и КС-грамматики	15
1.2. Регулярные языки и автоматные грамматики	19
1.2.1. Основные понятия автоматных грамматик	19
1.2.2. Промежуточные классы грамматик	23
1.2.3. Конечные автоматы	28
1.2.4. Автоматы с магазинной памятью	37
1.2.5. Машина Тьюринга	41
1.3. Машинный лингвистический анализ языков	46
1.3.1. Теория формальных грамматик и машинный анализ языков	46
1.3.2. Грамматики простого предшествования	49
1.4. Трансляторы	53
1.4.1. Основные понятия	53
1.4.2. Структура языка программирования с точки зрения транслятора	54
1.4.3. Состав и структура современного транслятора ..	61
2. Проблемы создания языковых средств информационных технологий	66
2.1. Работа со структурированными данными в автомати- зированных информационных системах	66
2.1.1. Информационная система – модель предметной области	66
2.1.2. Описание предметной области	67
2.1.3. Информационные база и процессор	69
2.1.4. Систематизация основных понятий статического описания предметной области	71
2.1.5. Описание динамики в предметной области АИС ..	74
2.2. Входные языки автоматизированных информационных систем	77
2.2.1. Структура входных языков обработки текстов запросов	77

2.2.2. Языковые средства документальных информационных систем	80
2.2.3. Проблемы информационного поиска	82
2.3. Сетевые языковые средства АИС	85
2.3.1. Основные понятия	85
2.3.2. Основы языка SQL	87
2.3.3. Дополнительные возможности SQL	88
2.4. Технология гипертекста и интеллектуальные сетевые интерфейсы	92
2.5. Поиск и обработка данных в глобальных сетях	97
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	101
ГЛОССАРИЙ*	

* Глоссарий расположен в середине учебного пособия и предназначен для самостоятельного заучивания новых понятий.

ДИДАКТИЧЕСКИЙ ПЛАН

Формальные языки и программирование. Формальные грамматики и языки. Основные понятия теории формальных грамматик. Соотношения Туэ и ассоциативные исчисления. Классификация грамматик. Грамматики непосредственных составляющих и КС-грамматики. Регулярные языки и автоматные грамматики. Основные понятия автоматных грамматик. Промежуточные классы грамматик. Конечные автоматы. Автоматы с магазинной памятью. Машина Тьюринга. Машинный лингвистический анализ языков. Теория формальных грамматик и машинный анализ языков. Грамматики простого предшествования. Методы нисходящего грамматического разбора. Трансляторы. Структура языка программирования с точки зрения транслятора. Состав и структура современного транслятора.

Проблемы создания языковых средств информационных технологий. Работа со структурированными данными в автоматизированных информационных системах. Информационная система – модель предметной области. Описание предметной области. Информационная база и процессор. Систематизация основных понятий статического описания предметной области. Описание динамики в предметной области АИС. Входные языки автоматизированных информационных систем. Структура входных языков обработки текстов запросов. Языковые средства документальных информационных систем. Проблемы информационного поиска. Сетевые языковые средства АИС. Основы языка SQL. Технология гипертекста. Поиск и обработка данных в глобальных сетях.

ЛИТЕРАТУРА

Базовая

1. Рейуорд-Смит В. Теория формальных языков: Вводный курс. М., 1988.

Дополнительная

2. Фостер Дж. Автоматический синтаксический анализ. М., 1975.
3. Попов Э.В. Общение с ЭВМ на естественном языке. М., 1982.
4. Ершов А.П. Введение в теоретическое программирование: Беседы о методе. М., 1977.

Примечание. Тематический обзор составлен на основе всех указанных источников.

ТЕМАТИЧЕСКИЙ ОБЗОР*

1. ФОРМАЛЬНЫЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ

1.1. Формальные грамматики и языки

1.1.1. Основные понятия теории формальных грамматик

В первой юните были рассмотрены вопросы анализа и синтеза процессов в естественном языке как в некоторой информационной системе. В соответствующих исследованиях было выработано большое количество рациональных правил, процедур, структур обработки лингвистической информации, которые оказались плодотворными в области информационных технологий. При этом, как показали исследования в области моделирования естественного интеллекта или систем принятия решений, естественный язык является удобной средой для моделирования процессов мышления.

В широко используемых классических автоматных моделях была развита 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 позиции заложенных в них грамматик. Поэтому можно рассматривать автоматные и лингвистические модели совместно и называть их автоматно-лингвистическими моделями.

Автоматные и лингвистические модели строятся на базе теории формальных грамматик – одного из основных подходов к описанию бесконечного формального языка конечными средствами.

Формальный язык – это совокупность неделимых исходных знаков и правил построения из них слов и словосочетаний без всякой связи с их возможной семантикой, а **теория формальных языков** – наука о формальных языках, в частности о структуре и способах представления бесконечных классов формальных языков.

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,

* Жирным шрифтом выделены новые понятия, которые необходимо усвоить. Знание этих понятий будет проверяться при тестировании.

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

$$g = ffg,$$

$$B = abba.$$

Пустая цепочка ε – цепочка, не содержащая 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 – цепочка $Z = ffgghh$. Обозначим операцию конкатенации символом o . Свойства этой операции можно записать следующим образом:

1) свойство замкнутости:

$$o: A^* \times A^* \rightarrow A^*;$$

2) свойство ассоциативности:

$$(\forall X \in A^*, Y \in A^*, Z \in A^*)$$

$$[(XoY)oZ = Xo(YoZ)],$$

где через A^* обозначено множество всех возможных цепочек (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^* .

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

Пример. 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.

1.1.2. Соотношения Туэ и ассоциативные исчисления

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

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

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

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 $n \geq 1$.

Откуда множество всех возможных цепочек алфавита:

$$A^* = \bigcup_{n=0}^{\infty} A^n \quad (\cup - \text{объединение множеств}).$$

Такое множество называется *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 $X \circ Y$ для простоты будем писать XY .

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

Соотношениями Туэ 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^* ($i = 1, 2, \dots, n$) 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:

$$X_0, X_1, \dots, X_n,$$

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

Пример. 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.

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 1 , если она соотносима с цепочкой ϵ (пустой цепочкой) в смысле соотношений Туэ:

$$(\epsilon, 1_1 p_1), (\epsilon, 1_2 p_2), \dots, (\epsilon, 1_n p_n),$$

где 1_i обозначает левую скобку i -го сорта, а p_i — соответствующую ей правую скобку. Легко видеть, что, например, цепочка:

$$1_1 1_2 p_2 p_1 1_3 p_3$$

принадлежит к ограниченному языку 1 , так как приводится к пустой цепочке последовательным применением данных соотношений:

$$1_1 1_2 p_2 p_1 1_3 p_3 \rightarrow 1_1 1_2 p_2 p_1 \rightarrow 1_1 p_1 \rightarrow \epsilon.$$

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:

$$(P_1 \rightarrow Q_2), (P_2 \rightarrow Q_1), \dots, (P_n \rightarrow Q_n).$$

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

$$X \Rightarrow Y,$$

если существуют такие цепочки U и V , что:

$$X = UP_iV,$$

$$Y = UQ_iV,$$

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

Говорят также, что X порождает Y .

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

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

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

$$X_0 \Rightarrow^* X_n.$$

Граматики Хомского, о которых идет речь в следующем параграфе, соответствуют формальным комбинаторным системам, в основу которых положены продукции.

1.1.3. Классификация грамматик

Теория формальных языков (формальных грамматик) занимается описанием, распознаванием и переработкой языков. Описание любого языка должно быть конечным, хотя сам язык может содержать бесконечное множество цепочек. **Формальная грамматика** – один из основных подходов к описанию бесконечного формального языка конечными средствами. 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 L , который задается определенным типом описания. Теория формальных языков позволяет ответить на ряд вопросов, возникающих во многих прикладных задачах, в которых используются автоматически-лингвистические модели. Например, могут ли языки из класса L распознаваться быстро и просто; принадлежит ли данный язык классу L и т.д. Важной проблемой является построение

алгоритмов, если они существуют, которые давали бы ответы на определенные вопросы о языках из класса L , например: “Принадлежит или нет к языку L цепочка X ?”

Существуют два основных способа описания отдельных классов языков. Первый из них *основан на ограничениях, налагающихся на систему productions, 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*.

Определим четыре типа грамматик.

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 ($V_N \cap V_T = \emptyset$). 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 следующим образом:

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

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

$$\alpha \rightarrow \beta.$$

β является элементом усеченной итерации словаря, поэтому среди productions 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 P , а $S \in V_N$ – начальный символ.

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

$$|\alpha| \leq |\beta|$$

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* (тип 3) называют такие, для которых $\alpha \in V_N$, а $\beta = V_T V_N^*$, 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 продукциями P данной грамматики. Другими словами:

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

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 иерархия грамматик, которой соответствует иерархия формальных языков*, каждый из них может быть порожден некоторой формальной грамматикой. Тип языка может быть определен типом той грамматики, с помощью которой он может быть порожден.

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. 1 приведена иерархия языков и соответствующие ей иерархии грамматик и автоматов как распознающих устройств.

Таким образом, грамматики типа 0 представляют собой *порождающие устройства* очень общего характера. А те формальные языки, с которыми имеют дело автомато-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. К ним относятся многие языки программирования, синтаксис которых описывается с помощью так называемой нормальной формы Бэкуса. 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:

$$A \rightarrow BC,$$

$$A \rightarrow aD,$$

где $A, B, C, D \in V_N, a \in V_T$.

Наконец, языки типа 3, введенные и изученные в связи с исследованием модели нейрона (нервной клетки), которые называют также 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 к языкам типа 3 относятся цепи Маркова, играющие важную роль в теории вероятностей и массового обслуживания.

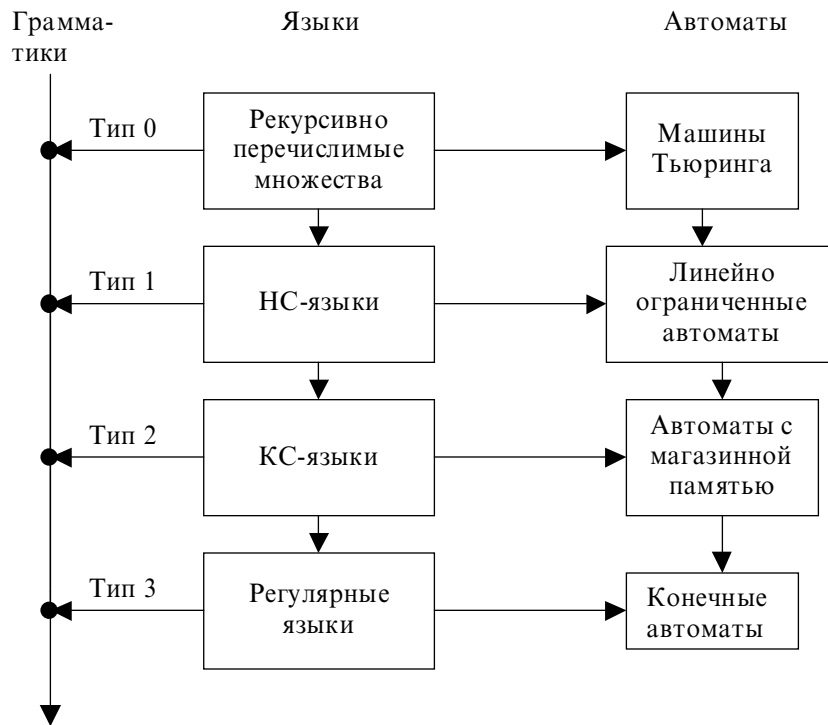


Рис.1. Иерархия языков, грамматик и автоматов

1.1.4. Грамматики непосредственных составляющих и КС-грамматики

Грамматики типа 0 и порождаемые ими языки 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. Почему языки типа 1 называются контекстно-зависимыми? Можно показать, 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:

$$\alpha A \beta \rightarrow \alpha \gamma \beta,$$

где $\alpha, \beta \in (V_T \cup V_N)^*$, $A \in V_N$; $\gamma \in (V_T \cup V_N)^+$, т.е. 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 (конструкции языка), а вывод цепочек языка можно рассматривать как информацию о синтаксической структуре этих цепочек.

Пример. Пусть необходимо построить язык для описания равнобедренных треугольников, каждая сторона которых состоит из конечного числа ориентированных отрезков определенной длины.

На рис. 2а приведены примеры таких треугольников. В зависимости от направления обозначим эти отрезки символами 1, r и x так, как показано на рис. 2б. Двигаясь по часовой стрелке от левого нижнего угла каждого треугольника, получим следующие описания этих треугольников:

$$111 rrr xxx,$$

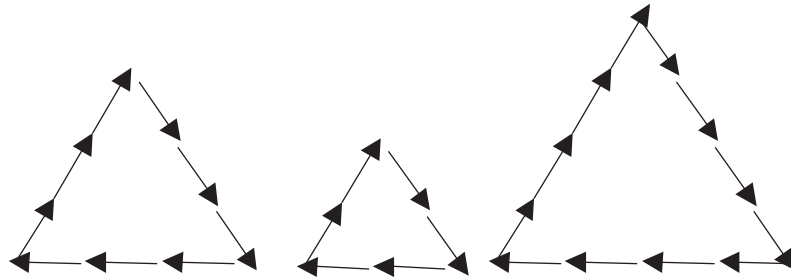
$$11 rr xx,$$

$$1111 rrrr xxxx.$$

Можно считать, что это цепочки некоторого языка описания:

$$L = \{1^n r^n x^n, n \geq 1\}$$

равнобедренных треугольников указанного типа.



a)



б)

Рис. 2. Графическая интерпретация языка $\{1^n, r^n, x^n\}$ (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 $V_T = \{1, r, x\}$, $V_N = \{S, R, X\}$, а множество продукций P состоит из элементов:

$$S \rightarrow 1SRX,$$

$$S \rightarrow 1RX,$$

$$XR \rightarrow RX,$$

$$1R \rightarrow 1r,$$

$$rR \rightarrow rr,$$

$$rX \rightarrow rx,$$

$$xX \rightarrow xx.$$

Чтобы получить цепочку $1^n r^n x^n$, надо сделать следующее количество шагов:

$$N = n(n+5)/2,$$

причем первую продукцию необходимо применить $n - 1$ раз, затем один раз – вторую продукцию, $1/2n (n - 1)$ 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 $\{1^n r^n x^n\}$. Более того, можно доказать, что данному языку не может соответствовать никакая бесконтекстная грамматика.

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

$$A \rightarrow \beta,$$

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

Пример. 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:

$$S \rightarrow W + S,$$

$$S \rightarrow W - S,$$

$$S \rightarrow W,$$

$$W \rightarrow W^* W,$$

$$W \rightarrow W/W,$$

$$W \rightarrow (S),$$

$$W \rightarrow a,$$

$$W \rightarrow b.$$

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

$$\begin{aligned} S &\Rightarrow W - S \Rightarrow (S) - S \Rightarrow (W + S) - S \Rightarrow (a + S) - S \Rightarrow (a + W) - S \Rightarrow \\ &\Rightarrow (a + W^* W) - S \Rightarrow (a + b^* W) - S \Rightarrow (a + b^* b) - S \Rightarrow (a + b^* b) - W \Rightarrow \\ &\Rightarrow (a + b^* b) - W/W \Rightarrow (a + b^* b) - a/W \Rightarrow (a + b^* b) - a/b. \end{aligned}$$

Способ описания вывода цепочки из начального символа, приведенный выше, не совсем удобен. Более наглядным является использование так называемых деревьев вывода. При этом корнем дерева будет являться вершина, имеющая метку начального символа S .

На рис. 3 приведено дерево вывода для цепочки, рассмотренной в данном примере.

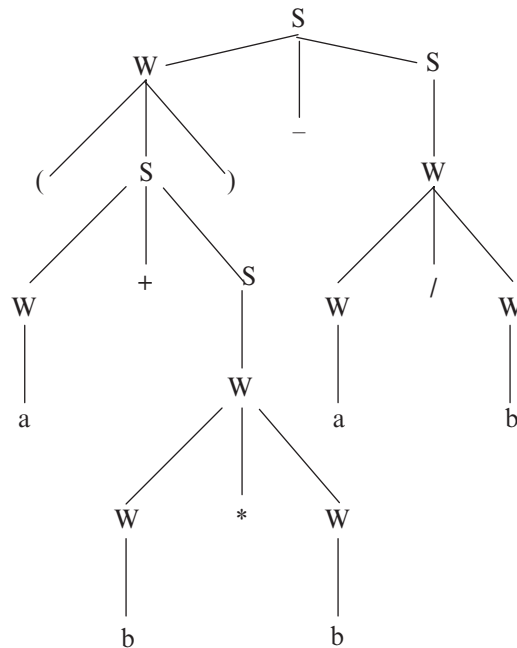


Рис. 3. Дерево вывода цепочки языка алгебраических выражений

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

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

$$A \rightarrow BC,$$

$$A \rightarrow a,$$

где $A, B, C \in V_N$; $a \in V_T$.

Тот факт, что для любой КС-грамматики можно построить нормальную форму, строго доказан.

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 $\alpha, \beta \in V_T \cup V_N)^+$.

Такие символы называют *самовставляющимися*, а грамматики – *грамматиками с самовставлением*. Таким образом, отсутствие свойства *самовставления* может служить свидетельством того, что данная грамматика является регулярной (автоматной). Грамматика из приведенного выше примера не является автоматной в силу наличия продукций:

$$S \rightarrow W,$$

$$W \rightarrow (S),$$

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

1.2. Регулярные языки и автоматные грамматики

1.2.1. Основные понятия автоматных грамматик

Материал, излагаемый в данном разделе, имеет большое значение при создании методов и средств анализа компиляторами входных текстов на соответствующих языках программирования. Следующие понятия данной предметной области являются основными:

автомат – абстрактная кибернетическая машина, обрабатывающая входную последовательность и определяющая ее принадлежность некоторому формальному языку или выдающая некоторую выходную последовательность;

конечный автомат – простая разновидность автомата, которая однократно считывает входную строку слева направо, при этом в любой момент времени конечный автомат находится в некотором внутреннем состоянии, меняющемся после считывания очередного символа;

порождающая грамматика – формальная грамматика, позволяющая построить любую правильную цепочку символов;

автоматная грамматика – разновидность порождающей грамматики, в которой допустимо вхождение в правую часть порождающего выражения не более одного нетерминального символа и не более одного терминального символа;

регулярный язык – язык, распознаваемый конечным автоматом;

регулярная грамматика – грамматика, порождающая регулярные языки.

Автоматные грамматики, порождающие класс регулярных (автоматных) языков, относятся к наиболее простому типу грамматик. Продукции

грамматик этого типа имеют вид:

$$A \rightarrow aB,$$

$$A \rightarrow a,$$

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:

$$A \rightarrow Ba,$$

$$A \rightarrow a.$$

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

Класс регулярных языков узок, но исключительная простота алгоритмов распознавания этих языков делает их привлекательными для использования в лексическом анализе 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 и могут восприниматься как отдельные символы, что упрощает дальнейший анализ текста программы.

Пример. 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:

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

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

Можно показать, что любой конечный язык, т.е. язык, содержащий конечное множество цепочек, $L = \{\alpha_1, \dots, \alpha_n\}$ является автоматным языком. Действительно, учитывая, что любая цепочка языка $\alpha_i = a_{i1} a_{i2} \dots a_{ijk_i}$, можно задать множество продукций:

$$P = \{S \rightarrow a_{i1}A_{i1}, A_{i1} \rightarrow a_{i2}A_{i2}, \dots, A_{ijk_i, i-1} \mid i = 1, \dots, n\}.$$

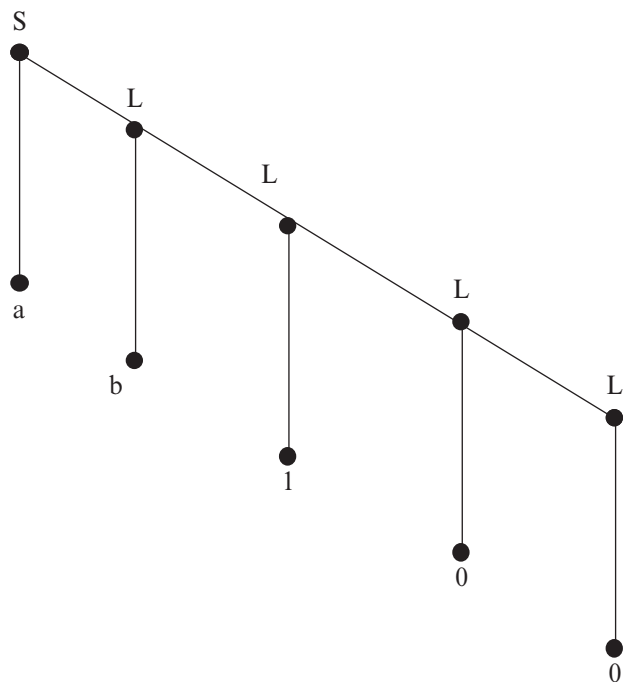


Рис. 4. Дерево вывода цепочки автоматного языка

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

Доказано, что можно 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, \dots, R_n – представляющие выражения, то $(R_{i_1}, \dots, R_{i_m})^*$ – тоже представляющее выражение ($ik \in \{1, \dots, n\}, k = 1, \dots, m$);
- 4) если R_1, \dots, R_n – представляющие выражения, то $R_1 \cup \dots \cup R_n$ – 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 ($i = 1, \dots, n$) представляют собой языки L_i ($i = 1, \dots, n$), то $(R_{i1}, \dots, R_{im})^*$ представляет собой язык $\{L_{i1} \cup \dots \cup L_{im}\}^*$;

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

Каждому представляющему выражению можно поставить в соответствие граф.

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

Рис. 5. Примеры представляющих выражений и соответствующих им графов

На рис.5 приведены примеры представляющих выражений и соответствующих им графов:

первое представляющее выражение задает язык, состоящий из одной цепочки: $L_1=\{ab\}$;

второе представляющее выражение задает бесконечный язык, являющийся итерацией алфавита $\{a\}$: $L_2=\{\epsilon, a, aa \dots\}$;

третье представляющее выражение задает свободную полугруппу с образующими $\{a, b\}$;

четвертое представляющее выражение задает язык, порождаемый автоматной грамматикой из примера, приведенного выше, и состоящий из идентификаторов (цепочек символов, начинающихся с буквы).

1.2.2. 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.

При использовании формальных грамматик в качестве механизма для порождения решений в СИИ (системах искусственного интеллекта) 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 определенности порождающих грамматик. В пределе это может превратить грамматики в алгоритмы, которые работают с формализованными знаниями как с данными. 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 неоднозначность вывода, позволяющая включать в контур управления человека.

Было предложено следующее обобщение бесконтекстных грамматик. 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

$$A \rightarrow \alpha,$$

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:

$$\begin{aligned} S &\Rightarrow^* \beta \text{ или} \\ &(r_1, \dots, r_n) \\ S &\Rightarrow_{r_1 r_2 \dots r_n} \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n = \beta. \end{aligned}$$

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

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

Пример. В примере (п. 1.1.4) приведена грамматика непосредственных составляющих для порождения описаний равносторонних треугольников в виде цепочек языка $L = \{1^n r^n x^n\}$. 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:

$$V_T = \{1, r, x\}; V_N = \{S, L, W\};$$

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

$$(1) (S \rightarrow LW) (2, 3) (\emptyset),$$

$$(2) (L \rightarrow IL) (4) (\emptyset),$$

- (3) $(L \rightarrow I)$ (5) (\emptyset) ,
 (4) $(W \rightarrow rWx)$ (2, 3) (\emptyset) ,
 (5) $(W \rightarrow rx)$ (\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.

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

Для порождения той же цепочки с помощью грамматики, приведенной в первом примере (п.1.1.4), необходимо сделать существенно большее число шагов $(1/2 n (n + 5))$.

Приложение теории формальных грамматик, связанное с грамматическим разбором или синтаксическим анализом цепочек языков, 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** – процесса отождествления объекта с одним из известных системе объектов. В этом случае, **распознающая грамматика** – это формальная грамматика, которая позволяет определить правильность (неправильность) любой предъявленной цепочки символов.

Такая трактовка синтаксического анализа лежит в основе синтаксического подхода к распознаванию образов. Грамматический разбор (синтаксический анализ) играет важную роль в системах общения человека с ЭВМ, в которых используются формальные модели естественного языка.

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.

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 вывод, воспринимать новую информацию, отвечать на вопросы и интерпретировать команды.

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

Порождающие грамматики, рассмотренные ранее, не могут служить целям понимания языка в силу того, что в них полностью отсутствует семантический аспект. Поэтому предлагаются расширения моделей, 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 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:

$$A_{i_0} \rightarrow A_{i_1} \dots A_{i_k}.$$

Семантическое правило – это функция f_{ijk} , переводящая определенные признаки символов $A_{i_0}, A_{i_1}, \dots, A_{i_k}$ в значение некоторого признака r символа A_{i_j} ($0 \leq j \leq k$).

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. 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:

$$(1) S \rightarrow L,$$

$$(2) S \rightarrow LL,$$

$$(3) L \rightarrow B,$$

$$(4) L \rightarrow LB,$$

$$(5) B \rightarrow 0,$$

$$(6) B \rightarrow 1.$$

Символы грамматики имеют следующие признаки:

$$A_0(B) = \{v\}.$$

$$A_1(B) = \{s\},$$

$$A_0(L) = \{v, 1\},$$

$$A_1(L) = \{s\},$$

$$A_0(S) = \{v\}.$$

Множества признаков терминальных символов пусты.

Множества величин признаков таковы:

D_v – множество рациональных чисел,

D_s – множество целых чисел,

D_1 – множество целых чисел.

Семантика признаков следующая:

v — “величина” символа,

s — “позиция” символа,

1 — “протяженность” символа.

Семантические правила имеют следующий вид:

для продукции (1) – $v(S) = v(L)$, $s(L) = 0$;
 для продукции (2) – $v(S) = v(L_1) + v(L_2)$, $s(L_1) = 0$, $s(L_2) = -1(L_2)$;
 для продукции (3) – $v(L) = v(B)$, $s(B) = s(L)$, $1(L)=1$;
 для продукции (4) – $v(L_1) = v(L_2)+v(B)$, $s(B) = s(L_1)$;
 $s(L_2) = s(L_1)+1$, $1(L_1) = 1(L_2)+1$;
 для продукции (5) – $v(B)=0$;
 для продукции (6) – $v(B) = 2^{s(B)}$.

1.2.3. Конечные 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. Как показали 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.

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

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

где:

$V = \{a_1, \dots, a_m\}$ – входной алфавит (конечное множество символов);

$Q = \{q_0, q_1, \dots, q_{n-1}\}$ – 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 .

На рис. 6 изображен конечный автомат в начальном состоянии q_0 , считывающий первый символ a_{i1} входной цепочки α_i . Стрелкой показано направление движения читающей головки. Отображение δ можно представить в виде совокупности так называемых команд, которые обозначаются следующим образом:

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

где: $q, q' \in Q; a \in V$.

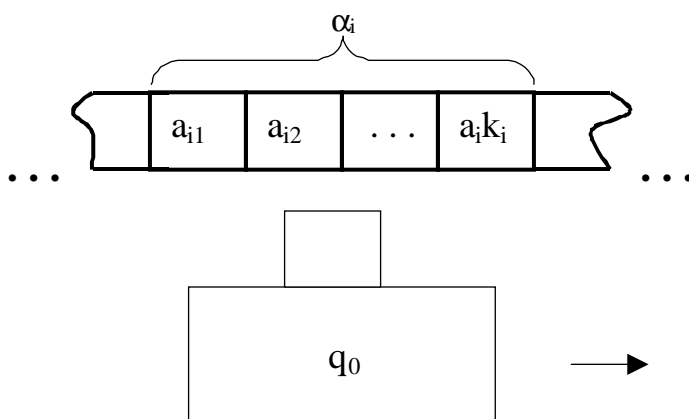


Рис. 6. Интерпретация конечного автомата

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. 7).

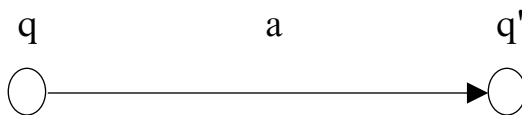


Рис. 7. Графическое представление команды автомата

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_i, 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* автоматом 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 $(q, a) \rightarrow q'$. 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. 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. 8.

Из диаграммы видно, что язык, допускаемый этим автоматом,

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

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 являющимся 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.

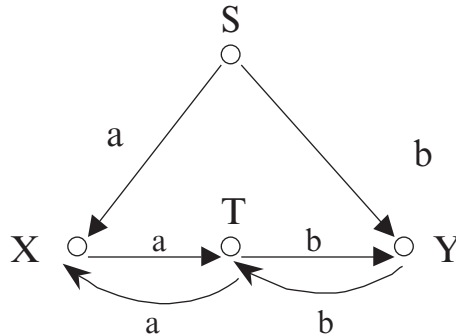


Рис. 8. Диаграмма состояний
детерминированного конечного автомата, допускающего язык $(aa, bb)^+$

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 $Q = \{q_0, q_1, \dots, q_{n-1}\}$, $V = \{a_1, \dots, a_m\}$, а список заключительных состояний представлен вектором F .

На рис. 9 представлена матрица переходов для автомата из вышеприведенного примера.

Недетерминированный конечный автомат есть пятерка вида:

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

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

Рис. 9. Матрица перехода для автомата, допускающего язык $(aa, bb)^+$

где:

$V = \{a_1, \dots, a_m\}$ – входной алфавит;

$Q = \{q_0, q_1, \dots, q_{n-1}\}$ – 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 productions:

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

продукции которой имеют вид:

$$A_i \rightarrow a_j A_k \text{ или } A_i \rightarrow a_{j_l}.$$

Построим для нее недетерминированный конечный автомат:

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

где:

$$V = V_T;$$

$$Q = V_N \cup \{T\}, \text{ причем символ } T \text{ не должен содержаться в } V_N;$$

$$q_0 = S;$$

$$F = \{T\}.$$

Отображение δ строится следующим образом:

каждой продукции вида $A_i \rightarrow a_j$ ставится в соответствие команда $(A_i, a_j) \rightarrow T$;

каждой продукции вида $A_i \rightarrow a_j A_k$ ставится в соответствие команда $(A_i, a_j) \rightarrow A_k$;

других команд нет.

Построенный таким образом автомат обозначим A_G .

Ввиду того, что любой регулярный язык может быть порожден автоматной грамматикой с правосторонними productions, 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 $(q_i, a_j) \rightarrow q_k$ ставится в соответствие продукция $q_i \rightarrow a_j q_k$, если $q_k \in Q$;
 каждой команде $(q_i, a_j) \rightarrow q_k$ ставится в соответствие еще одна продукция $q_i \rightarrow a_j$, 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;

$\delta(q, \alpha a) = \bigcup_{q_i \in \delta(q, \alpha)} \delta(q_i, a)$, 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 большими возможностями, чем детерминированный, классы языков, которые они допускают, совпадают.

Пример. 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:

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

Можно построить недетерминированный конечный автомат $A_G = (V, Q, \delta, q_0, F)$. При этом компонентами автомата будут:

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

Отображение δ представлено на рис. 10 в виде диаграммы состояний.

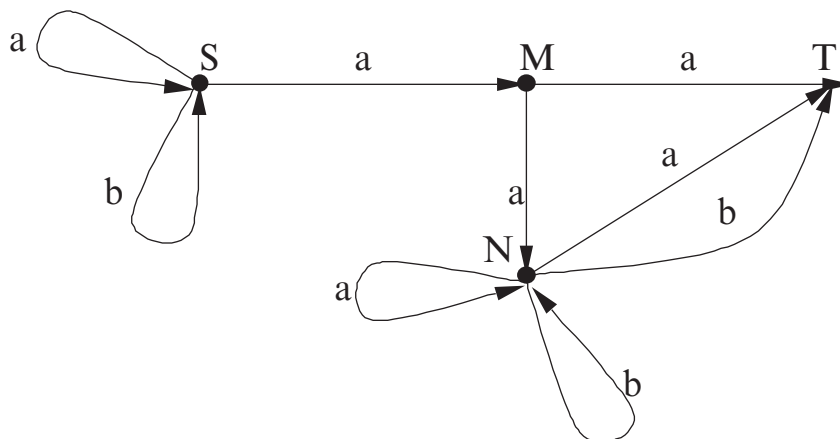


Рис. 10. Диаграмма состояний недетерминированного конечного автомата, допускающего язык $(a, b)^* aa\{a, b\}^*$

Теперь 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:

$$(\{S\}, a) \rightarrow \{S, M\};$$

$$(\{S\}, b) \rightarrow \{S\};$$

$$(\{S, M\}, a) \rightarrow \{S, M, N, T\};$$

$$(\{S, M\}, b) \rightarrow \{S\};$$

$$(\{S, M, N, T\}, a) \rightarrow \{S, M, N, T\};$$

$$\begin{aligned}(\{S, M, N, T\}, b) &\rightarrow \{S, N, T\}; \\(\{S, N, T\}, a) &\rightarrow \{S, M, N, T\}; \\(\{S, N, T\}, b) &\rightarrow \{S, N, T\}.\end{aligned}$$

На рис. 11 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' ; $\{S, N, T\}$ – символом T'_1 ; $\{S, M, N, T\}$ – символом T'_2 .

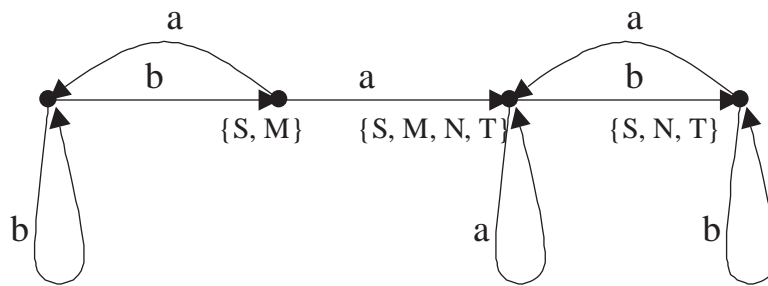


Рис.11. Диаграмма состояний детерминированного конечного автомата, допускающего язык $\{a, b\}^* aa\{a, b\}^*$

Продукциями грамматики $G_{A'}$ будут следующие:

$$\begin{aligned}S' &\rightarrow aM', S' \rightarrow bS', M' \rightarrow bS', M' \rightarrow aT'_1, T'_1 \rightarrow aT'_1, T'_1 \rightarrow bT'_1, T'_2 \rightarrow aT'_1, \\T'_2 &\rightarrow bT'_2, T'_1 \rightarrow a, T'_1 \rightarrow b, T'_2 \rightarrow a, T'_2 \rightarrow b.\end{aligned}$$

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:

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

где:

V_1, V_2 – соответственно входной и выходной алфавиты;
 Q, q_0 и F определяются так же, как и для конечных автоматов;

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 ;
- функции выходов ψ , отображающей $Q \times V_1$ в V_2 .

Отображение μ в этом случае можно представить совокупностью команд вида:

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

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

Можно ввести в рассмотрение ось времени, предполагая, что работа автомата протекает в дискретные такты времени $t = 0, 1, 2 \dots$.

Пусть конечный автомат в момент времени t_0 находится в состоянии $q_i(t_0)$. Если на его вход поступает символ $a_r(t_0) \in V_1$, то по команде:

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

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 рекуррентные соотношения:

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

$$b(t) = \psi[q(t), a(t)],$$

где: $q(t), q(t+1) \in Q, a(t) \in V_1, b(t) \in V_2$.

Эти соотношения вместе с начальным условием:

$$q(1) = q_0$$

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

$$a(1) a(2) \dots a(r)$$

в последовательность выходных символов:

$$b(1) b(2) \dots b(r).$$

Частными случаями детерминированных конечных автоматов (преобразователей) являются следующие варианты автоматов:

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

$$a_r \rightarrow b_s;$$

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

$$q_i \rightarrow \{q_j, b_s\};$$

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

$$(q_i, a_i) \rightarrow q_j.$$

1.2.4. Автоматы с магазинной памятью

Автоматы и преобразователи с магазинной (стековой) памятью играют значительную роль при построении автомато-лингвистических моделей различного назначения, связанных с использованием бесконтекстных (контекстно-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.

На рис. 12 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:

- 1) стереть символ из верхней ячейки (при этом все остальные символы перемещаются на одну ячейку вверх);
- 2) стереть символ из верхней ячейки и записать в стек непустую цепочку символов (при этом содержимое стека 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;

$V_M = \{z_0, z_1, \dots, z_{p-1}\}$ – 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;

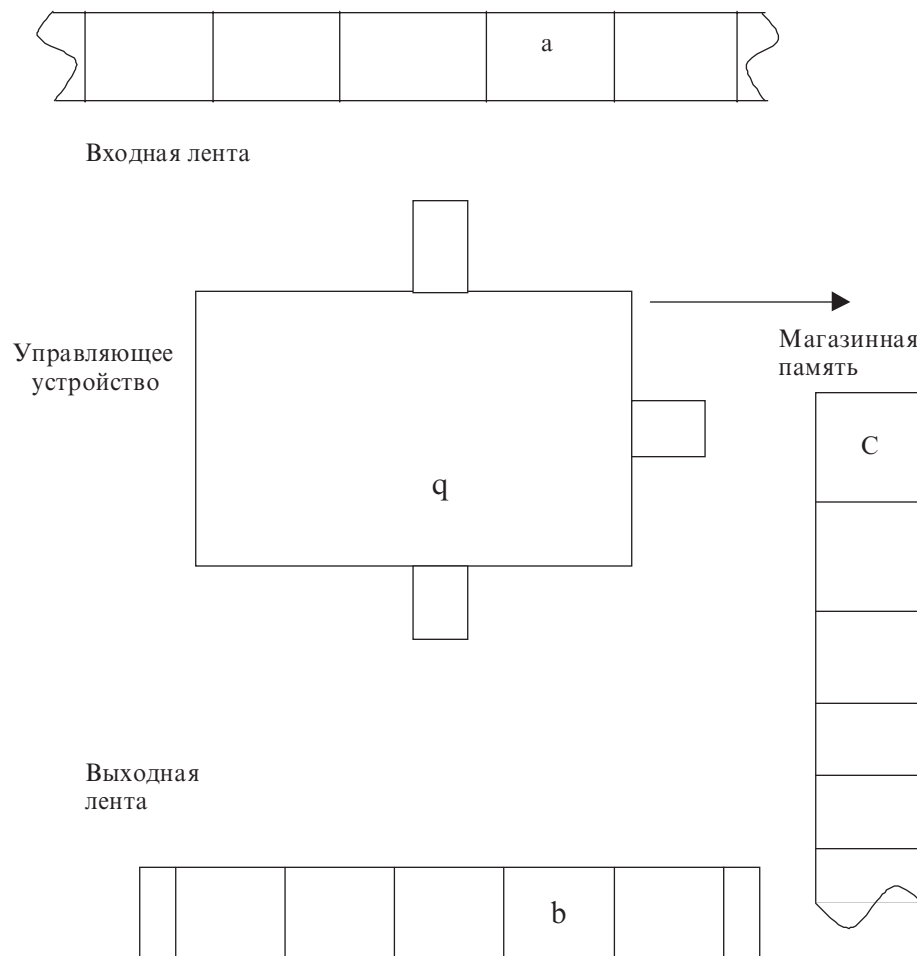


Рис.12. Интерпретация преобразователя с магазинной памятью

$z_0 \in V_M$ – 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 $Q \times (V \cup \{\epsilon\}) \times V_M$ 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 недетерминированного магазинного автомата.

Эту функцию можно представить совокупностью команд вида:

$$(q, a, z) \rightarrow (q_1, \gamma_1), \dots, (q_m, \gamma_m),$$

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

Если:

на входе читающей головки автомата находится символ a ;

автомат находится в состоянии q ;

верхний символ стека z ,

то автомат может перейти к состоянию q_i , записав при этом в стек цепочку $\gamma_i (1 \leq i \leq m)$ вместо символа z , передвинуть входную головку на один символ вправо так, как это показано на рис. 12, и перейти в состояние q_i . Крайний левый символ γ_i должен при этом оказаться в верхней ячейке магазина. Команда $(q, \varepsilon, z) \rightarrow (q_1, \gamma_1), \dots, (q_m, \gamma_m)$ означает, что независимо от входного символа и не передвигая входной головки, автомат перейдет в состояние q_i , заменив символ z магазина на цепочку $\gamma_i (1 \leq i \leq m)$.

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

$$(q, a, z) \rightarrow (q_1, \gamma_1), \dots, (q_m, \gamma_m),$$

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

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

$$a: (q, \gamma) | - (q', \gamma').$$

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 .

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 $q_f \in F$.

Доказано, что множество языков, допускаемых произвольными

магазинными автоматами согласно первому способу, совпадает с множеством языков, допускаемых согласно второму способу.

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. Рассмотрим бесконтекстный язык $L = \{\alpha\beta\}$, словарь которого $V = \{ab\}$, а цепочка β является “зеркальным отражением” цепочки α , т.е.:

$$\begin{aligned}\alpha &= a_1 \dots a_n, \\ \beta &= b_1 \dots b_n,\end{aligned}$$

причем $a_i = b_{n-i+1}$, $n \geq 1$, $1 \leq i \leq n$.

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

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

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

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

Для этой грамматики построим соответствующий ей недетерминированный автомат с магазинной памятью:

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

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

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

а отображение δ соответствует множеству команд:

$$\begin{aligned}(q_0, a, S) &\rightarrow (q_0, SA), (q_0, A), \\ (q_0, b, S) &\rightarrow (q_0, SB), (q_0, B), \\ (q_0, a, A) &\rightarrow (q_0, \varepsilon), \\ (q_0, b, B) &\rightarrow (q_0, \varepsilon).\end{aligned}$$

На рис. 13 изображены этапы просмотра цепочки $abba$ и соответствующие этим этапам состояния рабочей ленты (магазинной памяти). Считывание последнего символа цепочки сопровождается переходом автомата в состояние, при котором рабочая лента содержит пустую цепочку. Таким образом, входная цепочка $abba$ допускается данным автоматом (принадлежит языку $L_1(M)$).

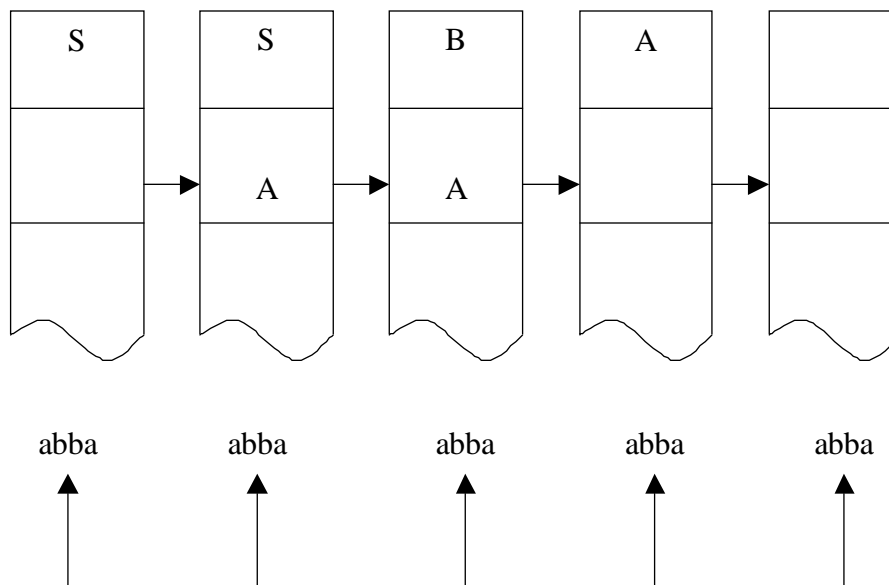


Рис. 13. Этапы просмотра цепочки abba магазинным автоматом

1.2.5. Машина Тьюринга

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

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 $a_i = a_{i1}, a_{i2}, \dots, a_{in}$, все остальные ячейки считаются заполненными специальным символом B – пробел, который не является входным (рис. 14).

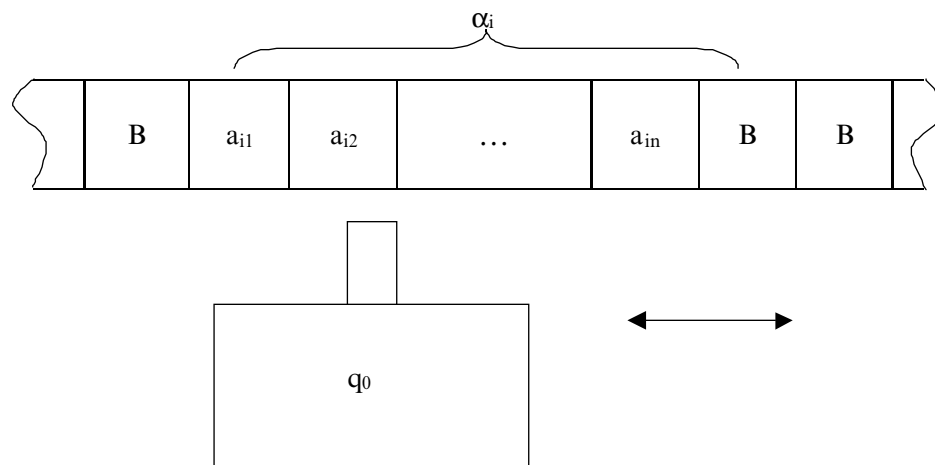


Рис. 14. Интерпретация машины Тьюринга

Формально машина Тьюринга определяется как:

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

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;

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

Отображение (функцию) δ можно задать как совокупность команд вида:

$$(q, A) \rightarrow (q', A', L), \text{ либо } (q, A) \rightarrow (q', A', P).$$

Ситуация машины Тьюринга T – это тройка вида (q, β, i) ,

где:

$q \in Q$;

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

i ($0 \leq i \leq n+1$) – 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 :

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

Пусть среди команд отображения δ имеется следующая:

$$(q, A_i) \rightarrow (q', X, \mathcal{L}).$$

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:

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

Аналогично для команды $(q, A_i) \rightarrow (q'X, \mathcal{P})$ движение машины Тьюринга T записывается как:

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

Возможны и такие ситуации:

$$(q, A_1 \dots A_n, 0),$$

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

К ним применимы команды вида $(q, B) \rightarrow (q', X, \mathcal{L})$, либо $(q, B) \rightarrow (q', X, \mathcal{P})$.

Первая из этих команд меняет указанные ситуации соответственно следующим образом:

$$(q, A_1 \dots A_n, 0) \mid - (q', X A_1 \dots A_n, 0),$$

$$(q, A_1 \dots A_n, n+1) \mid - (q', A_1 \dots A_n X, n).$$

Вторая из этих команд меняет их так:

$$(q, A_1 \dots A_n, 0) \mid - (q', X A_1 \dots A_n, 2),$$

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

Если ситуации (q_1, β_1, i_1) и (q_2, β_2, i_2) связаны между собой некоторым числом элементарных действий, то между ними имеет место отношение

$$(q_1, \beta_1, i_1) \mid -^* (q_2, \beta_2, i_2).$$

Язык, допускаемый машиной Тьюринга T , – это:

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

где: $q_f \in F$, $\beta \in V_1^*$.

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

Пример. Рассмотрим язык описания равносторонних треугольников из примера подраздела 1.1.4 – $L = \{1^n r^n x^n \mid n \geq 1\}$.

Этот язык допускает следующая машина Тьюринга:

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

где:

$$V_1 = \{1, r, x, \}, V_2 = \{1, r, x, L, R, X, Z, B\},$$

$$Q = \{q_0, q_1, \dots, q_{12}\}; F = \{q_{12}\}.$$

Отображение δ задается следующими командами:

$$\begin{aligned} (q_0, 1) &\rightarrow (q_1, 1, Л); (q_7, R) \rightarrow (q_7, R, Л); \\ (q_1, B) &\rightarrow (q_2, z, П); (q_7, L) \rightarrow (q_7, 1, Л); \\ (q_2, 1) &\rightarrow (q_3, 1, П); (q_7, L) \rightarrow (q_7, L, Л); \\ (q_3, 1) &\rightarrow (q_2, 1, П); (q_7, z) \rightarrow (q_8, Z, П); \\ (q_3, r) &\rightarrow (q_4, R, П); (q_8, L) \rightarrow (q_8, L, П); \\ (q_4, r) &\rightarrow (q_4, r, П); (q_8, R) \rightarrow (q_8, R, П); \\ (q_4, x) &\rightarrow (q_5, X, П); (q_8, X) \rightarrow (q_8, X, П); \\ (q_5, x) &\rightarrow (q_5, x, П); (q_8, Z) \rightarrow (q_{12}, Z, Л); \\ (q_5, B) &\rightarrow (q_8, Z, Л); (q_8, 1) \rightarrow (q_9, L, П); \\ (q_6, X) &\rightarrow (q_6, X, Л); (q_9, 1) \rightarrow (q_9, 1, П); \\ (q_6, R) &\rightarrow (q_6, R, Л); (q_9, R) \rightarrow (q_9, R, П); \\ (q_6, L) &\rightarrow (q_6, 1, Л); (q_9, r) \rightarrow (q_{10}, R, П); \\ (q_6, Z) &\rightarrow (q_{12}, Z, Л); (q_{10}, R) \rightarrow (q_{10}, r, П); \\ (q_6, x) &\rightarrow (q_7, x, Л); (q_{10}, X) \rightarrow (q_{10}, X, П); \\ (q_7, x) &\rightarrow (q_7, x, Л); (q_{10}, x) \rightarrow (q_{11}, X, П); \\ (q_7, X) &\rightarrow (q_7, x, Л); (q_{11}, x) \rightarrow (q_{11}, x, П); \\ (q_7, r) &\rightarrow (q_7, r, Л); (q_{11}, Z) \rightarrow (q_6, Z, Л). \end{aligned}$$

Последовательность действий для входной цепочки $\alpha = 1rx$ следующая:

$$\begin{aligned} (q_0, 1rx, 1) &\vdash (q_1, 1rx, 0) \vdash (q_2, Z1rx, 2) \vdash (q_3, ZLrx, 3) \vdash (q_4, ZLRx, 4) \vdash \\ &\vdash (q_5, ZLRX, 5) \vdash (q_6, ZLRXZ, 4) \vdash (q_6, ZLRXZ, 3) \vdash (q_6, ZLRXZ, 2) \vdash \\ &\vdash (q_6, ZLRXZ, 1) \vdash (q_{12}, ZLRXZ, 2). \end{aligned}$$

Так же как для автоматов, можно ввести понятие *недетерминированной* машины Тьюринга. Ее отличие от детерминированной машины заключается в том, что функция δ отображает множество $Q \times V_2$ в множество подмножеств $Q \times (V_2 - \{B\}) \times \{L, P\}$.

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.

При построении автоматно-лингвистических моделей, предназначенных для распознавания языков, возникает задача разрешимости для них *проблемы распознавания*. 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?

Не для всех языков типа 0 эта проблема разрешима, т.е. *существует 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 подразделах, а также так называемые *линейно-ограниченные автоматы*, допускающие языки типа 1.

Линейно-ограниченный автомат задается следующим образом:

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

где:

$V_1 = \{a_1, \dots, a_m, Z_1, Z_p\}$ – конечный входной алфавит;

$V_2 = \{A_1, \dots, A_k\}$ – конечное множество ленточных символов, причем

$V_1 \subseteq V_2$;

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

$q_0 \in Q$ – начальное состояние;

$F \subseteq Q$ – множество заключительных состояний;

δ – функция, отображающая $Q \times V_2$ множество подмножеств в $Q \times V_2 \times \{L, P\}$.

Множество V_1 содержит два специальных символа Z_1 и Z_p , называемых граничными маркерами, которые не позволяют головке управляющего устройства уйти с той части ленты, на которой задана входная цепочка.

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:

$$L(M) = \{\alpha \mid \alpha \in (V_1 - \{Z_1, Z_p\})^* \wedge (q_0 Z_1 \alpha Z_p, 1) \vdash^*(q_f, \beta, i)\},$$

где: $q_f \in F$, $\beta \in V_2$, $1 \leq i \leq n$ (n – длина исходной цепочки).

1.3. Машинный лингвистический анализ языков

1.3.1. Теория формальных грамматик и машинный анализ языков

Теория формальных грамматик во многом обязана своему появлению многочисленным исследованиям естественных языков.

Так, например, возникновение понятий автоматного языка, регулярного множества, марковской цепи было связано со статистическим исследованием последовательностей гласных и согласных в литературном тексте, а введенное Хомским понятие формальной грамматики возникло в результате исследований, направленных на создание строгих описаний грамматических закономерностей естественного языка.

В настоящее время естественный язык продолжает оставаться объектом изучения специалистов в области кибернетики. Основная цель исследований естественного языка – это построение таких описаний 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 – наиболее универсальное средство для моделирования и описания творческого мышления человека в системах искусственного интеллекта, 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 исследования, описание механизмов, посредством которых люди строят и понимают высказывания, с помощью традиционных грамматик весьма несовершенно. Было установлено, что естественные языки нельзя рассматривать как *бесконтекстные*.

В тоже время, именно 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, правда, не будет обладать достаточной объяснительной силой. *Объяснительной силой* описания фразы языка называют способность синтаксической структуры этой фразы (например, дерева вывода) обеспечивать ее семантическую интерпретацию. А без семантической интерпретации не имеет смысла говорить о моделировании процесса понимания естественного языка.

Граматики *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.

При введении различных классов грамматик не акцентировалось внимание на том, что их определения дают, по существу, механизмы порождения цепочек. *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 исследованиях по методам анализа языковых текстов (цепочек языка) были предложены два метода грамматического разбора:

нисходящий и восходящий. В соответствии с этим все распознаватели (анализаторы) 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:

$$A \rightarrow \alpha, A \rightarrow \beta,$$

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

Метод 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:

$$A \rightarrow \alpha, B \rightarrow \alpha,$$

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. Для анализа цепочек, порождаемых 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. 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 анализируемых символов. Таким образом, в данном случае мы имеем дело не с конечным автоматом, а с конечным преобразователем.

1.3.2. Грамматики простого предшествования

Одним из методов, позволяющих осуществить эффективный восходящий разбор, является метод, предложенный для анализа специального подкласса бесконтекстных грамматик, называемых грамматиками предшествования. **Грамматика предшествования** – контекстно-свободная грамматика, на символах которой заданы отношения, позволяющие определить во входной строке возможные границы синтаксических конструкций, рассматривая лишь пары соседних символов.

На множестве терминальных символов такой грамматики вводятся так называемые отношения предшествования \doteq , $>$, $<$, 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:

$$A \rightarrow \alpha ab\beta,$$

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

Будем говорить, что для пары (a, b) выполняется отношение $>$, если среди множества продукций P есть продукция вида:

$$A \rightarrow \alpha aB\beta,$$

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

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

где: $A, B \in V_N$, $\alpha, \beta, \gamma \in V^*$.

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

1) среди множества продукций P содержится продукция вида:

$$A \rightarrow \alpha b B \beta,$$

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

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

где: $A, B \in V_N$; $\alpha, \beta, \gamma \in V^*$;

2) среди множества продукций P содержится продукция вида:

$$A \rightarrow \alpha B C \beta,$$

причем

$$B \Rightarrow^* \gamma a, C \Rightarrow^* b \delta,$$

где: $A, B, C \in V_N$; $a, b \in V_T$; $\alpha, \beta, \gamma, \delta \in V^*$.

Предложен алгоритм, позволяющий без возвратов анализировать любую цепочку, порождаемую грамматикой предшествования, если на эту грамматику наложены следующие ограничения:

нет продукций вида $A \rightarrow \alpha, B \rightarrow \alpha$,

где:

$A, B \in V_N$; $\alpha \in \{V_N \cup V_T\}^*$;

нет продукций вида $A \rightarrow \alpha B C \beta$,

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 ограниченный класс бесконтекстных языков, поэтому производится расширение этого класса последовательным ослаблением ограничений:

– вводятся отношения предшествования и для нетерминальных символов;

– допускается неоднозначность этих отношений.

Методы предшествования (восходящие) основаны на использовании контекста из одного или двух символов. Дальнейшим их развитием явились $LR(k)$ -грамматики. 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.

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, что для любой бесконтекстной грамматики можно построить эквивалентную ей $LR(k)$ -грамматику для $k \geq 1$, на практике они используются в основном лишь для $k = 1$.

$LR(k)$ -распознаватели занимают положение, промежуточное между восходящей и нисходящей стратегиями. Однако условно их можно отнести к классу нисходящих распознавателей, так как в них все-таки преобладает нисходящая стратегия разбора.

1.3.3. Методы нисходящего грамматического разбора

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

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

$$S \rightarrow \alpha_{i1} A_{i1} \alpha_{i2} \dots A_{iin-1} \alpha_{in},$$

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

Пусть β анализируемая цепочка. Тогда, если выбрать одну из k этих продукций, цепочка β должна содержать в качестве подцепочек цепочки $\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{in}$, 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 связана в данном случае 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:

$$C \rightarrow AbB.$$

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.

Из методов грамматического разбора, непосредственно связанных с анализом предложений на естественном языке, рассмотрим *процедурный метод* грамматического 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 воспринимающего состоит в том, чтобы опознать присутствие таких признаков и использовать их для семантической интерпретации этой фразы.

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:

- главное;
- второстепенное;
- утвердительное;
- побудительное;
- 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 в качестве формализма для описания грамматики использует специальный язык

PROGRAMMAR, а система грамматического разбора представляет собой интерпретатор с этого языка. Он производит разбор, используя нисходящую стратегию. При этом характерной особенностью и достоинством процедурного метода 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.

Благодаря семантической интерпретации в ходе разбора PROGRAMMAR не имеет автоматического механизма возврата.

Вопросы семантической интерпретации, играющие определяющую роль при синтаксическом анализе естественного языка, базируются на проблеме представления знаний.

1.4. Трансляторы

1.4.1. Основные понятия

В разделе 1.4. “Трансляторы” показано, как теория формальных грамматик реализуется в прикладной области программирования.

Программирование – в широком смысле – это процесс, включающий все технические операции по созданию программы от анализа требований до конечной реализации. В более узком смысле под программированием понимается процесс кодирования, трансляции, компоновки и тестирования программы в рамках определенного проекта. Программирование, как правило, связывают с конкретными *языками программирования*, под которыми понимаются искусственные языки (системы обозначений) со строго определенными синтаксисом и семантикой, предназначенными для точного описания программ или алгоритмов для электронных вычислительных машин (ЭВМ).

Теория языков программирования базируется, в первую очередь, на теории формальных языков и грамматик (язык программирования – ограниченный естественный язык) и в ней вводятся следующие основные понятия для языков и технологии программирования:

лингвистический объект – грамматически допустимая конструкция языка;

синтаксис – набор правил, которые определяют основные внутренние структуры и последовательности символов, допустимые в языках программирования;

семантика – значения языковых единиц (слов, грамматических форм слов, словосочетаний, предложений);

метаязык – язык, используемый только для описания другого языка.

Как правило, программа реализует некий алгоритм. **Алгоритм** – конечный набор предписаний, определяющий решение задачи посредством конечного количества операций, а **программа** – данные, предназначенные для управления конкретными компонентами системы обработки данных в целях реализации определенного алгоритма.

Программа, которая написана на определенном языке программирования, чтобы быть выполненной на ЭВМ, должна быть оттранслирована (откомпилирована). **Трансляция** – это преобразование исходной системы данных в другую рабочую систему данных с аналогичными отношениями, а **транслятор (компилятор)** – программа или техническое средство, выполняющее трансляцию программы.

Теория трансляторов, как уже говорилось, основывается на теории формальных грамматик, включая методы **синтаксического анализа** – процесса принятия решения о том, является ли цепочка введенных символов предложением данного языка программирования. Важнейшими компонентами любого транслятора являются:

лексический анализатор – компонента, осуществляющая проверку правильности представления лексических конструкций исходного языка программирования и приведение программы к виду, допускающему ее обработку синтаксическим анализатором;

синтаксический анализатор – компонента, осуществляющая проверку исходных операторов на соответствие синтаксическим правилам данного языка программирования.

Кроме трансляции, для реализации выполнения программы на ЭВМ иногда используется режим **интерпретации** – реализации смысла некоторого синтаксически законченного текста, представленного на конкретном языке с помощью **интерпретирующего компилятора**, который осуществляет независимую компиляцию каждого отдельного оператора исходной программы.

И прежде чем перейти непосредственно к структуре языка программирования, как лингвистической системе, напомним, что **программный модуль** – это программа, рассматриваемая как целое в контекстах хранения в наборе данных, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

1.4.2. Структура языка программирования с точки зрения транслятора

Грамматические правила лексики языка программирования (ЯП) рассматриваются транслятором с точки зрения существования различ-

ных категорий слово-ориентированных языковых единиц, называемых *лексемами*. Лексемы распознаются компилятором. Грамматические правила *структуры фраз* подробно определяют допустимые способы группирования этих лексем в выражения, операторы и прочие смысловые единицы языка.

Лексемы ЯП образуются из последовательности операций, выполняемых с программой компилятором языка.

Программа начинает свое существование как последовательность стандартных символов, представляющих собой ее исходный код, создаваемый при работе в текстовом редакторе.

Базовая программная единица представляет собой файл. Обычно такой файл соответствует файлу операционной системы (ОС), находящемуся в оперативной или внешней памяти и имеющему созданное по правилам ОС имя и расширение.

На *фазе компиляции*, отвечающей за распознавание лексем, файл исходного кода программы подвергается лексическому анализу (т.е. разбиению на лексемы и пробелы). Пробельными обобщенно именуются собственно символы пробелов, горизонтальные и вертикальные символы табуляции, символы новой строки и комментарии. Пробельные символы служат для обозначения мест начала и конца лексем и, сверх этой функции, для исключения из компиляции всех избыточных символов, не входящих в состав лексем.

Стандартные символы, обычно рассматриваемые как пробельные, могут входить в строки литералов, и в данном случае будут защищены от нормального процесса разбиения на лексемы и пробелы (они станут рассматриваться транслятором как часть строки).

Комментарии представляют собой текстовые части, предназначенные для аннотирования программы. Комментарии используются исключительно самим программистом, и перед лексическим анализом они исключаются из исходного текста программы.

Указания на комментарии осуществляются с помощью специальных символов, различных в различных ЯП.

Как правило, компилятор распознает лексемы основных классов: *ключевые слова, идентификаторы, константы, строковые литералы, операции и знаки пунктуации* (также называемые *разделителями*). Формальное описание лексемы имеет следующий вид:

лексема:
ключевое слово;
идентификатор;
константа;
строковый литерал;

операция;
знак пунктуации.

Во время лексического анализа исходного кода лексемы выделяются методом, при котором из строки символов обязательно выбирается лексема максимальной длины. Например, слово `external` будет рассматриваться как отдельный идентификатор, а не как ключевое слово `extern`, за которым следует идентификатор `al`.

Ключевыми словами называются слова, зарезервированные для специальных целей, которые не должны использоваться в качестве обычных имен идентификаторов.

Идентификаторы представляют собой произвольные имена, присваиваемые классам, объектам, функциям, переменным, определяемым пользователем типам данных и т.д. Обычно идентификаторы могут содержать буквы от *A* до *Z* и от *a* до *z*, символ подчеркивания (`_`) и цифры от 0 до 9. Существует, как правило, только два ограничения:

- первый символ должен являться буквой;
- уникальность и контекст идентификаторов.

Хотя имена идентификаторов могут быть произвольными (в пределах изложенных правил), в случае использования одного и того же имени более чем для одного идентификатора в пределах одного контекста и разделении ими одного пространства имен возникает ошибка.

Константами называются лексемы, представляющие собой фиксированные числовые или символьные значения. Тип данных константы определяется компилятором по таким ключевым характеристикам, как числовое значение и формат, используемые при записи константы в исходном коде.

Операциями называются лексемы, вызывающие некоторые вычисления с переменными и прочими объектами, указанными в выражении. Набор операций современного ЯП включает в себя, помимо обычных арифметических и логических, операций средства манипуляции с данными на битовом уровне, средства доступа к компонентам структур данных и др.

Контекст, видимость, продолжительность и тип компоновки определяют части программы, из которых могут быть сделаны допустимые ссылки на идентификатор с целью доступа к соответствующему объекту.

Объектом, с точки зрения транслятора, называется идентифицируемая область памяти, которая может содержать фиксированное значение переменной (или набор таких значений). Используемое в данном случае слово “объект” не следует путать с термином, используемым в объектно-ориентированном программировании (ООП). Каждая величина имеет связанное с ней имя и тип (который также называют типом данных). Имя используется для доступа к объекту. Имя

может являться простым идентификатором либо сложным выражением, уникальным образом “указывающим” на данный объект.

Тип используется для следующих целей:

- определения требуемого количества памяти при ее исходном распределении;
- интерпретации битовых кодов, находимых в объектах при последующих к ним обращениях;
- в ситуациях контроля типа, требуемого для обнаружения возможных случаев недопустимого присваивания.

Современные ЯП поддерживают многие стандартные (предопределенные), а также определяемые пользователем типы данных, включая целочисленные типы разных размеров, со знаком и без него, числа с плавающей точкой различной точности представления, структуры, объединения, массивы и классы (в объектно-ориентированных ЯП).

Объявления устанавливают необходимые соотношения распределения памяти между идентификаторами и объектами. *Каждое объявление связывает идентификатор с некоторым типом данных.* Большинство объявлений, известных как *объявления определения*, также задают создание (где и когда) объекта, иначе говоря, распределение физической памяти и ее возможную инициализацию. Прочие объявления, называемые *объявлениями ссылки*, просто делают указанные в них идентификаторы известными компилятору. Один и тот же идентификатор может иметь множество объявлений ссылки, особенно в многофайловых программах, однако для каждого идентификатора допустимо только одно объявление определения.

Вообще говоря, идентификатор не может быть правильно использован в программе до соответствующей ему точки объявления в исходном коде.

Для связи идентификаторов с объектами транслятору требуется, чтобы каждый идентификатор имел как минимум два атрибута: *класс памяти* и *тип* (иногда его называют *типом данных*). Компилятор определяет эти атрибуты по явным или неявным объявлениям в исходном коде программы.

Класс памяти задает размещение объекта (сегмент данных, регистр, стек) и продолжительность его времени существования (все время работы программы, либо же при выполнении некоторых конкретных блоков кода). Класс памяти может быть установлен синтаксисом объявления, расположением в исходном коде или обоими этими факторами.

Тип, как говорилось выше, определяет размер памяти, распределяемый объекту, и то, каким образом программа будет интерпретировать битовые коды, находящиеся в памяти, распределенной объекту. Тип

данных можно рассматривать как множество значений, которые может принимать идентификатор данного типа, совокупно с множеством операций, выполнение которых допустимо для значений этого типа.

Контекстом идентификатора называется часть программы, в которой данный идентификатор может быть использован для доступа к связанному с ним объекту. Могут существовать, например, следующие *категории контекста*:

- блок (или локальный);
- функция;
- файл;
- класс.

Контекст зависит от того, как и где объявлены идентификаторы.

Контекст идентификатора в случае *контекста типа блока* (или локального контекста) начинается в точке объявления и заканчивается в конце блока, содержащего данное объявление (такой блок называется *объемлющим блоком*). Объявления параметров в определении функции также имеют контекст типа блока и ограничены контекстом блока, где эта функция определена.

Идентификаторами, имеющими *контекст типа функции*, являются метки операторов. Имена меток могут быть использованы в операторах *goto* в любой точке функции, где объявлена данная метка. Имена меток в пределах функции должны быть уникальными.

Идентификаторы с *контекстом файла*, называемые часто *глобальными*, объявляются вне всех блоков и классов – их контекст лежит между точкой объявления и концом исходного файла.

*Классом ООП** можно считать именованный набор компонентов, включая сюда структуры данных и действующие с ними функции. *Контекст класса* относится, за некоторыми исключениями, к именам компонентов конкретного класса. Классы и их объекты имеют множество специальных правил доступа и определения контекста.

Пространство имен – это контекст, в пределах которого идентификатор должен быть уникальным.

Видимостью идентификатора называется область исходного кода программы, из которого допустим нормальный доступ к связанному с идентификатором объекту.

Обычно контекст и видимость совпадают, однако бывают случаи, когда объект временно скрыт вследствие наличия идентификатора с тем же именем. Объект при этом не прекращает своего существования, но исходный идентификатор не может служить для доступа к нему до тех пор, пока не закончится контекст дублирующего идентификатора.

* Объектно-ориентированное программирование (ООП).

Видимость не может выходить за пределы контекста; но контекст может превышать видимость.

Термин *единица трансляции* относится к файлу исходного кода вместе с включаемыми файлами. Синтаксически *единица трансляции* определяется как последовательность внешних объявлений:

единица-трансляции:
внешнее-объявление
единица-трансляции внешнее-объявление

внешнее-объявление:
определение-функции
объявление

Выполняемая программа обычно создается компиляцией нескольких независимых единиц трансляции, а затем *компоновкой* получившихся объектных файлов с уже существующими библиотеками. Проблема возникает, когда один и тот же идентификатор объявлен более одного раза в одном и том же контексте. *Компоновка* – это процесс, который позволяет правильно связать каждое вхождение идентификатора с одним конкретным объектом или функцией. Все идентификаторы имеют один из трех атрибутов компоновки, тесно связанных с их контекстом: внешняя компоновка, внутренняя компоновка или отсутствие компоновки. Эти атрибуты определяются местоположением и форматом объявлений, а также явным (или неявным по умолчанию) использованием спецификатора класса памяти.

Ниже приводятся правила внешней и внутренней компоновки:

1. Идентификатор объекта или файла, имеющий файловый контекст, будет иметь внутренний тип компоновки.

2. Если объявление идентификатора объекта или функции содержит спецификатор класса памяти “внешний”, то идентификатор имеет тот же тип компоновки, что и видимое объявление идентификатора с файловым контекстом. Если такого видимого объявления не имеется, то идентификатор будет иметь внешний тип компоновки.

3. Если функция объявлена без спецификатора класса памяти, то ее тип компоновки определяется, как если бы был использован спецификатор класса памяти “внешний”.

4. Если идентификатор объекта с файловым контекстом объявлен без спецификатора класса памяти, то идентификатор имеет внешний тип компоновки.

Все шесть взаимосвязанных атрибутов (класс памяти, тип, контекст,

видимость, продолжительность и тип компоновки) могут быть разными способами определены при помощи объявлений.

Объявления могут быть объявлениями определения (их обычно просто называют объявлениями) и объявлениями ссылки (иногда называемыми неопределяющими объявлениями). *Объявление определения*, как и следует из названия, выполняет две функции, объявления и определения; неопределяющие же объявления требуют наличия где-либо далее в программе определений. *Объявление ссылки* просто вводит в программу одно или более имен идентификаторов. Определение фактически распределяет объекту память и связывает идентификатор с этим объектом. В число объектов, которые могут быть объявлены, входят:

- переменные;
- функции;
- классы и компоненты классов;
- типы;
- компоненты структур;
- компоненты объединений;
- массивы прочих типов;
- перечислимые константы;
- метки операторов.

Спецификатор типа с одним или более опциональным модификатором используется для задания типа объявляемого идентификатора.

Функции (процедуры) представляют собой центральный объект программирования. Функции объявляются в исходных файлах либо делаются доступными при компоновке с откомпилированными библиотеками. Функция может быть объявлена в программе несколько раз, при условии, что эти объявления совместимы. Объявления, если они имеются, должны соответствовать определению функции. Существенным различием между *определением* и *объявлением* является то, что определение содержит собственно тело функции. Компилятор использует эту информацию для контроля достоверности вызова функции. Компилятор также может приводить аргументы к нужному типу.

В целом, *определение функции* состоит из следующих разделов (грамматика позволяет создание и более сложных конструкций):

- спецификаторы класса памяти;
- тип возврата;
- имя функции;
- список объявления параметров, который может быть пустым, заключенный в круглые скобки;

– тело функции, представляющее собой коды, выполняемые при вызове функции.

Список объявления формальных параметров имеет синтаксис, аналогичный синтаксису обычных объявлений идентификаторов.

Функция вызывается с фактическими аргументами, помещенными в той же последовательности, что и соответствующие им формальные аргументы. Преобразования фактических аргументов выполняются, как если бы они инициализировались значениями формальных аргументов при объявлении типов.

1.4.3. Состав и структура современного транслятора

Важнейшей задачей при разработке компилятора с языка программирования (в качестве примера рассматривается ООЯП С++) является обеспечение максимально полного соответствия его входного языка стандартному определению. Эта задача является весьма не простой. Невозможно обеспечить полную синхронизацию процесса разработки с независимо идущим процессом стандартизации, который протекает достаточно интенсивно и сопровождается постоянными модификациями как синтаксиса, так и семантики языка.

Компилятор в целом можно рассматривать как композицию следующих процессоров, каждый из которых реализован в виде независимой программы:

- препроцессор Си++;
- компилятор переднего плана Си++ ;
- генератор объектного кода.

Построение компилятора в виде отдельных подсистем достаточно традиционно. Помимо очевидного выигрыша по времени при одновременной разработке трех компонент, а также упрощения тестирования и отладки, указанная структура обеспечивает более легкую адаптацию системы в целом, позволяя, например, путем добавления новых генераторов кода переносить компилятор на различные платформы.

Препроцессор – выполняет препроцессорную обработку исходного текста согласно правилам соответствующего стандарта Си++. Несмотря на значительную семантическую независимость конструкций, обрабатываемых препроцессором, от самого языка Си++, формально они являются частью определения языка.

Компилятор переднего плана – это компонента, которая является центральной частью всего компилятора Си++. Она воспринимает на входе текст программы на Си++ и формирует на выходе файл с семантически эквивалентным представлением этой программы на

некотором промежуточном языке, который можно трактовать как “обобщенный ассемблер”.

Обычно компилятор переднего плана построен по традиционной двухпроходной схеме. На первом проходе:

- осуществляется лексический и синтаксический анализ исходного текста;
- выполняется большой массив семантических проверок;
- генерируется табличное, древовидное представление программы.

Первичный лексический анализ (распознавание и кодирование лексем) выполняется по традиционной схеме конечного автомата. Формальное описание языка Си++ основано на множестве лексем и включает семантические действия, которые, как правило, сводятся к вызовам функций обработки объявлений, генерации поддеревьев и вычислению контекстных признаков.

Совокупность семантических таблиц, организованная в соответствии со структурой *областей видимости* исходной программы, содержит атрибуты всех программных сущностей, а дерево программы отражает структуру исполняемых конструкций. Элементы семантических таблиц (“семантические слова”) и узлы дерева связаны взаимными ссылками. Ссылки из дерева в таблицы представляют, как правило, использующие вхождения объявленных сущностей. Обратные ссылки обычно имеют характер атрибутов (например, ссылка из семантического слова переменной на дерево выражения для инициализатора этой переменной).

Второй проход компилятора заключается в серии рекурсивных обходов дерева программы. При этом выполняется окончательное вычисление атрибутов объектов, необходимых для генерации промежуточного кода (в частности, вычисляются размеры и смещения для стековых объектов), и обрабатываются вызовы функций. На завершающей фазе второго прохода производится непосредственная генерация промежуточного представления.

В компиляторе используется генератор объектного кода, который, как и другие компоненты компилятора, представляет собой независимую программу. Генератор объектного кода легко может быть заменен на эквивалентный по интерфейсу генератор для некоторой другой платформы.

Неотъемлемой частью комплекса, обеспечивающего работу компилятора, является *стандартная библиотека языка Си++*, которая представляет собой значительную (как по объему, так и по “идеологической” насыщенности) часть стандарта языка Си++. Описание библиотеки в совокупности составляет более половины общего объема стандарта. Помимо внушительного размера, предлагаемый стандартный комплекс

библиотечных ресурсов языка Си++ воплощает большое количество важных принципов и подходов, отражающих современное состояние программистской теории и практики. Прежде всего, необходимо отметить несколько существенных черт, свойственных стандартной библиотеке. Она должна обеспечивать поддержку воплощенной в языке концепции объектно-ориентированного программирования. Практически все библиотечные ресурсы спроектированы в виде иерархий классов.

Другая важная черта – использование механизма исключительных ситуаций. Все возможные “нештатные” эффекты в процессе выполнения библиотечных функций специфицированы в виде исключительных ситуаций, в библиотеке предусмотрена иерархия стандартных классов, предназначенных именно для передачи значений при возбуждении исключительных ситуаций.

Классическое объектно-ориентированное программирование использует базовый механизм – механизм наследования с последовательным увеличением детализации при переходе от класса родителя к классу потомку. В случае стандартной библиотеки Си++ активное использование шаблонных структур – шаблонных алгоритмов и шаблонных иерархий классов – позволяет библиотеке, с одной стороны, сохранять высокий уровень абстракции, а с другой – предоставлять разработчику реально действующий код. Кроме того, исходное построение библиотеки в виде совокупности четко выделенных блоков не только позволяет достичь простоты и единообразия интерфейсов, но дает пользователю большую широту в выборе способов их взаимодействия. Речь идет о повторном использовании программных модулей, в том числе стандартизации наиболее часто используемых алгоритмов и структур данных, которое стало возможным благодаря развитым средствам абстракции у современных языков программирования.

Общую структуру стандартной библиотеки можно представить как объединение нескольких крупных компонент, каждая из которых предоставляет набор примитивов и типовых решений, позволяющих достаточно легко построить набор конкретных классов и функций для некоторой прикладной области. Практически каждая компонента библиотеки существенно использует возможности, предоставляемые другими компонентами, за счет чего достигается максимальная гибкость, общность и эффективность.

Стандартная библиотека Си++ предоставляет:

- расширяемый набор классов и определений, необходимых для поддержки понятий самого языка;
- классы поддержки диагностики пользовательских приложений;
- утилиты общего назначения;

- обобщенные алгоритмы;
- средства локализации программ;
- классы и функции для математических вычислений;
- средства работы со строками;
- ввод/вывод.

Компилятор, как уже говорилось, осуществляет лингвистический контроль (анализ) входных текстов на исходном ЯП и их преобразование в исполняемые коды ЭВМ. Для использования вновь созданных компиляторов необходимо их тестирование на специально созданных входных пакетах.

На основании критериев качества и требований к пакетам тестов формулируются следующие принципы разработки пакета аттестационных тестов для проверки компиляторов на соответствие стандарту:

- набор аттестационных тестов должен покрывать весь стандарт языка;
- программы, входящие в пакет, должны быть независимыми в том смысле, что результаты компиляции или запуска любой из них не должны влиять на условия или результаты компиляции или запуска любой другой тестовой программы;
- каждая тестовая программа должна проверять соответствие между реализацией некоторой языковой конструкции и ее описанием в стандарте языка программирования.

Тестовый пакет имеет иерархическую структуру, как и структура текста стандарта языка C++. Тесты должны тестировать языковые ситуации, описанные в соответствующей части стандарта C++. Все тесты, содержащиеся в определенном каталоге, соответствующем части стандарта языка C++, образуют группу тестов для этой части. Основная цель каждой группы состоит в проверке реализации некоторого требования, содержащегося или выводимого из текста соответствующей части стандарта языка.

Каждая группа тестов разделяется на две подгруппы, состоящие из А-тестов и Б-тестов соответственно. А-тесты – это правильные программы, проверяющие точность реализации некой языковой конструкции, а Б-тесты – программы, содержащие ошибки, необходимые для проверки правильности реакции компилятора на данные ошибки.

Кроме этого, группы тестов разделяются на подгруппы в соответствии с целями тестирования и тестовыми спецификациями. Каждая такая спецификация определяет некоторую конструкцию, описанную в стандарте языка C++, которая должна быть запрограммирована, и эффект, который должен быть получен при обработке этой конструкции проверяемой реализацией компилятора. Каждой отдельной спецификации должен соответствовать ровно один тест.

Стандарт языка C++ является неформализованным (частично формализованной является лишь синтаксическая часть языка). Это ведет к тому, что процесс создания тестов тоже является частично формализованным. В этом случае одним из возможных путей создания тестов является глубокое изучение стандарта языка C++, в ходе которого выявляются и фиксируются ограничения, накладываемые на реализацию компилятора и языковые конструкции, появляющиеся в программах и программных элементах языка C++. Таким образом, при разработке тестовых спецификаций используется частично формализованная запись и специальные методы создания и уточнения тестовых атрибутов.

Атрибут теста – это любой атрибут или свойство языковой конструкции или выражения, встречающиеся в рассматриваемой части стандарта, или любое свойство контекста данной конструкции, которое может рассматриваться как предмет отдельного тестирования. Для каждого тестового атрибута определяется набор возможных значений. Каждое значение должно удовлетворять следующим двум требованиям:

- значение присутствует в стандарте;
- значение определяет отдельную тестируемую сущность.

Определяется набор эффектов, получаемых при компиляции. Каждый эффект должен удовлетворять следующим двум требованиям:

- существует возможность определения наличия или отсутствия данного эффекта;
- данный эффект должен быть как можно проще.

Спецификации для Б-тестов и методы их создания отличаются от соответствующих методов для А-тестов. Каждая такая спецификация состоит из двух частей.

Первая часть, называемая “Б-значения”, состоит из набора ошибочных значений тестовых атрибутов, определенных в соответствующей спецификации А-тестов. Каждое из ошибочных значений должно удовлетворять следующим свойствам:

- значение присутствует в стандарте языка C++;
- значение определяет отдельную тестируемую сущность.

Вторая часть называется “Ограничения” и состоит из формулировок ограничений, накладываемых на программы языка C++ и явно выделенных в соответствующей части стандарта или логически следующих из ее текста. Для некоторых ограничений явно указываются пути их нарушения. Каждое ограничение, накладываемое на программы языка C++, может быть отражено как в части “Б-значения”, так и в части “Ограничения”, но не в обеих сразу. Эффективность Б-тестов – диагностика ошибок в процессе компиляции. Это означает, что Б-тесты являются тестами этапа компиляции. По крайней мере, один тест должен

быть создан для каждого неправильного значения тестовых атрибутов. Аналогично, для каждого ограничения должен быть создан, по крайней мере, один тест. Если для некоторого ограничения добавлен набор возможностей его нарушения, то подразумевалось написание набора тестов, проверяющих каждое из этих нарушений.

Итак, подведем итоги.

Как уже отмечалось, все тесты в пакете подразделены на два класса, в соответствии с задачей тестирования и природой критериев успешного завершения или отказа.

Тест относится к классу А-тестов, если он является правильной выполнимой (в соответствии со стандартом языка C++) программой. А-тесты проверяют способность реализации обрабатывать правильные, с точки зрения стандарта, программы правильным образом. А-тест считается успешно завершённым, если его компиляция прошла успешно, и вызов получившегося кода прошел без ошибок. Причем ошибками в последнем случае являются не только ошибки, выданные соответствующей реализацией, но и сообщения самой программы о том, что данная реализация обработала ее не соответствующем стандарту способом.

Тест относится к классу Б-тестов, если он является неправильной (нарушающей требования стандарта языка C++) программой. Б-тест проверяет способность реализации обнаруживать нарушения стандарта в компилируемой программе. Б-тест считается успешно завершённым, если во время компиляции определены все ошибки, содержащиеся в нем.

Полученные тесты являются компиляционными тестами, содержащими нарушения стандарта, проверяемые на этапе компиляции.

2. ПРОБЛЕМЫ СОЗДАНИЯ ЯЗЫКОВЫХ СРЕДСТВ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

2.1. Работа со структурированными данными в автоматизированных информационных системах

2.1.1. Информационная система – модель предметной области

Предметная область – это совокупность объектов реального или предполагаемого мира, рассматриваемых в пределах данного контекста, который понимается как отдельное рассуждение, фрагмент научной теории или теория в целом и ограничивается рамками данного контекста.

В научных и инженерных кругах, занимающихся вопросами теории и практики автоматизированных информационных систем (АИС), под предметной областью понимают систематизированную совокупность объектов, свойств объектов, связей между объектами и функциями, выполняемые объектами. При этом под простым объектом предметной области понимается носитель совокупности характеризующих его свойств, через которые проявляется сущность объекта и которые неотделимы от него.

Данное определение является достаточно абстрактным, но только так и можно коротко охарактеризовать предмет дальнейшего рассмотрения, не прибегая к перечислению огромного количества областей человеческой деятельности, которые и являются предметными областями автоматизированных информационных систем, использующих информационные языки различного назначения.

Информационная база (ИБ) любой АИС представляет собой статичную информационную модель предметной области АИС, т.е. содержит систематизированное описание совокупности объектов, свойств объектов и связей между объектами в предметной области. Например, предметными областями АИС различного назначения могут быть:

библиотека – для библиотечной АИС;

учебные курсы или дисциплины;

архив конструкторско-технологических документов;

и т.п.

2.1.2. Описание предметной области

Единая информационная база может одновременно или в различные моменты времени обеспечить совершенно разные функциональные требования пользователей АИС. Совместно используя общую информацию, эти пользователи устанавливают диалог друг с другом через систему с помощью специализированных информационных языков. Очевидно, для того чтобы эта связь была полезной и надежной, должно существовать общее понимание информации, представленной в ИБ. Так как два пользователя могут никогда не встретиться, это общее понимание должно относиться к чему-либо внешнему по отношению к ним обоим. Объекты и события, к которым относится общее понимание информации – это объекты и события предметной области, которые обычно описываются в **документах** – материальных объектах с информацией, закреплённой созданным человеком способом для ее передачи во времени и пространстве. Для упорядочивания объектов в АИС используются **классификаторы** – документы, представляющие

систематизированный свод наименований и кодов группировок и (или) объектов классификации.

Предметная область состоит из реальных и абстрактных объектов, которые называют сущностями. Можно считать также, что она состоит из классов сущностей – например, люди, подразделения фирмы, даты. Классификация основывается на сходстве и учитывает характеристики, общие для нескольких сущностей. Выбор характеристик для группировки сущностей в классы произволен и осуществляется прагматически, в зависимости от целей анализа. В предметной области также рассматриваются некоторые общие свойства, которыми обладают сущности, классифицирующие их, связывающие и т.д. в данной предметной области. Они могут обозначаться как “классификации”, “правила”, “законы” или “ограничения”, касающиеся состояния и поведения сущностей в предметной области.

Все то, что считается частью предметной области, зависит от времени, то есть выбранные объекты и события могут со временем изменяться. Это в равной степени относится и к классификациям, правилам, законам, но, как правило, частота их изменения будет меньше.

Повторим, что информация, содержащаяся в АИС, дает описание предметной области. Конкретное физическое представление этой информации называют *базой данных* АИС. Классификации, правила и т.д. представляют основной интерес для системного аналитика, проектирующего АИС. При анализе предметной области он должен установить именно эти факты, обсудить их с заказчиком (пользователем) и описать.

Разработка любой АИС начинается с системного анализа предметной области, в результате которого создается **концептуальная схема** предметной области – представляющая собой непротиворечивую совокупность высказываний, истинных для данной предметной области, включая возможные состояния, классификации, законы, правила. Концептуальная схема предназначена для пользователей и разработчиков АИС. В том числе, концептуальная схема включает в себя единое описание содержимого информационной базы и описание того, какие действия – типа обновления и выборки – допустимы над этой информацией.

Концептуальное представление концентрирует внимание на смысле информации, именно концептуальная схема описывает это представление. На базе концептуальной схемы создаются *внешние схемы*, описания которых сосредоточены на том, как представлена информация для пользователя (т.е. описана основа для создания информационного языка пользователя). Описания внутреннего физического представления, данных в вычислительной системе содержатся во *внутренней схеме*.

Концептуальная схема, помимо того, что она является в первую очередь моделью предметной области на концептуальном уровне, полученной в результате системного анализа, играет также ключевую роль в проектировании баз данных.

2.1.3. Информационные база и процессор

Теоретической основой описания предметной области является интерпретируемая формальная система логики и теория формальных языков. То есть для описания предметной области требуется:

- 1) определение алфавита, в котором можно однозначно и автоматически распознать, входит ли данный символ в алфавит;
- 2) обеспечение конечного набора эффективных правил (алгоритмов), определяющих, какие строки символов являются правильно построенными и какие правильно построенные строки следует рассматривать как предложения;
- 3) обеспечение конечного набора эффективных правил, определяющих, является ли данное предложение аксиомой;
- 4) обеспечение конечного набора эффективных правил, определяющих выводимость данного предложения из исходного множества предложений;
- 5) обеспечение конечного набора правил интерпретации, приписывающих каждому предложению значение таким образом, что предложения интерпретируются как утверждения о предметной области – однозначно истинное или ложное.

Под *аксиомой* понимается любое замкнутое предложение, утверждаемое в качестве таковой авторитетным источником.

Аксиомы и правила вывода выбираются так, чтобы:

каждая аксиома интерпретировалась как истинное утверждение о предметной области;

каждое предложение, непосредственно выводимое из множества предложений, интерпретируемых как истинные утверждения о предметной области, само интерпретировалось как истинное утверждение о предметной области.

Этот процесс представляет собой точное воспроизведение обычного процесса дедуктивного вывода, выраженного в терминах формальной системы, для того чтобы можно было применять для информационных систем ЭВМ.

Требования 1 – 4 относятся к синтаксису или грамматике концептуальной схемы и информационной базы. Требование 5 связано с семантикой концептуальной схемы и информационной базы.

Подходящая формальная система для описания предметной области может, в принципе, предъявлять минимальные требования к той части концептуальной схемы, которая встроена в информационный процессор. То есть *встроенная минимальная* концептуальная схема может быть ограничена до необходимого минимума. Но на практике многие лингвистические конструкции, которые теоретически не нужны, могут быть включены в фактический информационный процессор по соображениям эффективности и удобства использования. Известно, что человеческое восприятие предметной области, а также передача этого восприятия другому лицу могут существенно отличаться у разных людей.

Проектировщик информационной системы должен иметь возможность выразить концептуальную схему в терминах, соответствующих рассматриваемой предметной области и восприятию ее пользователем. В частности, не должно налагаться никаких ограничений на сущности, наличие которых допускается в предметной области, или на свойства, которыми они могут обладать. Проектировщик информационной системы ограничен только требованиями основных принципов логики, встроенных в информационный процессор, т.е. требованиями на поддерживания *непротиворечивости*.

Учитывая также, что концептуальная схема должна быть простой в использовании и понимании для разнообразных пользователей, необходим механизм добавления лингвистических конструкций произвольной сложности (макроконструкций) в формальную систему, для того чтобы пользователи могли с ней взаимодействовать на любом требуемом уровне абстракции.

Концептуальная схема и информационная база абсолютно неизменны, пока нечто не произведет изменения в них. Это нечто называют информационным процессом. То есть, развивая дальше концепцию АИС, вводят понятие **информационной системы**, состоящей из концептуальной схемы, информационной базы и информационного процессора.

Информационный процессор производит изменения в информационной базе или концептуальной схеме только при получении сообщения. Сообщение содержит информацию и/или выражает команды. Сообщения исходят из некоторой части реального мира, называемой средой, которая может быть или не быть внешней по отношению к предметной области. При получении соответствующего сообщения, содержащего команду, информационный процессор может также выдавать информацию, имеющуюся в концептуальной схеме и информационной базе.

Пользователями информационной системы могут быть ЭВМ или

другие системы, а также люди. Пользователь – это некто или нечто, посылающий команды и сообщения в информационную систему и получающий сообщения от информационной системы.

В АИС информационный процессор является вычислительной системой или какой-либо ее частью. В традиционных, неавтоматизированных ИС роль информационного процессора играет человек, при условии, что он не нарушает установленных правил. С другой стороны, вычислительные системы могут действовать как пользователи информационной системы. Примером может служить сеть ЭВМ, обменивающихся сообщениями друг с другом. Если каждая имеет набор правил, независимый от других, тогда каждая играет роль пользователя других АИС.

2.1.4. Систематизация основных понятий статического описания предметной области

Далее проведена систематизация основных понятий, используемых при разработке, описании и применении концептуальных схем и информационных баз для АИС. Некоторые из этих понятий встречаются в литературе по базам данных, иногда с совершенно разным смыслом. Поэтому здесь даются краткие, точные, хотя и неформальные определения понятий, согласованных, где это возможно, со значением слов в естественном языке (например, понятие “реальный мир” должно пониматься в обычном языковом смысле).

Будем считать, что можно выделить такую часть реального мира, которую можно описать на некотором выбранном нами точно и формально определенном языке. Все объекты, которые мы наблюдаем или существование которых мы допускаем в этой выбранной части мира, называются *сущностями*.

Сущность – любой конкретный или абстрактный объект, включая связи между объектами. Например, если мы выбираем часть мира, в которой производится регистрация автомобилей, тогда сущностями являются:

автомобиль “Лада”;

дата 2 апреля 1999 года;

и т.д.

Абстрактная сущность – связь между другими сущностями, например, “право владения машиной “Лада” г-ном Серовым”.

Высказывание – возможное состояние сущностей, о котором можно утверждать или отрицать, что такое состояние имеет место. Высказывание может относиться к одной сущности, несколькими

отдельным сущностям, к группе сущностей. Различают высказывания о фактическом состоянии отдельных сущностей и высказывания о том, какое поведение сущностей является допустимым или возможным. Слова “правило” и “ограничение” относятся в основном к допустимым высказываниям.

С помощью описания высказываний, *предложений* обсуждают сущности и их состояние, то есть обмениваются информацией о сущностях посредством описаний высказываний, относящихся к ним. **Предложение** – лингвистический объект, выражающий высказывание, а **лингвистический объект** – грамматически допустимая конструкция языка. Лингвистические объекты сами могут считаться сущностями.

Предложения состоят из термов и предикатов. **Терм** – лингвистический объект, ссылающийся на сущность. **Предикат** – лингвистический объект, аналогичный глаголу, что-либо сообщающий о сущности (или сущностях), к которой относится терм в предложении.

Лингвистические объекты могут не играть никакой другой роли в описаниях, кроме использования в качестве каких-либо имен, тогда их называют лексическими объектами или именами.

Лексический объект или имя – (простой) лингвистический объект, который используется только для ссылки на сущность. Как правило, лексический объект состоит из одного или нескольких существительных.

Необходимо подчеркнуть, что не запрещено и, более того, часто весьма удобно, чтобы одна сущность имела несколько лексических объектов, с ней связанных, тогда эти лексические объекты являются синонимами.

Синонимы – различные термы, ссылающиеся на одну сущность. В основном, нет также запрета на то, чтобы несколько идентичных лексических объектов были связаны с различными сущностями. Тогда эти лексические объекты являются *омонимами* – одинаковыми термами, ссылающимися на различные сущности.

Теперь можно дать еще одно конструктивное определение предметной области. *Предметная область* – это все сущности, которые были, есть или когда-либо могут быть и которые представляют интерес для целей разработки АИС.

Предметная область по-другому может быть названа областью возможных сущностей, которая ограничивается возможными сущностями, представляющими интерес для целей разработки АИС.

Класс сущностей – все возможные сущности в предметной области, для которых выполняется данное высказывание. Каждый класс сущностей полностью определяется его возможными членами.

Естественно, любая сущность может быть членом многих классов, поэтому классы, вообще говоря, могут пересекаться.

Высказывание, определяющее класс, может быть произвольной сложности. Например:

класс Изготовителей Автомобилей состоит из всех возможных сущностей, производящих автомобили;

класс Владельцев Автомобилей состоит из всех возможных сущностей, относящихся к классам Изготовителей Автомобилей, Гаражей или Лиц, владеющих автомобилем.

Сами классы являются сущностями и им могут быть даны имена.

Тип сущности – это высказывание, устанавливающее, что сущность является членом определенного класса сущностей, подразумевая также существование такого класса сущностей. Например, предложения “Сущность является Изготовителем Автомобилей (тип)” и “Сущность принадлежит к Изготовителям Автомобилей (класс)” передают в точности одну и ту же информацию.

На тип можно ссылаться посредством имени типа. Довольно часто в таких случаях в качестве имени типа используется имя (существительное) в единственном числе, тогда как в качестве имени класса используется множественное число.

С понятием типа часто связывают понятие *экземпляр* – отдельная сущность, для которой выполняется определенное высказывание по поводу ее типа, то есть принадлежащая к определенному классу сущностей.

При проектировании информационных систем понятие класса и типа используются, в частности, для установления совокупностей необходимых высказываний.

Кроме необходимых высказываний, зафиксированных в концептуальной схеме, в некотором пространстве сущностей могут выполняться другие высказывания. Описание этих дополнительных высказываний составляет **информационную базу** – совокупность предложений, согласованных друг с другом и с концептуальной схемой, выражающих высказывания, отличающиеся от необходимых высказываний и выполняющиеся в некотором пространстве сущностей.

Совокупность предложений, составляющих концептуальную схему и информационную базу, вместе описывают все высказывания, относящиеся к некоторому пространству сущностей, и, тем самым, дают описание пространства высказываний для этого пространства сущностей. Предполагается, что эти высказывания выполняются для этого пространства сущностей, воспринимаемого как “реальность”. Следовательно, эта совокупность предложений, составляющих концептуальную схему и информационную базу, должна обязательно

быть непротиворечивой, если она претендует на то, чтобы быть истинным описанием этих высказываний.

Таким образом, пространство сущностей состоит в точности из тех конкретных и абстрактных объектов – сущностей, на которые можно сослаться с помощью термов в предложениях, содержащихся в информационной базе и концептуальной схеме, вместе взятых. Вполне возможно описание одной предметной области или одного определенного пространства сущностей в более, чем одной концептуальной схеме и информационной базе. Но предполагается, что обычно только одна информационная база одновременно будет частью одной информационной системы.

2.1.5. Описание динамики в предметной области АИС

Информационная база и концептуальная схема со временем изменяются, отражая изменения в выбранной части мира, составляющей предметную область, так как в информационной базе и концептуальной схеме должны быть только предложения, истинные в этой предметной области. Изменяться могут:

- сущности, появляющиеся или исчезающие в выбранной части;
- сущность, изменяющая свое состояние или связи с другими сущностями;
- классификация сущностей или какие-либо правила или ограничения, относящиеся к изменяющимся сущностям;
- выбранная часть мира расширяется или сокращается.

Все такие изменения могут повлечь за собой изменения и информационной базы, и концептуальной схемы. Хотя влияние первых двух видов изменений может ограничиться только информационной базой, последние два вида наверняка вызовут также изменения и в концептуальной схеме. *Манипулирование информацией* в информационной базе или концептуальной схеме осуществляется с помощью *элементарных действий*. Определено три вида элементарных действий: *вставка, удаление и выборка*.

Вставка – это добавление предложения к информационной базе или концептуальной схеме. Другие предложения, не являющиеся логически выводимыми до вставки, могут стать таковыми и, таким образом, становятся выводимой частью информационной базы или концептуальной схемы. Заметим, что логически выводимое предложение само по себе фактически не вставляется.

Удаление – ранее вставленное предложение удаляется из информационной базы или концептуальной схемы. Любое выводимое

предложение, которое не может быть выведено без исключенного предложения, больше не будет выводимым, т.е. не будет больше выводимой частью информационной базы или концептуальной схемы.

Заметим, что выводимое предложение могло быть одновременно вставлено явно. В этом случае, исключение другого предложения, от которого существенно зависит выводимость данного предложения, не приводит к автоматическому удалению явно вставленного предложения. Исключение этого другого предложения только сделает невозможным выведение явно вставленного предложения.

Последнее элементарное действие **выборка** – это поиск и выдача предложения, которое было вставлено в информационную базу или концептуальную схему, или логически выводимо из других предложений в информационной базе или концептуальной схеме.

Выборка выводимого предложения из информационной базы или концептуальной схемы возможна только в случае, если АИС знает, как вывести это предложение из других доступных или выводимых предложений в концептуальной схеме и информационной базе.

Допустимы комбинации элементарных действий, предназначенные для получения определенного результата. Такие сочетания определяются следующим образом.

Действие – одно или несколько элементарных действий, которые как одно целое изменяют совокупность предложений на другую совокупность предложений в информационной базе или концептуальной схеме и/или осуществляют поиск и выдачу совокупности предложений в информационной базе или концептуальной схеме. Типичным примером действия является замена определенного предложения другим, т.е. удаление, за которым следует вставка. Поскольку этот класс действий встречается часто, удобно определить его как особый вид действия – модификация.

Для исключения недопустимых действий и управления ими необходимо определить для действий правила и ограничения. Поэтому вводится определение действий, рассматриваемых как атомарные “единицы исполнения”. *Допустимое действие* – действие, удовлетворяющее установленным правилам или ограничениям. Допустимое действие изменяет непротиворечивую совокупность предложений в информационной базе или концептуальной схеме на другую непротиворечивую совокупность предложений и/или выбирает непротиворечивую совокупность предложений, присутствующих в информационной базе или концептуальной схеме.

Определенные допустимые действия могут изменять предположительно непротиворечивую, но в действительности “неверную”

совокупность предложений, не соответствующую действительности, в непротиворечивую и верную. Такие допустимые действия потребуются *для корректировки разрушенных информационных баз или концептуальных схем*. Таким допустимым действиям разрешается игнорировать некоторые правила о допустимых или требуемых последовательностях состояний совокупностей предложений. Например, если ошибочно указано, что некий человек женат, то для изменения этой информации на указание, что этот человек одинок, может потребоваться такое специальное допустимое действие.

Элементарное действие вызывается элементарной командой информационной системы. *Элементарная команда* – это приказ, или сигнал для запуска выполнения элементарного действия.

Действие и допустимое действие вызываются командой информационной системы. *Команда* – приказ или сигнал для запуска выполнения действия или допустимого действия. Но если допустимость действия будет нарушена, в ответ на команду действие может быть отвергнуто. Описание команд и действий дается на специальном языке, в виде *командных предложений* – лингвистических объектов, выражающих команду или элементарную команду.

Необходимо также иметь средства для записи комбинации элементарных действий и ее идентификации как единого целого, определяющего неделимое действие или допустимое действие. Такое описание действия – это лингвистический объект, описывающий действие или допустимое действие.

Взаимодействие между средой и информационной системой осуществляется посредством *сообщений* – совокупности одного или нескольких предложений и/или командных предложений, используемых для обмена информацией между средой и информационной системой. Сообщения обрабатываются информационным процессором информационной системы. *Информационный процессор* – механизм, который в ответ на команду выполняет действие над концептуальной схемой и информационной базой.

Информационный процессор распознает, относятся ли сообщения, полученные из среды, к данному языку. Сообщения, не относящиеся к данному языку, отбрасываются. Правильные сообщения могут выражать изменение в проблемной области, или требовать извлечения одного или нескольких предложений, имеющих в концептуальной схеме или информационной базе, или выводимых из присутствующих в них предложений.

2.2. Входные языки автоматизированных информационных систем

2.2.1. Структура входных языков обработки текстов запросов

В разделе 2.1 были рассмотрены формальные обоснования для создания эффективных языков пользователя автоматизированных информационных систем. Все **формализованные информационные языки (ФИЯ)** – это языки, созданные на базе естественных языков путем наложения ограничений на их лексику и грамматику, а также путем применения специальных обозначений для элементов этих языков. ФИЯ предназначены для описания и обработки данных в ИС. Языки, которые служат для составления предписаний на ввод, обновление, поиск, обобщение, редактирование и выдачу информации называются **входными языками АИС**. Среди входных языков обычно выделяют следующие:

язык описания данных (ЯОД) – формальный язык со специальными конструкциями, которые используются для описания схем баз данных;

язык описания хранения данных – язык, предназначенный для описания физического уровня представления данных;

язык манипулирования данными (ЯМД) – язык, предназначенный для формулирования запросов на поиск, обмен данными между прикладной программой и базой данных, а также для расширения языка программирования либо как самостоятельный язык.

В общем случае используется термин **язык запросов (справочный язык)** – язык, обеспечивающий взаимодействие конечного пользователя с информационной системой. Далее будет рассмотрена структура учебного входного языка, который содержит все основные, стандартные компоненты и атрибуты входных языков АИС.

Как уже указывалось, входные языки информационных систем служат для составления предписаний на ввод, обновление, поиск, обобщение, редактирование и выдачу информации. Они должны иметь в своем составе средства, позволяющие обозначать следующие элементы предписаний:

- идентификационные признаки пользователей (авторов предписаний);
- признаки массивов (файлов) информации, к которым производится обращение;
- операции над массивами;
- исходные данные для операций (вводимые или запрашиваемые сведения);

- имена программ и комплексов программ, реализующих операции над массивами;
- адреса, по которым следует направлять результаты выполнения операций;
- редакционные признаки выдаваемой информации;
- разделительные признаки для обозначения границ предписаний и их структурных элементов.

В описываемом учебном входном языке все предписания на обработку информации начинаются словом НАЧАЛО и заканчиваются словом КОНЕЦ. В промежутке между этими разделительными признаками могут записываться те компоненты, которые пользователь считает необходимым включить в состав предписания. Компоненты имеют следующие названия:

- | | |
|--------------|---------------|
| 1) АБОНЕНТ, | 7) ВЫБРАТЬ, |
| 2) ЗАДАЧА, | 8) ПРИСВОИТЬ, |
| 3) МАССИВ, | 9) ВЫДАТЬ, |
| 4) ВВЕСТИ, | 10) АДРЕС, |
| 5) ЗАТЕРЕТЬ, | 11) ФОРМА. |
| 6) НАЙТИ, | |

Обязательными компонентами всех предписаний являются компоненты, обозначенные словами АБОНЕНТ, ЗАДАЧА, МАССИВ, остальные – факультативные (но по крайней мере одна из компонент пп. 4-6 должна в предписании присутствовать).

В компоненте АБОНЕНТ указывается идентификационный код абонента (фамилия, название организации и т. п.). Запись этой компоненты имеет вид:

АБОНЕНТ – (ИВАНОВ)

Код абонента заключается в круглые скобки, а между левой скобкой и словом АБОНЕНТ проставляется тире. Слева и справа от тире оставляются пробелы.

В компоненте ЗАДАЧА указывается либо имя траектории решения задачи, либо (реже) ее описание, представляющее собой последовательность имен программ, участвующих в решении задачи. Каждая задача выполняет комплексную операцию по вводу, обновлению, поиску и обобщению информации. Имена траекторий решения задач и имена программ в их описаниях обозначаются русскими словами или аббревиатурами. В состав этих имен могут включаться также цифровые индексы.

Запись имени траектории решения задачи имеет вид:

ЗАДАЧА – (ВВОД)

Слово в скобках обозначает имя комплексной операции ввода информации. Запись полной траектории:

ЗАДАЧА-(ПГМ1=СИНКОН, ПГМ2=КОДСЛОВ, РЕЖИМ=О,
ПГМ3 = СИНАН, ПГМ4 = ФОРММОД, РЕЖИМ = Ф, ПГМ5 =АНКПРИНТ)

В описании траектории решения задачи справа от знаков равенства указываются имена программных модулей и режимы их работы, а слева — указатели функциональной роли элементов описания и порядковые номера программ. В приведенном примере указаны сокращенные имена программных модулей – синтаксический контроль (СИНКОН), кодирование слов (КОДСЛОВ), синтаксический анализ (СИНАН), формирование модуля (ФОРММОД), выдача информации на ПРИНТ в анкетной форме (АНКПРИНТ), а также обозначения режимов работы — обновление (О), формирование (Ф).

Обычно программные модули выполняются в порядке их следования. Для изменения последовательности выполнения программ в описании траекторий вводятся операторы безусловного и условного перехода. В операторе безусловного перехода, имеющем структуру типа:

ПЕРЕХОД = ПГМ_i [имя модуля],

справа от знака равенства указывается аббревиатура ПГМ и порядковый номер программы, к которой следует перейти, а в скобках – имя этой программы. В операторе условного перехода:

ЕСЛИ Р = 1 ТО ПГМ_i [имя модуля] ИНАЧЕ ПГМ_j [имя модуля]

указывается номер программы *i*, к которой следует перейти при Р = 1, и номер программы *j*, к которой следует перейти при Р = 0 (значение признака Р вырабатывается предшествующей программой).

Имена траекторий указываются в компоненте ЗАДАЧА в тех случаях, когда для основных режимов работы информационной системы заблаговременно составляются описания траекторий (создается библиотека траекторий) и есть возможность выбирать эти траектории по их именам.

В компоненте МАССИВ указываются имена или признаки массивов

(файлов) информации, к которым производится обращение. При этом предполагается, что все сведения, хранящиеся в информационной системе, распределены по различным массивам в зависимости от их тематической принадлежности, а описания массивов сведены в каталог (в массив описаний массивов). Предполагается также, что формализованные описания массивов имеют такую же структуру, что и описания других объектов, учитываемых в системе, и поиск в каталоге массивов ведется по тем же правилам, что и в остальных массивах.

Мы привели пример этого языка, разработанного в 70-х гг., чтобы, рассматривая современные входные языки, можно было убедиться в том, что исходные лингвистические принципы построения входных языков АИС не изменились. А это говорит, в свою очередь, о том, что выбранные принципы верны и проверены практикой.

В процессе развития АИС исторически в этих системах выделились следующие виды входных языков:

язык идентификаторов – язык, в качестве предложений которого используются идентификаторы – последовательности букв и цифр;

язык форматного типа – язык, предложения которого состоят из последовательности слов фиксированной длины;

язык позиционного типа – подтип форматных языков, отличающийся от них неопределенностью длины элементов;

языки анкетного типа – язык реализуется в виде комплекта анкет, и каждому типу запроса соответствует отдельная анкета;

нормированные языки – строго определенное подмножество естественного языка, предложения которого, по крайней мере по отношению предметной области, семантически однозначны.

2.2.2. Языковые средства документальных информационных систем

Документальные системы применяются для поиска документов по их обобщенным описаниям. В описания документов могут включаться их наименования, краткое изложение содержания, наименования организаций-издателей документов, время и место издания, учетные номера документов и т. п. Перед вводом в ЭВМ описания документов формализуются и представляются в виде последовательностей наименований признаков и их значений. Поиск документов ведется по их формализованным описаниям, а в качестве результатов поиска могут выдаваться любые фрагменты этих описаний.

Формализованные описания документов по своей структуре аналогичны формализованным описаниям любых других объектов, и для

их ввода в ЭВМ, обновления и поиска можно использовать стандартные входные языки АИС. Но в документальных системах часто возникает необходимость хранить в памяти ЭВМ наряду с формализованными и неформализованные описания документов (рефераты, аннотации), чтобы выдавать эти описания по запросам потребителей. Это может оказывать некоторое влияние на структуру входных языков этих систем.

Входных языков, предназначенных специально для документального поиска, очень много. Часто они отличаются друг от друга больше внешней формой, чем логическими возможностями. В большинстве из них используются:

- логические операторы типа И, ИЛИ, НЕ;
- отношения порядка типа =, <, > и их отрицания;
- различного рода синтагматические ограничения (например, требование, чтобы термины запроса входили в одно и то же предложение реферата).

Применяются также средства маскирования элементов запросов, их усечения и средства поиска по весовым критериям.

При использовании весовых критериев каждому термину запроса или документа присваивается числовой индекс – его “вес”. В процессе поиска веса терминов запроса, совпавшие с терминами документа, суммируются, а полученная сумма сравнивается с заданным порогом значимости. Если эта сумма превосходит пороговое значение или равна ему, то документ считается релевантным, если нет, то он отвергается. Веса найденных документов могут также использоваться для их ранжирования по степени релевантности.

Языковые средства документальных систем мы кратко рассмотрим на примере системы Асод, предназначенной для документального и фактографического поиска. Она позволяет вести поиск документов по ключевым словам, встречающимся в их описаниях, с применением логических операторов И, ИЛИ, НЕ и ограничителей, регламентирующих требуемое взаимное расположение ключевых слов в документе.

В качестве ограничителей применяются средства для указания допустимых расстояний между словами, встречающимися в одном и том же документе, для указания принадлежности слов к одному и тому же предложению или параграфу и др. Запрос может быть сформулирован в виде списка ключевых слов с приписанными им весами и с указанием порогового значения суммарного веса. Кроме того, к каждому ключевому слову может быть применена операция усечения и замены его списком синонимов.

При фактографическом поиске в системе Асод используются операторы сравнения, применяемые к форматированным полям

описаний документов. В качестве операторов сравнения, наряду с операторами $=$, $<$, $>$ и их отрицаниями, используются также оператор маскирования, оператор сканирования (проверка вхождения заданной последовательности символов в любое место анализируемого поля) и оператор проверки попадания численного значения поля в заданный интервал.

2.2.3. Проблемы информационного поиска

Предписание на поиск объектов, удовлетворяющих заданным условиям, формулируется в компоненте НАЙТИ (см.2.2.1). При формулировке условий поиска используются отношения $=$, $>$, $<$ и их отрицания $\text{НЕ } =$, $\text{НЕ } >$, $\text{НЕ } <$. Логические связи между поисковыми признаками указываются с помощью операторов И /ИЛИ (операторов конъюнкции и дизъюнкции).

Поисковый признак может состоять из:

одного наименования признака, обозначающего класс всех возможных значений этого признака (например, Y_i);

наименования признака с предшествующим отрицанием, обозначающего класс любых признаков, кроме заданного (например, $\text{НЕ } Y_i$);

наименования признака и его значения, соединенных отношениями $=$, $>$, $<$, $\text{НЕ } =$, $\text{НЕ } >$, $\text{НЕ } <$;

одного значения признака с предшествующим знаком отношения.

Таким образом, во входном языке допустимы следующие структуры признаков:

Y_i , $\text{НЕ } Y_i$, $Y_i = Z_i$, $Y_i \text{ НЕ } = Z_i$, $Y_i > Z_i$, $Y_i \text{ НЕ } > Z_i$, $Y_i < Z_i$,
 $Y_i \text{ НЕ } < Z_i$, $= Z_i$, $\text{НЕ } = Z_i$, $> Z_i$, $\text{НЕ } > Z_i$, $< Z_i$, $\text{НЕ } < Z_i$.

В компоненте НАЙТИ предусматривается возможность указания двух групп поисковых признаков, заключаемых в круглые скобки – имен объектов и любых других признаков объектов. Перед обеими группами может стоять знак отрицания НЕ. Группа имен объектов может иметь одну из следующих структур:

- 1) $O - (X_i)$;
- 2) $O - \text{НЕ } (X_i)$;
- 3) $O - (X_1 \text{ ИЛИ } X_2 \text{ ИЛИ } \dots \text{ ИЛИ } X_n)$;
- 4) $O - \text{НЕ } (X_1 \text{ ИЛИ } X_2 \text{ ИЛИ } \dots \text{ ИЛИ } X_n)$;
- 5) $O - (\text{ВСЕ})$.

Последняя структура используется в тех случаях, когда в поисковом предписании нет необходимости указывать имена объектов. Если нужно указывать несколько объектов, то их имена отделяются друг от друга знаком дизъюнкции.

Группа поисковых признаков объектов, не являющихся их именами, может быть представлена в дизъюнктивной или конъюнктивной нормальной форме. В первом случае это последовательность конъюнкций признаков, соединенных знаками дизъюнкции, во втором – последовательность дизъюнкций, соединенных знаками конъюнкции.

Число признаков в конституентах (в конъюнкциях и дизъюнкциях) может быть различным. Перед первым признаком дизъюнктивной нормальной формы ставится связка ИЛИ, а перед первым признаком конъюнктивной нормальной формы – связка И. Эти связки необходимы для указания приоритетов логических операций: в первом случае менее приоритетной операцией является операция ИЛИ, во втором случае – операция И. Если в компоненте НАЙТИ указываются имена искомых объектов и нет необходимости в указании дополнительных признаков этих объектов, то вместо них проставляется слово ВСЕ.

Обозначим перечень имен искомых объектов символом X , а логическую функцию поисковых признаков без учета знака отрицания перед скобкой символом \mathfrak{X} . Тогда компонента НАЙТИ может быть представлена в обобщенном виде как одна из следующих структур:

- 1) НАЙТИ: $O - (ВСЕ), P - (\mathfrak{X})$;
- 2) НАЙТИ: $O - (ВСЕ), P - НЕ (\mathfrak{X})$;
- 3) НАЙТИ: $O - (X), P - (ВСЕ)$;
- 4) НАЙТИ: $O - НЕ (X), P - (ВСЕ)$;
- 5) НАЙТИ: $O - НЕ (X), P - (\mathfrak{X})$;
- 6) НАЙТИ: $O - НЕ (X), P - НЕ (\mathfrak{X})$;
- 7) НАЙТИ: $O - (ВСЕ), P - (ВСЕ)$.

Здесь:

- первая запись означает – найти все объекты, удовлетворяющие условию \mathfrak{X} ;
- вторая – найти все объекты, не удовлетворяющие условию \mathfrak{X} ;
- третья – найти заданный перечень объектов;
- четвертая – найти объекты, не входящие в заданный перечень;
- пятая – найти среди объектов, не входящих в заданный перечень, объекты, удовлетворяющие условию \mathfrak{X} ;
- шестая – найти среди объектов, не входящих в заданный перечень, объекты, не удовлетворяющие условию \mathfrak{X} ;

– последняя запись применяется в тех случаях, когда нужно выдать информацию о всех объектах.

Другим специфичным случаем задания поисковых признаков является случай, когда в качестве их значений выступают классификационные коды и требуется искать информацию не по полным кодам, а по их фрагментам. Фрагменты кодов выделяются с помощью “масок”. Маски задаются путем замены цифр или групп цифр классификационных кодов на символы X (буквы “икс”) или группы таких символов. Символы X и группы этих символов могут проставляться на произвольных позициях кодов. Если, например, требуется вести поиск по классификационному признаку $Y = K_1 K_2 \dots K_n$ (где K_1, K_2, \dots, K_n – цифры), исключая позиции 1, 2, 5, 6, $n - 1$ его значения, то этот признак должен быть задан в виде записи:

$$Y = XXK_3K_4XXK_7 \dots K(n-2)XK_n.$$

Вместо знака равенства здесь могут также использоваться любые другие отношения ($NE =, >, NE >, <, NE <.$).

В компоненте **ВЫБРАТЬ** указываются перечни наименований признаков, которые следует выбирать из БД для объектов, удовлетворяющих условиям поиска. Компонента может иметь одну из следующих структур:

- 1) **ВЫБРАТЬ** – ($Y_1/Y_2/\dots/Y_n$);
- 2) **ВЫБРАТЬ** – (ВСЕ);
- 3) **ВЫБРАТЬ** – (ТО ЖЕ).

В первом случае задается перечень наименований выбираемых признаков объектов (в результатах поиска они располагаются в последовательности, указанной в компоненте **ВЫБРАТЬ**). Во втором случае выбираются все признаки объектов и располагаются в результатах поиска в том порядке, в котором они вводились в АИС. В третьем случае из БД выбираются признаки, упомянутые в условиях поиска.

Иногда возникает необходимость исключения из БД (затирания) сведений о группах объектов, определяемых логическими условиями поиска. Условия поиска формулируются в компоненте **НАЙТИ**, а признаки объектов, подлежащие исключению из массивов, – в компоненте **ВЫБРАТЬ**. Режим затирания сведений задается путем указания имени этого режима в компоненте **ЗАДАЧА**.

2.3. Сетевые языковые средства АИС

2.3.1. Основные понятия

Основные идеи современных информационных технологий базируются на концепции, согласно которой данные должны быть организованы в базы данных с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Эти базы данных создаются и функционируют под управлением специальных программных комплексов, называемых системами управления базами данных (СУБД).

Увеличение объема и структурной сложности хранимых данных, расширение круга пользователей информационных систем привели к широкому распространению наиболее удобных и сравнительно простых для понимания реляционных (табличных) СУБД. Для обеспечения одновременного доступа к данным множества пользователей, нередко расположенных достаточно далеко друг от друга и от места хранения баз данных, созданы сетевые многопользовательские версии СУБД. В них тем или иным путем решаются специфические проблемы параллельных процессов, целостности (правильности) и безопасности данных, а также санкционирования доступа.

Совместная работа пользователей в сетях с помощью унифицированных средств общения с базами данных возможна только при наличии стандартного языка манипулирования данными, обладающего средствами для реализации перечисленных выше возможностей. Таким языком стал SQL, разработанный в 1974 г. фирмой IBM для экспериментальной реляционной СУБД. В 1987 г. SQL стал международным стандартом языка баз данных, а в 1992 г. вышла вторая версия этого стандарта.

Языки манипулирования данными (ЯМД), созданные до появления реляционных баз данных и разработанные для многих систем управления базами данных (СУБД) персональных компьютеров, были ориентированы на операции с данными, представленными в виде логических записей файлов. Это требовало от пользователей детального знания организации хранения данных и достаточных усилий для указания не только того, какие данные нужны, но и того, где они размещены и как шаг за шагом получить их.

Непроцедурный язык SQL (Structured Query Language – структуризованный язык запросов) ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Особенность предложений этого языка состоит в том, что они ориентированы в большей степени на конечный результат обработки

данных, чем на процедуру этой обработки. SQL сам определяет, где находятся данные, какие индексы и даже наиболее эффективные последовательности операций следует использовать для их получения: не надо указывать эти детали в запросе к базе данных.

Непрерывный рост быстродействия, а также снижение энергопотребления, размеров и стоимости компьютеров привели к резкому расширению возможных рынков их сбыта, круга пользователей, разнообразия типов и цен. Естественно, что расширился спрос на разнообразное программное обеспечение. Борясь за покупателя, фирмы, производящие программное обеспечение, стали выпускать на рынок все более и более интеллектуальные и, следовательно, объемные программные комплексы. Приобретая (желая приобрести) такие комплексы, многие организации и отдельные пользователи часто не могли разместить их на собственных ЭВМ. Для обмена информацией и ее обобществления были созданы сети ЭВМ, где обобществляемые программы и данные стали размещать на специальных обслуживающих устройствах – файловых серверах.

СУБД, работающие с файловыми серверами, позволяют множеству пользователей разных ЭВМ (иногда расположенных достаточно далеко друг от друга) получать доступ к одним и тем же базам данных. При этом упрощается разработка различных автоматизированных систем управления организациями, учебных комплексов, информационных и других систем, где множество сотрудников (учащихся) должны использовать общие данные и обмениваться создаваемыми в процессе работы (обучения). Однако при такой идеологии вся обработка запросов из программ или с терминалов пользовательских ЭВМ выполняется на этих же ЭВМ. Поэтому для реализации даже простого запроса ЭВМ часто должна считывать из файлового сервера и (или) записывать на сервер целые файлы, что ведет к конфликтным ситуациям и перегрузке сети.

Для исключения указанных и некоторых других недостатков была предложена технология “Клиент-Сервер”, по которой запросы пользовательских ЭВМ (Клиент) обрабатываются на специальных серверах баз данных (Сервер), а на ЭВМ возвращаются лишь результаты обработки запроса. При этом, естественно, нужен единый язык общения с Сервером и в качестве такого языка выбран SQL. Поэтому все современные версии профессиональных реляционных СУБД (DB2, Oracle, Ingres, Informix, Sybase, Progress, Rdb) и даже нереляционных СУБД (например, Adabas) используют технологию “Клиент-Сервер” и язык SQL.

2.3.2. Основы языка SQL

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволило создать компактный язык с небольшим (менее 30) набором предложений. SQL может использоваться как интерактивный (для выполнения запросов) и как встроенный (для построения прикладных программ). В нем существуют:

- предложения определения данных (определение баз данных, а также определение и уничтожение таблиц и индексов);
- запросы на выбор данных (предложение SELECT);
- предложения модификации данных (добавление, удаление и изменение данных);
- предложения управления данными (предоставление и отмена привилегий на доступ к данным, управление транзакциями и другие).

Кроме того, он предоставляет возможность выполнять в этих предложениях:

- арифметические вычисления (включая разнообразные функциональные преобразования), обработку текстовых строк и выполнение операций сравнения значений арифметических выражений и текстов;
- упорядочение строк и (или) столбцов при выводе содержимого таблиц на печать или экран дисплея;
- создание представлений (виртуальных таблиц), позволяющих пользователям иметь свой взгляд на данные без увеличения их объема в базе данных;
- запоминание выводимого по запросу содержимого таблицы, нескольких таблиц или представления в другой таблице (реляционная операция присваивания);
- агрегирование данных: группирование данных и применение к этим группам таких операций, как среднее, сумма, максимум, минимум, число элементов и т.п.

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

INTEGER – целое число (обычно до 10 значащих цифр и знак);

SMALLINT – “короткое целое” (обычно до 5 значащих цифр и знак);

DECIMAL(p,q) – десятичное число, имеющее p цифр ($0 < p < 16$) и знак, с помощью q задается число цифр справа от десятичной точки ($q < p$, если $q = 0$, оно может быть опущено);

FLOAT – вещественное число с 15 значащими цифрами и целочисленным порядком, определяемым типом СУБД;

CHAR(n) – символьная строка фиксированной длины из n символов ($0 < n < 256$);

VARCHAR(n) – символьная строка переменной длины, не превышающей n символов ($n > 0$ и разное в разных СУБД, но не меньше 4096);

DATE – дата в формате, определяемом специальной командой (по умолчанию mm/dd/yy); поля даты могут содержать только реальные даты, начинающиеся за несколько тысячелетий до н.э. и ограниченные пятым – десятым тысячелетиями н.э.;

TIME – время в формате, определяемом специальной командой (по умолчанию hh.mm.ss);

DATETIME – комбинация даты и времени;

MONEY – деньги в формате, определяющем символ денежной единицы (\$, руб, ...) и его расположение (суффикс или префикс), точность дробной части и условие для показа денежного значения.

В некоторых СУБД дополнительно существует тип данных LOGICAL, DOUBLE и ряд других. СУБД INGRES предоставляет пользователю возможность самостоятельного определения новых типов данных, например плоскостные или пространственные координаты, единицы различных метрик, пяти- или шестидневные недели (рабочая неделя, где сразу после пятницы или субботы следует понедельник), дроби, графика и т.п.

2.3.3. Дополнительные возможности SQL

Безопасность информации означает защиту данных от несанкционированного раскрытия, изменения или уничтожения. SQL позволяет индивидуально защищать как целые таблицы, так и отдельные их поля. Для этого имеются две возможности:

- механизм представлений, используемый для скрытия засекреченных данных от пользователей, не обладающих правом доступа;
- подсистема санкционирования доступа, позволяющая предоставить указанным пользователям определенные привилегии на доступ к данным и дать им возможность избирательно и динамически передавать часть выделенных привилегий другим пользователям, отменяя впоследствии эти привилегии, если потребуется.

Обычно при установке СУБД в нее вводится какой-то идентификатор, который должен далее рассматриваться как идентификатор наиболее привилегированного пользователя – *системного администратора*.

Каждый, кто может войти в систему с этим идентификатором (и может выдержать тесты на достоверность), будет считаться системным администратором до выхода из системы. Системный администратор может создавать базы данных и имеет все привилегии на их использование. Эти привилегии или их часть могут предоставляться другим пользователям (пользователям с другими идентификаторами). В свою очередь, пользователи, получившие привилегии от системного администратора, могут передать их (или их часть) другим пользователям, которые могут их передать следующим и т.д.

Привилегии предоставляются с помощью предложения GRANT (предоставить), общий формат которого имеет вид:

GRANT привилегии ON объект TO пользователи;

здесь:

- “привилегии” – список, состоящий из одной или нескольких привилегий, разделенных запятыми, либо фраза ALL PRIVILEGES (все привилегии);
- “объект” – имя и тип объекта (база данных, таблица, представление, индекс и т.п.);
- “пользователи” – список, включающий один или более идентификаторов санкционирования, разделенных запятыми, либо специальное ключевое слово PUBLIC (общедоступный).
- К таблицам относятся привилегии SELECT, DELETE, INSERT и UPDATE [(столбцы)], позволяющие соответственно:
- считывать (выполнять любые операции, в которых используется SELECT);
- удалять, добавлять или изменять строки указанной таблицы (изменение можно ограничить конкретными столбцами).

Например, предложение:

GRANT SELECT, UPDATE (Владелец) ON Автомобиля TO boss;

позволяет пользователю, который представился системе идентификатором boss, использовать информацию из таблицы Автомобиля, но изменять в ней он может только значения столбца Владелец.

Если пользователь U_1 предоставил какие-либо привилегии другому пользователю U_2, то он может впоследствии отменить все или некоторые из этих привилегий. Отмена осуществляется с помощью предложения REVOKE (отменить), общий формат которого очень похож на формат предложения GRANT:

REVOKE привилегии ON объект FROM пользователя.

Например, можно отобрать у пользователя право изменения значений столбца Владелец:

REVOKE UPDATE (Владелец) ON Автомобиля FROM boss.

В SQL предусмотрена еще одна возможность, позволяющая пользователю обеспечивать безопасность своих данных – это механизм управления транзакциями.

Транзакция, или логическая единица работы – это в общем случае последовательность ряда таких операций, которые преобразуют некоторое непротиворечивое состояние базы данных в другое непротиворечивое состояние, но не гарантируют сохранения непротиворечивости во все промежуточные моменты времени.

Никто, кроме пользователя, генерирующего ту или иную последовательность SQL-предложений, не может знать о том, когда может возникнуть противоречивое состояние базы данных и после выполнения каких SQL-предложений оно исчезнет, т.е. база данных вновь станет актуальной. Поэтому в большинстве СУБД создается механизм обработки транзакций, при инициировании которого все изменения данных рассматриваются как предварительные до тех пор, пока пользователь (реже система) не выдаст предложения:

COMMIT (фиксировать), превращающее все предварительные обновления в окончательные (“зафиксированные”);

ROLLBACK (откат), аннулирующее все предварительные обновления.

Пользователь должен сам решать, включать ли механизм обработки транзакций и если включать, то где издавать COMMIT (ROLLBACK), т.е. какие последовательности SQL-предложений являются транзакциями.

Большинство СУБД позволяют любому числу транзакций одновременно осуществлять доступ к одной и той же базе данных, и в них существуют те или иные механизмы управления параллельными процессами, предотвращающие нежелательные воздействия одних транзакций на другие. Как правило, это механизм блокирования, главная идея которого достаточно проста. Если транзакции нужны гарантии, что некоторый объект (база данных, таблица, строка или поле), в котором она заинтересована, не будет изменен каким-либо непредсказуемым образом в течение требуемого промежутка времени, она устанавливает блокировку этого объекта. Результат блокировки заключается в том, чтобы изолировать этот объект от других транзакций и, в частности, предотвратить его изменение средствами этих транзакций. Для первой транзакции, таким образом, имеется возможность выполнять

предусмотренную в ней обработку, располагая определенными знаниями о том, что объект в запросе будет оставаться в стабильном состоянии до тех пор, пока данная транзакция этого пожелает.

Все предложения SQL, которые можно ввести с терминала, можно использовать также в прикладной программе.

Многие современные СУБД имеют собственные языки программирования, ряд которых включает в себя SQL. Другие работают с программами, написанными на одном из распространенных алгоритмических языков (Си, Паскаль или Фортран), в которые включаются предложения SQL. Для обмена информацией с частями программы, написанными на любых из этих языков, существуют специальные конструкции SQL, позволяющие работать с переменными и (или) отдельными строками таблиц. К исходным текстам языков программирования, которые включают в себя конструкции SQL, предъявляется ряд требований. Переменные включающего языка:

- могут появляться в предложениях манипулирования данными языка SQL только в определенных фразах и предложениях;
- должны иметь типы данных, совместимые с типами данных тех столбцов базы данных, с которыми они должны сравниваться, значения которых им должны быть присвоены или которым должны быть присвоены значения переменных;
- могут иметь имена, совпадающие с именами столбцов базы данных (система различает их по месторасположению в предложениях SQL или по специальному символу, устанавливаемому перед именем переменной, когда надо использовать ее значение).

После выполнения любого предложения SQL происходит обновление системной переменной SQLCODE (в нее заносится числовой индикатор состояния). Нулевое значение SQLCODE означает, что данное предложение выполнено успешно. Положительное значение означает, что предложение выполнено, но имела место некоторая исключительная ситуация. Например, значение +100 указывает, что не было найдено никаких данных, удовлетворяющих запросу. Наконец, отрицательное значение указывает, что имела место ошибка и предложение не выполнено. Поэтому за каждым предложением SQL в программе должна следовать проверка значения SQLCODE и должно предприниматься соответствующее действие, если это значение оказалось не таким, которое ожидалось.

Основная проблема “встраивания” предложения SELECT в программу заключается в том, что SELECT, как правило, порождает таблицу с множеством строк и столбцов, а включающий язык не обладает

хорошими средствами, позволяющими оперировать одновременно более чем одной записью (строкой). По этим причинам необходимо обеспечить своего рода мост между уровнем множеств языка SQL и уровнем записей включающего языка. Такой мост обеспечивают курсоры. Курсор состоит, по существу, из некоторого рода указателя, который может использоваться для просмотра множества записей. Поочередно указывая каждую запись в данном множестве, он обеспечивает возможность обращения к этим записям по одной одновременно.

В тех же приложениях, где надо отыскивать и обрабатывать множество подходящих записей из одной таблицы или совокупности таблиц базы данных следует использовать курсоры, позволяющие организовать последовательный доступ к строкам какой-либо таблицы (соединению таблиц).

Предложение:

```
DECLARE имя_курсора CURSOR  
FOR подзапрос
```

определяет имя курсора и связанный с ним подзапрос. С его помощью идентифицируется некоторое множество столбцов и строк указанной таблицы (совокупности таблиц), которое становится активным множеством для данного курсора. Курсор идентифицирует также позицию в этом множестве (сначала это позиция его первой записи). Активные множества всегда рассматриваются как упорядоченные.

Описанные с помощью DECLARE CURSOR множества используются рядом предложений SQL для удаления отмеченных строк (DELETE), их модификации (UPDATE) или присвоения значений перечисленных в SELECT столбцов переменным. Однако перед выполнением этих команд необходимо активизировать курсор, который в этот момент не открыт. Для этого используется предложение OPEN (OPEN имя_курсора).

Следует упомянуть еще два предложения, связанные с курсорами. Это предложение для дезактивации курсора (CLOSE имя_курсора) и предложение для уничтожения курсора (DROP CURSOR имя_курсора).

2.4. Технология гипертекста и интеллектуальные сетевые интерфейсы

Развитие локальных и глобальных сетей, массовое использование ЭВМ не только в профессиональных областях деятельности, но и в сфере досуга и развлечений оказали сильное влияние на развитие

информационных языков, с помощью которых пользователи получают доступ к информационным сетевым ресурсам. Наблюдается быстрая эволюция от традиционных средств доступа к данным к сложным видам интерфейсов пользователя.

Не менее важным стимулом для развития интерфейсов пользователя явилась возможность навигации по информационным узлам и WEB-серверам Internet. Технологии, основанные на **гипертексте** – способе организации информации и доступа к ней, при котором между различными текстами и (или) фрагментами текстов установлены связи, а выделение связи автоматически обеспечивает доступ к соответствующему тексту или фрагменту текста, значительно упрощают и делают более эффективным поиск в *неструктурированных* текстах.

Гипертекстовая система представляет информацию в виде некоторого графа, в узлах которого содержатся текстовые элементы (предложения, абзацы, страницы, целые статьи либо книги), а между узлами имеются связи, с помощью которых можно переходить от одного текстового элемента к другому. При этом информационные элементы гипертекста могут находиться на глобально распределенных сетевых серверах. **Информационные элементы гипертекста** – это объекты, которые создает и которыми манипулирует разработчик (пользователь). Эти объекты включают мысли, диаграммы, рисунки, идеи, обсуждения, аргументы, алгоритмы. Физическими представлениями объекта могут быть текст, битовое представление изображений, графика, звуки, мультимпликация, процессы и т.д.

Еще одним фактором, способствующим развитию информационных языковых средств обработки данных, стали **мультимедийные средства** – средства, позволяющие одновременно проводить операции с неподвижными изображениями, видеофильмами, анимированными графическими образами, текстом, речевым и звуковым сопровождением.

Технологии клиент-сервер и концепция открытых систем концентрируют в себе современные пользовательские интерфейсы и способствуют их развитию за счет предоставления все новых возможностей в обработке данных и знаний.

Под *клиентом* здесь понимается прикладная программа, которая оформляет запрос пользователя телекоммуникационной сети на получение сетевых услуг в соответствии с принятым сетевым протоколом. Клиент получает запрошенную услугу от сетевого *сервера* – прикладной программы, которая принимает запрос из сети от клиента на предоставление некоторой сетевой услуги и предоставляет клиенту эту услугу, если она входит в его компетенцию.

Открытые системы – это АИС различного назначения, размещенные в различных узлах телекоммуникационной сети. Их работу как единого интегрированного целого обеспечивает система отраслевых, государственных и международных стандартов в области информационных технологий, специфицирующих интерфейсы, услуги и поддерживающих форматы данных для достижения взаимодействия и переносимости приложений, данных и персонала.

Традиционные языковые интерфейсы, которые рассматривались до сих пор, “не понимают” содержания сообщения в момент осуществления преобразования. Им не требуется знаний ни о характере поведения пользователя, ни о внутренней структуре рабочих процессов, ни о данных, с которыми они манипулируют, а нужны только некоторые правила преобразований форматов.

Интерфейсы, описываемые далее, имеют несколько общих характеристик, которые позволяют называть их в некотором смысле *интеллектуальными*. Основная особенность состоит в том, что преобразования, включенные в интерфейс, до сих пор проводившиеся через механизм определений, должны осуществляться в контексте отображаемой предметной области. Такой интерфейс должен обладать некоторыми знаниями о “мире задачи”, в котором функционируют он и пользователь.

Люди постоянно приобретают и совершенствуют свои знания (свою “модель мира”). Они интерпретируют полученную информацию и словами выражают то, что хотят сказать в свете этой модели. Процесс, с помощью которого это делается, часто описывается в форме распознавания образов. Поступающая информация сопоставляется с образцами, содержащимися в модели мира, чтобы определить, какие из них пригодны, т.е. какая из интерпретаций предпочтительнее.

Вторая особенность интеллектуальных интерфейсов заключается в том, что они также используют форму распознавания образов для интерпретации входных сообщений от пользователя в свете системной модели мира. Возникают две проблемы:

- сам по себе механизм распознавания образов;
- обеспечение модели мира, которая приобретает и хранит образцы.

Эти возможности опираются на все время возрастающую мощность ресурсов ЭВМ и специальное оборудование. Требуется большая компьютерная мощность для обработки правил, используемых компьютерной системой при принятии даже простых решений (таких, как выявление бракованных битов в битовом изображении), которые человек делает почти подсознательно в одно мгновение.

Третьей особенностью интеллектуальных интерфейсов является проблема представления образцов в системной модели, а также получение новых образцов и совершенствование старых. Компьютер должен сам обучаться на основании опыта и создавать такую же модель мира, как и у его пользователя.

Использование таких дополнительных средств, как речевые и визуальные представления, увеличивает полосу пропускания необходимых средств связи, и, следовательно, скорость, с которой должна передаваться информация.

Речевой вывод обеспечивается с помощью специальных чипов, преобразующих текст в речь. Речь состоит из серии *фонем*, или значащих звуков (см. юниту 1), аппаратным базисом является генератор тонов, который может синтезировать некоторый диапазон подходящих звуков. Проблема состоит в том, чтобы обеспечить набор правил, которые указывали бы, какая фонема должна быть использована для данного текстового слога. Ряд таких правил был разработан лингвистами и используется в современных интерфейсах. Лучшие результаты получаются, если используются записанные, а не синтезированные звуки. Чтобы обеспечить восприимчивость с адекватным временем отклика, требуется процессор значительной мощности.

Речевой ввод и техническое зрение (ввод изображений от телекамеры) являются примерами сопоставления образцов. Они расширяют возможности считывания документов. Образцы, с которыми проводится сопоставление, крайне многочисленны и сложны. Речевой ввод требует распознавания из непрерывных речевых волн правильной последовательности фонем и их преобразования в текстовые слоги в зависимости от контекста. В речевом выводе нет проблемы определения того, где кончается одно слово и начинается другое. Текст удобно разбит пробелами и символами пунктуации. Непрерывная речь не так удобно разбита паузами (паузы часто могут делаться и внутри слов).

Большинство серийных устройств речевого ввода распознают лишь ограниченный диапазон отдельных произношений.

Интеллектуальные интерфейсы требуют использования модели мира, чтобы обеспечивать контекст для интерпретации на вводе и построения фраз при выводе. Их миром являются прикладные задачи, которые они должны поддерживать. Каждый пользователь имеет свою собственную модель этого мира приложений и использует свою модель для формирования входного сообщения в систему и интерпретации ее выходных сообщений. Требование естественности диалога является фактическим требованием того, чтобы системная и пользовательская модели мира приложений были сопоставимы. Если эти модели

радикально отличаются, то несопоставимые представления приведут к ошибкам взаимопонимания, которые могут сильно затруднить взаимодействие пользователя с системой.

Все интерфейсы, даже самые простые, содержат модель мира своего приложения. Методы структурирования диалога, используемые фразы и соглашения содержат предложения о мире своего приложения. В любом интерфейсе эта модель отражает взгляд на приложение, принятый проектировщиком при разработке системы. Эта модель *статическая*, и пользователи, если они хотят успешно взаимодействовать с системой, должны подгонять свои модели, сопоставляя их с ней.

Большинство из более или менее удачных интерфейсов человек-компьютер потерпели провал из-за того, что модель, на которой проектировщик основывал систему, существенно отличается от модели, сложившейся у пользователя. Чтобы компенсировать этот недостаток, большое значение придается методам системного анализа, которые помогают проектировщику создавать модель диалога, сходную с моделью, сложившейся у пользователя.

В телекоммуникационных сетях в роли интерфейса к базе данных (знаний) может выступать Web браузер.

Базы знаний обычно содержат огромное количество информации, поэтому поиск нужной информации становится экстремально критической функцией. Большинство современных методов поиска включают:

- инструментальные средства;
- средства интеллектуального поиска;
- визуальные модели.

Инструментальные средства поиска используются для информационной навигации в Internet. Все они могут быть адаптированы и для корпоративных сетей. Иногда разрабатываются центральные хранилища интерфейсов (*карты знаний*), которые связываются с знаниями. Пользователи могут выбирать карту для навигации при поиске знаний, хранящихся в многочисленных базах знаний, причем не зная точно, в какой именно базе данных следует искать.

С помощью средств интеллектуального поиска ведут поиск нужных данных в информационной среде Internet или корпоративных сетей. Например, изучаются интересы пользователей по наборам классифицированных ими сообщений или документов, а также используются эвристические методы для сбора дополнительных, более точных сведений. Базируясь на синтаксисе сообщений, определяют ключевые фразы, которые помогают понять задачу пользователя. Например, один из *эвристических* подходов предполагает *извлечение любых слов*,

целиком состоящих из заглавных букв, таких как ISDN, так как это, вероятно, соответствует представлению аббревиатур или технических имен. Другой эвристический метод заключается в том, чтобы не обращать при этом внимания на слова, если они используются для усиления, например “NOT”. Еще один способ – включение перечислений, нумерованных списков, секций заголовков и описаний диаграмм. Все это позволяет находить документы, предугадывая запросы пользователя.

Среди новых тенденций в области проектирования систем поиска информации можно выделить метод визуальных моделей (визуализации знаний).

Система создает интеллектуальный контекст (словарь-справочник), используя метаданные, выделенные из исходных документов, включая структурированную информацию в БД и целевых документах, или неструктурированные данные в офисных документах и Web-страницах. Для неструктурированных документов выполняется *лингвистический анализ* и автоматически помечаются проанализированные документы. Сервер интеллектуального контекста анализирует помеченную информацию, идентифицирует взаимосвязи между документами и строит многомерное информационное пространство, используя специальный язык пометок (Information Space Markup Language). Пользователь “летит” сквозь информационное пространство, манипулируя мышью.

Например, средство визуализации VizControl предлагает несколько форматов визуализации. Каждый из них развивает метод “фокус контекст”, когда интересующие пользователя данные выводятся на передний план и в тоже время сохраняется структура даже очень больших наборов данных. Одно из таких инструментальных средств, гиперболический браузер, использует гиперболическую геометрию для расширения информационного пространства при работе с иерархическими структурами, которые расширяются экспоненциально с увеличением глубины. Таким образом, гиперболический браузер может показать 1000 узлов в окне размером 600х600 пикселей, в центре которого высвечивается текст довольно большого объема (для сравнения, условный 2D-браузер может показать на экране лишь около 100 узлов). Пользователь перемещается по информационному пространству, щелкая “мышью” на узле или передвигая указатель “мыши” по гиперболической плоскости.

2.5. Поиск и обработка данных в глобальных сетях

Интернет представляет собой глобальную сеть компьютеров, обменивающихся данными с помощью общего языка (протокола). Работа Интернет похожа на работу международной телефонной системы –

отсутствует единый владелец или управляющий, но все объединены и работают как в одной большой сети. Служба WWW, или просто Web, имеет графический, удобный для поиска документов интерфейс. Эти документы, а также связи между ними, образуют пространство Web.

Файлы или страницы Web связаны между собой. Чтобы перейти к другой странице, необходимо установить указатель на специальный текст или графический объект, называемый *гиперссылкой*, и нажать кнопку мыши. Страницы Web могут находиться на компьютере, расположенном в любом месте земного шара. При подключении к Web вы получаете доступ к всемирной информационной службе.

Гиперссылки являются текстом или графическими объектами, содержащими адреса Web. Если установить указатель на гиперссылку и нажать кнопку мыши, произойдет переход к странице, размещенной на соответствующем узле Web. Каждая страница, включая основную страницу узла Web, имеет свой адрес универсального указателя ресурсов URL, например <http://www.microsoft.com/home.htm>. Адрес URL состоит из имени компьютера, содержащего необходимую страницу, и полного пути к этой странице.

Корпоративной сетью считается любая выделенная сеть, использующая коммуникационные стандарты Интернет, а также сервисные приложения, обеспечивающие доставку данных пользователям сети. Например, предприятие может создать сервер Web для публикации объявлений, производственных графиков и других служебных документов. Служащие осуществляют доступ к необходимым документам с помощью средств просмотра Web.

Серверы Web корпоративной сети могут обеспечить пользователям услуги, аналогичные услугам Интернет, например работу с гипертекстовыми страницами (содержащими текст, гиперссылки, графические изображения и звукозаписи), предоставление необходимых ресурсов по запросам клиентов Web, а также осуществление доступа к базам данных.

Для управления средством просмотра на его панели инструментов содержится необходимый набор функций и команд. Рядом с панелью инструментов находится адресная панель, содержащая адрес текущей страницы Web. Чтобы перейти к новой странице Web, следует ввести в адресное поле ссылку URL на эту страницу. Кроме того, к новой странице можно перейти щелчком гиперссылки.

Чтобы создать узел Web на компьютере, можно воспользоваться службами узла Web. С помощью служб узла Web совместный доступ к документам Web выполняется так же просто, как и к файлам в корпоративной сети (доступ осуществляется средством просмотра

Web). Для обработки запросов средства просмотра используется протокол обмена гипертекстом HTTP (Hypertext Transfer Protocol) и протокол передачи файлов FTP (File Transfer Protocol). С помощью службы FTP пользователи могут не только загружать файлы с узла Web, но и копировать файлы на него.

С помощью служб узла Web на персональном компьютере можно создать личный узел Web. Личные узлы Web полезны для предоставления информации членам рабочей группы, а также для публикации личных основных страниц в корпоративной сети. Чаще всего личный узел применяется для:

- публикации личной основной страницы;
- публикации служебных документов, например расписаний, записок и отчетов;
- создания приложений для запуска на узле Web.

Служба Web является интерактивной системой запросов и ответов. Средство просмотра Web запрашивает данные путем отправки запроса URL на сервер Web. Сервер Web посылает ответ в виде страницы HTML (специальный язык). Страница HTML может быть статической (уже присутствующей и отформатированной), динамической (сервер создает ее в ответ на запрос пользователя), а также содержать список доступных файлов и папок узла Web.

Каждая страница корпоративной сети или Интернета имеет свой адрес URL. Средство просмотра Web запрашивает страницу путем отправки запроса URL на сервер Web. Сервер использует полученный адрес для поиска и отображения страницы. Адрес URL состоит из имени протокола, домена и пути к необходимому ресурсу. *Протокол* – это метод соединения, применяемый для получения доступа к необходимой информации, например протокол HTTP. Имя домена – это имя компьютера в системе DNS (Domain Name System); путь к ресурсу – путь к файлам, размещенным на компьютере. В следующей таблице представлены возможные варианты адресов URL:

Протокол	Имя домена	Путь к ресурсу
http://	www.microsoft.com	/backoffice
https:// (HTTP с защитой)	www.company.com	/catalog/orders.htm
gopher://	gopher.college.edu	/research/astronomy/index.htm
ftp://	orion.bureau.gov	/stars/alpha quadrant/starlist.txt

В адресе URL могут содержаться инструкции, которые должен выполнить сервер Web перед возвратом страницы. Эти инструкции записываются после указания пути. Сервер Web передает данные для обработки программе или сценарию, а затем возвращает результат в виде страницы Web.

В ответ на запрос средства просмотра сервер Web возвращает страницу HTML.

Пользователи могут составлять запросы без указания имени файла. Для узла Web или отдельного каталога можно создать документ, отображаемый при обработке таких запросов, или настроить сервер на просмотр содержимого каталога. Если в каталоге отсутствует документ, отображаемый по умолчанию, и включен режим отображения содержимого каталога, пользователю возвращается страница HTML со списком файлов и папок каталога (гипертекстовый вариант представления проводника или диспетчера файлов). Пользователь может загрузить файл, щелкнув его в списке.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *Составьте логическую схему базы знаний по теме юниты.*

2. Изобразите на схеме классификацию формальных грамматик.

3. Задан автомат:

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

где:

$V = \{0, 1\}$ - входной алфавит;

$F = \{0, 1\}$ - выходной алфавит).

$Q = \{q_0, q_1, q_2\}$ - алфавит состояний автомата;

δ - функция переходов;

γ - функция выходов;

$q_0 \in Q$ - начальное состояние автомата.

Функции перехода и выхода заданы в таблице предписаниями:

δ	γ
$(q_0, 0) \mapsto q_1$	$(q_0, 0) \mapsto 0$
$(q_0, 1) \mapsto q_0$	$(q_0, 1) \mapsto 1$
$(q_1, 0) \mapsto q_2$	$(q_1, 0) \mapsto 1$
$(q_1, 1) \mapsto q_1$	$(q_1, 1) \mapsto 0$
$(q_2, 0) \mapsto q_0$	$(q_2, 0) \mapsto 1$
$(q_2, 1) \mapsto q_2$	$(q_2, 1) \mapsto 0$

На вход подается последовательность 0, 1, 0, 1.

Нарисуйте для данного случая диаграмму состояний автомата.

4. Для задания 3 составьте таблицу состояний автомата.

5. Составьте блок-схему прохождения исходного текста программы через транслятор (описание в п. 1.4.2, 1.4.3).

6. Изобразите на схеме классификацию входных языков АИС (раздел 2.2).

7. Составьте таблицу основных функций (возможностей) языка SQL.

8. Создайте с помощью редактора Word 7 гипертекст на базе 3-х файлов:

файл, где будут храниться только номера заданий из данного текста;

файл, где будут храниться только тексты заданий из данного текста;

файл, где будут храниться только тексты ответов на задания из данного текста.

ЛИНГВИСТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

ЮНИТА 2

ОСНОВЫ ТЕОРИИ ФОРМАЛЬНЫХ ЯЗЫКОВ И ЯЗЫКИ ОБРАБОТКИ ДАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Редактор Н.В. Друх

Оператор компьютерной верстки Д.В. Федотов

Изд. лиц. ЛР № 071765 от 07.12.1998

Сдано в печать

НОУ "Современный Гуманитарный Институт"

Тираж

Заказ
