**Таблица переходов детерминированного конечного автомата**

Наряду с представлением графом функция переходов ДКА может быть задана таблицей, что, безусловно, больше подходит для программной реализации конечного автомата (табл. 1).

Рассмотрим таблицу переходов ДКА А12, распознающего язык идентификаторов.

Таблица 1. Таблица переходов конечного автомата А12;

В таблице записано состояние, в которое переходит автомат, находясь в состоянии, соответствующем данной строке таблицы, и получив входной символ, обозначенный в соответствующем столбце.

Конечное состояние автомата В помечено в таблице жирным шрифтом.

Наряду с состояниями N и В предусмотрено дополнительное состояние Е — состояние ошибки. Это сделано для того, чтобы функция переходов была определена для всех возможных пар символ-состояние. Иначе переход из состояния N при поступлении на вход символа b был бы не определен.

Попав в состояние Е, автомат остается в нем. Состояние Е не является конечным. На практике при программной реализации, кроме символов входного алфавита, потребуется, скорее всего, определить реакции автомата и на любые другие символы, которые, очевидно, должны переводить автомат в состояние Е.

**Программная реализация автоматного**

**Распознавателя**

В листинге 1 приведен эскиз программы, моделирующей работу детерминированного конечного автомата. Эта программа и является универсальным распознавателем (синтаксическим анализатором) автоматных языков.

 В ней есть лишь некоторые условности: предполагается, что состояния обозначаются заглавными латинскими буквами (S — начальное состояние), а входной алфавит — малые латинские буквы. Не конкретизированы также способы считывания символов и проверки их наличия на входе, а также то, как автомат реагирует на принятие или непринятие входной цепочки — эти части программы записаны по-русски.

*Листинг 1. Универсальный распознаватель автоматных языков*

**type**

tCondition = (S, А, В, С, ..., Е, ...) ; { Состояния }

tAlpha = 'а'..'z'; { Алфавит }

tJump = array [tCondition, tAlpha] of tCondition;

{ Таблица переходов }

tFinish = set of tCondition;

/ Тип множества конечных состояний }

**var**

Cond : tCondition; { Текущее состояние }

Ch : tAlpha; { Входной символ }

Р : tJump; { Функция переходов }

Fin : tFinish; { Конечные состояния }

 { Здесь задаются значения Р и Fin }

Cond := S;

**while** Есть символы **do begin**

Читать(Ch);

Cond := P[Cond, Ch]

**end;**

**if** Cond **in** Fin **then**

Цепочка принята

**else**

Цепочка не принята

**Дерево разбора в автоматной грамматике**

Говоря о задаче синтаксического анализа, мы сводили ее к построению дерева разбора. Между тем, в предыдущих разделах на роль распознавателя автоматных языков предложен конечный автомат, который дерево не строит. Нет ли здесь противоречия?

Нет. Дерево разбора цепочки в автоматной грамматике может быть однозначно построено, если известна последовательность переходов конечно-автоматного распознавателя. То есть распознающий автомат не только дает ответ на вопрос о принадлежности цепочки языку, но и позволяет выявить структуру цепочки.

Структура при этом представлена последовательностью переходов автомата.

Рассмотрим, какой вид имеет дерево разбора терминальной цепочки в автоматной грамматике. Из трех видов правил автоматной грамматики правила вида

 A →a

и

А → ε

 могут быть использованы в процессе порождения цепочки ровно один раз, после чего процесс порождения заканчивается. Все остальные подстановки выполняются по правилам вида

A→аВ.

Каждая такая подстановка приводит к появлению в дереве новой внутренней вершины, помеченной нетерминалом. Ее левая дочерняя вершина помечается терминалом (рис. 12).



*Рис. 12. Дерево разбора в автоматной грамматике*

Если для грамматики типа 3 построен ДКА, то входная цепочка однозначно определяет последовательность проходимых автоматом состояний и дерево вывода.

**Пример автоматного языка**

Рассмотрим язык целых чисел со знаком. Примеры правильно записанных чисел:

177 +22 -1 0 02

Построим конечный автомат, который распознает этот язык. Зададим этот автомат с помощью диаграммы переходов. Эта диаграмма будет служить также и формальным определением самого языка.

Начальное состояние автомата обозначим S (рис. 13). Находясь в этом состоянии, автомат ожидает символ, с которого может начинаться запись числа.



Рис.13. Конечный автомат, распознающий целые числа со знаком

Это знаки «+», «-» и цифры. Соответственно, из состояния S должны исходить дуги, помеченные этими символами.

Десять дуг, помеченных цифрами от 0 до 9, заменим одной, пометив ее символом «*ц*».

 После того как принят знак числа, автомат должен перейти в состояние, в котором он ожидает первую цифру.

Обозначим такое состояние А. Таким образом, переходы по символам «+» и «-» ведут из состояния S в состояние А.

Если, находясь в начальном состоянии S, автомат получил цифру, он должен перейти в состояние (обозначим его В), в котором могут быть приняты последующие цифры, если они есть.

Из состояния А при получении цифры автомат также переходит в состояние В.

Состояние В следует пометить как конечное, поскольку переход в это состояние означает, что на вход автомата поступила правильная запись целого числа.

Дуга, помеченная символом «*ц*», ведущая из состояния В в него же, позволяет автомату принять вторую и последующие цифры числа, если они есть.

По диаграмме переходов можно записать и грамматику, порождающую язык целых чисел со знаком. Каждой дуге соответствует правило. Конечные состояния порождают правила с пустой цепочкой в правой части.

S→+A|-A| *ц* B

А→ *ц* В

В→ *ц* В|ε

На практике, как уже говорилось, приходится учитывать возможность поступления на вход автомата не только символов входного алфавита, но и любых других символов.

В этой ситуации можно предполагать, что из любого состояния исходит дуга, ведущая в состояние ошибки Е (рис. 14). Кроме того, удобно считать, что входная цепочка всегда завершается специальным символом «конец текста», который обозначают ⊥



Рис. 14. Конечный автомат с состоянием ошибки и дополнительным конечным состоянием

При использовании символа ⊥ к автомату следует добавить состояние К, которое будет единственным конечным, и в которое из «бывших» конечных состояний будут направлены дуги, помеченные ⊥.