Не составило бы труда заполнить для полученного автомата таблицу переходов и использовать универсальный распознаватель (см. листинг 1), моделирующий поведение ДКА. Но здесь мы рассмотрим другую возможность.

Пользуясь диаграммой переходов (см. рис. 13 и 14), напишем программу, которая ведет себя подобно конечному автомату, но не моделирует его поведение напрямую.

В начале работы, то есть находясь в исходном состоянии, программа считывает первый символ входной цепочки и проверяет его допустимость:

Читать(Символ);

**if** **not** (Символ **in** [' + ', ' -', '0'.. '9']) then

Ошибка

Будем считать, что обращение к процедуре Ошибка останавливает работу распознавателя.

Далее (по-прежнему оставаясь в начальном состоянии) проверяем, какой именно из допустимых символов поступил на вход.

Если это знак, читаем следующий символ, переходя к состоянию А и предусматривая действия, которые автомат выполняет, находясь в этом состоянии:

**else if** Символ **in** [ ' + ', ' - ' ] **then begin**

Читать(Символ);

{ Состояние А }

**if** Символ **in** ['0'..'9'] **then**

Читать(Символ)

**else**

Ошибка

**end**

**else** { Символ - цифра (в состоянии S) }

Читать(Символ);

{ Состояние В }

Если в состоянии S поступил символ цифры, считывается следующий входной символ и происходит переход к состоянию В.

После выполнения приведенного фрагмента программа или останавливается по причине ошибки, или приходит в состояние, аналогичное состоянию В конечного автомата.

Находясь в состоянии В, автомат должен принимать все поступившие на вход цифры, оставаясь при этом в состоянии В:

{ Состояние В }

**while** Символ **in** [ '0'.. '9'] **do**

Читать(Символ);

Выход из цикла происходит, если очередной считанный символ — не цифра.

Если это символ ⊥ — «конец текста», то входная цепочка закончена и автомат переходит в состояние К (см. рис. 14), принимая входную цепочку.

Если цикл прекращен по причине поступления символа, отличного от цифры и ⊥, автомат переходит в состояние ошибки:

**if** Символ = ‘⊥’ **then**

Цепочка принята

**else**

Ошибка;

Как видим, распознаватель автоматного языка, каким и является наша программа, можно написать, не прибегая к прямому моделированию поведения конечного автомата с использованием таблицы переходов.

Такой подход может иметь свои преимущества. Технология программирования распознавателя оказывается довольно простой: программа пишется по диаграмме переходов ДКА, которая исполняет роль схемы алгоритма.

**Синтаксические диаграммы автоматного языка**

В построенной программе-распознавателе состояния конечного автомата не фигурировали явно. Они просто соответствовали некоторым точкам программы.

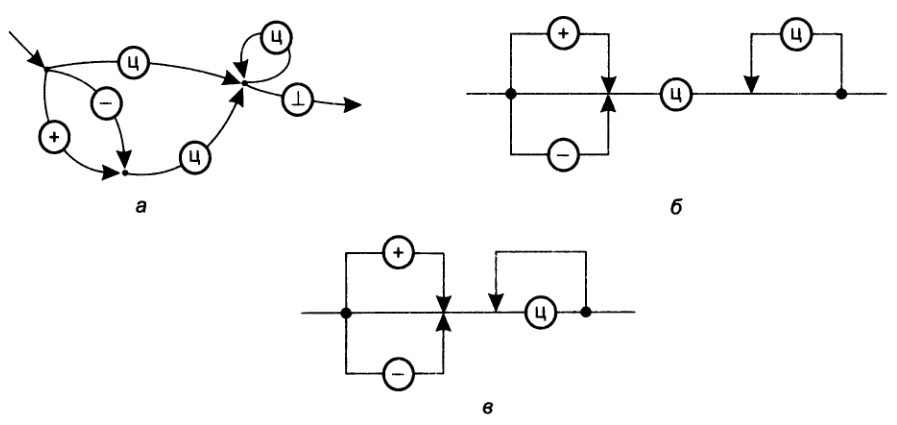
В противоположность состояниям, символы, которыми помечены дуги диаграммы переходов, явно используются в распознавателе.

Устраним из диаграммы переходов обозначения состояний (заглавные буквы и кружки). Те места диаграммы, где были состояния автомата, превращаются в точки ветвления и соединения дуг.

Терминальные символы, отмечающие переходы - дуги, наоборот разместим в кружках на этих дугах.

Результат такого преобразования для диаграммы переходов автомата, распознающего целые числа, или, что то же самое, для графа автоматной грамматики, порождающей целые числа, показан на рис. 15, а.

Полученный граф носит название синтаксической диаграммы автоматного языка или автоматной грамматики.



*Рис.15 Синтаксические диаграммы грамматики целых чисел*

Синтаксическая диаграмма может использоваться для тех же целей, что грамматика и граф переходов автомата, то есть для порождения и для распознавания предложений языка.

Любой путь от входа диаграммы (соответствует начальному состоянию автомата) к ее выходу (конечному состоянию К) порождает цепочку символов, являющуюся правильным предложением языка (сентенцией грамматики). Решение же задачи распознавания сводится к поиску такого пути от входа к выходу диаграммы, который соответствует заданной цепочке.

По сравнению с порождающей грамматикой и конечным автоматом синтаксические диаграммы гораздо наглядней и лучше подходят для спецификации языка при его конструировании. Очень удобна диаграмма в роли схемы алгоритма при написании синтаксического анализатора.

Используя другую манеру начертания дуг, переместив символ «ц» на дугу, ведущую в бывшее состояние В и отказавшись от явного изображения конца текста, получим более простую диаграмму, задающую синтаксис целых со знаком (рис. 15, б).

Упрощение диаграммы можно продолжить и дальше (рис. 16, в), сохраняя ее эквивалентность исходной (рис.15, а). Однако, при программировании анализатора на Паскале эта последняя диаграмма не удобней предыдущей.

Перепишем анализатор целых чисел, пользуясь синтаксической диаграммой, показанной на рис. 15, б, как схемой алгоритма (листинг 2).

Чтобы избавиться от условностей, будем считать, что программа имеет доступ к глобальной переменной *ch*, которая хранит текущий символ.

Чтение следующего входного символа выполняет процедура *Nextch*, которая помещает прочитанное значение в переменную *ch*.

Считается, что константа EOT (от End Of Text) обозначает конец текста. Реакция на ошибку возложена на процедуру *Error*, которая выдает сообщение об ошибке и останавливает работу программы-распознавателя.

В случае принятия входной цепочки никакого специального сообщения не предусматривается.

*Листинг 2. Распознаватель целых со знаком*

NextCh; { Прочитать первый символ }

**if** Ch **in** ['+', '-' ] **then**

NextCh;

**if** Ch **in** ['0'..'9'] **then**

NextCh

**else**

Error;

**while** Ch **in** [' 0 ' . . ' 9 ' ] **do**

NextCh;

i**f** Ch <> EOT **then**

Error;

Следует отметить ряд важных черт получившейся программы.

Она состоит из нескольких частей, разделенных в листинге пустыми строками.

Первая и последняя части, как нетрудно понять, должны присутствовать всегда: перед началом анализа надо получить первый символ, а по завершении — убедиться, что в момент, соответствующий выходу из диаграммы, входная цепочка исчерпана.

Три других части строго соответствуют структуре синтаксической диаграммы (см. рис. 15, б).

На диаграмме выделяются три последовательно соединенных участка, и программа содержит три последовательно записанных и выполняемых фрагмента.

Первый из них (if-then) проверяет наличие (необязательного) знака.

Второй (if-then-else) — наличие обязательной цифры. Цикл while (который, как известно, может не выполниться ни разу) соответствует циклу на диаграмме, задающему последовательность из нуля или более цифр.

Программа-распознаватель может быть написана по синтаксической диаграмме автоматной грамматики с использованием формальных приемов.