Лекція 5. Организация лексического анализа

Назначение и необходимость фазы лексического анализа

Лексический анализ - первая фаза процесса трансляции, предназначенная для группировки символов входной цепочки в более крупные конструкции, называемые лексемами. С каждой лексемой связано два понятия:

- ***класс лексемы***, определяющий общее название для категории элементов, обладающих общими свойствами (например, идентификатор, целое число, строка символов и т. д.);
- ***значение лексемы***, определяющее подстроку символов входной цепочки, соответствующих распознанному классу лексемы. В зависимости от класса, значение лексемы может быть преобразовано во внутреннее представление уже на этапе лексического анализа. Так, например, поступают с числами, преобразуя их в двоичное машинное представление, что обеспечивает более компактное хранение и проверку правильности диапазона на ранней стадии анализа.

В принципе, задачи, решаемые сканером, можно возложить на синтаксический анализатор. Но такой подход неудобен по следующим причинам:

1. Замена, цепочек символов, представляющих элементарные конструкции языка, делает внутренне представление программы более удобным для дальнейшего анализа распознавателем. Последний манипулирует не отдельными символами, а законченными элементарными конструкциями, что облегчает их общее восприятие и дальнейший семантический анализ. Кроме этого, при построении лексем может осуществляться простейшая семантическая обработка. Например, преобразование и проверкачисловых констант.
2. Уменьшается длина программы, поступающей в синтаксический анализатор, за счет устранения из нее несущественных для дальнейшего анализа пробелов, комментариев, игнорируемых символов. Уменьшение размера текста повышает производительность распознавателей, особенно для многопроходных компиляторов.
3. Один и тот же язык программирования может иметь различные внешние представления элементарных конструкций. Поэтому, наличие нескольких лексических анализаторов, порождающих на выходе одно и то же множество лексем, позволяет не переписывать синтаксический анализатор. Написать новый лексический анализатор намного проще, чем синтаксический. Примером языка, имеющего различные входные наборы символов, является PL/1. Для него определены 48-символьный и 60-символьный алфавиты.
4. Лексический анализатор использует более простые, по сравнению с синтаксическим анализатором, методы разбора. Следовательно, на одних и тех же цепочках и при выполнении разбора одних и тех же конструкций его производительность будет выше.
5. Блок лексического анализа естественным образом вписывается в иерархическую структуру компилятора, что тоже немаловажно при системном подходе к разработке трансляторов.

Транслитератор

Несмотря на то, что лексический анализатор обрабатывает входную цепочку, удобнее на его вход подавать не просто отдельные символы, а символы, сгруппированные по категориям. Поэтому, перед лексическим анализатором осуществляется дополнительная обработка, сопоставляющая с каждым символом его класс, что позволяет сканеру манипулировать единым понятием для целой группы символов, иногда достаточно большой. Устройство, осуществляющее сопоставление класса с каждым отдельным символом, называется **транслитератором**. Наиболее типичными классами символов являются:

- ***буква*** - класс, с которым сопоставляется множество букв, причем необязательно только одного алфавита;
- ***цифра*** - множество символов, относящихся к цифрам, чаще всего от 0 до 9;
- ***разделитель*** - пробел, перевод строки, возврат каретки перевод формата;
- ***игнорируемый*** - может встречаться во входном потоке, но игнорируется и поэтому просто отфильтровывается из него (например, невидимый код звукового сигнала и другие аналогичные коды);
- ***запрещенный*** - символы, который не относятся к алфавиту языка, но встречается во входной цепочке;
- ***прочие*** - символы, не вошедшие ни в одну из определенных категорий.

Пара, класс символа и его значение, поступают на вход сканера, который выбирает для анализа тот ее элемент, который наиболее удобен в данный момент. Например, при анализе идентификатора удобнее манипулировать понятием "буква", тогда как при анализе действительного числа можно сразу смотреть значение буквы **"E"** или **"e"**, означающей начало порядка. Следует также отметить, что во всех дальнейших рассуждениях будем считать, что понятия "буква" и "цифра" являются терминалами, как полученные в транслитераторе до начала лексического анализа. В предыдущей трактовке эти понятия считались нетерминальными символами.

Грамматики и распознаватели для лексического анализа

Лексические анализаторы обычно используются для распознавания элементарных конструкций, синтаксическое описание которых можно осуществить с применением праволинейных грамматик. Это наиболее простой класс грамматик, эквивалентных по мощности детерминированным конечным автоматам. Использование праволинейных грамматик для построения автоматов широко освещается в литературе [Ахо78, Льюис]. Простота лексического анализа заключается также в том, что лексемы - это единственные нетерминалы, используемые в ходе распознавания. Даже, если в грамматике, используемой для описания лексем, существуют промежуточные нетерминалы, они исчезают при построении конечного автомата, преобразуясь в его состояния. Входными символами конечного автомата являются терминальные символы и их классы символов, формируемые транслитератором.

Связь между диаграммой Вирта и конечным автоматом

Построение конечного автомата можно осуществить также с использованием ряда грамматик с левой рекурсией и диаграмм Вирта. На практике, вместо праволинейной грамматики, удобнее

использовать диаграммы Вирта. Они намного нагляднее при описании правил. Кроме этого, между диаграммами Вирта, не содержащими нетерминалов, и конечными автоматами существует однозначное соответствие. Практически это два эквивалентных способа представления одной модели, которая одновременно может служить как механизмом порождения, так и механизмом распознавания.

Конечный автомат, эквивалентный диаграмме Вирта, состоящей только из терминалов, строится следующим образом:

1. Начальная дуга диаграммы преобразуется в начальное состояние конечного автомата.
2. Конечная дуга диаграммы образует заключительное состояние конечного автомата.
3. Выходы отдельных дуг, соединяющих символы, и точки ветвления остальных дуг диаграммы образуют множество остальных состояний конечного автомата.
4. Конечные состояния диаграммы являются допускающими состояниями конечного автомата.
5. Терминальный символ диаграммы Вирта, расположенный между дугами, преобразуется в связь между соответствующими состояниями, помеченную входным символом, равным этому терминалу.
6. Связи, обеспечивающие переход в допускающие состояния, помечаются множеством оставшихся символов. Это множество не пересекается с множеством символов, обеспечивающих другие переходы из текущего в другие состояния (обозначены как "прочие").

Построение конечного автомата в соответствии с данными правилами можно рассмотреть на примере описания идентификатора. Диаграмма Вирта для идентификатора и полученный по ней конечный автомат приведены на рис. 4.1.

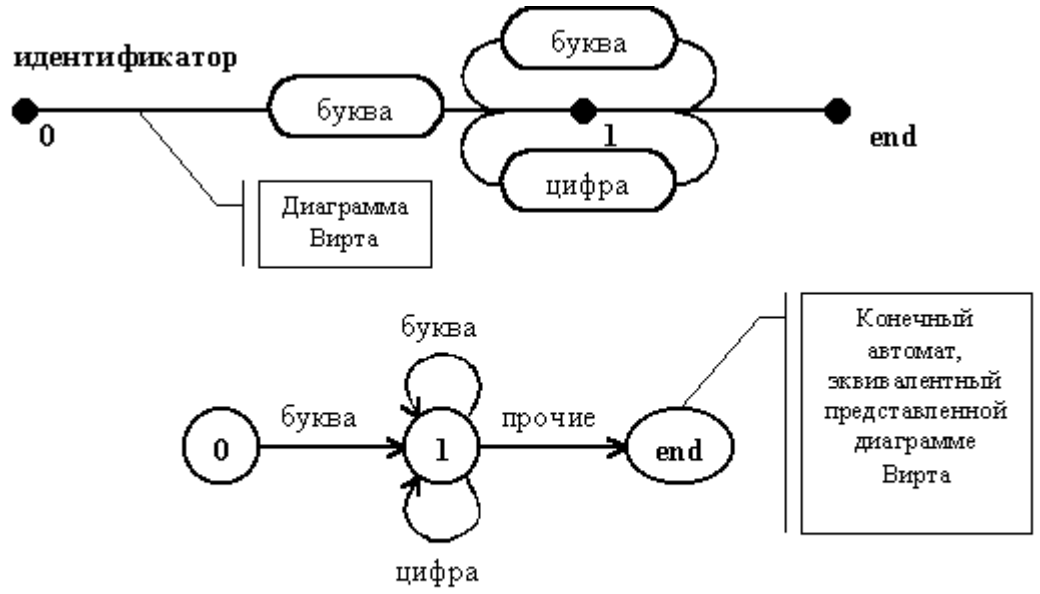


Рис. 4.1. Преобразование диаграммы Вирта в эквивалентный конечный автомат.

Для наглядности на дугах отмечены точки, соответствующие состояниям конечного автомата. Они обозначены теми же номерами, что и состояния. Заключительное состояние отмечено

меткой **"end"**. Следует еще раз отметить, что буква и цифра на диаграмме рассматриваются как терминалы, так как эти понятия формируются транслитератором.

Диаграммы Вирта могут использоваться и для того, чтобы минимизировать общее представление правил, что соответствует методам минимизации, применяемым к конечным автоматам. Эта минимизация осуществляется за счет объединения одинаковых подграфов диаграмм. Пример возможной минимизации понятия "идентификатор представлен на рис. 4.2.

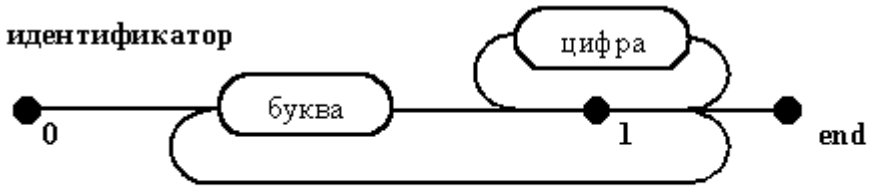


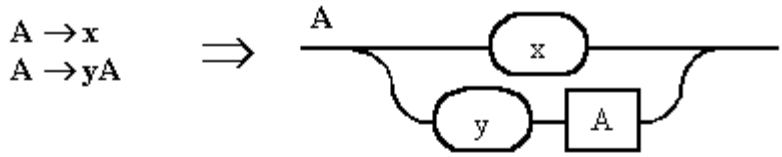
Рис. 4.2. Минимизированная диаграмма Вирта, задающая идентификатор.

Следует отметить, что не всегда минимизация диаграммы Вирта ведет к минимизации соответствующего ей конечного автомата. Приведенный рисунок показывает, что, несмотря на сокращение общего числа символов, разметка, соответствующая состояниям конечного автомата, осталась неизменной.

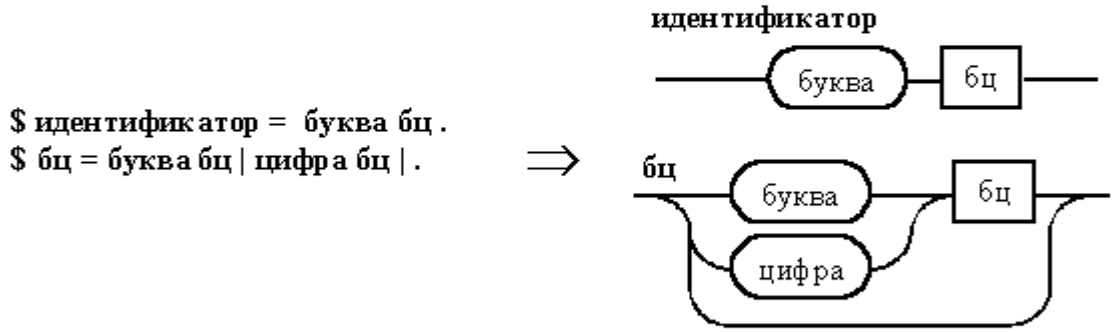
Наличие однозначного и легко понятного соответствия между диаграммами Вирта и конечными автоматами позволяет не строить последние, а переходить к написанию программы лексического анализатора непосредственно от диаграмм Вирта. Это сокращает технологическую цепочку, связывающую формальное описание элементов языка программирования и их программную реализацию.

Связь между диаграммами Вирта и праволинейными грамматиками. Преобразование правой рекурсии в итерацию

Использование диаграмм Вирта для непосредственного написания программы позволяет говорить о том, что, при наличии праволинейной грамматики, можно не строить конечный автомат, а преобразовать исходную грамматику в диаграммы Вирта. Непосредственное представление праволинейной грамматики диаграммами Вирта не вызывает каких-либо проблем (рис. 4.3), но полученный результат не позволяет переходить к написанию программы из-за наличия в правых частях правил нетерминалов.



а) преобразование простой грамматики



б) преобразование идентификатора

Рис. 4.3. Непосредственное преобразование праволинейных грамматик в диаграммы Вирта.

В ряде случаев от нетерминалов можно избавиться простой подстановкой. Но, если в правиле имеется рекурсия, обычную подстановку осуществить не удастся. Однако можно воспользоваться тем, что в праволинейной грамматике допускается только правая рекурсия, которая легко заменяется итерацией.

Замена правой рекурсии на итерацию в диаграммах Вирта производится следующим простым способом:

- дуга, **входящая** в рекурсивно определяемый нетерминал, замыкается своим **выходом** на самое начало правила, образуя, таким образом, цикл;
- сам нетерминал, оказавшийся в подвешенном состоянии вычеркивается, а выходившая из него дуга убирается.

Пример, иллюстрирующий, общие принципы преобразования правой рекурсии, представлен на рис. 4.4.

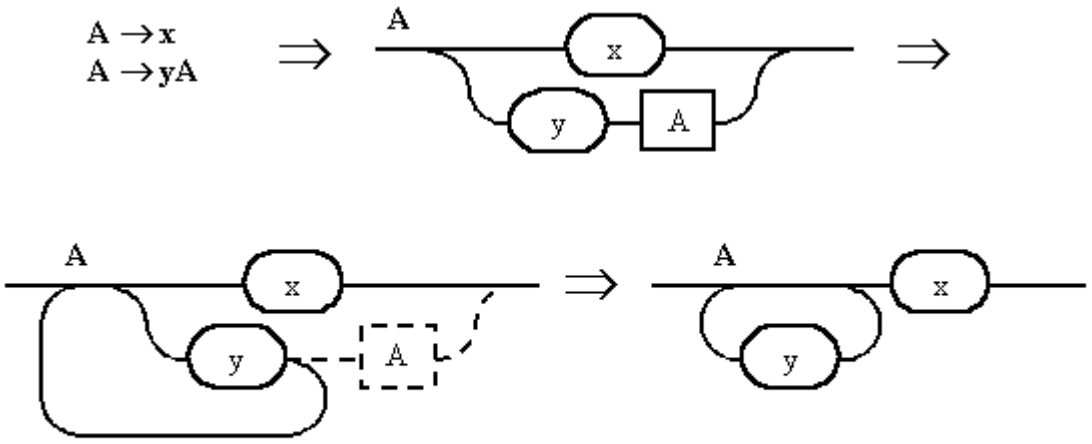


Рис. 4.4. Преобразование праволинейной грамматики в итеративную диаграмму Вирта.

Он показывает возможность такого преобразования, исходя из следующих индуктивных рассуждений. Нетерминал **A** на первом шаге порождает терминальную цепочку **x** или терминальную цепочку **y**, за которой следует нетерминал **A**, из которого на следующем шаге можно породить терминальную цепочку **x** или терминальную цепочку **y**, за которой следует нетерминал **A**, из которого... Эти бесконечные рассуждения на тему "У попа была собака..." можно заменить одной структурированной фразой: Нетерминал **A** порождает терминальную цепочку **y**, повторяющуюся ноль или большее число раз, за которой следует терминальная цепочка **x**. На рис. 4.5 приведено преобразование в итеративную диаграмму Вирта правил праволинейной грамматики, описывающих идентификатор. После избавления от лишних нетерминалов мы получаем хорошо знакомый результат.

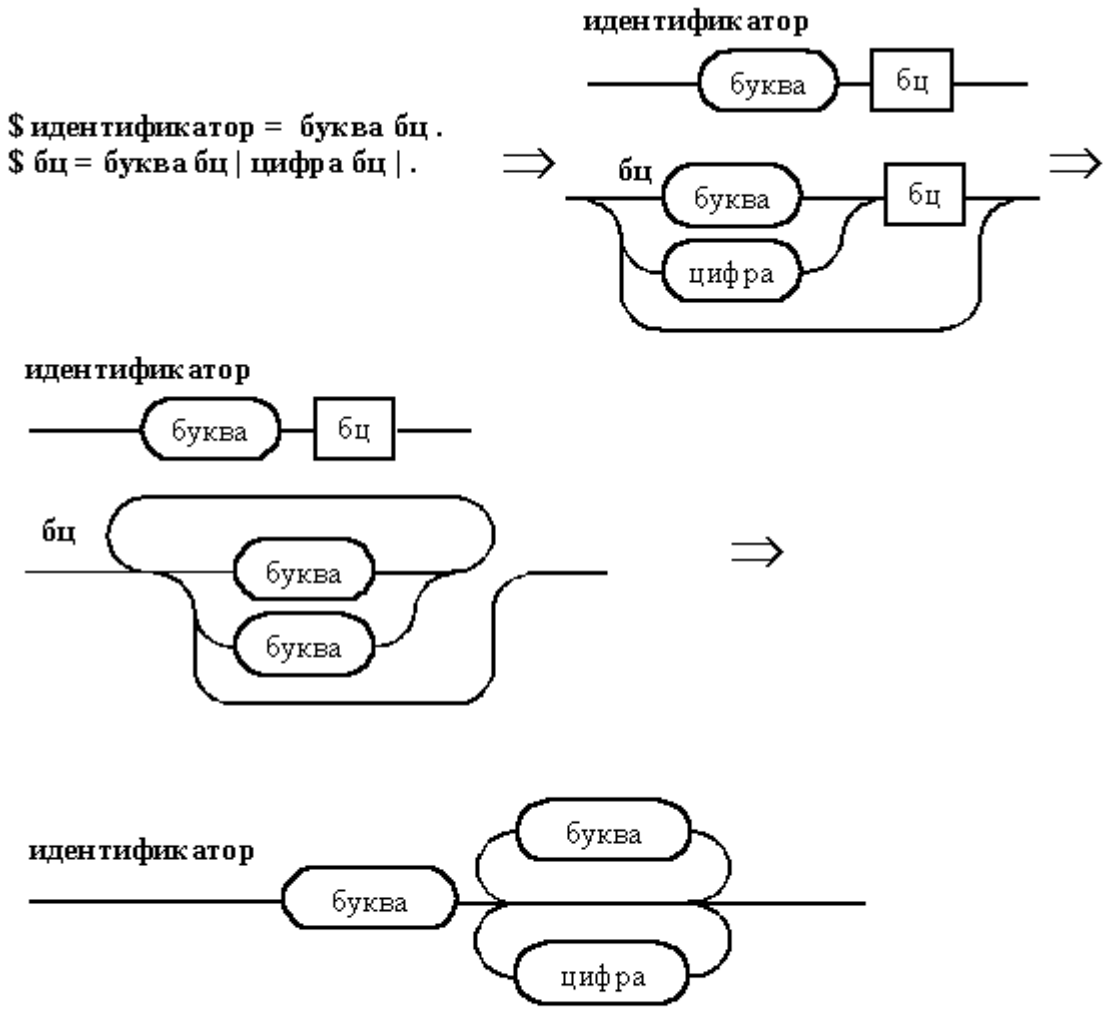


Рис. 4.5. Преобразование праволинейной грамматики идентификатора в итеративную диаграмму Вирта.

Связь между диаграммами Вирта и грамматиками с левой рекурсией. Преобразование левой рекурсии в итерацию

Аналогичные преобразования в итерацию могут осуществляться и для правил, содержащих левую рекурсию:

- дуга, **выходящая** из рекурсивно определяемого нетерминала, замыкается своим **входом** на самый конец правила, образуя, таким образом, цикл;
- сам нетерминал, оказавшийся без адресата, вычеркивается, а входившая в него дуга убирается.

Таким образом, мы имеем зеркальную интерпретацию по отношению к правой рекурсии, что вполне очевидно. Пример, иллюстрирующий, общие принципы преобразования левой рекурсии, представлен на рис. 4.6.

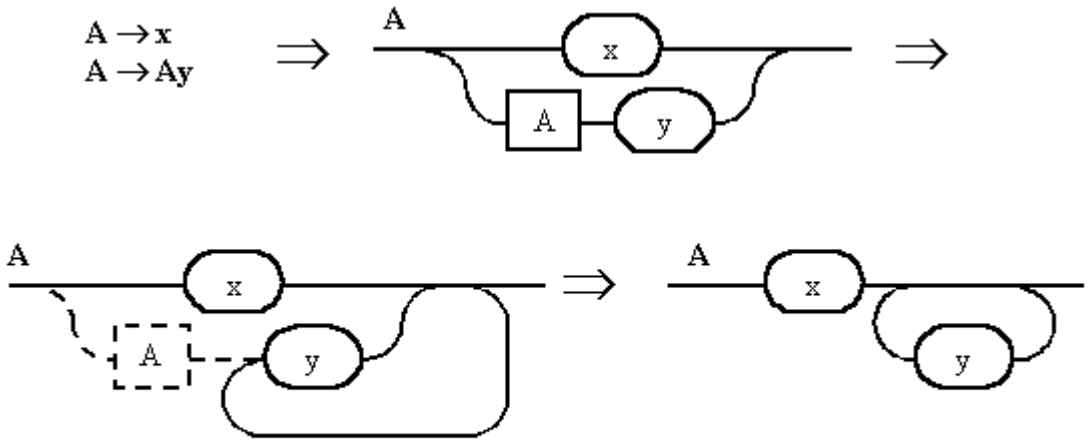


Рис. 4.6. Преобразование грамматики с левой рекурсией в итеративную диаграмму Вирта.

Он показывает возможность такого преобразования, исходя из следующих индуктивных рассуждений. Нетерминал **A** на первом шаге порождает терминальную цепочку **x** или нетерминал **A** (из которого на следующем шаге можно породить терминальную цепочку **x** или нетерминал **A** (из которого...), за которой следует терминальная цепочка **y**), за которым следует терминальная цепочка **y**. А эти бесконечные рассуждения на тему "И надпись написал, что..." можно заменить еще одной структурированной фразой: Нетерминал **A** порождает терминальную цепочку **x**, за которой следует терминальная цепочка **y**, повторяющаяся ноль или большее число раз. На рис. 4.7 приведено преобразование в итеративную диаграмму Вирта правил грамматики с левой рекурсией, описывающих идентификатор. Налицо опять тот же результат.

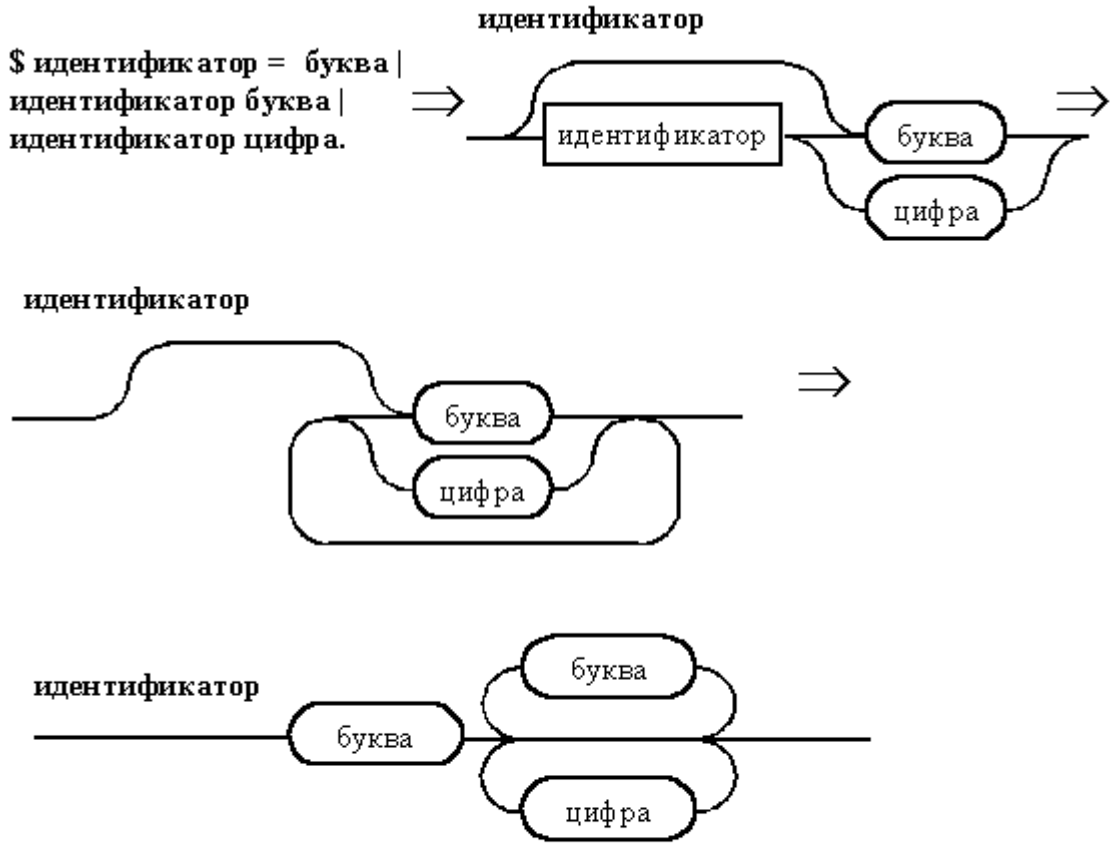


Рис. 4.7. Преобразование грамматики идентификатора с левой рекурсией в итеративную диаграмму Вирта.

Приведенные примеры показывают, что на практике, для описания лексем, удобнее пользоваться итеративными диаграммами Вирта, содержащими терминалы. Они могут быть легко получены из любого другого представления синтаксиса языка. Итеративные диаграммы Вирта, состоящие из терминальных символов практически эквивалентны конечным автоматам, что позволяет использовать их непосредственно для программной реализации. Для описания лексем можно использовать не только праволинейную грамматику (что рекомендуется теорией), но и грамматики с левой рекурсией (на практике дают более наглядное представление). И те, и другие легко сводятся к итеративным диаграммам Вирта, являющимся удобной основой для программной реализации лексического анализатора.

Методы лексического анализа

Выделяются методы непрямого и прямого лексического анализа.

Непрямой лексический анализ, или лексический анализ с возвратами, заключается в последовательной проверке версий о классах лексем. Если проверка текущей версии не подтверждается, то происходит откат назад по цепочке символов и осуществляется проверка следующей версии.

Прямой лексический анализ позволяет определить значение лексемы без откатов назад по цепочке символов.

Организация непрямого лексического анализатора

Непрямой лексический анализатор состоит из отдельных автоматов, каждый из которых распознает одну заданную лексему. Все автоматы имеют одинаковую структуру и отличаются только внутренними состояниями, что связано с различием распознаваемых лексем. Общая структура непрямого лексического анализатора приведена на рис. 4.8.

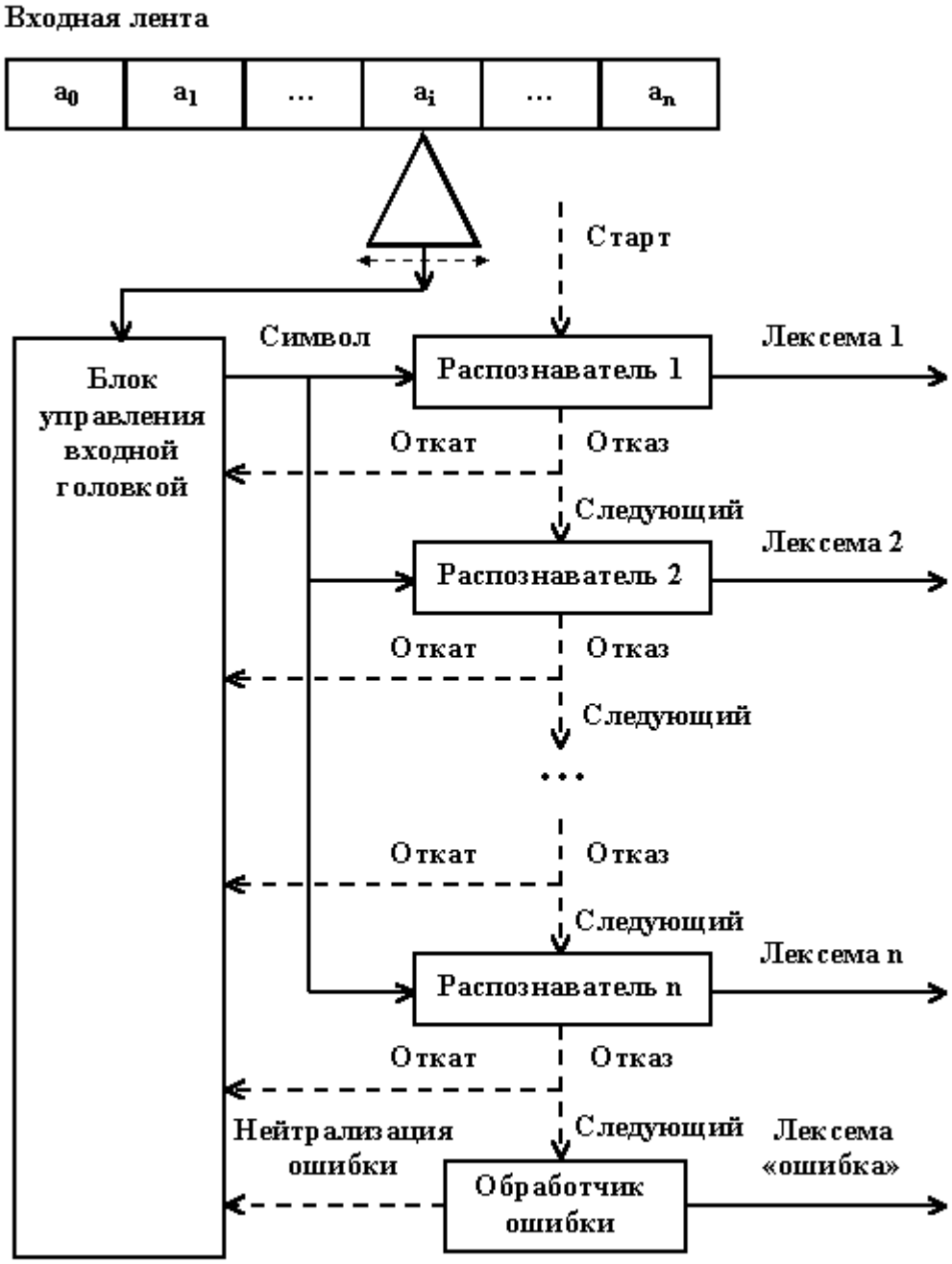


Рис. 4.8. Обобщенная структура непрямого лексического анализатора

Он имеет входную головку, читающую символы с входной ленты. Символы анализируются блоком управления входной головкой и передаются активному автомату. Затем входная головка смещается на один символ вправо.

Автоматы связаны между собой в последовательную цепь, что определяет порядок передачи управления между ними в случае, если активный автомат отвергает входную цепочку символов, то есть, не может распознать предписанную ему лексему. Последовательность автоматов в цепи определяется возможным предпочтительным анализом одних лексем перед другими. Наличие приоритета связано с тем, что некоторые лексемы имеют сходные начальные подцепочки и одна из лексем может включать другую. Например, действительное число может начинаться с цепочки цифр. С нее же может начинаться и целое число. Поэтому, в начале необходимо провести проверку на действительное число, а затем, если версия не подтвердится, попытаться распознать целое число. Существуют различные группы лексем, требующие приоритетной расстановки распознавателей. Они образуют группы автоматов, выстроенных в соответствии с приоритетом. Порядок размещения отдельных групп обычно не является существенным.

Анализ очередной лексемы начинается с запуска автомата, расположенного первым. Он читает первый символ a_i обрабатываемой подцепочки. Далее идет распознавание лексемы, в результате чего читаются n символов. Текущим символом становится a_{i+n}. Если версия о лексеме подтверждается, то сканер выдает ее и завершает работу. Его последующий запуск начинается с чтения символа a_{i+n}, который вновь передается первому автомату. Если же автомат не может распознать лексему, то он выдает сигнал отказа, в блок управления входной головкой и активизирует следующий по приоритету автомат. Блок управления осуществляет возврат входной головки к символу a_i и передает его новому активному автомату. Процесс возврата входной головки осуществляется до тех пор, пока лексема не будет распознана. Если не один из автоматов не распознает лексему, то управление передается обработчику ошибки. Тот выдает лексему ***"ошибка"*** и осуществляет ее нейтрализацию. В простейшем случае нейтрализация заключается в сдвиге входной головки на следующий символ a_{i+1} перед повторным запуском лексического анализатора.

К достоинствам непрямого лексического анализатора следует отнести:

- прозрачность общей регулярной структуры, которая легко может изменяться и наращиваться;
- простота каждого отдельно автомата, распознающего одну, достаточно элементарную, конструкцию;
- практическая применимость подхода в самых разнообразных языках, независимо от сложности выделения в нем тех или иных лексем.

Иллюстрацией последнего пункта может служить следующий фрагмент оператора цикла, являющийся правильной конструкцией в языке программирования FORTRAN-4:

DO I = 3, 10, 3

Так как в данном языке пробелы могут не использоваться, то рассмотренная запись будет корректной, если ее записать следующим образом:

DOI=3,10,3

Поэтому, при анализе сразу не ясно, что перед нами: рассмотренный оператор цикла или оператор присваивания, начинающийся с переменной ***DOI***:

DOI=3

Только возврат назад позволяет корректно просканировать эту и множество других изощренных конструкций Фортрана.

К недостаткам непрямого лексического анализатора следует отнести низкую скорость распознавания. Это связано с постоянными возвратами входной головки к исходному состоянию, если версия о лексеме не подтверждается.

Теоретически данный недостаток можно исправить, если перестроить структуру непрямого лексического анализатора таким образом, чтобы все автоматы работали одновременно. Это вполне допустимо, так как отдельные автоматы являются вполне автономными устройствами. Общая структура такого лексического анализатора приведена на рис. 4.9.

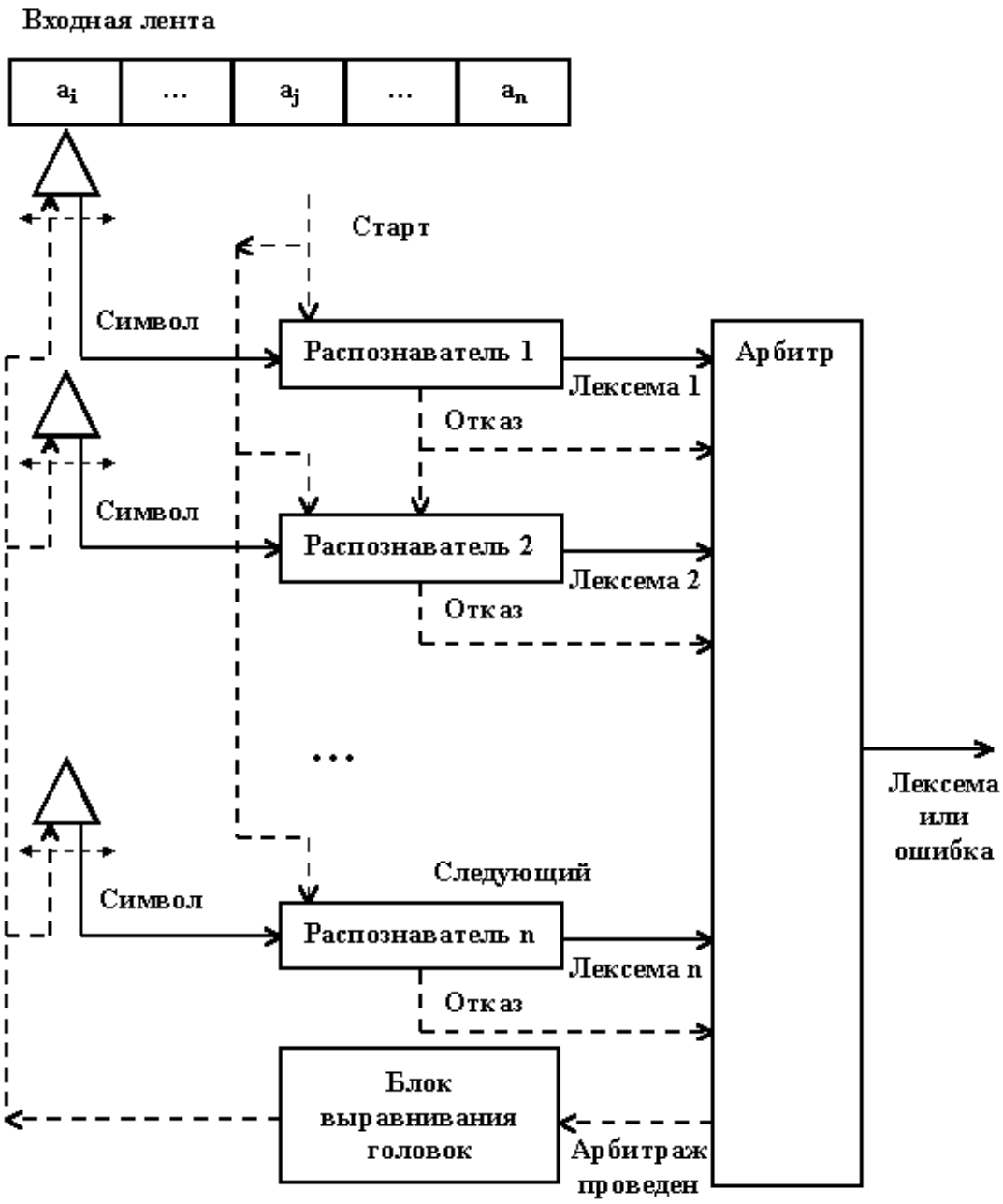


Рис. 4.9. Структура лексического анализатора с параллельной работой распознавателей

Каждый из распознавателей имеет свою входную головку. Перед запуском лексического анализатора все они читают один и тот же символ a_i. После запуска каждый из распознавателей анализирует входную цепочку одновременно с другими, допуская или отвергая ее к концу своей работы.

Распознанные лексемы и отказы передаются арбитру, который определяет нужную выходную лексему в соответствии с избранной схемой их приоритетов. Перед завершением цикла распознавания текущей лексемы входные головки всех автоматов выравниваются по положению входной головки автомата, породившего выходную лексему. В том случае, если

все автоматы выдадут отказ, арбитр формирует на выходе ошибочную лексему и выравнивает все входные головки в соответствии с избранным способом нейтрализации ошибки (можно поступить так же, как и в предыдущем анализаторе: осуществить выравнивание на символе a_{i+1}).

Не смотря на повышение скорости распознавания, такой метод не находит, в настоящее время, практического применения. Это связано с отсутствием вычислительных систем, обеспечивающих реальный параллелизм на уровне коротких последовательностей команд. Кроме того, низким является объем полезных вычислений, так как на каждую распознанную лексему приходится большой объем бесполезных вычислений в автоматах, порождающих другие лексемы и отказы

Организация прямого лексического анализатора

Прямой лексический анализатор строится на основе одного детерминированного автомата, объединяющего множество автоматов, распознающих отдельные лексемы. Такой автомат на каждом шаге читает один входной символ и переходит в следующее состояние, приближающее его к распознаванию текущей лексемы или формированию ошибки. Для лексем, имеющих одинаковые подцепочки, автомат имеет общие фрагменты, реализующие единое множество состояний. Отличающиеся части реализуются своими фрагментами. Обобщенная структура прямого лексического анализатора приведена на рис. 4.10.

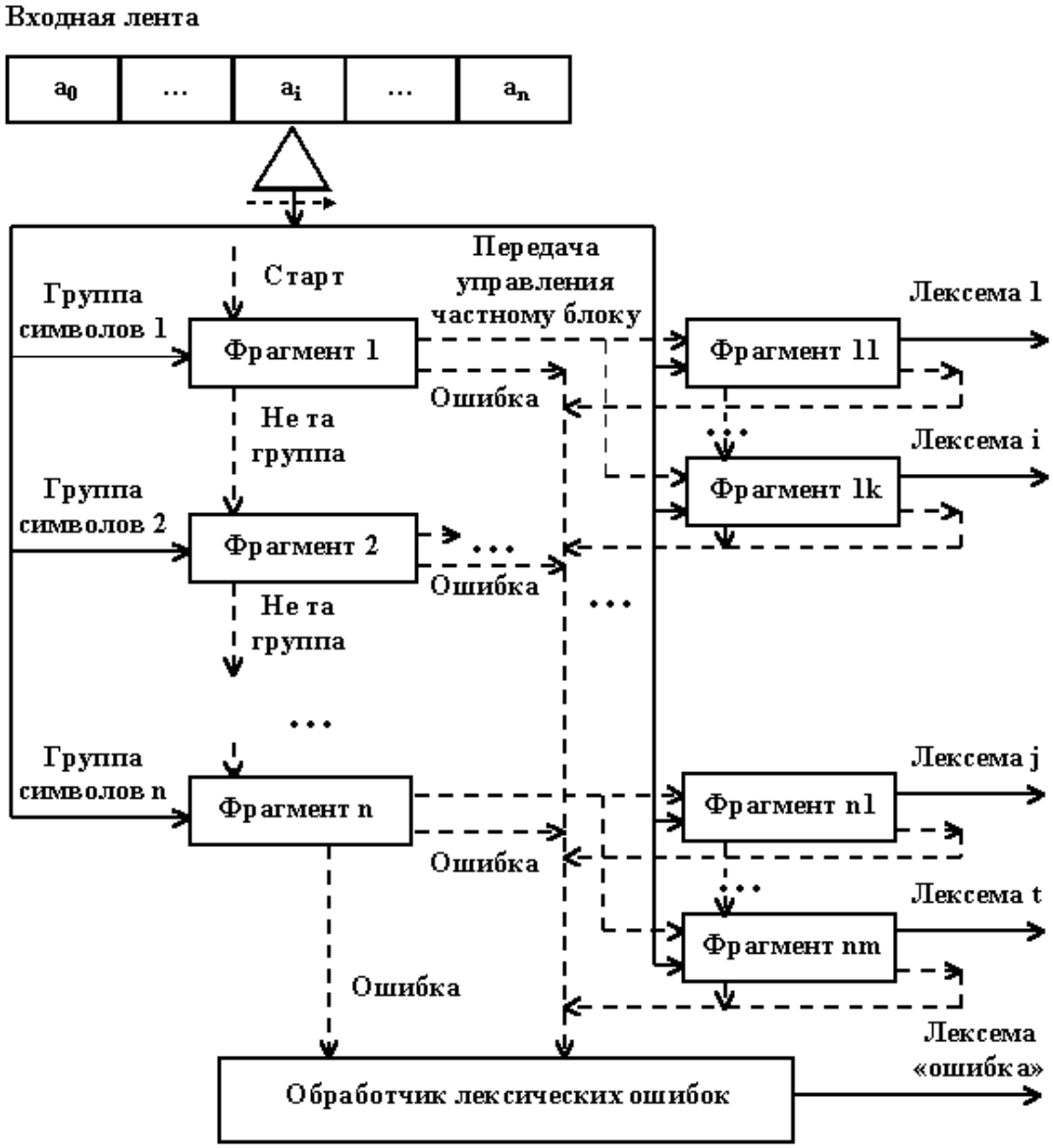


Рис. 4.10. Обобщенная структура прямого лексического анализатора

Он содержит входную головку, которая передает текущий входной символ в первое состояние одного из начальных блоков, реализующего фрагмент детерминированного автомата. Каждое из таких состояний распознает множество входных символов, не пересекающихся с множеством входных символов других состояний. Разбиение автомата на фрагменты проводится на основе выделения состояний, специализирующихся на распознавании общих или частных подцепочек отдельных лексем. В самом общем случае прямой лексический анализатор может рассматриваться как единый неструктурированный блок. Поэтому его дробление на отдельные блоки достаточно условно. Очередной символ читается обычно каждым состоянием автомата. В каждом фрагменте условно можно выделить начальное состояние, с которого начинается разбор подцепочки, закрепленной за фрагментом. Если символ не входит в группу, распознаваемых начальным состоянием фрагмента, то происходит отказ и передача управления начальному состоянию следующего альтернативного фрагмента,

анализирующего другую группу входных символов. Если входной символ не может быть распознан другими состояниями фрагмента, то обработка отказа передается обработчику ошибок, который формирует ошибочную лексему и нейтрализует ошибку принятым способом. Передача управления обработчику ошибок осуществляется и в том случае, если все начальные состояния связанных между собой фрагментов формируют отказ. Разбор заканчивается выдачей распознанной или ошибочной лексемы.

К достоинствам прямого лексического анализатора относятся:

- высокая производительность за счет отсутствия возвратов входной головки;
- меньшее общее число состояний, что получается за счет слияния одинаковых фрагментов отдельных автоматов и проведения дополнительной минимизации.

К недостаткам следует отнести:

- на разработку прямого лексического анализатора требуется больше времени;
- данный анализатор практически невозможно реализовать для некоторых языков программирования. Примером такого языка может служить FORTRAN-4.