## Глава 2. Лексический анализ

### 2.4. Распознавание токенов

Основной задачей сканера является чтение входных символов исходной программы с целью их группирования в лексемы (которые с точки зрения языка должны рассматриваться как единое целое), представляющие соответствующие токены. Если задан шаблон, описывающий лексему токена, распознавание токена выполняется достаточно просто. Однако возникает сложность распознавания, если лексема одного токена может быть префиксом лексемы другого токена. В таких случаях для корректного распознавания потребуется чтение дополнительных символов входного потока, следующих за текущим символом (*опережающее чтение*).

Например, если в языке в качестве разделителя используется символ '**:**', а в качестве операции присваивания используется обозначение '**:=**', то, встретив символ '**:**', необходимо прочитать следующий за ним символ. Если он окажется символом '**=**', то получена лексема '**:=**' токена «операция присваивания». В противном случае символ '**:**' является лексемой токена «двоеточие», при этом прочитан один дополнительный символ входного потока, который следует вернуть во входной поток (этот символ может являться началом другой лексемы).

Приведем еще один пример. Рассмотрим синтаксически правильные входные строки '3.75' и '3..75'. Первая строка образует одну лексему, относящуюся к токену «числовая константа», а вторая – три лексемы: '3', '**..**', '75', определяющие конструкцию «диапазон», где лексемы '3' и '75' являются числовыми константами. Чтобы распознать лексему '3' как токен «числовая константа», требуется прочитать два дополнительных символа (две точки), которые следует вернуть во входной поток.

Для облегчения реализации опережающего чтения и возврата символов во входной поток рекомендуется использовать входной буфер, из которого лексический анализатор может выполнять чтение и в который может возвращать прочитанные символы путем простого перемещения указателя. Использование входного буфера повышает также эффективность анализатора, так как считывание блока символов обычно существенно более эффективно, чем посимвольное считывание.

Можно использовать следующую схему буферизации [1], которая включает два по очереди загружаемых буфера (рис. 2.2).

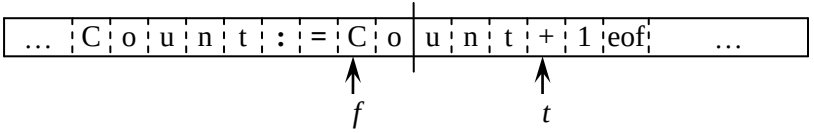



Рис. 2.2. Схема буферизации входного потока

Оба буфера имеют один и тот же размер $N$, причем $N$ обычно равно размеру дискового блока, например 4096 байт. При этом можно считать $N$ символов в буфер одной командой чтения, не используя системный вызов для каждого символа по отдельности. Если во входном файле осталось менее $N$ символов, конец исходного файла маркируется специальным символом конца файла **eof**, отличным от любого из возможных символов исходной программы. Предусматриваются два указателя: указатель $f$, маркирующий начало текущей обрабатываемой лексемы, и указатель $t$, который сканирует символы до тех пор, пока выполняется соответствие шаблону.

Как только определена очередная лексема, указатель *t* устанавливается таким образом, чтобы указывать на символ, являющийся ее правым концом. Затем, после того как определен токен для лексемы, передаваемый синтаксическому анализатору, указатель *f* устанавливается на символ, непосредственно следующий за только что обнаруженной лексемой. На рис. 2.2 указатель *t* указывает на символ за концом текущей лексемы Count (идентификатор переменной) и должен быть перемещен на одну позицию влево (возврат дополнительно прочитанного символа во входной поток).

При перемещении указателя *t* первоначально необходимо выполнить проверку, не достигнут ли конец одного из буферов, и, если достигнут, заполнить другой буфер символами из входного потока, и перенести *t* в начало этого только что заполненного буфера.

Таким образом, для каждого считанного символа необходимо выполнить два сравнения: одно – не достигнут ли конец буфера, второе – какой именно символ считан. Однако проверку конца буфера можно совместить с проверкой считанного символа, если расширить каждый буфер для хранения в его конце специального ограничителя (сторожа). Ограничитель представляет собой специальный символ, который не может быть частью исходной программы.

Множество лексем, соответствующих некоторому токену, можно рассматривать как формальный язык этого токена (язык идентификаторов, язык констант, язык символов пунктуации, язык символов операций и т. д.). Языки токенов практически всегда относятся к классу регулярных языков, поэтому в качестве формального описания шаблонов токенов достаточно использовать регулярные грамматики или регулярные выражения.

Распознавателем регулярного языка является конечный автомат. Существуют процедуры построения (*синтеза*) конечного автомата для заданной регулярной грамматики или заданного регулярного выражения, которые будут рассмотрены ниже в соответствующих разделах.

При синтезе конечных автоматов необходимо учитывать следующую особенность. Вместо построения отдельных конечных автоматов для каждого ключевого слова можно использовать специальную таблицу ключевых слов. Если лексический анализатор распознает лексему как идентификатор, то он простым просмотром таблицы ключевых слов может определить, является эта лексема ключевым словом или нет.

После синтеза всех конечных автоматов-распознавателей для всех токенов наиболее предпочтительным подходом является объединение всех конечных автоматов в один конечный автомат, который выбирает наибольшую лексему, соответствующую некоторому шаблону. Тогда программная реализация лексического анализатора сводится к программной реализации этого конечного автомата с соответствующим формированием токенов и реализацией возврата символов во входной поток при опережающем чтении.

### 2.5. Конечный автомат

Формально конечный автомат-распознаватель (КА) определяется как пятерка $M = (K, T, \delta, k_0, F)$, где

$K$ – конечное множество состояний;

$T$ – конечный входной алфавит;

$\delta$: $K \times T \to 2^K$ – функция переходов автомата, здесь $2^K$ обозначает булеан (степень) множества $K$ (множество всех подмножеств множества $K$, мощность булеана равна $2^{|K|}$);

$k_0 \in K$ – начальное состояние автомата;

$F \subseteq K$ – множество конечных (финальных, заключительных, принимающих) состояний.

Приведено определение *недетерминированного* конечного автомата (НКА), у которого значением функции переходов $\delta(k, a)$, $k \in K$, $a \in T$, может являться любое подмножество состояний из $K$. Если областью значений функции переходов является множество состояний $K$, то есть $\delta$: $K \times T \to K$, автомат называется *детерминированным* (ДКА). В ДКА множество $\delta(k, a)$ содержит не более одного элемента для всех $k \in K$ и $a \in T$.

Запись функции переходов $\delta(k_i, a) = k_j$ для ДКА означает, что при чтении входного символа $a \in T$ в текущем состоянии $k_i \in K$ автомата осуществляется переход в состояние $k_j \in K$. В НКА при чтении символа $a$ в текущем состоянии $k_i$ возможен переход в любое из нескольких альтернативных состояний из множества $\delta(k_i, a)$.

Функцию переходов можно представить в виде графа (диаграммы) переходов или таблицы переходов.

Граф переходов автомата представляет собой ориентированный граф, в котором вершины помечаются состояниями автомата, а дуги – символами входного алфавита. Начальное состояние будем выделять небольшой входной стрелкой, а конечные состояния будем изображать в виде прямоугольника.

Таблица переходов является обычным матричным представлением функции переходов δ. Формально, согласно определению функции δ: $K \times T \to 2^K$, в ее матричном представлении строки должны соответствовать состояниям, а столбцы – символам входного алфавита. Но часто удобнее представлять эту матрицу в транспонированном виде, а именно: строки соответствуют символам входного алфавита, а столбцы – состояниям автомата, на пересечении строки и столбца записывается соответствующее значение функции. Для выделения конечных состояний будем заключать их в квадратные скобки. Первый столбец всегда соответствует начальному состоянию.

Рассмотрим НКА $M = (\{k_0, k_1, k_2, k_f\}, \{a, b\}, \delta, k_0, \{k_f\})$ со следующей функцией переходов δ:

$\delta(k_0, a) = \{k_1, k_f\}$; $\delta(k_2, a) = \{k_2, k_f\}$;

$\delta(k_0, b) = \varnothing$; $\delta(k_2, b) = \varnothing$;

$\delta(k_1, a) = \{k_1, k_f\}$; $\delta(k_f, a) = \varnothing$;

$\delta(k_1, b) = \{k_2\}$; $\delta(k_f, b) = \varnothing$.

На рис. 2.3. приведено представление функции переходов δ в виде графа переходов, а на рис. 2.4 – в виде таблицы переходов.

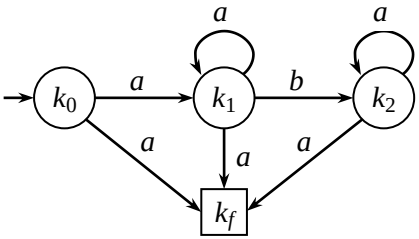
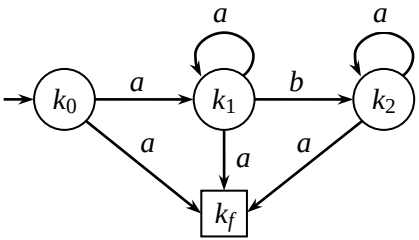



Рис. 2.3. Граф переходов НКА




| | $k_0$ | $k_1$ | $k_2$ | $[k_f]$ |
|---|---|---|---|---|
| $a$ | $\{k_1, k_f\}$ | $\{k_1, k_f\}$ | $\{k_2, k_f\}$ | $\varnothing$ |
| $b$ | $\varnothing$ | $\{k_2\}$ | $\varnothing$ | $\varnothing$ |

Рис. 2.4. Таблица переходов НКА из рис. 2.3

По мере чтения символов входной строки КА переходит из состояния в состояние в соответствии с функцией переходов. Если после чтения последнего символа входной строки КА находится в одном из конечных состояний, то говорят, что строка принимается автоматом. В противном случае строка не принадлежит языку, принимаемому автоматом.

Расширим определение функции переходов на строку, определив ее как функцию $\delta: K \times T^* \to K$ (для НКА $\delta: K \times T^* \to 2^K$). Тогда $\delta(k, x)$ – состояние, которое будет достигнуто из состояния $k$ после ввода строки $x \in T^*$. Если $x = ay$ $(a \in T,\ y \in T^*)$, то расширенную функцию $\delta$ можно определить рекурсивно: $\delta(k, x) = \delta(\delta(k, a), y)$.

Таким образом, множество строк $T(M) \subseteq T^*$, принимаемых автоматом $M$, можно определить следующим образом:

$T(M) = \{x \in T^* \mid \delta(k_0, x) \in F\}$.

Использование НКА в качестве распознавателя приводит к существенным потерям времени при лексическом анализе, поскольку приходится перебирать все возможные варианты. Поэтому следует использовать всегда ДКА. К счастью, каждому НКА можно поставить в соответствие ДКА, принимающий тот же язык.

Поскольку множество всех подмножеств множества $K$ конечно и равно $2^{|K|}$, то каждое подмножество можно рассматривать как отдельное состояние, т. е. получится $2^{|K|}$ возможных состояний детерминированного автомата. На практике, чтобы не рассматривать все $2^{|K|}$ возможных состояний, ограничиваются рассмотрением множества тех состояний, которые могут быть достигнуты из начального состояния $\{k_0\}$, т. е. $K' = \delta(\{k_0\}, a)$ для всех $a \in T$. Далее рассматривается множество состояний $\delta(K', a)$ для всех $a \in T$, $K' \subseteq K$. Процесс продолжается до тех пор, пока полученные таким образом состояния не начнут совпадать с уже существующими состояниями автомата. В качестве множества конечных состояний полученного ДКА выбирается такое множество $K' \subseteq K$, что $K' \cap F \neq \varnothing$, т. е. множество $K'$, образующее состояние ДКА, содержит хотя бы одно из конечных состояний исходного НКА.

Процесс определения функций переходов ДКА, эквивалентного НКА на рис. 2.3, представлен на рис. 2.5.

| | $\{k_0\}$ | $[\{k_1, k_f\}]$ | $\{k_2\}$ | $[\{k_2, k_f\}]$ |
|---|---|---|---|---|
| $a$ | $\{k_1, k_f\}$ | $\{k_1, k_f\}$ | $\{k_2, k_f\}$ | $\{k_2, k_f\}$ |
| $b$ | $\varnothing$ | $\{k_2\}$ | $\varnothing$ | $\varnothing$ |

Рис. 2.5. Определение функций переходов ДКА

Из начального состояния $\{k_0\}$ достижимо состояние $\{k_1, k_f\}$ при чтении входного символа $a$. Поэтому добавляется соответствующий столбец. Определим состояния, в которые возможен переход из $\{k_1, k_f\}$:

$\delta(\{k_1, k_f\}, a) = \delta(k_1, a) \cup \delta(k_f, a) = \{k_1, k_f\} \cup \varnothing = \{k_1, k_f\}$;

$\delta(\{k_1, k_f\}, b) = \delta(k_1, b) \cup \delta(k_f, b) = \{k_2\} \cup \varnothing = \{k_2\}$.

Появилось новое состояние $\{k_2\}$ (добавляем в таблицу соответствующий столбец). Аналогично, из состояния $\{k_2\}$ возможен переход в новое состояние $\{k_2, k_f\}$, из которого достижимо только состояние $\{k_2, k_f\}$. Новые состояния не появились, процесс завершен. Конечными состояниями будут состояния $\{k_1, k_f\}$ и $\{k_2, k_f\}$, поскольку они содержат конечное состояние $k_f \in F$ исходного НКА.

Состояние, соответствующее пустому множеству играет роль фиктивного состояния. С позиции лексического анализа это означает наличие лексической ошибки и может быть выделено как специальное состояние «ошибка». Это состояние в действительности может быть опущено, в результате чего будет получена неполная функция переходов.

Далее можно пронумеровать полученные состояния неотрицательными целыми числами (начальному состоянию дадим номер 0): $\{k_0\} - 0$, $\{k_1, k_f\} - 1$, $\{k_2\} - 2$, $\{k_2, k_f\} - 3$. В результате получится таблица переходов (рис. 2.6) и граф переходов (рис. 2.7) ДКА, эквивалентного исходному НКА.

| | 0 | [1] | 2 | [3] |
|---|---|---|---|---|
| *a* | 1 | 1 | 3 | 3 |
| *b* | | 2 | | |

Рис. 2.6. Таблица переходов ДКА

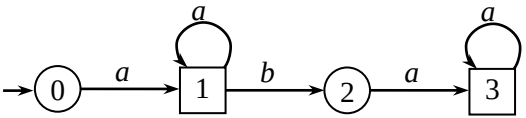



Рис. 2.7. Граф переходов ДКА