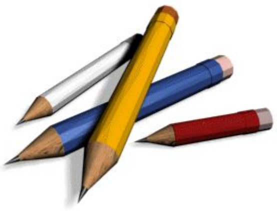
Лексический анализ

Лексический анализ

Основная задача лексического анализа

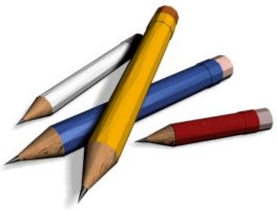
разбить входной текст, состоящий из последовательности отдельных литер (символов), на последовательность лексем (слов)



Лексический анализ

Любой символ входной последовательности может

- принадлежать к какой-либо лексеме
- принадлежать к разделителю (разделять лексемы)

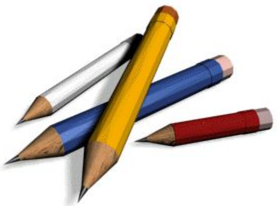


Лексический анализ

В зависимости от ситуации одна и та же последовательность символов может быть и частью лексемы, и частью разделителя.

```
if (a>b) /* if */
```

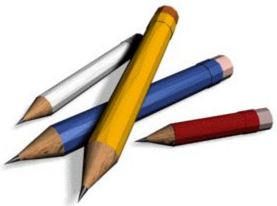
лексема *разделитель*



Лексический анализ

В некоторых случаях разделители между лексемами могут отсутствовать.

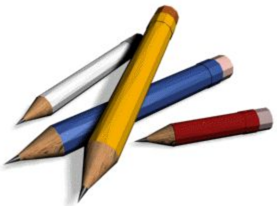
```
if ( a > b ) a -= b ; else b -= a ;
if(a>b)a-=b;else b-=a;
```



Кроме определения границ лексем, при работе лексического анализатора требуется определить их тип:

– идентификаторы, в т.ч.
    – ключевые слова;
  – пользовательские идентификаторы;
– пунктуаторы;
– числа;
– строки;

и т.д.



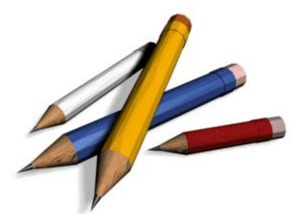
Лексический анализ

Лексический анализатор выдает последующим фазам компиляции информацию в зависимости от типа лексемы:

для синтаксического анализа

– ключевые слова – значение лексемы;
– пользовательские идентификаторы – тип лексемы;
– пунктуаторы – значение лексемы;
– числа – тип лексемы;
– строки – тип лексемы;

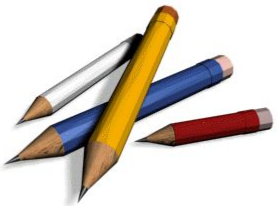
и т.д.



Лексический анализ

для семантического анализа

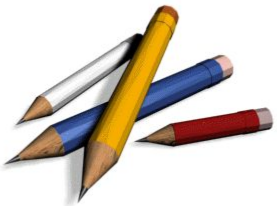
– пользовательские идентификаторы – значение лексемы;
– числа – значение лексемы;
– строки – значение лексемы;
и т.д.



для всех последующих фаз

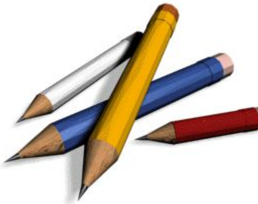
– расположение лексемы в исходном тексте программы (имя файла, номер строки, позиция в строке)

(для локализации синтаксических ошибок и ошибок времени выполнения)



Регулярные множества

На этапе лексического анализа удобно считать, что лексемы каждого типа являются элементами отдельных *языков*, называемым *регулярными множествами.*



Формальное определение регулярного множества

Регулярное множество в алфавите *V* определяется *рекурсивно* следующим образом:

(1) $\varnothing$ (пустое множество) – регулярное множество в алфавите *V*;

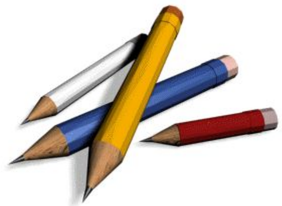
(2) {ε} – регулярное множество в алфавите *V* (ε – пустая цепочка);

(3) {*a*} – регулярное множество в алфавите *V* для каждого $a \in V$;

(4) если *P* и *Q* – регулярные множества в алфавите *V*, то регулярными являются и множества

- (а) $P \cup Q$ (объединение),
- (б) *PQ* (конкатенация, т.е. множество $\{pq \mid p \in P,\ q \in Q\}$),
- (в) P^* (итерация: $P^* = \sum_{n=0}^{\infty} P^n$);

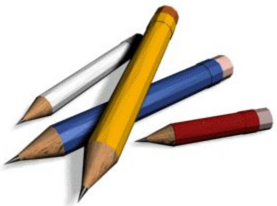
(5) ничто другое не является регулярным множеством в алфавите *V*.



Неформальное определение регулярного множества

Множество в алфавите *V* регулярно тогда и только тогда, когда оно

- $\varnothing$,
- {ε},
- {*a*}, где $a \in V$,
- его можно получить из этих множеств применением конечного числа операций объединения, конкатенации и итерации.



Регулярные выражения

Средством записи регулярных множеств являются *регулярные выражения.*

Лексический анализ

Формальное определение регулярного выражения

Регулярное выражение в алфавите V определяется *рекурсивно* следующим образом:

(1) $\varnothing$ – регулярное выражение, обозначающее множество $\varnothing$;

(2) ε – регулярное выражение, обозначающее множество {ε};

(3) *a* – регулярное выражение, обозначающее множество {*a*};

(4) если *p* и *q* – регулярные выражения, обозначающие регулярные множества *P* и *Q* соответственно, то

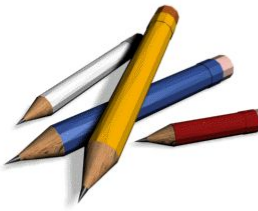
- (а) (*p*|*q*) – регулярное выражение, обозначающее регулярное множество $P \cup Q$,
- (б) (*pq*) – регулярное выражение, обозначающее регулярное множество *PQ*,
- (в) (*p**) – регулярное выражение, обозначающее регулярное множество *P**;

(5) ничто другое не является регулярным выражением в алфавите V.

Для краткости записи регулярных выражений используются следующие соглашения:

– лишние скобки в регулярных выражениях опускаются с учетом приоритета операций:
 1. операция итерации (наивысший приоритет)
 2. операция конкатенации
 3. операция объединения (наименьший приоритет).
– запись *p*+ обозначает выражение *pp**

Например: $(a \mid ((ba)(a^*))) \rightarrow a \mid ba+$



Примеры регулярных выражений:

a* (ε | *a*) | *b

обозначает множество {*a*, *b*, *aa*};

***a* (*a* | *b*) ***

обозначает множество всевозможных цепочек, состоящих из *a* и *b*, начинающихся с *a*;

(*a* | *b*)* (*a* | *b*) (*a* | *b*)*

обозначает множество всех непустых цепочек, состоящих из *a* и *b*, т.е. множество $\{a, b\}^+$;

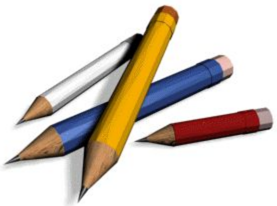
((0 | 1) (0 | 1) (0 | 1))*

обозначает множество всех цепочек, состоящих из нулей и единиц, длины которых делятся на 3.

Лексический анализ

Для каждого регулярного множества можно найти регулярное выражение, обозначающее это множество, и наоборот.

Для каждого регулярного множества существует бесконечно много обозначающих его регулярных выражений.



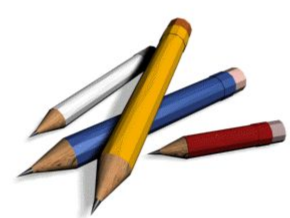
При записи регулярных выражений оказывается удобно дать им индивидуальное обозначение.

Пример 1. Регулярное выражение для множества идентификаторов.

$$Letter = a \mid b \mid c \mid \ldots \mid x \mid y \mid z$$

$$Digit = 0 \mid 1 \mid \ldots \mid 9$$

$$Identifier = Letter\ (\ Letter \mid Digit\ )^*$$



Пример 2. Регулярное выражение для множества вещественный чисел с плавающей запятой.

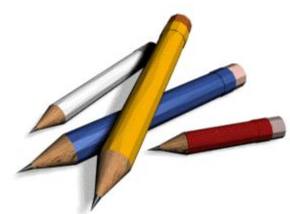
Digit = 0 | 1 | ... | 9

Integer = *Digit* +

Fraction = .*Integer* | ε

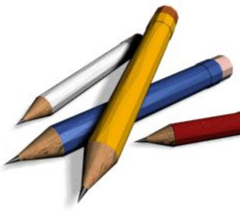
Exponent = (*E*(+ | – | ε)*Integer*) | ε

Number = *Integer Fraction Exponent*



Лексический анализ

Для определения принадлежности последовательности символов регулярному множеству (т.е. для реализации процедуры *распознавания языка*) чаще всего используются *конечные автоматы*.



Формальное определение конечного автомата

Конечным автоматом (КА) называют пятерку ($\mathbf{Q}$, $\mathbf{V}$, f, q_0, $\mathbf{F}$), где

Q – конечное множество состояний автомата;

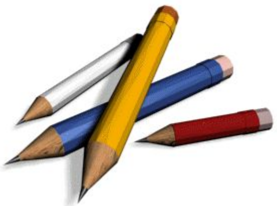
V – конечное множество допустимых входных символов (алфавит автомата);

f – функция переходов, отображающая декартово произведение множеств $\mathbf{V}\times\mathbf{Q}$ во множество подмножеств **Q**:

$$f(a, q)=\mathbf{R},\ a\in\mathbf{V},\ q\in\mathbf{Q},\ \mathbf{R}\subseteq\mathbf{Q};$$

q_0 – начальное состояние автомата, $q_0\in\mathbf{Q}$;

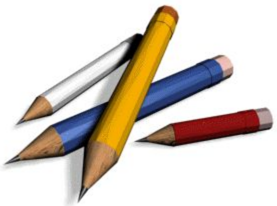
F – непустое множество заключительных состояний автомата, $\mathbf{F}\subseteq\mathbf{Q}$, $\mathbf{F}\neq\varnothing$.



Работа конечного автомата представляется в виде последовательности шагов.

На каждом шаге автомат находится в одном из своих состояний $q \in \mathbf{Q}$, называемым текущим состоянием. В начале работы автомат всегда находится в начальном состоянии q_0.

На следующем шаге он может перейти в другое состояние или остаться в текущем. То, в какое состояние автомат перейдет на следующем шаге работы, определяет функция переходов f.

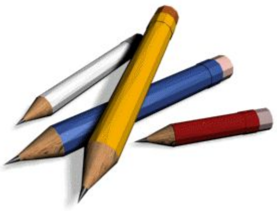


Лексический анализ

Функция переходов зависит не только от текущего состояния, но и от того, какой символ из алфавита **V** был подан на вход автомата.

Значением функции переходов *f* является некоторое *множество* следующих состояний автомата.

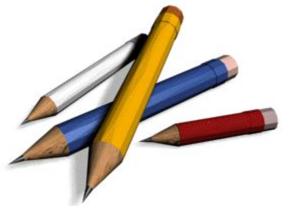
Конечный автомат может перейти в любое из этих состояний.



Лексический анализ

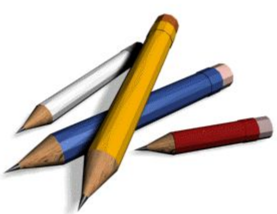
Работа конечного автомата продолжается до тех пор, пока на его вход поступают символы.

Если после окончания работы конечного автомата он находится в одном из *заключительных состояний*, то говорят, что конечный автомат *принял цепочку (допускает цепочку).*



Лексический анализ

Множество цепочек, принимаемых (допускаемых) конечным автоматом, называют *языком, распознаваемым (допускаемым) конечным автоматом.*



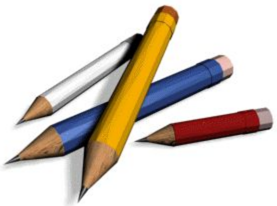
Неформальное определение конечного автомата

Лексический анализ

Лексический анализ

Конечный автомат называют *полностью определенным (всюду определенным)*, если в каждом его возможном состоянии функция перехода определена для всех возможных входных символов:

$$\forall a \in \mathbf{V}\ \forall q \in \mathbf{Q}\ \exists\ f(a, q)=\mathbf{R},\ \mathbf{R} \subseteq \mathbf{Q}.$$



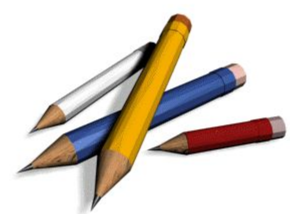
Лексический анализ

Конечный автомат называют *детерминированным*, если для любой допустимой комбинации входного символа и текущего состояния значение функции переходов содержит не более одного следующего состояния.

$$\forall a \in \mathbf{V}\ \forall q \in \mathbf{Q}\ |f(a, q)| \leq 1$$

В противном случае конечный автомат называют *недетерминированным*.

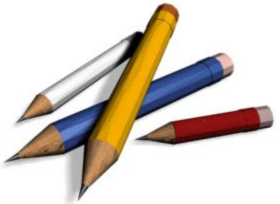
$$\exists\ a \in \mathbf{V}\ \exists\ q \in \mathbf{Q}\ |f(a, q)| > 1$$



Лексический анализ

Особенностью недетерминированных конечных автоматов (НКА) является то, что находясь в некотором состоянии и получая на входе один и тот же символ, они могут перейти в любое из *нескольких* возможных последующих состояний.

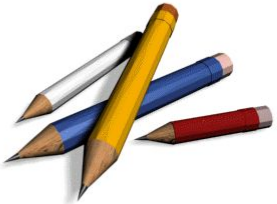
Непосредственная практическая реализация НКА затруднительна. Однако можно построить ДКА, распознающий тот же самый язык, что и НКА.



Большое практическое значение имеет следующая

Теорема.
Пусть $M = (\mathbf{Q}, \mathbf{V}, f, q_0, \mathbf{F})$ – детерминированный конечный автомат, не являющийся всюду определенным.

Тогда существует всюду определенный детерминированный конечный автомат $M' = (\mathbf{Q}', \mathbf{V}, f', q_0, \mathbf{F})$, такой что $L(M) = L'(M')$.



Лексический анализ

Доказательство (конструктивное).

Дополним множество Q состояний ДКА новым состоянием: $Q' = Q \cup \{q'\}$, $q' \notin Q$.

Определим новую функцию $f\,'$ переходов ДКА следующим образом:

– $f\,'(a, q) = f\,(a, q)$, если $f\,(a, q) \neq \varnothing$
– $f\,'(a, q) = \{q'\}$, если $f\,(a, q) = \varnothing$
– $f\,'(a, q') = \{q'\}$

Легко показать, что построенный таким образом автомат допускает тот же самый язык.

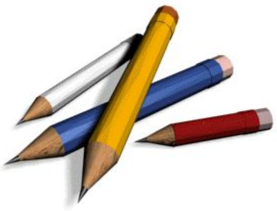
Лексический анализ

Новое состояние конечного автомата соответствует *ошибочной ситуации* (т.е. на вход автомата поступил недопустимый в данном состоянии символ).

Диаграмма переходов конечного автомата

Графически конечные автоматы принято изображать с помощью диаграмм переходов.

Диаграмма переходов конечного автомата – это направленный граф, вершины которого помечены символами состояний конечного автомата, и содержащий помеченные дуги, описывающие допустимые переходы, т.е. на графе изображается дуга (p, q), помеченная символом $a \in V$, если $q \in f(p, a)$.



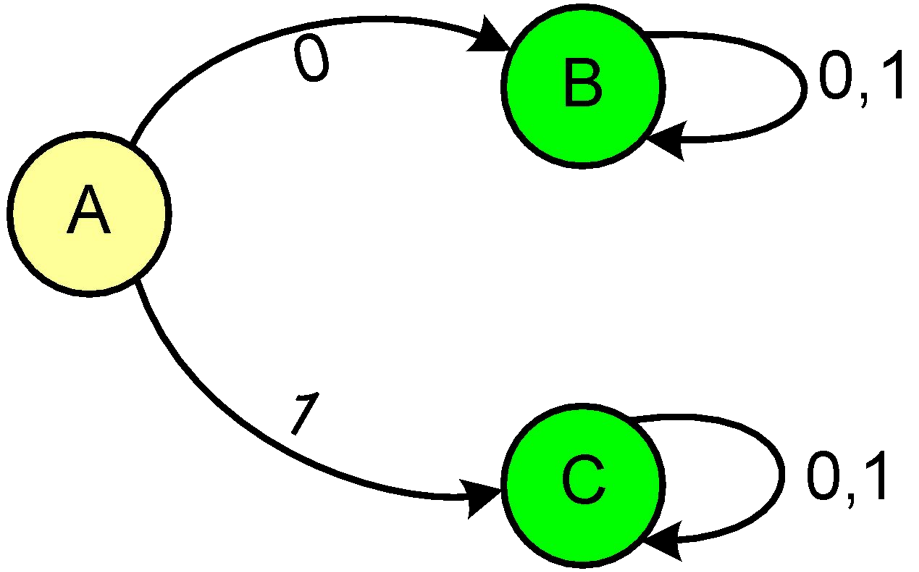
Пример 1. Диаграмма всюду определенного детерминированного конечного автомата

Q = {A,B,C}

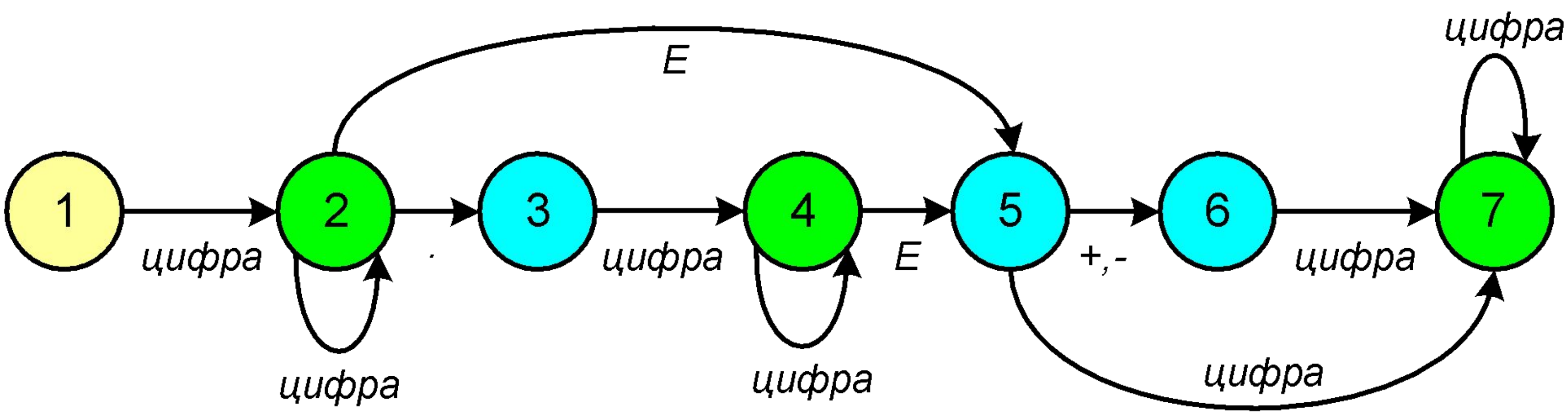
V = {0,1}

q_0 = A

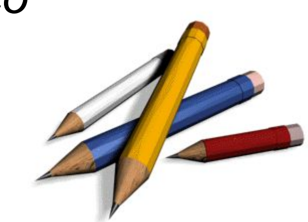
f	0	1
A	B	C
B	B	B
C	C	C



Пример 2. Диаграмма конечного автомата, принимающего множество положительных действительных чисел



Различными цветами показаны различные состояния конечного автомата: начальное, промежуточные, заключительные



Состояниям этого конечного автомата соответствуют:

1 – Начало числа

2 – Целая часть

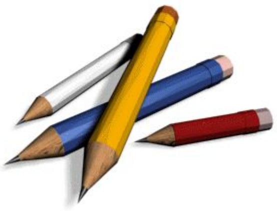
3 – Начало дробной части

4 – Дробная часть

5 – Начало экспоненциальной части

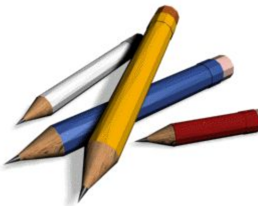
6 – Начало модуля показателя

7 – Показатель



Функция переходов этого конечного автомата определяется следующим образом:

f	.	+, –	E	0..9
1				2
2	3		5	2
3				4
4			5	4
5		6		7
6				7
7				7

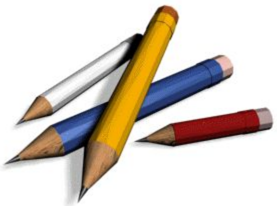


Способы программной реализации конечного автомата

1-й способ. С помощью подпрограммы

Все состояния КА рекомендуется описать в виде перечисления:

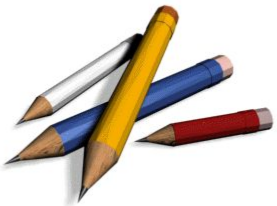
```
enum StateType { STATE_START, STATE_1, STATE_2, … };
```



Лексический анализ

Подпрограмма, реализующая функцию переходов конечного автомата, будет иметь следующую структуру:

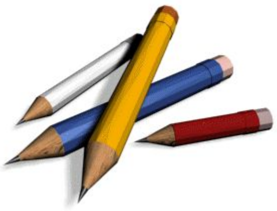
```
StateType f(StateType State, char c) {
switch(State) {
case STATE_START:
switch(c) {
case '1': return STATE_1;
case '2': return STATE_2;
...
default: return STATE_ERROR;
}
break;
case STATE_1: switch(c) {
...
}
break;
...
}
}
```



Лексический анализ

Для проверки входной последовательности символов достаточно нужное число раз вызвать функцию переходов:

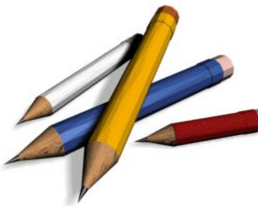
```
StateType State=START_STATE;
for(int i=0;i<strlen(stroka);i++) {
State = f(State,stroka[i]);
}
```



Лексический анализ

Затем следует определить (например, с помощью отдельной логической функции), является ли состояние конечного автомата заключительным, если да, то каким именно:

```
if ( isFinalState(State) ) {
// Ok!
switch(State) {
...
}
} else {
// Error!
}
```

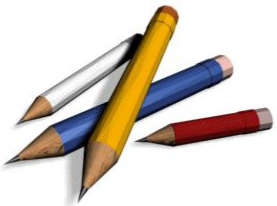


2-й способ. С помощью набора объектов

Все состояния КА могут быть описаны как отдельные объекты, у которых за переход в следующее состояние отвечает специальный метод.

Иерархию классов удобно начать с абстрактного класса:

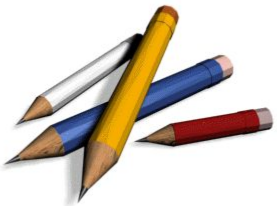
```
abstract class AbstractState {
abstract AbstractState getNextState (char c);
abstract boolean isFinalState();
}
```



Лексический анализ

Для каждого состояния или группы состояний КА следует описать конкретные классы, например:

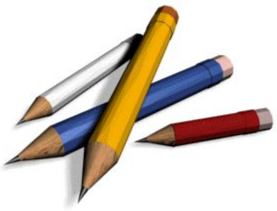
```
class StartState extends AbstractState {

AbstractState getNextState (char c) {
switch(c) {
case '1': return new StateOne();
case '2': return new StateTwo();
...
default: return new StateError();
}
}

boolean isFinalState() { return false; }

}
```



Лексический анализ

Процедура анализа последовательности символов будет иметь вид:

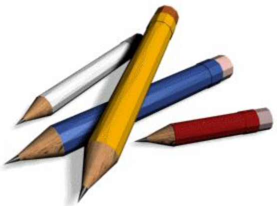
```
AbstractState state = new StartState();

for(int i=0;i<stroka.length();i++) {
state=state.getNextState(stroka.charAt(i));
}

if (state.isFinalState()) {
// Ok!
} else {
// Error!
}
```



Лексический анализ

Использование объектно-ориентированного программирования при реализации конечного автомата имеет ряд преимуществ.

Основное из них: значительно упрощается процесс добавления нового состояния КА (достаточно описать новый класс-состояние и внести изменение в те классы, из которых появляются новые переходы).

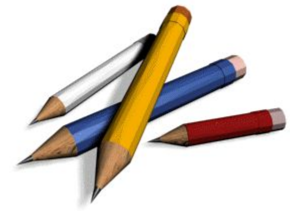


Таблично управляемый конечный автомат

Один из способов построения конечного автомата заключается в использовании таблично заданной функции переходов.

Основная идея программной реализации *таблично управляемого конечного автомата* заключается в следующей схеме:

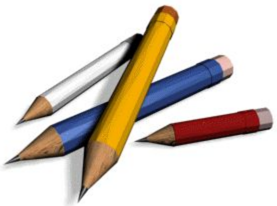
состояние' = таблица[состояние][символ]



Минимизация конечного автомата

Полная таблица переходов КА может быть очень большой, поэтому обычно используют различные алгоритмы *минимизации*.

Все алгоритмы минимизации основаны на *объединении нескольких различных состояний КА в одно.*



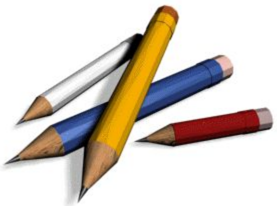
Лексический анализ

1 шаг

Множество состояний КА делится на две группы:

– множество заключительных состояний;

– множество всех остальных состояний.

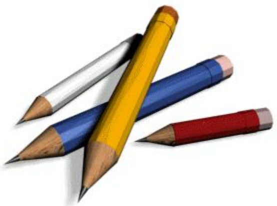


Лексический анализ

2 шаг

Каждая группа из получившего разбиения делится на подгруппы по следующему правилу

в одну подгруппу включаются такие состояния, из которых для каждого допустимого входного символа переходы осуществляются в состояния, принадлежащие одной и той же группе предыдущего разбиения.

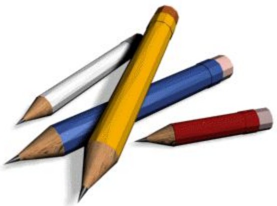


Лексический анализ

3 шаг

Заменить исходное разбиение на новое.

Шаги 2–3 должны повторяться до тех пор, пока разбиение на подгруппы не перестанет изменятся.



Лексический анализ

4 шаг

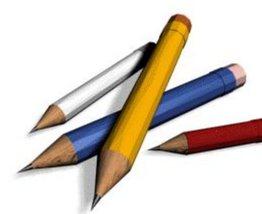
Новый конечный автомат формируется следующим образом:

– каждая группа получившегося разбиения становится состоянием нового КА;

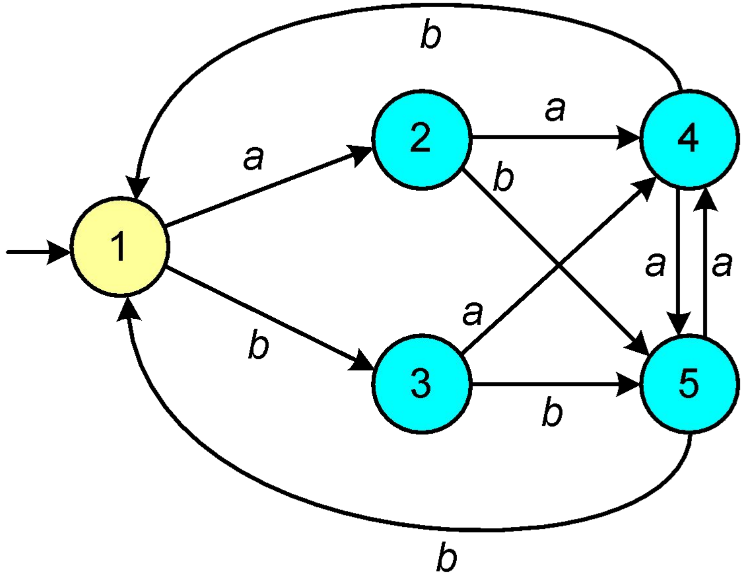
– группа, содержащая начальное состояние исходного КА, становится исходным состоянием нового КА;

– группы, содержащие заключительные состояния исходного КА, становятся заключительными состояниеми нового КА;

– функция переходов определяется очевидным образом.



Рассмотрим реализацию описанного алгоритма на примере следующего конечного автомата.



f	a	b
<u>1</u>	2	3
2	4	5
3	4	5
4	5	1
5	4	1

Шаг 1. Начальное разбиение множества состояний будет таким:

$G_1 = \{ 1 \}$ $G_2 = \{ 2, 3, 4, 5 \}$

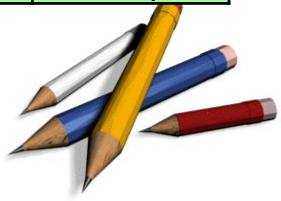
Шаг 2. Множество G_2 нужно разбить на две подгруппы: $\{ 2, 3 \}$ и $\{ 4, 5 \}$

Шаг 3. Новое разбиение будет таким

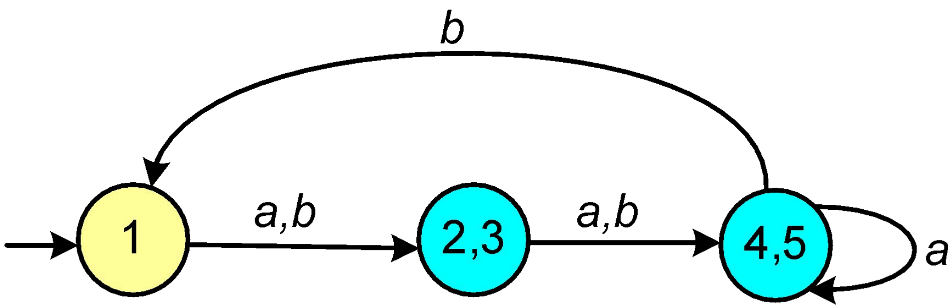
$G_1 = \{ 1 \}$ $G_2 = \{ 2, 3 \}$ $G_3 = \{ 4, 5 \}$

f	a	b
1	$2 \in G_2$	$3 \in G_2$
2	$4 \in G_3$	$5 \in G_3$
3	$4 \in G_3$	$5 \in G_3$
4	$5 \in G_3$	$1 \in G_1$
5	$4 \in G_3$	$1 \in G_1$

Выполнение еще одной итерации не приведет к изменению разбиения.



Шаг 4. В результате получим КА с 3 состояниями

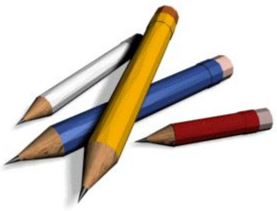


f	a	b
G_1	G_2	G_2
G_2	G_3	G_3
G_3	G_3	G_1

Построение конечного автомата по регулярному выражению

Другой способ автоматизации построения лексических анализаторов заключается в использовании регулярных выражений.

Основная идея заключается в построении *композиции* конечных автоматов, соответствующих подвыражениям исходного регулярного выражения.



Лексический анализ

На каждом шаге построения строящийся автомат имеет в точности одно заключительное состояние, в начальное состояние нет переходов из других состояний и нет переходов из заключительного состояния в другие.