Теория формальных языков и компиляторов
Часть 2. Конечно-автоматные распознаватели

Лекция 10. Свойства конечных автоматов

В этом разделе будет рассмотрен табличный способ задания функции переходов. Далее дадим определения детерминированных и недетерминированных автоматов. И рассмотрим пример программной реализации автоматного распознавателя.

10.1 Табличное задание функции переходов

В конце предыдущей лекции был рассмотрен полностью определенный конечный автомат:

$$A_1 = (\{I, M, B, F\}, \{a, b\}, \delta, I, \{F\}).$$

С функцией переходов:

$$\delta(I, a) = E; \delta(I, b) = B;$$

$$\delta (B, a) = M; \delta(B, b) = E;$$

$$\delta(M, a) = E; \delta(M, b) = \{B, F\}.$$

Более наглядное представление функции переходов можно получить, записав значения $\delta(s, c)$ в таблицу, как показано ниже.

Таблица 10.1

Полностью определенный конечный автомат A_1

Состояние	**Функция δ**	
	a	b
I	E	B
B	M	E
M	E	B, F
F	E	E
E	E	E

В таблице по строкам обозначены состояния, а по столбцам – символы алфавита. На пересечении строки и столбца указывается состояние, в которое происходит переход из состояния, соответствующего строке, и считывании символа, соответствующему столбцу. Так, если автомат находится в состоянии I и считывает символ b, то он перейдет в состояние B. Если же будет считан символ a, то возникнет ошибка (переход в E).

Из данной таблицы следует, что состояние ошибки E «замкнуто» само на себя, т.к. при любом символе автомат остается в данном состоянии.

Таким образом, КА может быть определен (своей функцией переходов) таблично. Соответствующий таблице граф полностью определенного конечного автомата показан на рис. 10.1.

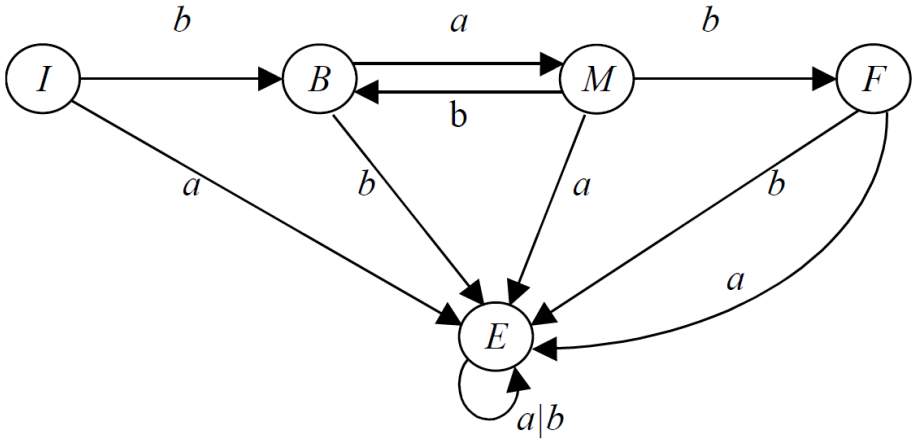


Рис. 10.1. Полностью определенный A_1

Представление автомата таблицей переходов имеет свои преимущества и недостатки. К преимуществам относится быстрое определение конфигурации автомата по входному символу над алфавитом Σ и текущему состоянию s∈S. Недостаток – громоздкость таблицы, когда алфавит велик и большинство переходов ведет в пустое множество состояний. Представление δ гораздо компактнее, однако, при этом трудно определить конфигурацию конечного автомата.

Приведем еще пример КА. Пусть конечный автомат A_2 задан табл. 10.2. В этом автомате, как видно из таблицы, отсутствуют переходы из состояний 1 и 2 по входу a. Соответствующий граф переходов показан на рис. 10.2.

Таблица 10.2

Автомат A_2

Состояние	**Функция δ**	
	a	b
0	0	0, 1
1	-	2
2	-	3

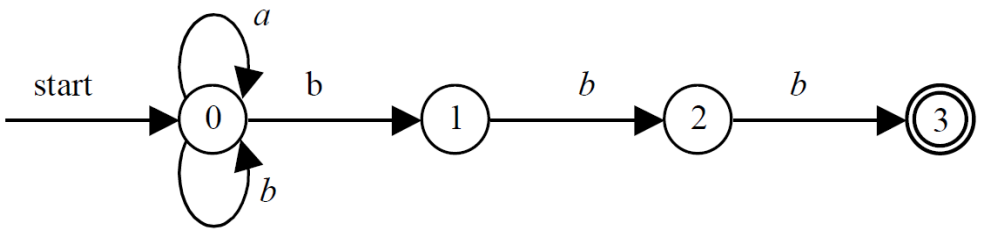


Рис. 10.2. Граф автомата A_2

Можно легко определить язык $L(A_2)$, допускаемый автоматом A_2:

$$L(A_2) = \{(a|b)^*bbb\}.$$

Все допустимые цепочки (слова) можно показать с помощью так называемых перемещений – move. Например, слово bbb $\in$ $L(A_2)$ имеет перемещение:

$$0 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{b} 3.$$

Заметим, что move может и не привести в заключительное состояние. Так, цепочка abb имеет move:

$$0 \xrightarrow{a} 0 \xrightarrow{b} 1 \xrightarrow{b} 2.$$

10.2 Детерминированные и недетерминированные автоматы

Рассмотрим конечный автомат, представленный на рисунке 10.3. Такой автомат является **недетерминированным** (НКА).

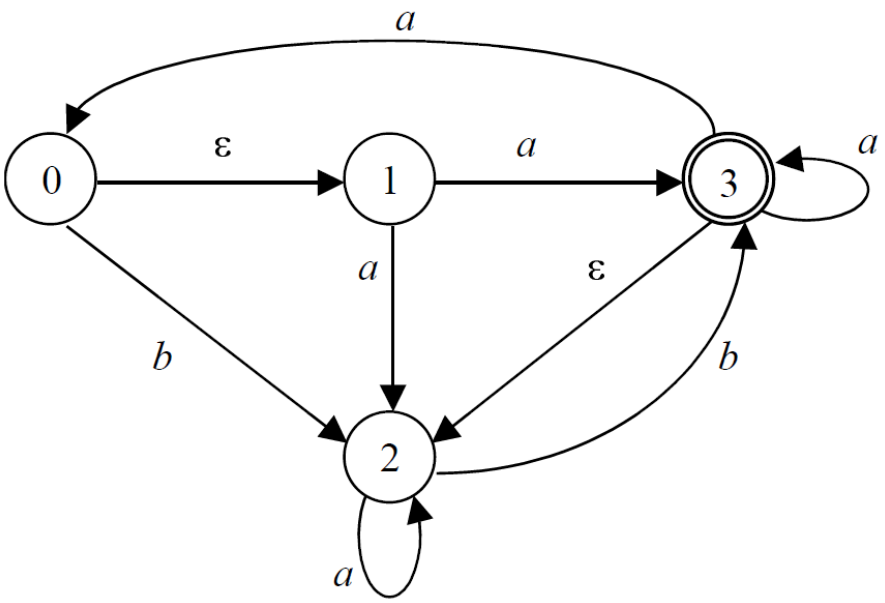


Рис. 10.3. Недетерминированный автомат N

НКА характеризуются следующими **особенностями**:

- либо существует переход по пустому символу ε (в данном примере – из состояния 0 в 1 и из 3 в 2),

- либо из одного состояния выходят несколько переходов, помеченных одним и тем же символом из словаря (здесь это состояния 1 и 3).

Переход из состояния 0 в состояние 1 может быть выполнен **спонтанно в любой момент времени без подачи входного сигнала** из Σ. Переход из состояния 1 по входному символу a разрешает автомату перейти либо в 3, либо в 2 (еще одна неопределенность).

Рассмотренный НКА не является полностью определенным.

Определение 10.1. Недетерминированным конечным автоматом N (Nondeterministic finite automation) называется пятерка

$$N = (S, \Sigma, I, \delta, F),$$

где S – конечное непустое множество (состояний); Σ – конечное непустое множество входных символов (входной словарь); $I \subseteq S$ – множество начальных состояний; δ: $S \times (\Sigma \cup \varepsilon) \to 2^S$ – функция переходов. Здесь 2^X – обозначен булеан X, т.е. множество всех подмножеств множества X; $F \subseteq S$ – множество заключительных состояний.

Слово «недетерминированный» вовсе нельзя читать как «неопределенный». Действительно, автомат, показанный на рис. 10.3, может спонтанно (без подачи входного символа) перейти в новое состояние 1 или остаться в прежнем 0. Автомат принимает, например, цепочку ω = abb. При этом путь, помеченный ω, выглядит так:

$$\begin{array}{ccccccc} & \omega & & a & & b & & b \\ 0 & \to & 1 & \to & 2 & \to & 2 & \to & 3. \end{array}$$

Определение 10.2. Недетерминированный конечный автомат N распознает входную цепочку ω, если существует путь, помеченный символами цепочки ω из начального в одно из заключительных состояний автомата возможно, с учетом ε-переходов. Недетерминированный КА распознаёт язык L(N), если он распознаёт все цепочки этого языка.

Распознает ли автомат N (рис. 10.3) цепочку bb? Да, поскольку есть путь из начального 0-состояния в заключительное – 3.

На рис. 10.4 показан детерминированный конечный автомат (ДКА) D, заданный своим графом.

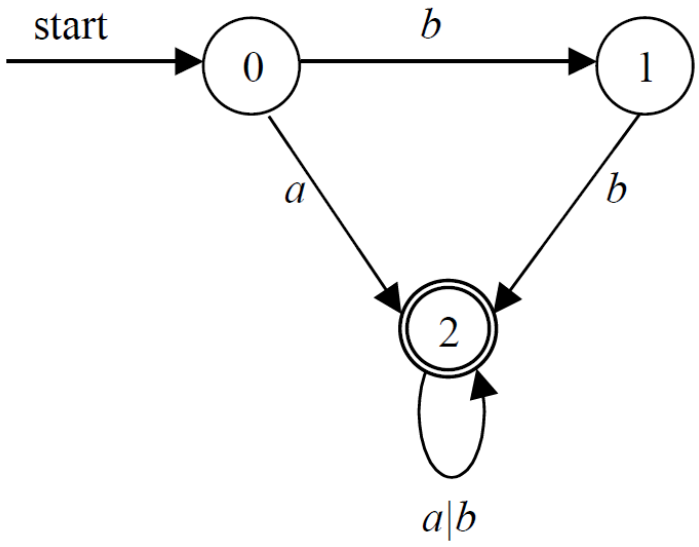


Рис. 10.4. Детерминированный автомат D

Автомат D, как легко убедиться, «путешествуя» по графу из начального состояния 0 в конечное 2, допускает тот же язык L(D), что и недетерминированный автомат N. Такие автоматы называют эквивалентными.

Определение 10.3. Два автомата $A_1 = (S_1, \Sigma_1, \delta_1, s_1, F_1)$ и $A_2 = (S_2, \Sigma_2, \delta_2, s_2, F_2)$ называются эквивалентными, если они распознают один и тот же язык.

Снова отметим особенности D в отличие от N:

- отсутствие ε-переходов;
- для каждого состояния s и входного символа t существует не более одной дуги, исходящей из s, помеченной как t.

Детерминированный конечный автомат (ДКА) имеет для входного символа не более одного перехода из каждого состояния. Табличное представление D отличается от N тем, что каждая запись в таблице всегда имеет одно состояние. Поэтому проще, чем в N, проверятся принадлежность цепочек языку L(D), так как имеется не более одного пути – из начального состояния в заключительное.

Определение 10.4. Конечный автомат D = (S, Σ, δ, s_0, F) называется детерминированным конечным автоматом (Deterministic finite automata), если в каждом его состоянии s∈S функции переходов δ(s, c), для любого входного символа c∈Σ содержит не более одного состояния, т.е.

$$\forall c \in \Sigma, \forall s \in S: \delta(s, c) = \{r\}, r \in S \text{ или } \forall c \in \Sigma, \forall s \in S: \delta(s, c) = 0.$$

ДКА значительно проще НКА и с точки зрения программной реализации. Далее будут рассмотрены примеры написания распознающей функции для детерминированного автомата.

Теорема 10.1. Для любого недетерминированного конечного автомата существует эквивалентный ему детерминированный конечный автомат. Доказательство этой теоремы выполняется конструктивным методом. Это означает, что по заданному недетерминированному конечному автомату N строится эквивалентный детерминированный конечный автомат D по определенному конечному алгоритму.

Так, недетерминированному автомату N, приведенному на рис. 10.5, эквивалентен детерминированный конечный автомат, представленный на рис. 10.6. Оба автомата принимают цепочки языка L = {(a|b)*abb}.

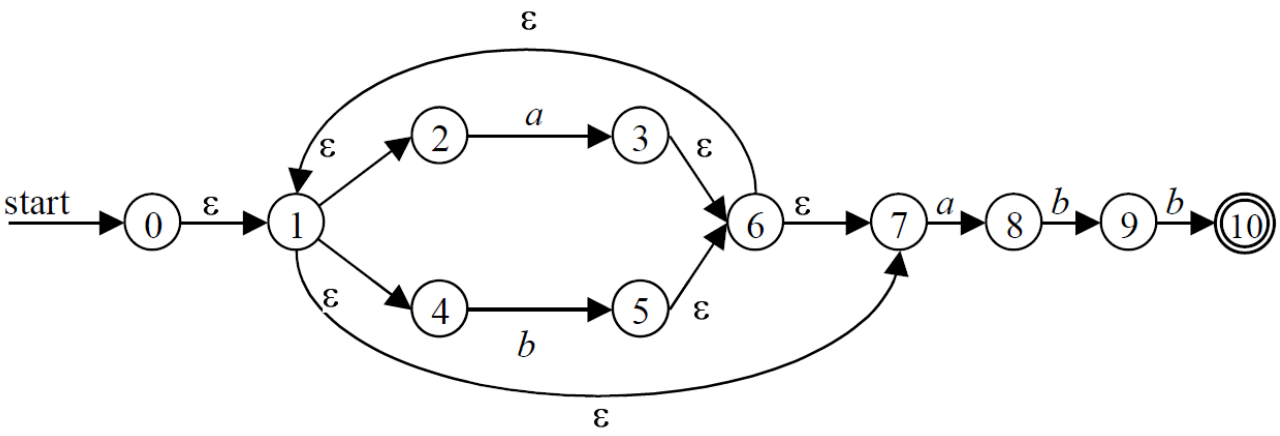


Рис. 10.5. Недетерминированный автомат для L = {(a|b)*abb}

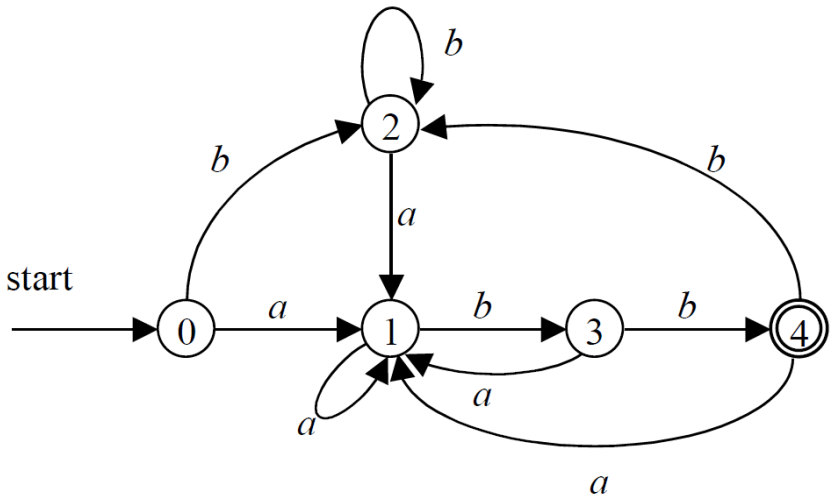


Рис. 10.6. Детерминированный автомат для L = {(a|b)*abb}

10.3 Программная реализация автоматного распознавателя

Рассмотрим пример программной реализации автомата-распознавателя, построенного с учетом рассмотренных теоретических положений. Пусть необходимо создать синтаксический анализатор, который проверяет введенную строку на принадлежность языку L = {(a|bb)(a|b)*}. Примерами корректных строк являются следующие: «a», «bb», «aaab» и т.д.

Можно построить автомат-распознаватель, определяющий данный язык. Его графическое представление изображено на рис. 10.4. Запишем табличное представление данного автомата:

Таблица 10.3

Автомат D

Состояние	**Функция δ**	
	a	b
0	2	1
1	-	2
2	2	2

Приведем пример простой программной реализации данного распознавателя. Пусть в каждом состоянии автомат вызывает соответствующую функцию для анализа текущего символа из входной

последовательности. Обозначим эти функции: в состоянии 0 анализ производится функцией state0(), в состоянии 1 – функцией state1() и т.д. Если текущий входной символ приводит к переходу автомата в новое состояние, то вызывается соответствующая функция. Если такого перехода нет (переход в состояние ошибки), то автомат сообщает об отрицательном результате разбора. Разбор будет завершен с корректным результатом, если получен признак конца строки и автомат находится в одном из конечных состояний. В остальных случаях автомат также сообщает об ошибке. Такими ситуациями могут быть: 1) все символы считаны, а автомат не перешел в конечное состояние; 2) автомат находится в конечном состоянии, во входной последовательности есть необработанные символы, но нет возможности перейти в новое состояние.

Пусть функция `char getSymbol()` возвращает очередной символ из входной последовательности. Тогда можно записать функцию, описывающую логику работы автомата в состоянии 0:

```
bool state0() {
    char symbol = getSymbol();
    if(symbol == ‘a’) return state2();
    if(symbol == ‘b’) return state1();
    return false;
}
```

В данном случае если текущий символ – ‘a’, то вызывается функция `state2()`, имитирующая смену состояния автомата. Если считан символ ‘b’, то вызовется функция `state1()`. В зависимости от результата работы соответствующей функции, будет возвращено логическое значение: `true` – разбор корректен или `false` – ошибка. В остальных случаях автомат вернет `false`, что означает ошибку разбора. При написании данной функции удобно пользоваться таблицей 10.3, чтобы учесть все возможные варианты изменения состояния автомата.

Аналогичным образом можно записать функцию `state1()`:

```
bool state1() {
     char symbol = getSymbol();
     if(symbol == ‘b’) return state2();
     return false;
}
```

В данном случае переход в следующее состояние возможен только когда текущий символ – ‘b’. При этом вызывается функция `state2()`. В остальных случаях автомат вернет `false`, что означает ошибку разбора.

Наконец, функция `state2()` записывается в виде:

```
bool state2() {
     char symbol = getSymbol();
     if(symbol == ‘a’) return state2();
     if(symbol == ‘b’) return state2();
     if(symbol == ‘\0’) return true;
     return false;
}
```

Здесь первый и второй условный оператор можно объединить, но показанный подход позволяет легко изменить логику поведения функции в случае, если требуется поменять конфигурацию автомата. Например, сделать так, чтобы символ ‘a’ переводил автомат в состояние 0. Тогда потребуется только изменить имя вызываемой функции.

При написании функций, соответствующих конечным состояниям, нужно предусмотреть случай корректного завершения разбора. Это сделано в третьем условном операторе: если строка завершилась, функция возвращает true.

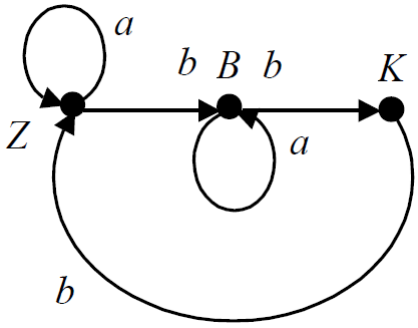
В результате, для запуска автомата необходимо настроить чтение разбираемой строки из входного потока (с клавиатуры, из файла, буфера или другого источника) и вызвать функцию `state0()`. В зависимости от значения, которое вернула функция, можно сообщить об успешности разбора:

```
if(state0()) cout << “Строка записана корректна (принадлежит языку L = {(a|bb)(a|b)*})”;
else cout << “Строка не принадлежит языку L = {(a|bb)(a|b)*})”;
```

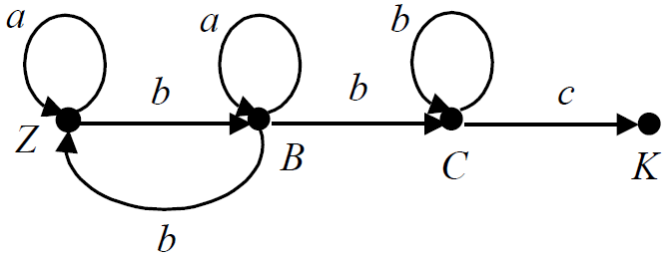
Упражнения

В №№ 1 – 4 записать табличное задание функции переходов и показать изменение состояния (перемещения) автомата при разборе 3…5 правильных и 3…5 ошибочных цепочек. Определите, является ли автомат детерминированным. После этого запишите фрагмент программы-распознавателя на базе данного конечного автомата.

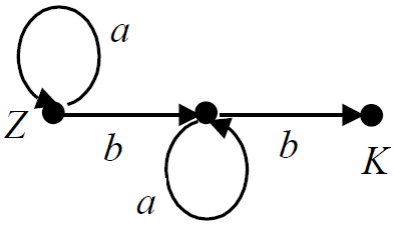
1.



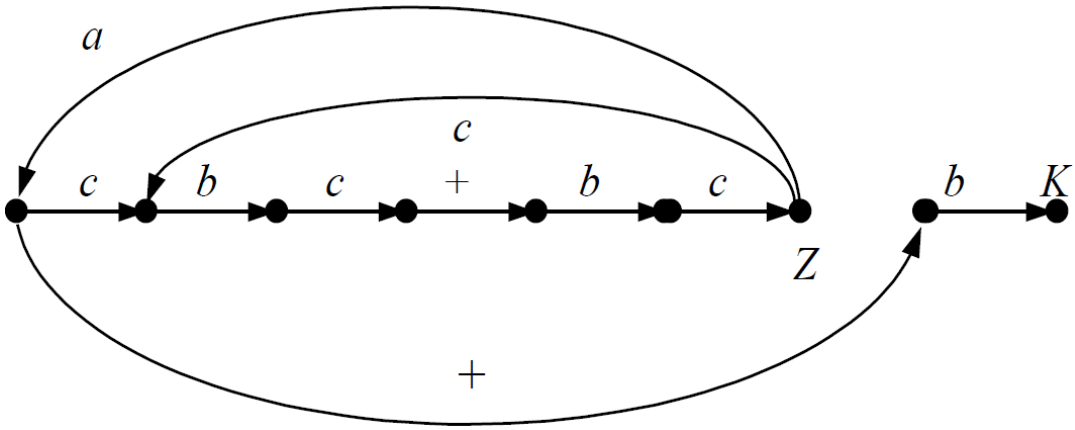
2.



3.



4.



Список использованных источников

1. Шорников Ю.В. Теория и практика языковых процессоров.