Анисимов Андрей Евгеньевич
Старший преподаватель Удмуртского государственного университета, один из соавторов книги «Сборник задач по основаниям программирования», которая является практическим приложением к фундаментальному труду Н.Н. Непейводы и И.Н. Скопина «Основания программирования», а также к книге Н.Н. Непейводы «Стили и методы программирования».

Автоматное программирование. Часть 2

В первой части статьи (журнал «Потенциал» № 3, 2008) были введены понятие конечного автомата и алгоритм его работы. Во второй части будет описано, как конечный автомат снабжают способностью к вычислениям и как писать для него программу.

4. Автоматы Мили: переходы и действия

Продолжим введение в увлекательный мир конечных автоматов. Конечные автоматы, рассмотренные в предыдущих главах, могли сделать только одно – допустить или не допустить входную цепочку. То есть определить, принадлежит ли она некоторому языку (регулярному множеству). Например, в примере 5 таким языком был язык дробных десятичных чисел (например, «3.1415926», «0.00001»), в примере 6 – язык идентификаторов («abcd», «File_Save»). Таким образом, результатом работы автомата является ответ «да» или «нет» на вопрос, допустима ли данная цепочка. Ничего более после своей работы конечный автомат не оставляет.

Однако в программировании (как и в жизни) часто бывает недостаточно простого ответа на простой вопрос. Нужно получить больше информации, сделав какие-либо вычисления, проверки, расчёты. То есть добиться большего результата, чем просто «да/нет».

Поэтому модернизируем наш конечный автомат[1] таким образом, чтобы он, помимо распознавания входной строки, умел ещё *попутно* делать некоторые вычисления, получать дополнительные результаты.

Для начала построим «обычный» конечный автомат A_5 и определим его язык. Множество состояний A_5 состоит из трёх элементов $\{q_0, q_1, q_2\}$; как обычно, начальное состояние – q_0, заключительное состояние – q_2. Входной алфавит

$$\sum = \{'+', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Для краткости записи (как в примере 5) в дальнейшем будем использовать слово «*digit*» для обозначения

[1] Не путать с АКМ! ☺

любой из десяти цифр алфавита.

Правила перехода конечного автомата A_5 приведены в таблице 1, граф состояний – на рисунке 1.

Таблица 1

Состояние	Условие	Переход
q_0	*digit*	q_1
q_1	*digit*	q_1
	'+'	q_0
	¶	q_2
q_2		

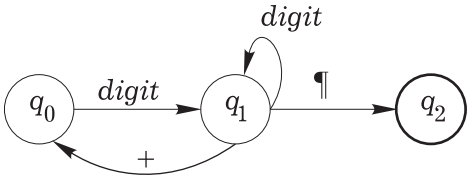


Рис. 1. Граф состояний автомата A_5

Попробуем «запустить» автомат A_5 на разных входных цепочках. Убедитесь самостоятельно, что цепочки «100», «1+2+3», «123+1+56» автомат допускает, а такие цепочки, как «+1», «5+», «3++2» – нет. Можно неформально описать язык конечного автомата A_5 так: «непустая последовательность целых неотрицательных чисел, разделяемых знаком плюс» [1].

Итак, мы сконструировали конечный автомат, позволяющий распознавать *запись суммы целых неотрицательных чисел*. А теперь давайте «научим» этот автомат ещё и вычислять значение этой суммы!

Для этого возможностей конечного автомата недостаточно. Дело в том, что после распознавания входной цепочки автомат не сохраняет никаких результатов вычислений. Да он, собственно, никаких вычислений-то почти не производит. Как же нам посчитать сумму?

Для этого поступим так. Снабдим конечный автомат дополнительной возможностью *выполнять определённые действия при совершении переходов*. То есть укажем, что при каждом переходе у автомата не только меняется состояние и сдвигается головка вправо по ленте, но и производятся некоторые действия. То есть свяжем каждый переход с определёнными вычислениями. Такой автомат называется *автоматом Мили*.

Вспомним, что правило для каждого перехода в таблице переходов автомата записывается отдельной

[1] Для самых любознательных приведём формальное определение языка: $\left\{digit^k \left\{+digit^*\right\} | k \geq 1\right\}$.

строкой. Для связывания действий с переходами добавим в нашу таблицу к трём уже имеющимся столбцам «Состояние», «Условие» и «Переход» ещё один столбец «Действие», в который будем помещать описание тех действий, которые необходимо совершить при каждом переходе автомата. Этот столбец пусть будет предпоследним в таблице (см. таблицу 2).

Таблица 2

Состояние	Условие	Действие	Переход
...	...	...	...

Ну а теперь построим те вычисления, которые должен выполнить конечный автомат A_5 для нахождения искомой суммы. Состояние q_0 – начальное, с него начинается работа. Выходящая из него стрелка *digit* (на графе, см. рис. 1) – это первая цифра числа в строке. Пусть искомое число мы будем накапливать в переменной n. Так как цифра самая первая, то эту цифру просто запишем в n (вот это и есть действие, которое мы связываем с переходом из состояния q_0 в состояние q_1 по символу *digit*).

Далее. Кольцевая стрелка *digit* из состояния q_1 в само q_1 позволяет последовательно распознать остальные цифры этого числа. Значит, при переходе по этой стрелке необходимо каждую новую цифру *приписывать справа* к уже имеющемуся числу n. Например, если в n уже находятся цифры «12», то после приписывания новой цифры «3» получится «123».

Теперь рассмотрим стрелку «+» из состояния q_1 в q_0. Эта стрелка означает, что закончена запись очередного числа и после него находится символ «+». Так как запись числа полностью распознана и его значение записано в n, то это число можно суммировать. Для накопления суммы чисел введём переменную s, значение которой начинается с нуля. Для суммирования найденного числа n увеличим значение переменной s на n. После этого перехода мы вновь оказываемся в q_0 и начинаем распознавать следующее число и так далее...

Процесс распознавания строки заканчивается тогда, когда автомат встретит после записи очередного числа маркер конца строки ¶. Такому переходу мы назначим действие суммирования последнего числа нашей строки. Переход в заключительное состояние q_2 останавливает работу автомата, и при этом в переменной s будет накоплен требуемый результат – сумма чисел строки. Например, для входной строки «123+1+56» значение s будет равно 180. Что и требовалось найти.

Итак, такое довольно длинное словесное описание конечного автомата A_5, снабжённого действиями для вычисления суммы, можно гораздо короче представить в виде таблицы переходов с действиями (таблица 3).

Дадим некоторые пояснения. Первая строка этой таблицы не содержит состояния. Там находятся действия, которые необходимо выполнить *до начала работы* конечного автомата. Последняя строка таблицы тоже не содержит состояний – это действия, которые необходимо выполнить *после окончания* успешной работы автомата. Теперь про сами действия. Действие $n = digit$ – это присваивание числу первой цифры. Действие $n = n * 10 + digit$ – это приписывание справа к числу n очеред-

ной цифры (например, чтобы приписать к числу 12 цифру 3, надо сделать следующее – 12*10+3, получится 123). Действие $s = s + n$ есть прибавление значения числа n к искомой сумме s.

Ну а теперь проанализируем процесс распознавания и вычисления конечным автоматом A_5 строки «123+1++56». Все переходы и действия автомата по шагам представлены в таблице 4.

Таблица 3

Состояние	Условие	Действие	Переход
		$s = 0$	
q_0	*digit*	$n = digit$	q_1
q_1	*digit*	$n * 10 + digit$	q_1
	'+'	$s = s + n$	q_0
	¶	$s = s + n$	q_2
q_2			
		Выход: s	

Таблица 4

№ шага	Состояние	Условие	Действие	Результат		Переход
				n	s	
0	(до начала)		$s = 0$		0	
1	q_0	1 (это *digit* !)	$n = digit$	1		q_1
2	q_1	2	$n = n * 10 + digit$	12		q_1
3	q_1	3	$n = n * 10 + digit$	123		q_1
4	q_1	+	$s = s + n$		123	q_0
5	q_0	1	$n = digit$	1		q_1
6	q_1	+	$s = s + n$		124	q_0
7	q_0	5	$n = digit$	5		q_1
8	q_1	6	$n = n * 10 + digit$	56		q_1
9	q_1	¶	$s = s + n$		180	q_2 (стоп!)
10	(после конца)		Выход: s (то есть 180)			

Итак, автомат не просто распознал и допустил строку «123+1+56», но попутно вычислил значение суммы целых чисел в ней, равное 180.

Для закрепления материала этой главы рекомендуется выполнить упражнение 6.

5. Написание программы конечного автомата

Итак, понятие конечного автомата введено. Были рассмотрены примеры конечных автоматов. Также стало известно, как наделить автомат способностью производить вычисления (автомат Мили). Трудолюбивый читатель, надеемся, внимательно разобравший примеры и выполнивший упражнения, также научился имитировать работу конечного автомата на бумаге. Теперь пришла очередь программировать.

Нашей целью является написание программы, имитирующей работу конечного автомата, задаваемого формально, то есть заданного в виде функции (таблицы) переходов. Точнее, познакомимся с методикой создания таких программ, позволяющей программисту почти *механически* писать тексты программ по заданному конечному автомату.

Для рассмотрения этой методики вернёмся к автомату A_5, успешно нами построенному в предыдущей главе. Напомним, что этот автомат умеет допускать «строки, состоящие из целых неотрицательных чисел, разделённых знаком плюс». Также мы снабдили автомат способностью находить сумму этих чисел, выполняя определённые действия при каждом переходе, сделав его автоматом Мили (см. таблицу 3).

Идея программы, которую мы будем писать, такова: каждому *состоянию* конечного автомата мы сопоставим *фрагмент программы*, который будет выполнять действия и переходы автомата для этого состояния. Фрагменты пометим метками, а переходы от состояния к состоянию (то есть от фрагмента к фрагменту) будут основаны на операторе безусловного перехода **goto**. Казалось бы, что **goto** надо избегать в программах всеми способами, это стало де-факто правилом в практическом программировании. Но! Как раз безусловные переходы в программе наиболее точно соответствуют рассматриваемому стилю автоматного программирования. Ведь автомат всегда находится в одном состоянии и после проверки условия однозначно «знает», куда дальше двигаться без полного перебора возможных путей[1].

В программе будут использованы переменные: **char ch** – текущий символ входной строки; **int n,s** – это переменные для накопления результата. Также используем две вспомогательные функции:

- **int is_digit (char c)** – возвращает логическое значение «истина», если входной символ *c* является цифрой;
- **int digit (char c)** – возвращает целочисленное значение цифры, представленной символом *c*.

Начало программы получается такое:

```
#include <stdio.h>
char ch;                    //текущий символ
int n,s;                    //переменные для накопления результата

int is_digit(char c)     //возвращает Истина, если с – цифра
  {return('0'<=c) && (c<='9');}
```

[1] Подробнее об особенностях программных реализаций конечных автоматов см. в книгах [1] и [2].

```
int digit(char c)       //возвращает числовое значение символа c
  {return int(c-'0');}
```

Функция void main() будет содержать основную часть программы. Её мы будем строить в точном соответствии с таблицей автомата A_5, снабжённого действиями (таблица 3). До начала работы автомата нужно выполнить действие – обнуление переменной s (первая строка таблицы):

```
s=0;                    //действие до начала работы автомата,
```

затем поместим головку автомата на первый символ входной строки:

```
ch=getchar();
```

то есть присвоим переменной **ch** значение первого символа входного потока с помощью стандартной функции **getchar()**, которая при каждом обращении к ней забирает из потока очередной символ и возвращает его. Значит, применение **getchar()** с точки зрения автомата – это сдвиг головки на одну позицию вправо по строке.

Ну а теперь переходим к главному – состояниям автомата. Начнём с q_0. В соответствии с таблицей 3 в состоянии q_0 на входе может быть только входной символ **digit** (цифра), при этом автомат совершает следующие операции:

а) действие **n = digit;**

б) сдвиг головки по входной строке вправо;

в) переход в новое состояние q_1.

Любой другой символ, кроме цифры, будет приводить к ошибке и остановке автомата. Эта ситуация в программе для q_0 может быть проанализирована так:

```
q0:
 if (is_digit(ch))
 {
    n=digit(ch);     //действие
    ch=getchar();    //сдвиг по входной строке вправо
    goto q1;         //переход в новое состояние
 }
  else
 {
    printf("Error!\n");
    return;          //остановка в случае ошибки
 }
```

Аналогично поступим с состоянием q_1. Только здесь на входе допустимы три различные символа (а не один, как с q_0) – цифр **digit**, знак '+' и конец строки '¶'. Поэтому разбор случаев здесь глубже.

```
q1:
   if (is_digit(ch))
   {
     n=n*10+digit(ch);  //действие
     ch=getchar();      //сдвиг
     goto q1;           //переход
   }
   else
   {
     if (ch=='+')
     {
       s=s+n;                    //действие
       ch=getchar();             //сдвиг
       goto q0;                  //переход
     }
     else
     {
       if (ch=='\n')
       {
         s=s+n;                  //действие
         goto q2;                //переход
       }
       else
       {
         printf("Error!\n");
         return;                //остановка в случае ошибки
       }
     }
   }
```

В этом фрагменте обратим внимание на отсутствие сдвига по символу '¶' (в языке С конец строки обозначается **'\n'**). Дело в том, что это маркер конца строки и сдвигаться дальше уже некуда.

Осталось рассмотреть состояние q_2. Оно заключительное, поэтому ни действий, ни сдвигов, ни переходов не совершается. Поэтому фрагмент пуст:

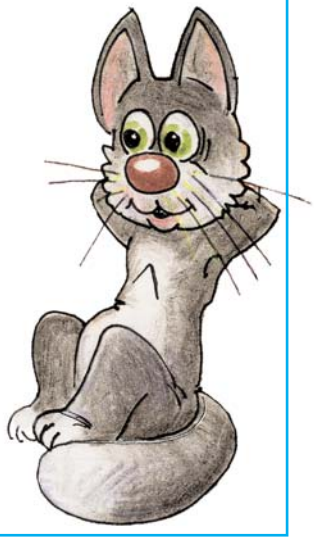
```
q2:   //заключительное состояние - остановка
```

В этот момент конечный автомат должен остановиться, успешно допустив строку. Однако в соответствии с таблицей 3 необходимо выполнить *действия после остановки автомата* (самая последняя строка таблицы). Запишем и их:

```
printf("s=%d\n",s);    //действие после остановки автомата
return;               //конец работы программы
```

Итак, по частям программа построена, приведём здесь её окончательный полный текст.

```
#include <stdio.h>
char ch;                  //текущий символ
int n,s;                  //переменные для накопления результата

int is_digit(char c)      //возвращает Истина, если с – цифра
  {return('0'<=c) && (c<='9');}

int digit(char c)         //возвращает числовое значение символа с
  {return int(c-'0');}

void main()
{
  s=0;                    //действие до начала работы автомата
  ch=getchar();

  q0:
    if (is_digit(ch))
    {
      n=digit(ch);        //действие
      ch=getchar();       // сдвиг по входной строке вправо
      goto q1;            //переход в новое состояние
     }
     else
     {
       printf("Error!\n");
       return;            //остановка в случае ошибки
     }

  q1:
    if (is_digit(ch))
    {
      n=n*10+digit(ch);//действие
      ch=getchar();     //сдвиг
      goto q1;          //переход
    }
     else
    {
       if (ch=='+')
       {
         s=s+n;           //действие
         ch=getchar();    //сдвиг
         goto q0;         //переход
       }
       else
       {
         if (ch=='\n')
```



```
        {
          s=s+n;              //действие
          goto q2;            //переход
        }
        else
        {
          printf("Error!\n");
          return;             //остановка в случае ошибки
        }
      }
    }

  q2:                        //заключительное состояние - остановка

  printf("s=%d\n",s);    //действие после остановки автомата
  return;                //конец работы программы
}
```

Предлагаем этот текст программы набрать, отладить и протестировать в какой-нибудь среде языка C/C++. Убедитесь, что конечный автомат A_5 и данная программа эквивалентны, то есть работают одинаково.

Важнейший вывод по разобранному примеру таков: структура программы построена и полностью соответствует таблице переходов конечного автомата A_5 с действиями (таблица 3). Таким образом, на этом примере рассмотрена *одна из методик построения программы*, которая реализует конечный автомат, заданный своей таблицей переходов.

Безусловно, такой подход к написанию программы далеко не единственный. В соответствии с [2] можно вообще указать три подхода к созданию программ – конечных автоматов:

а) «ручное» написание программы-автомата, как это было сделано выше; однако существуют и другие подходы к организации структуры такой программы – на основе циклов, оператора выбора [1], рекурсии, вызовов подпрограмм и другие;

б) «универсальная программа», на вход которой подаётся определение автомата в виде таблицы переходов, а программа затем эмулирует конечный автомат (интерпретация);

в) «автоматический транслятор», на вход которого подаётся определение автомата (таблица переходов), а на выходе – текст программы, как в п. *а*), только созданной не вручную, а автоматически.

Для закрепления материала этой главы рекомендуется выполнить упражнения 7 и 8.

6. Выводы

Предложенная здесь методика построения конечного автомата в виде таблицы переходов, наделения его способностью производить вычисления и дальнейшего построения программы конечного автомата проста и

[1] В языке C/C++ таким оператором является **switch**, позволяющий выбирать одну из нескольких ветвей. Поэтому прикладное применение автоматного стиля программирования называют **switch**-программированием.

понятна. Это делает идею автоматного программирования, суть которого сводится к программной реализации какого-либо конечного автомата, весьма доступной даже для начинающих программировать. Другое дело ответить на вопрос: *а когда нужно использовать методы конечных автоматов?*

В практике программирования часто возникают задачи, связанные с обработкой последовательностей, потоков данных, на которых необходимо произвести некоторые проверки или вычисления. Можно даже обобщить – большинство задач можно свести к проблемам обработки потоков. И вот здесь программисту нужно уметь увидеть ту часть задачи, для которой наиболее естественно применение методов и приемов стиля автоматного программирования.

Вряд ли можно назвать истиной утверждение, что «много задач на практике решается *исключительно* с помощью автоматов». Как правило, в программировании не бывает чистого применения одного из стилей – чаще возникает смешанное применение методик. Например, часто возникает сочетание объектно-ориентированного, автоматного стилей и программирования от событий.

Кроме этого, методика, предложенная в статье, вряд ли может быть использована прямолинейно при решении всех подобных задач. Однако чтобы уяснить идею автоматного программирования, необходимо чётко и ясно представлять себе, что такое конечный автомат, как он работает и вычисляет, как создавать для него программу. Получение навыков в программировании конечных автоматов – важная и необходимая составляющая в профессиональной подготовке современного программиста.

В заключение хотелось бы рекомендовать читателю продолжить изучение стиля автоматного программирования в различных источниках, в частности, по книге Н.Н. Непейводы «Стили и методы программирования» [2] и на материалах сайта кафедры «Технологии программирования» СПбГУ ИТМО[1] [4].

Упражнения

Здесь даны несколько полезных упражнений для практического закрепления материала второй части статьи. Рекомендуется выполнить не только упражнения «на бумаге», но и написать (где это требуется) и отладить программы. Ряд задач ссылается на упражнения из первой части статьи (см. «Потенциал» № 3, 2008 г.)

Упражнение 6. Определите конечный автомат Мили в виде *таблицы переходов с действиями* для всех автоматов из упражнения 5. В качестве действий можно провести следующие вычисления:

а) найти десятичную дробь, равную данной обыкновенной;

б) найти значение данного выражения;

в) проверить корректность даты;

г) найти цену в копейках (или перевести в доллары по текущему курсу).

[1] Что характерно – четырёхкратные чемпионы мира по программированию.

Упражнение 7. Напишите программу конечного автомата для данных в статье и упражнениях определений:

а) для автоматов

$A_1, A_2, A_3, A_4, A_{y_1}, A_{y_2}, A_{y_3}, A_{x_1}, A_{x_2}$

(автоматы без действий).

б) для автоматов Мили из упражнения 6 (автоматы с действиями).

Упражнение 8. Самостоятельно придумайте какой-нибудь язык (регулярное множество допустимых строк), для него определите конечный автомат в виде графа состояний и таблицы переходов; придумайте вычисления, которыми вы можете наделить автомат и напишите его таблицу переходов с действиями. По этой таблице напишите программу, реализующую полученный автомат Мили.

Список рекомендуемой литературы

1. Непейвода Н.Н., Скопин И.Н. Основания программирования. Москва–Ижевск: РХД, 2003. 880 с.

2. Непейвода Н.Н. Стили и методы программирования. – М.:ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2005. 320 с.

3. Сборник заданий по основаниям программирования: Учебное пособие/ Анисимов А.Е., Пупышев В.В. – М.:ИНТУИТ.РУ «Интернет-Университет Информационных Технологий»; БИНОМ. Лаборатория знаний, 2006. 348 с.

4. http:://is.ifmo.ru: сайт по автоматному программированию и мотивации к творчеству//Кафедра «Технологии программирования» Санкт-Петербургского государственного университета информационных технологий, механики и оптики.

Вопросы Вопросы Вопросы Вопросы

Начало на стр. 37

2. Почему неожиданно появляются и исчезают волны-гиганты? Известно немало историй о том, как тот или иной корабль, когда в море нет сильного волнения, неожиданно встречается с огромной волной высотой 25-35 метров. Представляете, каково попасть под 35-метровую волну! Какова причина возникновения таких мощных волн?

3. Можно ли услышать броуновское движение? Всем известно, что звук – это колебания воздуха и слышим мы его потому, что на ушную барабанную перепонку оказывается при этом переменное давление. Но ведь давление воздуха вблизи перепонки непрерывно изменяется вследствие броуновского движения его молекул. Не должны ли мы из-за этого ощущать постоянный шум в ушах? Если да, то почему мы не слышим броуновского движения?

Редакция журнала предлагает читателям ответить на вопросы на страницах 37, 52 и до 1 августа прислать свои ответы в редакцию по адресу: 109544 г. Москва, ул. Рабочая, д. 84, редакция журнала «Потенциал». Верные ответы на предлагаемые вопросы будут опубликованы в № 9 журнала «Потенциал» 2008 года. Тем, кто ответит на все вопросы правильно, редакция подготовила приз – бесплатные номера журнала «Потенциал» до конца 2008 года.