С авторами можно связаться по адресу: mail@avrorasystems.com, "для Шалыто".

Краткая версия статьи опубликована в журнале "Мир ПК", 2002. №2, С.144-149.
Более полная версия статьи доступна на сайте журнала "Мир ПК" (www.pcworld.ru).

От тьюрингова программирования к автоматному

© 2001 г. А.А. Шалыто, Н.И. Туккель

Федеральный научно-производственный центр – ФГУП "НПО "Аврора"",
Санкт-Петербургский государственный институт точной механики и оптики (технический университет)
Санкт-Петербург

Предлагается технология автоматного программирования, основанная на модели, расширяющей классическую модель машины Тьюринга и позволяющей единообразно и эффективно решать практические задачи со сложным поведением.

> В понимании теории программ большую помощь могут оказать один-два оператора, описывающих состояние машины...
>
> А.Тьюринг [1]

Известна роль машин Тьюринга как универсального средства реализации алгоритмов [2,3]. В [3] программирование этих машин названо "тьюрингово программирование". Однако применение такой парадигмы программирования на практике невозможно из-за чрезвычайной простоты тьюринговых операций и крайне ограниченного доступа к внешней памяти машины, что приводит к большой трудоемкости написания программ этого типа и значительному числу шагов при их выполнении. Так, например, умножение на машине Тьюринга двух на три требует (при задании этих чисел штрихами на ленте) 188 шагов [2].

Настоящая работа посвящена преобразованию классической модели машины Тьюринга в другую модель, обеспечивающую переход от тьюрингова программирования к автоматному. Последнее весьма эффективно для систем со сложным поведением, характерным для задач управления или сводимым к ним, как показано ниже.

Классическая схема машины Тьюринга (рис. 1), состоящая из абстрактного конечного автомата и бесконечной ленты, может быть изображена по-другому (рис. 2). Такое изображение характерно для теории управления и содержит контур обратной связи. Отметим, что на этих рисунках не изображен выход автомата, обеспечивающий управление передвижением ленты.

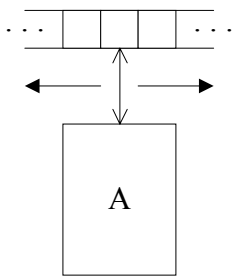

Рис. 1. Машина Тьюринга

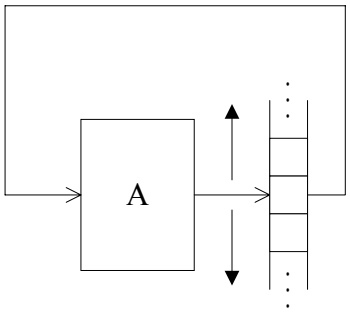

Рис. 2. Другое изображение машины Тьюринга

Абстрактный автомат, применяемый в машине Тьюринга, сам по себе не является универсальным и не удобен для практического применения. Второй из указанных недостатков снимается при переходе от абстрактного автомата к структурному (рис. 3), главное отличие которого заключается в параллелизме по входам и выходам.

Рассмотрим предлагаемую модель (рис. 4), базирующуюся на структурном автомате, в которую для обеспечения универсальности и практического применения введены формирователи входных воздействий 1, а самое главное – объекты управления 2.

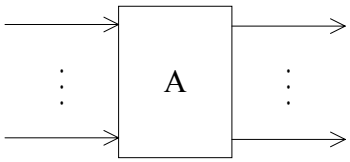

Рис. 3. Структурный автомат

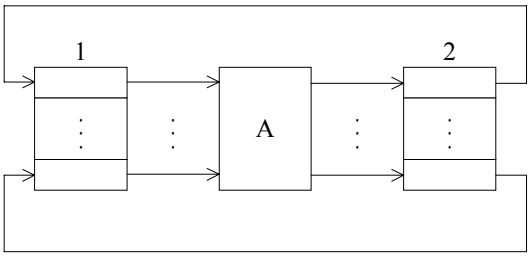

Рис. 4. Схема связей автомата

Эту модель будем называть "схема связей автомата". В ней формирователь 1 преобразует поступающую от источников информацию во входные воздействия, например, путем сравнения значения счетчика с нулем. В этой схеме объектами управления могут быть устройства, в которых по командам автомата возможно выполнение **сколь угодно сложных** операций (в том числе и над памятью).

В предложенной модели выходные воздействия формируются в определенных "точках" пространства состояний, задаваемого графом переходов автомата. Поэтому процесс управления объектами полностью задается этим графом в наглядной и понятной форме.

Эффективность применения предлагаемой модели покажем на примере классических задач распознавания цепочек символов.

Известно [4,5], что теория построения трансляторов базируется на иерархии моделей автоматов. При этом для задач распознавания цепочек символов с усложнением языка усложняется также модель применяемого автомата. Покажем, что при использовании предлагаемого подхода перечисленные ниже задачи могут быть решены в рамках одной модели (рис. 4).

Рассмотрим три классические [5] задачи распознавания цепочек для:

- скобок произвольной глубины;
- языка $\{1^n0^n \mid n\geq 0\}$;
- языка $\{1^n0^n1^n \mid n\geq 0\}$.

Несмотря на то, что первая задача кажется сложнее второй, это не так. В первой из них проверяется только парность скобок, а во второй аналогичная задача решается после определения правильности порядка символов.

Применение распознавателей, являющихся конечными автоматами без выхода, позволяет, в частности, решить задачу распознавания четности числа символов в цепочке. Однако даже возможность формирования выходных цепочек в классической модели конечного автомата не позволяет на ее основе решить перечисленные выше задачи. При этом первые два языка, задаваемые контекстно-свободными грамматиками, требуют использования модели автомата с магазинной памятью, а для третьего языка, определяемого контекстно-зависимой грамматикой, необходим переход к линейно-ограниченным автоматам [5].

В работе [6] на основе теории автоматов была предложена SWITCH-технология для программной реализации алгоритмов логического управления. При этом каждый структурный автомат может принадлежать к одному из следующих классов: автомат без выходов, автомат Мура, автомат Мили, смешанный автомат. В отличие от работы [5], где

применяются таблицы переходов, в [6] в качестве основного способа задания автоматов используются графы переходов.

Ограниченность применяемой в работе [6] модели конечного автомата для событийных ("реактивных") систем привела к ее расширению [7]. При этом вместо двоичных входов и выходов было предложено использовать входные и выходные воздействия. Входные воздействия могут быть событиями и входными переменными, а выходные воздействия – действиями, которые могут выполняться в вершинах, а также на дугах и петлях графа переходов. Для обеспечения универсальности входные переменные и действия реализуются функциями, число и вид которых не фиксировано, а определяется схемой связей [6], которая содержит также автомат и объекты управления.

В первых двух рассматриваемых задачах в качестве объектов управления выступают счетчик, заменяющий магазин, и два индикатора "Допустить" и "Отвергнуть". Счетчик может быть обнулен, а его значение – увеличено или уменьшено на единицу. С выхода счетчика на вход автомата поступает информация о том, равно ли значение счетчика нулю.

В третьей задаче в схему связей введен дополнительный объект управления – переменная, запоминающая значение счетчика. На вход автомата дополнительно поступает информация о том, что значение счетчика равно n.

При реализации (в рамках предлагаемой технологии) преобразование графа переходов в текст программы выполняется формально и изоморфно. При этом граф переходов автомата Мили реализуется одним оператором switch. Если на всех входящих дугах некоторой вершины этого графа присутствуют одинаковые действия, то они могут быть перенесены в нее. Таким образом, автомат Мили преобразуется в более компактный смешанный автомат, который, как и автомат Мура, реализуется по шаблону, содержащему два оператора switch [7].

Хотелось бы верить, что упомянутые в эпиграфе один или два оператора относятся к одному или двум операторам switch, используемым в предлагаемом подходе.

В классической теории трансляторов в описание автомата включены действия, обеспечивающие чтение с ленты. В предлагаемом подходе такая работа с лентой может быть заменена работой с массивом, обрабатываемым в цикле, тело которого содержит вызов функции, реализующей автомат.

Первая задача была решена двумя способами. Первый из них базируется на применении автомата с одним состоянием, для которого может быть построен граф переходов с одной вершиной и четырьмя петлями (рис. 5). Это решение совпадает с приведенным в [5] с точностью до работы с лентой и замены магазина счетчиком. По мнению авторов, более естественным является второй способ, при использовании которого распознавание выполняется не на петлях, а в соответствующих состояниях смешанного автомата: "Начальное", "Анализ", "Допустить", "Отвергнуть". В этом графе переходов переменные x2, x3 и x4 ортогональны.

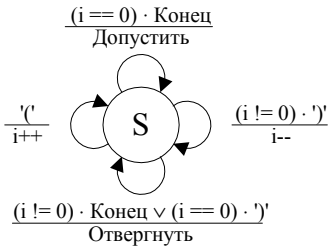

Рис. 5. Распознавание последовательностей скобок произвольной глубины. Традиционный подход

В соответствии с предложенной моделью для второго способа построена схема связей (рис. 6) и граф переходов автомата Мили (рис. 7).

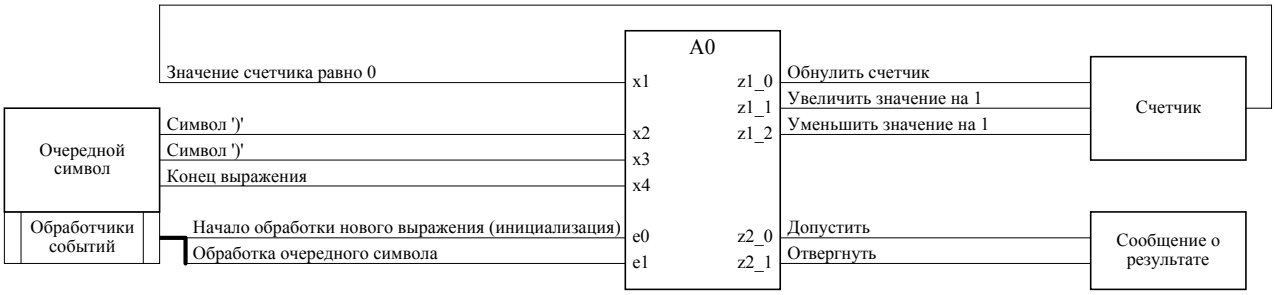

Рис. 6. Распознавание последовательностей скобок произвольной глубины. Схема связей автомата

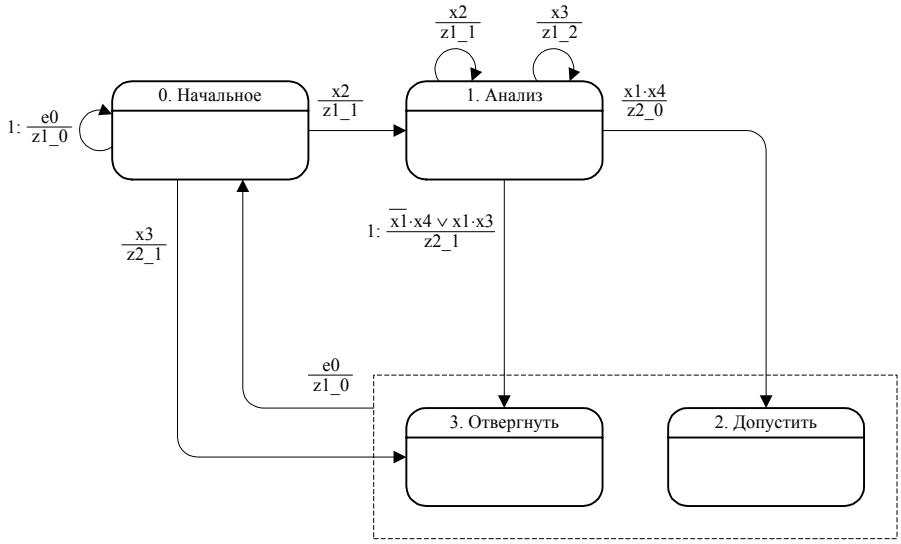

Рис. 7. Распознавание последовательностей скобок произвольной глубины. Граф переходов автомата Мили

На рис. 8 этот граф упрощен за счет перехода к смешанному автомату, в котором действия с дуг перенесены в вершины, что позволило использовать пометку z2_1 только один раз. При этом отметим, что пометки действий с петель в вершины не могут быть перенесены, так как на петле состояние автомата не изменяется, а в предложенной в [7] стандартной реализации действия в вершинах выполняются только при изменении состояний.

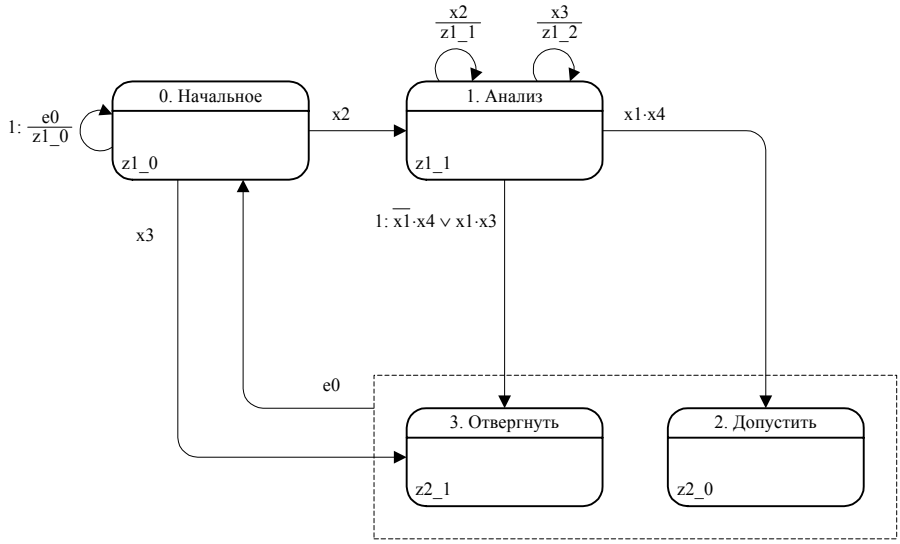

Рис. 8. Распознавание последовательностей скобок произвольной глубины. Граф переходов смешанного автомата

Для упрощения чтения графа переходов в нем все абстрактные обозначения (кроме e0) могут быть заменены смысловыми (рис. 9).

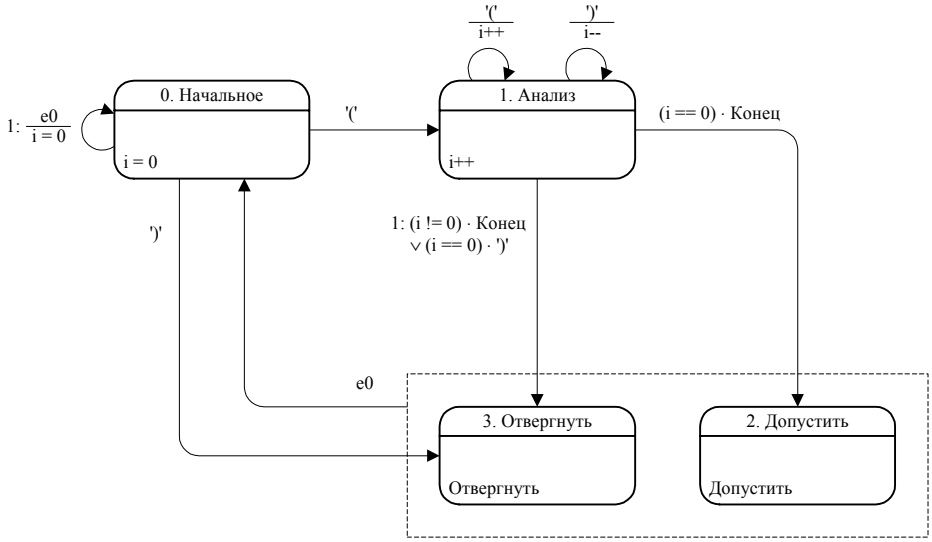

Рис. 9. Распознавание последовательностей скобок произвольной глубины. Граф переходов смешанного автомата со смысловыми обозначениями

В соответствии с предложенной в [7] программной реализацией автоматов, номер события передается функции, реализующей автомат, в качестве первого параметра. Вторым параметром передается код очередного символа.

Событие e0 предназначено для инициализации автомата перед началом обработки нового выражения, и поэтому петли в графах переходов, помеченные этим событием, имеют первый приоритет. Событие e1, указанное на рис. 6, не используется в графах переходов и предназначено для ввода очередного символа. Оно применяется в программе при вызове автомата в цикле.

В листингах 1–3 приведены программы для решения первой задачи вторым способом. Каждая из этих программ содержит цикл, моделирующий последовательный ввод символов

цепочки, и программную реализацию соответствующего автомата. При этом функция автомата вызывается из тела цикла.

Листинг 1 содержит программу, в которой автомат Мили (рис. 7) формально и изоморфно [7] реализован одним оператором switch.

Листинг 1. Автоматная программа распознавания последовательностей скобок произвольной глубины, реализующая автомат Мили

```
#include <stdio.h>
#include <string.h>

int     y0 = 0 ;    // Переменная состояния автомата
int     i = 0 ;     // Счетчик

void main()
{
  char  str[100] = "" ;
  int   j ;

  for(;;)
  {
    printf( "\nВведите строку: " ) ;
    scanf( "%s", str ) ;
    printf( "Введена строка: %s", str ) ;

    A0( 0, 0 ) ;
    for( j = 0 ; j <= strlen(str) ; j++ )
      A0( 1, str[j] ) ;
  }
}

void A0( int e, char c )
{
  switch( y0 )
  {
    case 0:
      if( e == 0 )                          { z1_0() ; }
      else
      if( x2(c) )                           { z1_1() ;          y0 = 1 ; }
      else
      if( x3(c) )                           { z2_1() ;          y0 = 3 ; }
    break ;

    case 1:
      if( !x1() && x4(c) || x1() && x3(c) )  { z2_1() ;          y0 = 3 ; }
      else
      if( x1() && x4(c) )                   { z2_0() ;          y0 = 2 ; }
      else
      if( x2(c) )                           { z1_1() ; }
      else
      if( x3(c) )                           { z1_2() ; }
    break ;

    case 2:
      if( e == 0 )                          { z1_0() ;          y0 = 0 ; }
    break ;

    case 3:
      if( e == 0 )                          { z1_0() ;          y0 = 0 ; }
    break ;
  }
}

int x1()
{ return i == 0 ; }

int x2( char c )
{ return c == '(' ; }

int x3( char c )
{ return c == ')' ; }

int x4( char c )
{ return c == 0 ; }
```

```
void z1_0()
{ i = 0 ; }
void z1_1()
{ i++ ; }

void z1_2()
{ i-- ; }

void z2_0()
{ printf( "\nДопустить.\n" ) ; }

void z2_1()
{ printf( "\nОтвергнуть.\n" ) ; }
```

Листинги 2 и 3 содержат программы, в которых смешанные автоматы, представленные на рис. 8 и 9 соответственно, реализованы двумя операторами switch. Эти реализации также построены формально и изоморфно.

Листинг 2. Автоматная программа распознавания последовательностей скобок произвольной глубины, реализующая смешанный автомат

```
#include <stdio.h>
#include <string.h>

int     y0 = 0 ;    // Переменная состояния автомата
int     i = 0 ;     // Счетчик

void main()
{
  char  str[100] = "" ;
  int   j ;

  for(;;)
  {
    printf( "\nВведите строку: " ) ;
    scanf( "%s", str ) ;
    printf( "Введена строка: %s", str ) ;
    A0( 0, 0 ) ;
    for( j = 0 ; j <= strlen(str) ; j++ )
      A0( 1, str[j] ) ;
  }
}

void A0( int e, char c )
{
  int y_old = y0 ;

  switch( y0 )
  {
    case 0:
      if( e == 0 )          { z1_0() ; }
      else
      if( x2(c) )                               y0 = 1 ;
      else
      if( x3(c) )                               y0 = 3 ;
    break ;

    case 1:
      if( !x1() && x4(c) || x1() && x3(c) )     y0 = 3 ;
      else
      if( x1() && x4(c) )                       y0 = 2 ;
      else
      if( x2(c) )           { z1_1() ; }
      else
      if( x3(c) )           { z1_2() ; }
    break ;

    case 2:
      if( e == 0 )                              y0 = 0 ;
    break ;
```

```
    case 3:
      if( e == 0 )                          y0 = 0 ;
    break ;
  }

  if( y_old == y0 ) return ;

  switch( y0 )
  {
    case 0:
      z1_0() ;
    break ;

    case 1:
      z1_1() ;
    break ;

    case 2:
      z2_0() ;
    break ;

    case 3:
      z2_1() ;
    break ;
  }
}

int x1()
{ return i == 0 ; }

int x2( char c )
{ return c == '(' ; }

int x3( char c )
{ return c == ')' ; }

int x4( char c )
{ return c == 0 ; }

void z1_0()
{ i = 0 ; }

void z1_1()
{ i++ ; }

void z1_2()
{ i-- ; }

void z2_0()
{ printf( "\nДопустить.\n" ) ; }

void z2_1()
{ printf( "\nОтвергнуть.\n" ) ; }
```

Листинг 3. Автоматная программа распознавания последовательностей скобок произвольной глубины, реализующая смешанный автомат с реализацией входных и выходных воздействий внутри функции автомата

```
#include <stdio.h>
#include <string.h>

int     y0 = 0 ;    // Переменная состояния автомата
int     i = 0 ;     // Счетчик

void main()
{
  char  str[100] = "" ;
  int   j ;

  for(;;)
  {
    printf( "\nВведите строку: " ) ;
    scanf( "%s", str ) ;
    printf( "Введена строка: %s", str ) ;

    A0( 0, 0 ) ;
```

```
    for( j = 0 ; j <= strlen(str) ; j++ )
      A0( 1, str[j] ) ;
  }
}

void A0( int e, char c )
{
  int y_old = y0 ;

  switch( y0 )
  {
    case 0:
      if( e == 0 )          { i = 0 ; }
      else
      if( c == '(' )                                          y0 = 1 ;
      else
      if( c == ')' )                                          y0 = 3 ;
    break ;

    case 1:
      if( (i != 0) && (c == 0) || (i == 0) && (c == ')') )    y0 = 3 ;
      else
      if( (i == 0) && (c == 0) )                              y0 = 2 ;
      else
      if( c == '(' )        { i++ ; }
      else
      if( c == ')' )        { i-- ; }
    break ;

    case 2:
      if( e == 0 )                                            y0 = 0 ;
    break ;

    case 3:
      if( e == 0 )                                            y0 = 0 ;
    break ;
  }

  if( y_old == y0 ) return ;

  switch( y0 )
  {
    case 0:
      i = 0 ;
    break ;

    case 1:
      i++ ;
    break ;

    case 2:
      printf( "\nДопустить.\n" ) ;
    break ;

    case 3:
      printf( "\nОтвергнуть.\n" ) ;
    break ;
  }
}
```

Интересно отметить, что упрощение графа переходов (переход от автомата Мили к смешанному автомату) приводит к усложнению программной реализации (два оператора switch вместо одного).

Вторая задача также была решена двумя способами. Первый из них базируется на применении автомата Мили с двумя состояниями, в котором отсутствует достижимость начального состояния (рис. 10).

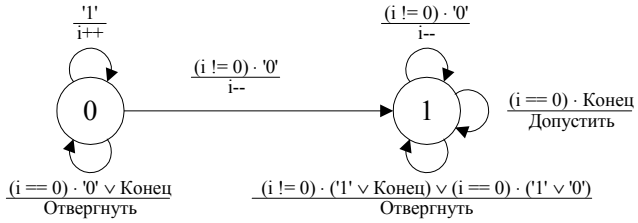

Рис. 10. Распознавание последовательностей $\{1^n0^n \mid n\geq 0\}$. Традиционный подход

Это решение также совпадает с приведенным в [5] с точностью до работы с лентой и замены магазина счетчиком. Второе решение основывается на применении смешанного автомата с пятью состояниями: "Начальное", "Считать `1`", "Считать `0`", "Допустить", "Отвергнуть". Схема связей этого автомата приведена на рис. 11, а его граф переходов – на рис. 12. В этом графе переходов переменные x2, x3 и x4 ортогональны.

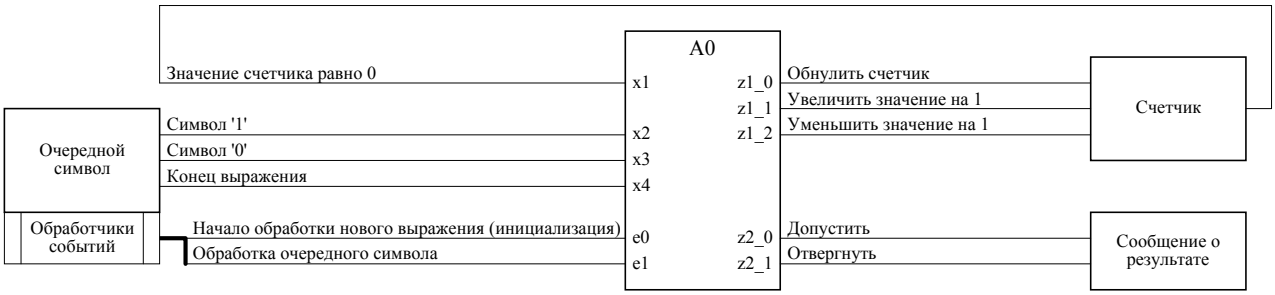

Рис. 11. Распознавание последовательностей $\{1^n0^n \mid n\geq 0\}$. Схема связей автомата

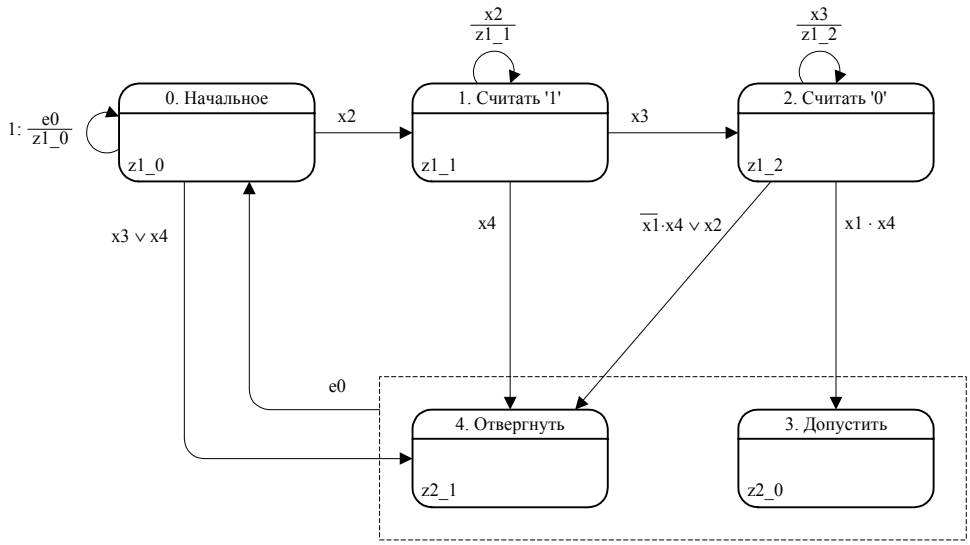

Рис. 12. Распознавание последовательностей $\{1^n0^n \mid n\geq 0\}$. Граф переходов автомата

Третья задача, которая не может быть решена магазинным автоматом, реализуется смешанным автоматом с шестью состояниями: "Начальное", "Считать `1`", "Считать `0`", "Повторно считать `1`", "Допустить", "Отвергнуть". Схема связей этого автомата приведена на рис. 13, а его граф переходов – на рис. 14. В этом графе переходов переменные x3, x4 и x5 ортогональны.

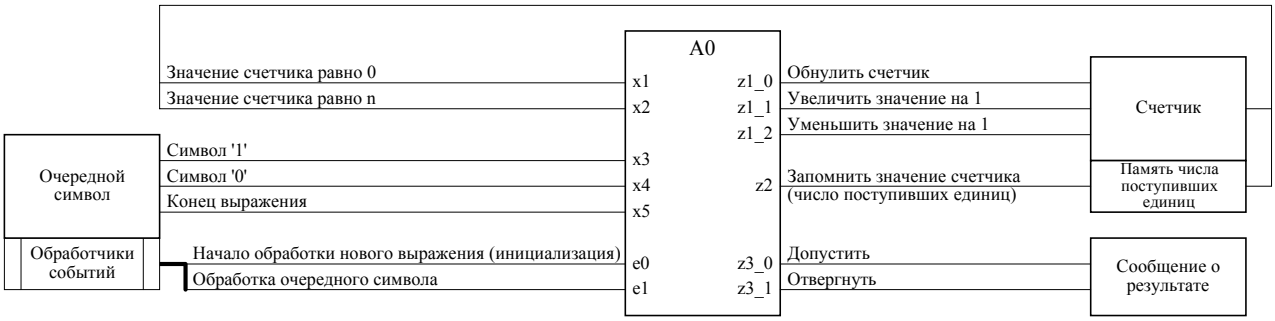

Рис. 13. Распознавание последовательностей $\{1^n0^n1^n \mid n\geq 0\}$. Схема связей автомата

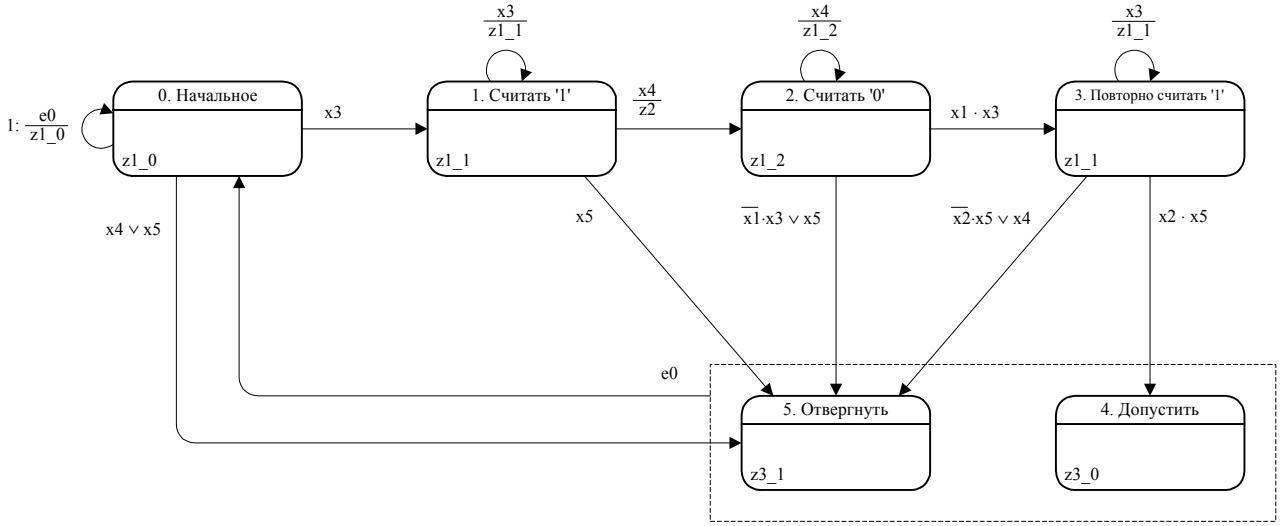

Рис. 14. Распознавание последовательностей $\{1^n0^n1^n \mid n\geq 0\}$. Граф переходов автомата

Из приведенных примеров следует, что при использовании предлагаемого подхода, изменение типа грамматики не приводит к необходимости использования модели другого типа, а вызывает лишь незначительное усложнение графа переходов.

Кроме того, на основе излагаемого подхода была решена задача распознавания цепочек символов языков: $\{0^n1^n, \ldots, 0^n1^n \mid n\geq 1\}$, $\{0^n1^n, \ldots, 0^n1^n, 0^n \mid n\geq 1\}$, $\{1^n0^n, \ldots, 1^n0^n \mid n\geq 1\}$ и $\{1^n0^n, \ldots, 1^n0^n, 1^n \mid n\geq 1\}$. Усложнение этой задачи по сравнению с предыдущей привело к дальнейшему увеличению числа объектов управления в схеме связей. Новыми объектами в этой схеме являются переменные a и b, определяющие порядок следования единиц и нулей, одна из которых, в принципе, может быть исключена. Для первых двух языков переменные a и b принимают значения '0' и '1', соответственно, а для оставшихся – '1' и '0'. Эта задача **также** реализуется смешанным автоматом с шестью состояниями: "Начальное", "Вычисление n", "Считать a", "Считать b", "Допустить", "Отвергнуть". Схема связей этого автомата приведена на рис. 15, а его граф переходов – на рис. 16. В этом графе переходов переменные x5, x6 и x7 ортогональны.

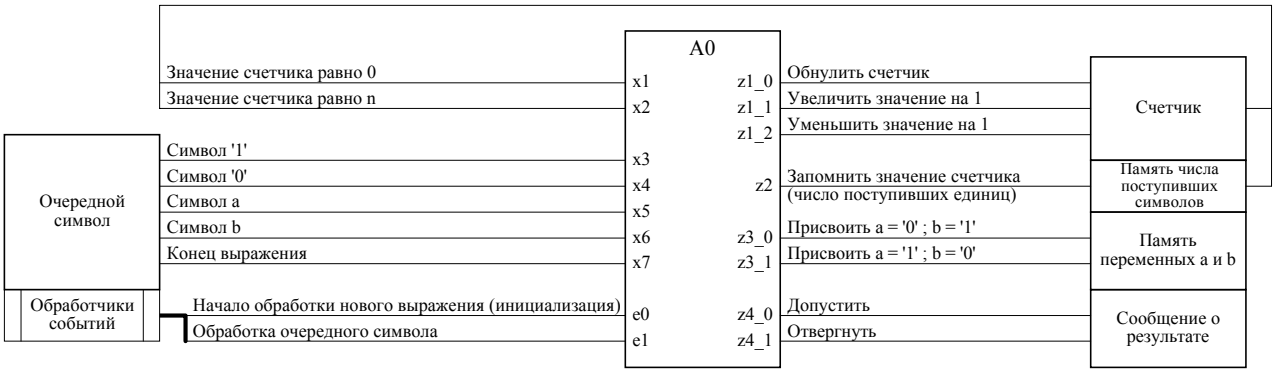

Рис. 15. Распознавание четырех языков. Схема связей автомата

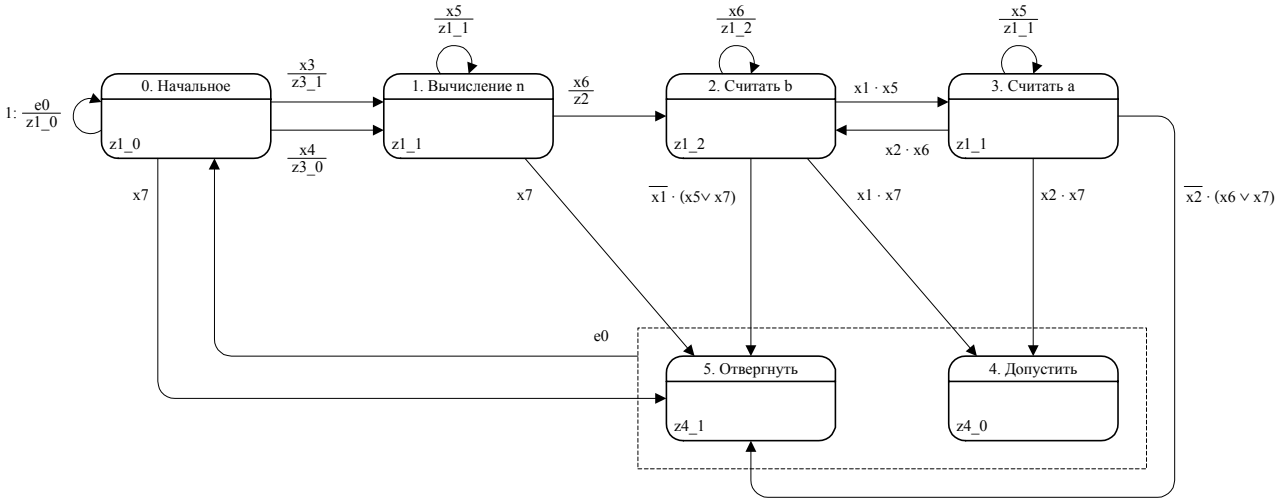

Рис. 16. Распознавание четырех языков. Граф переходов автомата

В рассматриваемой модели вместо одиночного автомата может использоваться система взаимосвязанных автоматов. Эти автоматы могут быть вложенными. Кроме этого способа взаимодействия они могут также вызывать друг друга из выходных воздействий и обмениваться номерами состояний, что позволяет весьма эффективно реализовывать параллельные процессы.

Графы переходов, используемые в предлагаемом подходе, являются не "картинками", а математическими моделями, что подтверждается автоматическим протоколированием, выполняемым в терминах автоматов [7]. Это чрезвычайно важно, так как "протоколы (история вычислений) являются конструкциями, вскрывающими механизм работы программы и, поэтому, постепенно среди теоретиков программирования сложилось представление, что множество протоколов лучше характеризует программу, нежели сам исходный программный текст" [8]. Изложенное связано с тем, что "идея доказательства правильности программ в значительной мере исчерпала себя, так как выяснилось, что в общем случае невозможно установить свойства результата работы программы или процедуры, не исполнив ее до конца... В теории вычислений доказывается, что в общем случае распознать определенные свойства в программе можно только динамически, прослеживая весь процесс вычисления при соответствующих входных воздействиях" [9].

В заключение отметим, что в отличие от применения автоматов в теории трансляторов, предлагаемый подход может использоваться при программировании различных задач со сложным поведением. Автоматное программирование позволяет в наглядной форме (в терминах состояний и переходов) описывать процессы управления

объектами, обеспечивая с целью уменьшения размерности решаемых задач сокрытие деталей, связанных с вычислениями.

Отметим также, что если Дж. фон Нейман изменил машину Тьюринга для эффективной аппаратной реализации алгоритмов (например, введением арифметическо-логического устройства) [10], то в настоящей работе делается аналогичная попытка преобразования этой машины с целью повышения эффективности программирования. При этом, если в машине Тьюринга автомат решает две задачи (управление и вычисление, не свойственное для него), то в предлагаемом подходе он решает только задачу управления, а вычисления осуществляются в объектах управления, предназначенных для этой цели. Например, автоматы (рис. 7, 8) вызывают функцию z1_1, увеличивающую значение переменной на единицу, что отражено в текстах реализующих их программ (листинг 1 и 2). При использовании в автомате смысловых обозначений (рис. 9), его реализация, приведенная в листинге 3, по сравнению с предыдущим листингом упрощается.

Автоматный подход в программировании весьма конструктивен, так как позволяет, в отличие от работы [9], под состоянием программы понимать не значения **всех** ячеек памяти, а разделять состояния на две группы: автоматные и остальные. При этом с помощью конечного (обычно небольшого и независящего от размерности задачи) числа состояний автомата удается управлять сколь угодно большим числом состояний объекта управления, которое может увеличиваться в зависимости от размерности решаемой задачи. Так, в рассмотренных примерах число состояний в автоматах не зависит от параметра n.

Эта же идея была использована авторами при объединении автоматного подхода с объектно-ориентированным. При этом было предложено в классах выполнять разделение атрибутов и методов на автоматные и остальные, что позволяет описывать поведение объектов в терминах состояний [11].

Рассматриваемая в настоящей статье парадигма автоматного программирования начинает все шире использоваться [12-19]. Для большей ее популяризации на сайте [11] создан раздел "Автоматы".

ЛИТЕРАТУРА

1. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы. М.: Вильямс, 2000.
2. Хопкрофт Д. Машины Тьюринга //В мире науки. 1984. № 7.
3. Трахтенброт В.А. Алгоритмы и вычислительные автоматы. М.: Советское радио. 1974.
4. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ. М.: Мир, 1978.
5. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. М.: Мир, 1979.
6. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
7. Шалыто А.А., Туккель Н.И. Программирование с явным выделением состояний //Мир ПК. № 8, 9.
8. Ершов А.П. Смешанные вычисления //В мире науки. 1984. № 6.
9. Лавров С.С. Программирование. Математические основы, средства, теория. СПб.: БХВ-Петербург, 2001.
10. Фон Нейман Дж. Общая и логическая теория автоматов /В кн. Тьюринг А. Может ли машина мыслить? Саратов: Колледж, 1999.
11. http://www.softcraft.ru
12. Герр Р. Новый поворот //PC Magazine / Russian Edition. 1998. № 10.
13. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.
14. Богатырев Р. Об асинхронном и автоматном программировании //Открытые системы. 2001. № 3.
15. Шалыто А.А. Использование граф-схем алгоритмов и графов переходов при программной реализации алгоритмов логического управления //Автоматика и телемеханика. 1996. № 6, 7.
16. Жаков В.И., Фильчаков В.В., Янкелевич А.А. Применение конечных автоматов для описания пользовательского интерфейса и синтеза приложений //Информационные технологии. 1997. № 4.
17. Сосонкин В.Л., Мартинов Г.М., Любимов А.Б. Интерпретация диалога в Windows-интерфейсе систем управления //Приборы и системы управления. 1998. № 12.
18. Любченко В.С. Мы выбираем, нас выбирают... (к проблеме выбора алгоритмической модели //Мир ПК. 1999. № 3.
19. Кузнецов Б.П. Психология автоматного программирования //BYTE/Россия. 2000. № 11.