Глава 17

Программная реализация автоматов с памятью

Известен [1, 2] метод аппаратной реализации автоматов с памятью, заданных граф-схемами (ГС) алгоритмов, которые для двоичных переменных могут быть названы бинарными графами (БГ).

При этом методы построения ГС для автоматов с памятью, реализуемых программно, не рассматривались, несмотря на то что в названии работы [1] присутствует термин «микропрограммный».

Прежде чем перейти к изложению методов программной реализации автоматов с памятью, перечислим основные отличия между граф-схемами, реализуемыми аппаратно и программно:

— ввод входных переменных в ГС первого типа осуществляется в каждой условной вершине (УВ), в то время как в ГС второго типа эти переменные обычно вводятся одним оператором «Ввод», являющимся первым оператором в теле ГС;

— в ГС первого типа в каждой операторной вершине (ОВ) указываются обозначения только тех выходных переменных, которые в ней принимают единичные значения, а для остальных, не указанных в этой вершине, по умолчанию считают, что они принимают в этой ОВ нулевые значения. В ГС второго типа в операторных вершинах указываются как единичные, так и нулевые значения выходных переменных, а для умалчиваемых выходных переменных полагают, что они сохраняют предыдущее значение;

— ГС первого типа могут содержать внутренние обратные связи, в то время как в ГС второго типа такие связи обычно не используются, а применяется одна внешняя обратная связь, соединяющая оператор «Вывод» с оператором «Ввод», что поддерживается язы-

ками инструкций большинства программируемых логических контроллеров;

— в ГС первого типа значения всех выходных переменных (в том числе и умалчиваемых) выдаются на выход в каждой ОВ, а в ГС второго типа на выход выдаются значения всех выходных переменных (в том числе и умалчиваемых) из тех ОВ, которые связаны с оператором «Вывод» непосредственно или через условные вершины. Кроме того, предыдущие значения выходных переменных могут сохраняться и выдаваться на выход при прохождении таких контуров, образованных внешней обратной связью, которые не содержат операторных вершин и называются поэтому «пустыми».

17.1. Метод независимых фрагментов

Предположим, что в общем случае автомат задан системой булевых формул (СБФ), состоящей из $2N$ формул, первые N из которых имеют вид: $z_{11} = f_1(z_1, \ldots, y_m)$, … , $z_{k1} = f_k(z_1, \ldots, y_m)$, $y_{11} = f_{k+1}(z_1, \ldots, y_m)$, … , $y_{m1} = f_N(z_1, \ldots, y_m)$, где $k + m = N$, а оставшиеся N формул являются переобозначениями переменных: $z_1 = z_{11}, \ldots, z_k = z_{k1}, y_1 = y_{11}, \ldots, y_m = y_{m1}$.

Предлагаемый метод построения структурированного бинарного графа (СБГ) по этой СБФ состоит в следующем:

— с помощью метода независимых фрагментов для автоматов без памяти, изложенного в разд. 16.3, по первым N формулам строится ядро тела СБГ;

— к ядру последовательно подсоединяется одна или несколько операторных вершин, в которых записываются оставшиеся N формул системы;

— к построенной структуре последовательно подсоединяется одна или несколько ОВ, в которых выполняется обнуление переменных, введенных лишь с целью переобозначения и имеющих в СБФ двойные индексы.

Пример 17.1. Построить СБГ для R-триггера, описываемого СБФ

$$z_1 = !R(S \vee z), \; z = z_1.$$

Построим на основе первой БФ оператор структуры

$$F = !RSz_1 | !Rzz_1$$

и выполним его минимизацию:

$$F = !R(Sz_1 | zz_1).$$

Искомый СБГ представлен на рис. 17.1.

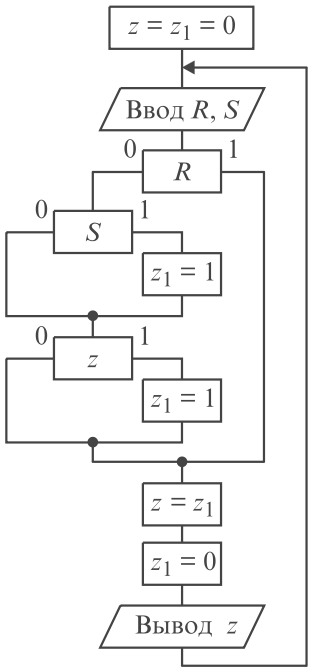


Рис. 17.1

Пример 17.2. Построить СБГ для счетного триггера, описываемого системой булевых формул:

$$z_1 = !xz \vee x!y,\ \ y_1 = !xz \vee xy,\ \ z = z_1,\ \ y = y_1.$$

Построим на основе первых двух БФ оператор структуры

$$F = !xzz_1 | x!yz_1 | !xzy_1 | xyy_1$$

и выполним его минимизацию:

$$F = !xz(z_1 | y_1) | x(!yz_1 | yy_1).$$

Введем в этот оператор квадратные скобки, отображающие использование управляющих конструкций «полный выбор»:

$$F = \left[!xz(z_1 | y_1) \vee x[!yz_1 \vee yy_1]\right].$$

Искомый СБГ представлен на рис. 17.2.

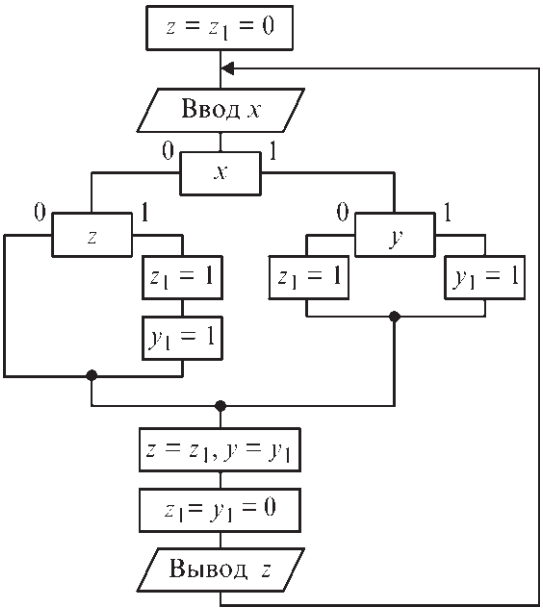


Рис. 17.2

Недостаток этого метода, изложенного в [3], состоит в том, что он строит СБГ, которые достаточно трудно понять, так как они не изоморфны эквивалентным им графам переходов (ГП).

17.2. Канонический и распределительный методы

Особенность применения этих методов для автоматов с памятью состоит в том, что после построения с их помощью тела бинарного графа и замыкания внешней обратной связи может появиться возможность исключения некоторых условных и операторных вершин, связанных с сохранением предыдущих значений выходных и (или) внутренних переменных.

Предположим, что требуется построить бинарный граф, реализующий *R*-триггер, по кодированной таблице переходов (КТП). Структура БГ зависит от порядка переменных в этой таблице. Построим БГ с помощью рассматриваемых методов для всех (шести) возможных порядков переменных, используя только три таблицы. Отметим, что в отличие от метода независимых фрагментов в данном случае отсутствует необходимость применять переобозначения переменных.

Пример 17.3. Построить бинарный граф по КТП

$$z(R, z, S) = \left|0111\ \ 0000\right|^T.$$

1. Используя канонический метод, построим СБГ1 (рис. 17.3).

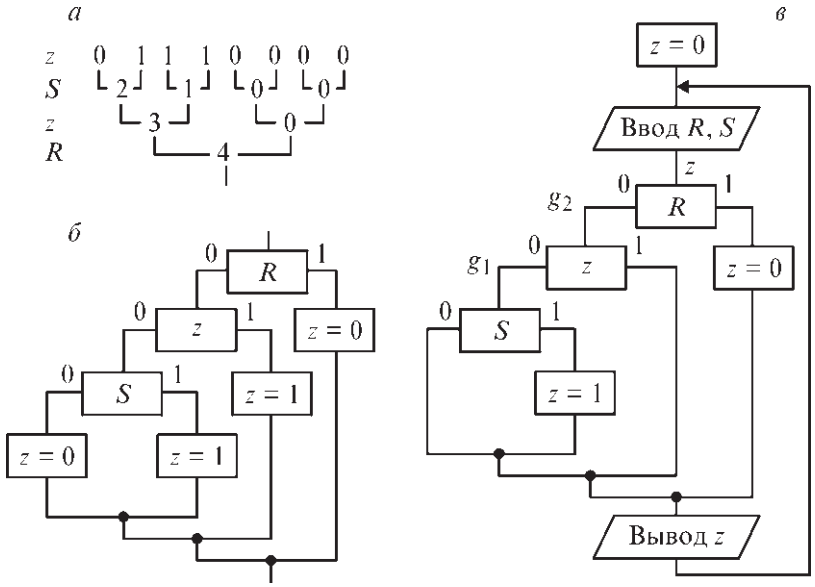


Рис. 17.3

Тело построенного СБГ1 состоит из пяти вершин, что характеризует его сложность. Так как СБГ1 имеет древовидную структуру, выполним его верификацию, как и для автоматов без памяти, от выхода к входу:

$$g_1 = z!S \vee 1 \,\&\, S = S \vee z;\ \ g_2 = g_1!z \vee zz = S \vee z;$$
$$z = g_2!R \vee 0 \,\&\, R = !R(S \vee z).$$

СБГ1 может быть запрограммирован, например, следующим образом:

$$z = R\,?\left(0 : \left(z\,?\,z : (S\,?\,1 : z)\right)\right).$$

2. Применяя распределительный метод, построим структуру, приведенную на рис. 17.4.

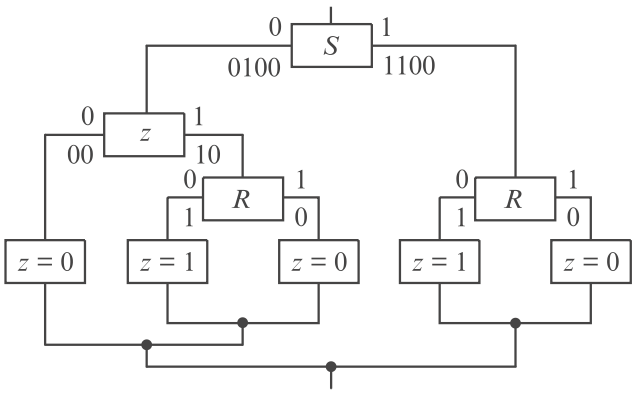


Рис. 17.4

На основе этой структуры могут быть построены БГ1 (рис. 17.5) и СБГ2 (рис. 17.6).

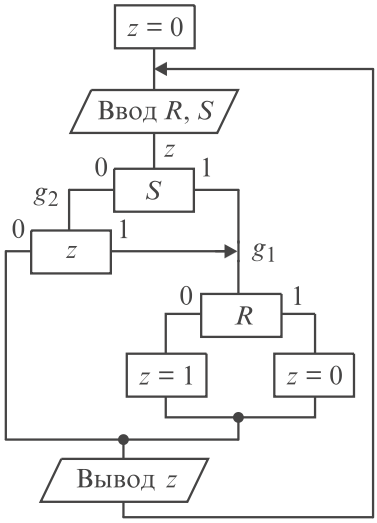


Рис. 17.5

Сложность этих бинарных графов составляет пять и семь соответственно.

Так как БГ1 не содержит последовательно соединенных блоков, то выполним его верификацию от выхода к входу:

$$g_1 = 1 \,\&\, !R \vee 0 \,\&\, R = !R;\ \ g_2 = z!z \vee g_1 z = !Rz;$$
$$z = g_2 !S \vee g_1 R = !R(S \vee z).$$

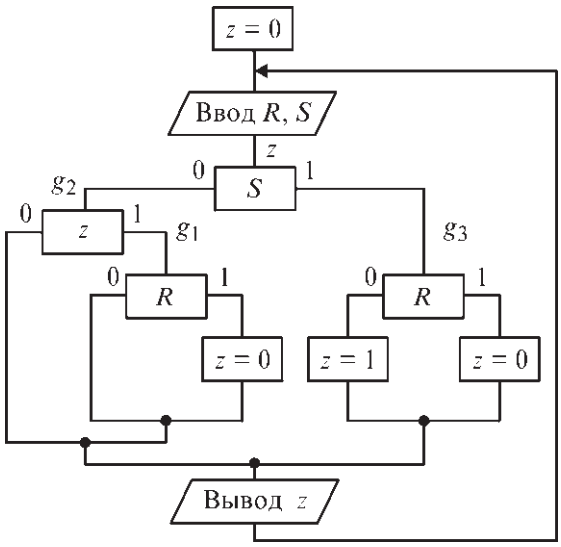


Рис. 17.6

При программировании тела БГ1 приходится использовать метку:

```
      if(S) goto g1;
      if(z)
g1: {if(R) z=0;
      else  z=1;}
```

Выполним верификацию СБГ2:

$$g_1 = z!R \vee 0 \,\&\, R = z!R;\ g_2 = !zz \vee g_1 z = !Rz;$$
$$g_3 = 1 \,\&\, !R \vee 0 \,\&\, R = !R;\ z = g_2!S \vee g_3 S = !R(S \vee z).$$

Пример 17.4. Построить бинарный граф по КТП

$$z(R, S, z) = \left| \, 0111 \;\; 0000 \, \right|^T.$$

1. Применяя канонический метод, построим СБГ3 (рис. 17.7).

Сложность этого графа равна четырем. Выполним его верификацию:

$$g = z!S \vee 1 \,\&\, S = z \vee S,\ z = g!R \vee 0 \,\&\, R = !R(S \vee z).$$

Обратим внимание на то, что при одном и том же столбце значений в этом и предыдущем примерах при использовании одного и того же метода для рассматриваемого автомата с памятью постро-

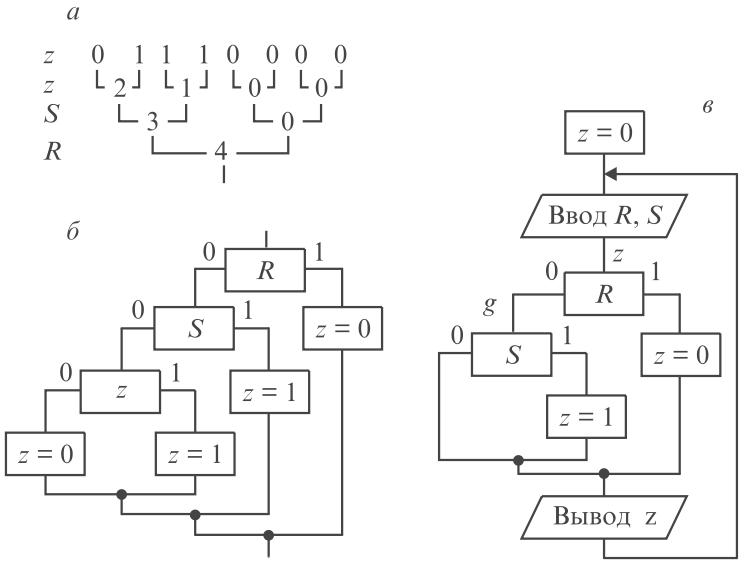


Рис. 17.7

ены СБГ разной сложности, что невозможно для автоматов без памяти.

Тело построенного СБГ программируется следующим образом:

```
if(R)   z=0;
else
if(S)   z=1;
```

2. Применяя распределительный метод, построим структуру, приведенную на рис. 17.8.

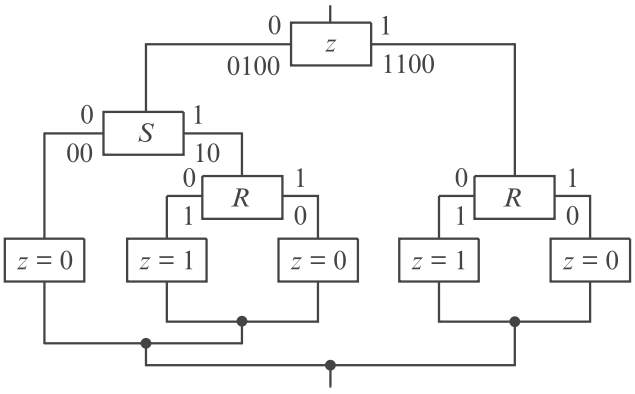


Рис. 17.8

На основе этой структуры могут быть построены БГ2 (рис. 17.9), БГ3 (рис. 17.10) и СБГ4 (рис. 17.11). Сложность этих графов составляет пять, пять и шесть соответственно.

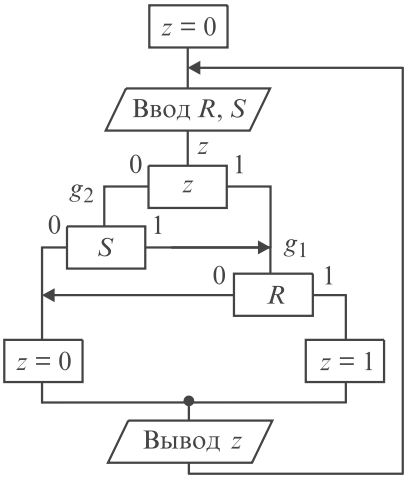


Рис. 17.9

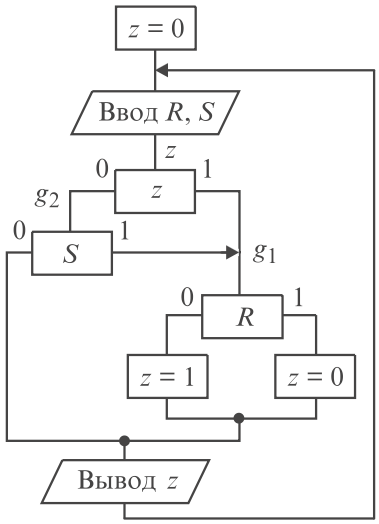


Рис. 17.10

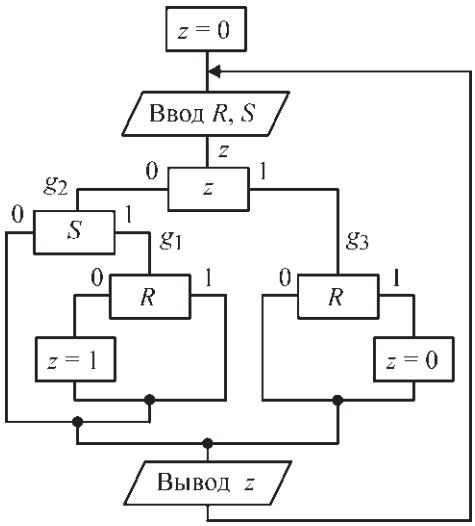


Рис. 17.11

Выполним верификацию БГ2 (рис. 17.9):

$$g_1 = 1 \,\&\, !R \vee 0 \,\&\, R = !R;\;\; g_2 = 0 \,\&\, !S \vee g_1 S = S!R;$$

$$z = g_2 ! z \vee g_1 z = (z \vee S)!R.$$

Выполним верификацию БГ3 (рис.17.1

$$g_1 = 1 \,\&\, !R \vee 0 \,\&\, R = !R;\ \ g_2 =$$

$$z = g_2 ! z \vee g_1$$

Выполним верификацию С

$$g_1 = 1 \,\&\, !R \vee zR =$$

СБГ4 соотв
фу переходо

Эти СБГ и ГП мог
мощью конструкции s

```
ca

ca
```

Пример 17.5. Построить структурированный бинарный граф по КТП

$$z(S, R, z) = \left| \, 0100 \;\; 1100 \, \right|^{T}.$$

1. Используя канонический метод и подъем одной из операторных вершин вверх, построим СБГ6 (рис. 17.14).

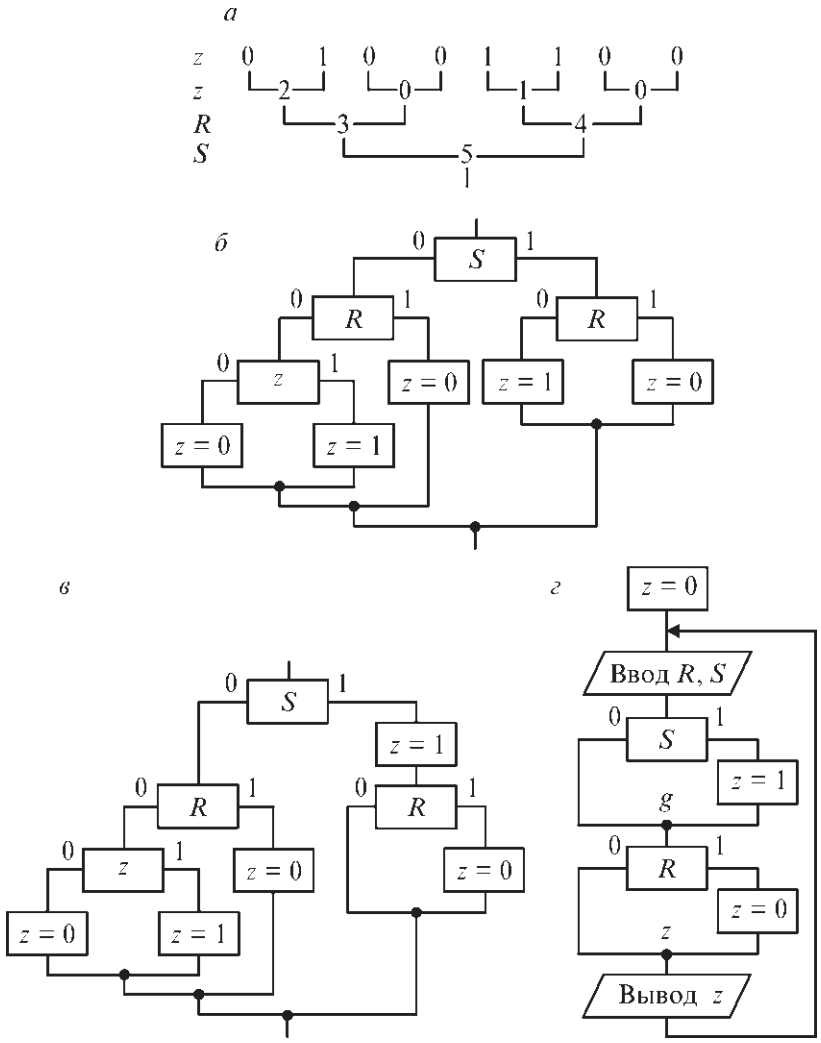


Рис. 17.14

Сложность этого графа равна четырем. Так как тело этого СБГ состоит только из последовательно соединенных блоков с одним

входом и одним выходом, то выполним его верификацию от входа к выходу:

$$g = z!S \vee 1 \,\&\, S = z \vee S,\ \ z = g!R \vee 0 \,\&\, R = !R(S \vee z).$$

Тело СБГ в этом случае реализуется простейшей программой:

```
if(S)  z=1;
if(R)  z=0,
```

которая, однако, понимается хуже предыдущей программы, так как она в явном виде не содержит предшествующих значений переменной z.

2. Применяя распределительный метод, построим структуру, приведенную на рис. 17.15.

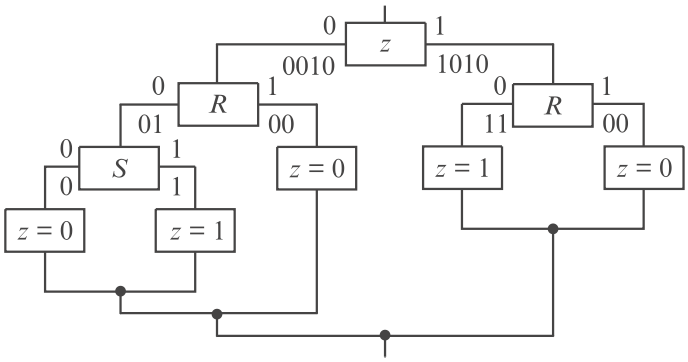


Рис. 17.15

На основе этой структуры построим СБГ7 (рис. 17.16).

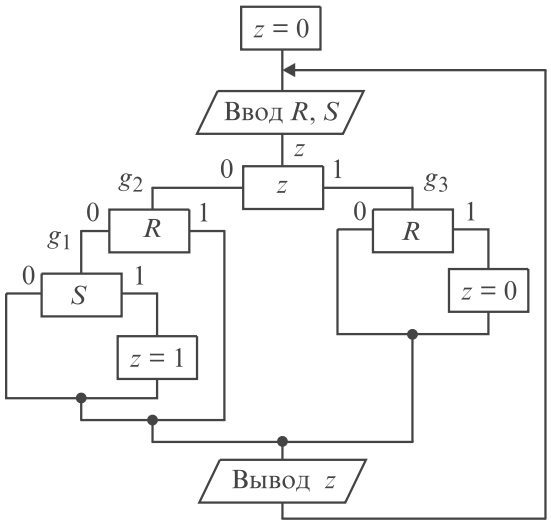


Рис. 17.16

Выполним верификацию этого графа:

$$g_1 = z!S \vee zS = S \vee z;\ \ g_2 = g_1!R \vee zR = z \vee S!R;$$
$$g_3 = z!R \vee 0\,\&\,R = !Rz;\ \ z = g_2!z \vee g_3 z = !R(S \vee z).$$

СБГ7 соответствует СБГ8 (рис. 17.17), который изоморфен графу переходов (рис. 17.18).

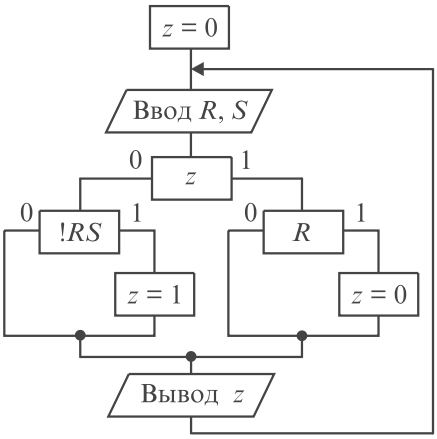


Рис. 17.17

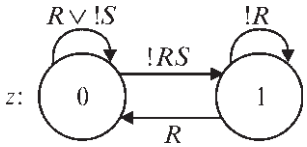


Рис. 17.18

Эти СБГ и ГП могут быть изоморфно запрограммированы с помощью конструкции `switch`:

```
        switch(z) {
case 0: if(!R&S)  z=1;
        break;
case 1: if(R)     z=0;
        break;    }
```

После завершения построения БГ с помощью рассматриваемых методов отметим, что пять из них (БГ1, БГ2, БГ3, СБГ1 и СБГ3) могут быть построены также и при использовании формульного метода. Например, тело БГ3 (рис. 17.10) изоморфно БГ4 (рис. 17.19), построенному по булевой формуле $z = (z \vee S)!R$.

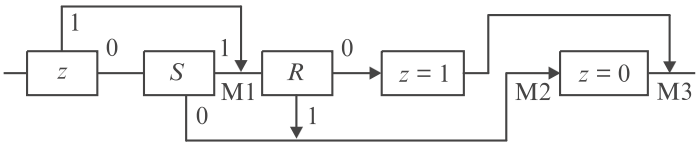


Рис. 17.19

Этот граф реализуется наиболее неструктурированной программой из рассмотренных:

```
    if(z)  goto M1;
    if(!S) goto M2;
M1: if(R)  goto M2;
    z=1;
    goto M3;
M2: z=0;
M3:
```

Граф БГ2 (рис. 17.9) обладает уникальным свойством: из всех рассмотренных графов он в минимизированном виде не имеет «пустых» (от операторных вершин) путей, т.е. на каждом из них устанавливается одно из значений выходной переменной z, являющейся одновременно внутренней (промежуточной). Это свойство, характерное для бинарных графов, реализующих автоматы без памяти, сохраняется в данном случае и для автомата с памятью, так как этот граф содержит условную вершину, помеченную переменной z. При отсутствии таких вершин наличие в графе «пустых» путей свидетельствует о том, что он реализует автомат с памятью. В этом случае весьма простой по структуре бинарный граф может иметь весьма сложное поведение, полностью понять которое в общем случае можно лишь построив эквивалентный граф переходов.

Сравнение построенных бинарных графов может быть выполнено по следующим критериям, применяемым традиционно: быстродействие, число вершин, структурированность и сложность реализующей программы.

По первому критерию наилучшей является СБГ3, по второму — СБГ3 и СБГ6. Структурированными являются все графы, за исключением БГ1—БГ3. Наиболее простой (из рассматриваемых ветвящихся программ) является программа, реализующая СБГ6.

Таким образом, наилучшими по традиционным критериям являются графы СБГ3 и СБГ6.

Оба эти графа обладают тем свойством, что условные вершины в них помечены только символами входных переменных, в то время как такие вершины, помеченные символом выходной переменной z, в этих графах отсутствуют.

Граф СБГ3 обладает тем преимуществом, что на каждом пути от входа к выходу выходной переменной z присваивается значение не более одного раза, а структура СБГ6 обеспечивает его изоморфную реализацию наиболее простой ветвящейся программой.

Однако оба эти графа обладают, по мнению автора, существенным недостатком, который может свести на нет указанные достоинства: они недостаточно понятны в том смысле, что при $R = S = 0$

сохраняют предыдущее значение переменной *z* без указания ее конкретного значения и поэтому прочесть это значение непосредственно по графу (без его «прокрутки») невозможно. В этих графах для определения значения переменной *z* на «пустых» путях необходимо использовать внемодельные средства — память ЭВМ или человека, в которой фиксируется предшествующее значение этой переменной.

Эта проблема имеет место и в общем случае. Поэтому не случайно Н. Вирт назвал свою книгу «Программы = алгоритмы + данные» [4], отмечая в ней, что при реализации алгоритмов граф-схемами последние в общем случае в явном виде указывают лишь связи по управлению, в то время как явные связи по данным в них отсутствуют.

Однако если ЭВМ обладает большой оперативной памятью, содержимое которой весьма легко прочесть, то такая память у человека крайне ограниченна [5], и поэтому при использовании БГ не только для программирования, но и в качестве языка общения между Специалистами различных областей знаний, по мнению автора, должен применяться новый критерий качества бинарных графов — п о н я т н о с т ь.

Так как понятность является субъективным фактором, введем формальный критерий понятности бинарного графа, реализующего автомат с памятью, — его изоморфизм с функционально эквивалентным графом переходов без флагов и умолчаний [3, 6]. При этом наиболее понятными, по мнению автора, являются бинарные графы, изоморфные таким графам переходов, которые соответствуют автоматам без выходного преобразователя или автоматам Мура, так как для каждого автомата, принадлежащего этим классам, граф переходов совпадает с графом достижимых маркировок [3].

По этому критерию наилучшими из рассмотренных графов являются СБГ5 и СБГ8. Несмотря на наличие «пустых» путей в них, они обладают такой структурой, что по этим графам, поднимаясь от оператора «Вывод» вверх, можно непосредственно определить предшествующее значение переменной *z* на этих путях, так как указанные графы начинаются с дешифратора значений этой переменной. Последний можно считать дешифратором внутренних состояний (в дальнейшем — «д е ш и ф р а т о р с о с т о я н и й») изоморфного автомата, граф переходов которого, в свою очередь, обладает изоморфизмом с конструкцией `switch` языка СИ, изоморфной также и СБГ с указанными свойствами.

В графах СБГ3 и СБГ6 переменная, характеризующая состояния, в условных вершинах не применяется, а в графах БГ1, СБГ1 и СБГ2 эта переменная используется в качестве некоторой внутренней переменной, делая их весьма непонятными.

Если в общем случае построение СБГ начинать с построения дешифратора состояний, а не событий, как это обычно принято, то можно утверждать, что СБГ с такой структурой содержат в явном виде связи не только по управлению, но и по данным. Они могут быть названы «автоматные схемы алгоритмов» (АСА).

Из изложенного следует, что имеет место взаимно однозначное соответствие (изоморфизм) между тремя видами описаний алгоритмов — АСА, ГП и конструкцией `switch`, по каждому из которых два других могут быть восстановлены непосредственно (без дополнительных вычислений) (рис. 17.20).

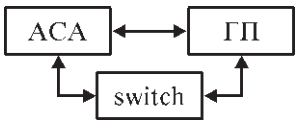


Рис. 17.20

17.3. Модификация канонического метода

Используем метод, изложенный в разд. 16.4.6, для реализации автоматов с памятью.

Пример 17.6. Реализовать последовательный одноразрядный двоичный сумматор.

Рис. 17.21 и 17.22 иллюстрируют применение рассматриваемого метода.

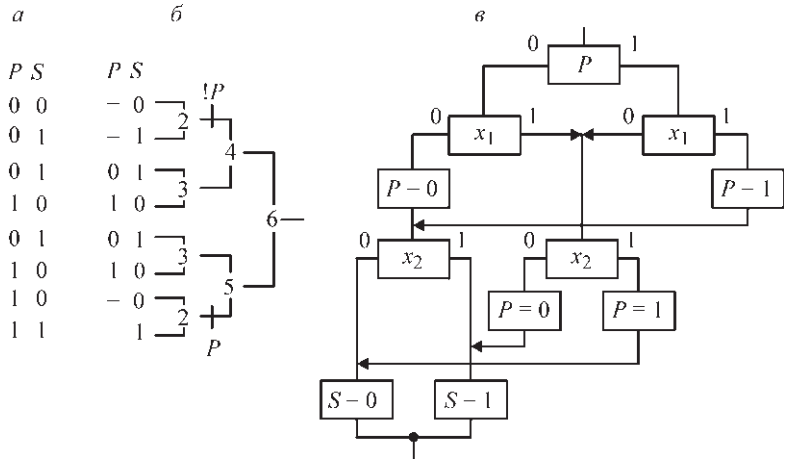


Рис. 17.21

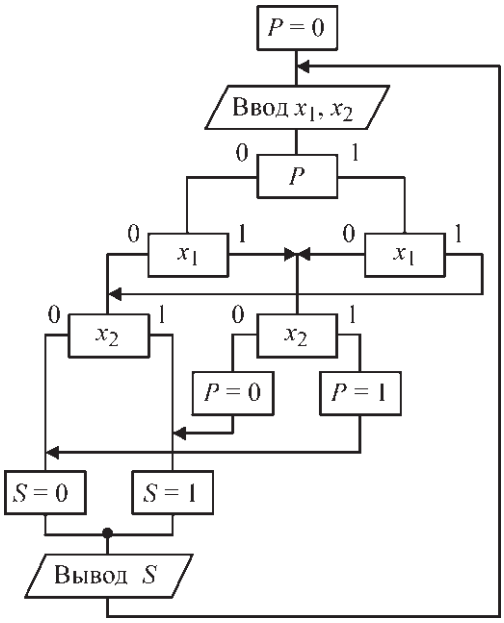


Рис. 17.22

17.4. Верификация бинарных графов

Если в бинарном графе с одной обратной связью (внешней), условные вершины которого помечены только символами входных переменных, на каждом пути от входа к выходу в операторных вершинах указаны значения всех выходных переменных, то такой граф реализует автомат без памяти.

Как показано в разд. 17.2, если в БГ указанного типа условные вершины могут быть помечены также и значениями выходных переменных, то такой граф реализует автомат с памятью.

Если в БГ с одной обратной связью указанного типа для каждой переменной существует хотя бы один путь, на котором в операторных вершинах не записано ее значение, то такой граф реализует автомат с памятью, являющийся автоматом без выходного преобразователя.

Для автоматов с памятью под верификацией бинарного графа будем понимать построение по нему эквивалентного ГП.

Пример 17.7. Верифицировать СБГ (рис. 17.23).

Этот БГ является комбинационным по переменной z_3 и автоматным по переменным z_1 и z_2.

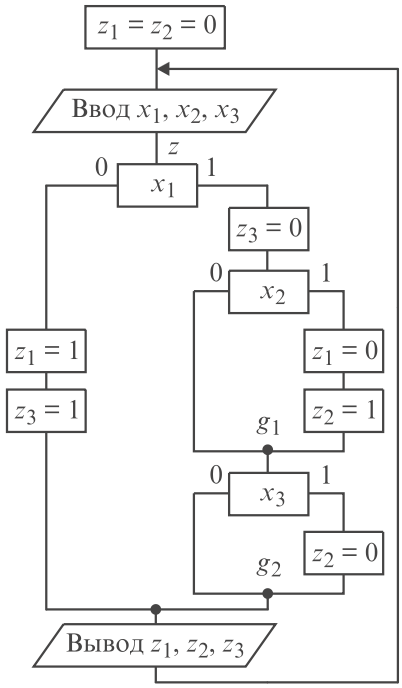


Рис. 17.23

Верификацию переменной z_3 будем проводить от выхода к входу:

$$z_3 = 1 \,\&\, !x_1 \vee 0 \,\&\, x_1 = !x_1.$$

Перейдем к верификации переменных z_1 и z_2. Так как по этим переменным заданный СБГ состоит из двух последовательных блоков, охваченных третьим, то для первых двух блоков верификацию будем проводить сверху вниз, а для третьего (после верификации последовательно соединенных блоков) — снизу вверх:

$$g_{11} = z_1 !x_2 \vee 0 \,\&\, x_2 = !x_2 z_1;$$
$$g_{12} = z_2 !x_2 \vee 1 \,\&\, x_2 = x_2 \vee z_2;$$
$$g_{21} = g_{11} !x_3 \vee g_{11} x_3 = !x_2 z_1;$$
$$g_{22} = g_{12} !x_3 \vee 0 \,\&\, x_3 = !x_3 (x_2 \vee z_2);$$
$$z_1 = 1 \,\&\, !x_1 \vee g_{21} x_1 = !x_1 \vee !x_2 z_1;$$
$$z_2 = z_2 !x_1 \vee g_{22} x_1 = !x_1 z_2 \vee !x_3 (x_1 x_2 \vee z_2).$$

Каждой из полученных двух формул соответствует граф переходов с двумя вершинами. Однако, так как эти графы должны вычисляться параллельно и иметь общие входные переменные, такое описание рассматриваемой части алгоритма проблему понятности не решает. Она решается, если по найденной СБФ построить один граф переходов автомата без выходного преобразователя (рис. 17.24).

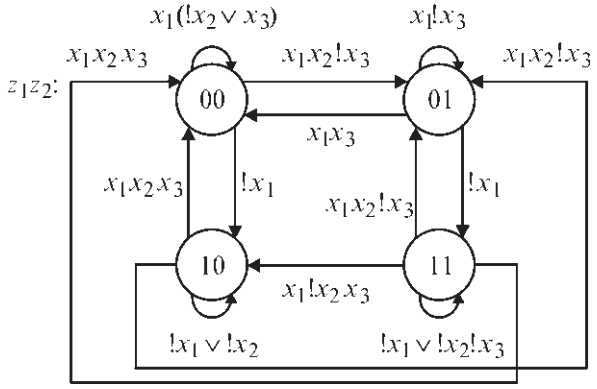


Рис. 17.24

Вводя переменную z_3 в построенный граф, получим ГП смешанного автомата (рис. 17.25), который эквивалентен заданному СБГ.

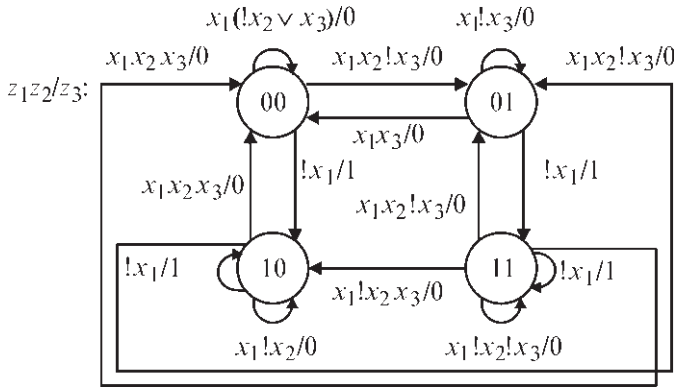


Рис. 17.25

Из приведенного примера следует, что сравнительно простой по структуре СБГ имеет весьма сложное поведение, описываемое построенным графом переходов.

Пример 17.8. Верифицировать бинарный граф (рис.17.22).

Перечислим пути в бинарном графе с учетом умалчиваемых значений переменной P:

$1.\,!P!x_1!x_2!S!P$; $2.\,!P!x_1x_2S!P$; $3.\,!Px_1!x_2!PS$; $4.\,!Px_1x_2P!S$;
$5.\,P!x_1!x_2!PS$; $6.\,P!x_1x_2P!S$; $7.\,Px_1!x_2!SP$; $8.\,Px_1x_2SP$.

Запишем эти выражения так, чтобы в каждом из них на двух последних позициях сначала располагалась переменная P или ее инверсия, а затем переменная S или ее инверсия:

$1.\,!P!x_1!x_2!S!P$; $2.\,!P!x_1x_2S!P$; $3.\,!Px_1!x_2S!P$; $4.\,!Px_1x_2!SP$;
$5.\,P!x_1!x_2S!P$; $6.\,P!x_1x_2!SP$; $7.\,Px_1!x_2!SP$; $8.\,Px_1x_2SP$.

Объединяя второе и третье выражения, получим $!P(x_1 \oplus \oplus x_2)S!P$, а шестое и седьмое — $P(x_1 \oplus x_2)!SP$. Исходя из изложенного, можно построить ГП автомата Мили (рис. 17.26).

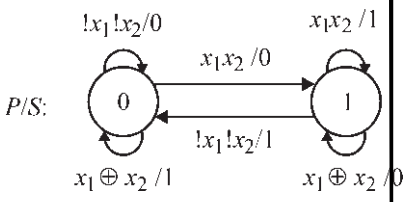
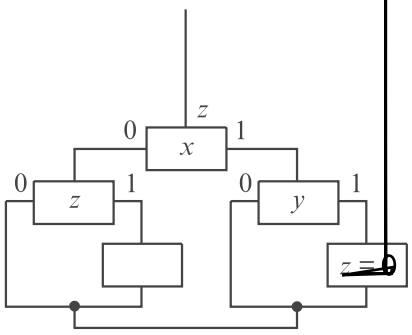


Рис. 17.26Рис. 17327

Построенный граф переходов существенно более компактен, обозрим и понятен по сравнению с исходным бинарным графом.

Пример 17.9. Верифицировать структурированный БГ (рис. 17.27).



Структура этого СБГ позволяет проводить его верификацию от входа к выходу:

$$g_{11} = z\,!\,z \vee zz = z; \qquad g_{21} = 1\,\&\,!y \vee 0\,\&\, y = !y;$$
$$g_{12} = 0\,\&\,!\,z \vee 1\,\&\, z = z; \qquad g_{22} = y\,!y \vee yy = y;$$

$$z_1 = g_{11}\,!\,x \vee g_{21}\,x = !\,xz \vee x\,!y;$$
$$y_1 = g_{12}\,!\,x \vee g_{22}\,x = !\,xz \vee xy.$$

По каждой из последних двух формул может быть построен ГП. Но так как эти ГП связаны между собой по всем переменным (входной, внутренней и выходной) и должны вычисляться параллельно, то понять алгоритм по этому описанию весьма сложно. Поэтому построим по полученной СБФ граф переходов автомата без выходного преобразователя (рис. 17.28).

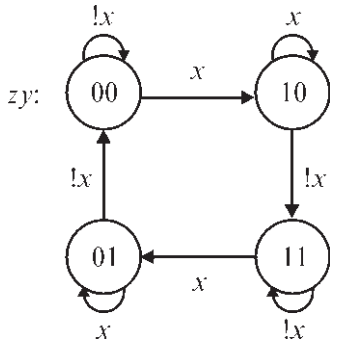


Рис. 17.28

Заданный алгоритм становится еще более понятным, если построенный граф преобразовать в граф переходов автомата Мура (рис. 17.29), так как при этом внутренняя и выходная переменные находятся на разных уровнях, что более соответствует физическому смыслу этих типов переменных.

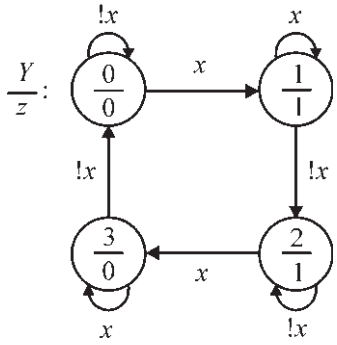


Рис. 17.29

Из рассмотрения последнего графа переходов следует, что заданный СБГ описывает работу счетного триггера, что весьма трудно понять по исходному заданию.

Из этого примера следует, что описание алгоритма с помощью графа переходов, в котором вершины пронумерованы десятичными числами, существенно более понятно по сравнению с описанием, в котором применяется БГ или СБГ без дешифратора состояний. Это объяснятся тем, что в таком графе переходов используются понятия «состояние» и «многозначное кодирование состояний», а в указанном БГ или СБГ применяются двоичные переменные, влияющие друг на друга и не позволяющие непосредственно по этой модели определить пространство состояний автомата и переходы в этом пространстве.

Пример верификации бинарного графа с внутренними обратными связями приведен в [3, 6].

17.5. Построение бинарных графов по графам переходов

Возможность использования методов построения бинарных графов, рассмотренных выше, ограничена, так как метод независимых фрагментов требует знания формул, описывающих автомат, а остальные методы, связанные с табличным заданием автомата, предназначены для решения задач небольшой размерности.

Покажем, что бинарные графы весьма просто могут быть построены по графам переходов. На первый взгляд кажется, что при наличии графа переходов нет необходимости в построении бинарного графа, однако она может возникнуть ввиду требований ГОСТа [7], который определяет применение бинарных графов для описания алгоритмов.

Для кодирования вершин графа переходов в общем случае могут применяться различные виды кодирования, что влияет на структуру бинарного графа, которая зависит также и от класса автомата с памятью, которому соответствует граф переходов [3, 6].

17.5.1. Автоматы без выходного преобразователя

Принудительное кодирование состояний. Если состояния автомата могут быть различимы только за счет использования значений выходных переменных, то будем называть такой вид кодирования принудительным [3].

Пусть задан ГП (рис. 17.30).

Так как все вершины этого графа помечены различными кортежами значений выходных переменных, то в данном случае может

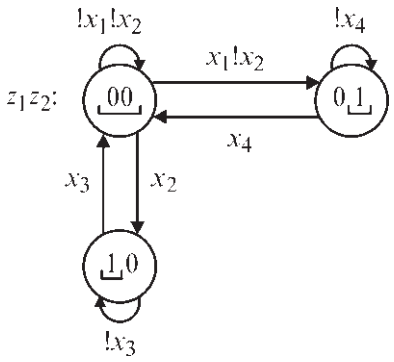


Рис. 17.30

применяться принудительное кодирование вершин, которое не требует введения дополнительных (внутренних) переменных для их различения.

Структурированный БГ, построенный по этому графу, приведен на рис. 17.31.

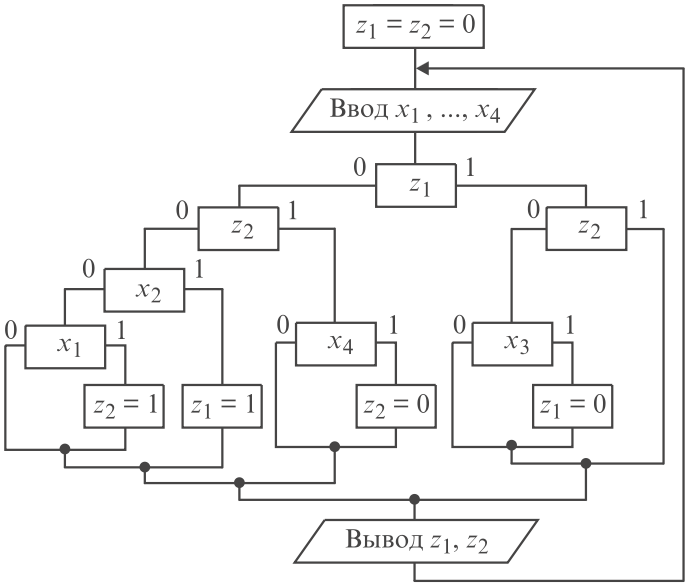


Рис. 17.31

Анализ заданного графа переходов позволяет упростить построенный СБГ за счет исключения в нем правой условной вершины с пометкой z_2 и подключения входа такой вершины с пометкой x_3 к единичному выходу условной вершины с пометкой z_1 (рис. 17.32).

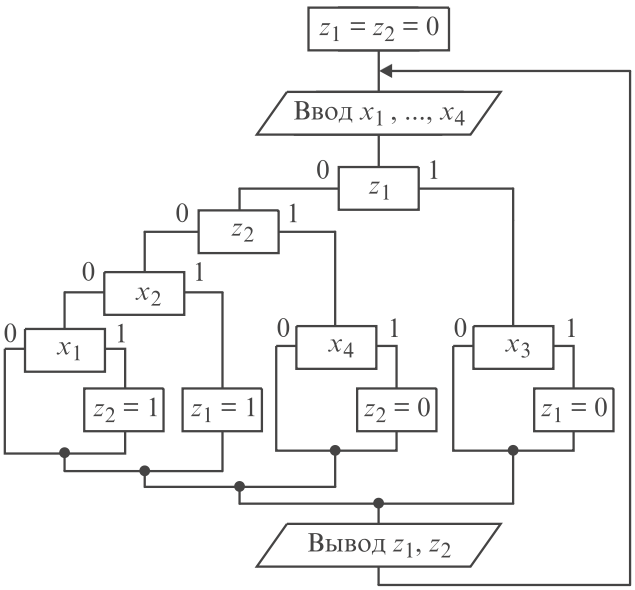


Рис. 17.32

Для упрощенного СБГ не может быть построен изоморфный ему граф переходов. Потеря изоморфизма приводит к тому, что этот СБГ становится плохо понимаемым, так как, на первый взгляд, кажется, что при $z_1 = 1$ значение z_2 может быть произвольным. Однако это не так, если учесть предысторию.

В силу того что в данном случае цель построения СБГ состоит прежде всего в том, чтобы весьма просто понимать его, а не в том, чтобы в нем «разбираться», то применение упрощенного СБГ по этому критерию нецелесообразно.

Обратим внимание, что в рассматриваемом примере для кодирования вершин могут использоваться неполные кортежи значений выходных переменных (неполные коды) [3]. Эти коды в графе переходов на рис. 17.30 отмечены скобками. Применение таких кодов целесообразно при программной реализации на основе СБФ, которая в данном случае имеет вид:

$$z_{11} = x_2 ! z_1 ! z_2 \vee ! x_3 z_1;\ z_2 = x_1 ! x_2 ! z_1 ! z_2 \vee ! x_4 z_2;\ z_1 = z_{11}.$$

При этом отметим, что построить ветвящийся бинарный граф, изоморфный ГП с неполными кодами, не удается.

Принудительно-свободное кодирование состояний. Если состояния автомата для их различения кодируются значениями выходных и дополнительно вводимых внутренних переменных, то такой вид кодирования будем называть принудительно-сво-

бодным [3], так как в этом случае первая часть каждого кода определяется принудительно значениями выходных переменных в этом состоянии, а вторая ввиду программной реализации автомата — свободно.

Граф переходов на рис. 17.33, описывающий алгоритм работы счетного триггера, не может быть реализован непосредственно, так как в нем соответствующие пары вершин неразличимы.

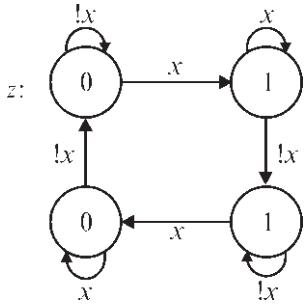


Рис. 17.33

Вводя дополнительную (промежуточную) переменную, используем принудительно-свободное кодирование, которое в данном случае является также и противогоночным (рис. 17.28). Этот граф переходов изоморфен СБГ на рис. 17.34. В теле этого графа первые два яруса образуют дешифратор состояний.

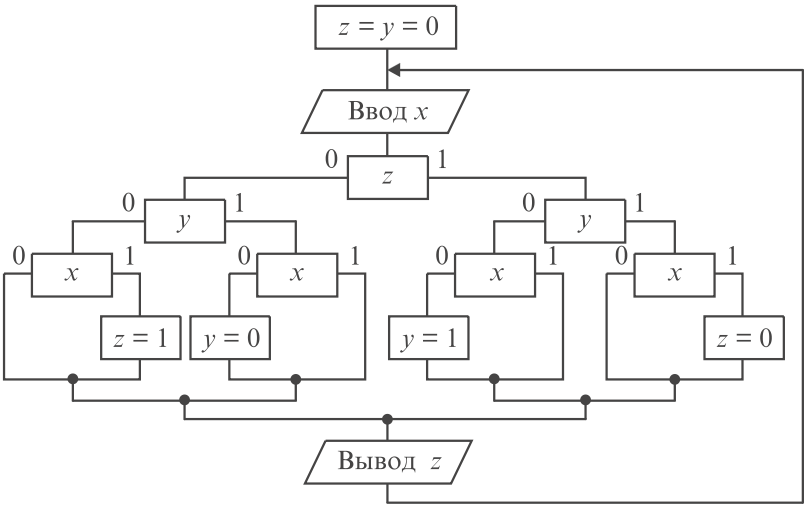


Рис. 17.34

В [3] показано, что этот СБГ может быть существенно упрощен за счет потери изоморфизма с графом переходов, если учесть,

что в данном случае для обеспечения корректной реализации в каждой вершине графа (рис. 17.28) может быть выбрано значение лишь одной переменной, которое совместно со значением входной переменной, помечающей петлю этой вершины, обеспечит сохранение состояния (рис. 17.35).

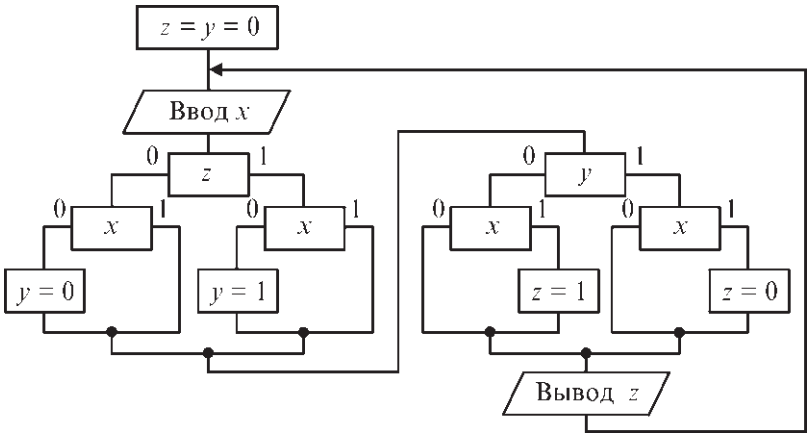


Рис. 17.35

Этот СБГ в отличие от предыдущего существенно менее понятен, так как в нем «разрушен» дешифратор состояний за счет применения в каждом блоке входной переменной *x* между переменными *z* и *y*, характеризующими состояния автомата.

На рис. 17.27 представлен еще более простой СБГ, реализующий счетный триггер [3]. Однако этот граф еще менее понятен, так как в нем от дешифратора состояний не осталось и следа.

Последний СБГ характерен для событийного программирования [8], так как он начинается с дешифратора значений входной переменной, называемого также дешифратором событий, каждое из которых совместно со значением выходной (внутренней) переменной приводит к формированию определенного значения внутренней (выходной) переменной.

Сформулируем несколько общих положений, характеризующих предлагаемый подход.

В [3] были введены понятия «автоматное программирование» и «автоматное проектирование программ», которые базируются на совместном использовании таких понятий, как «состояние» и «событие». В рамках предложенного в [3] подхода основным является понятие «состояние», так как при этом в любой момент времени можно в явном виде определить, в какой точке «пространства состояний» находится программа. Под состоянием при этом (в отличие от [9]) понимается не совокупность значений внутренних переменных, а некоторая абстракция, которая

лишь после кодирования приобретает связь с переменными. Например, для такого объекта, как вода, основным является то, что она может находиться в трех состояниях (жидкое, твердое и газообразное), в то время как значения переменной, характеризующие эти состояния, являются вторичными. Отметим, что это, в частности, позволяет применять ГП и при «нечетком управлении».

Если программирование проводить, как это делается традиционно, на основе использования внутренних переменных, а не состояний, то обычно приходится применять большое число таких переменных, являющихся, как правило, д в о и ч н ы м и, при непосредственном рассмотрении которых весьма трудно определить число состояний, кодируемых ими.

Если проектирование программы начинать с выделения состояний, то априорное знание их числа позволяет применять м н о г о з н а ч н о е кодирование, при котором для каждой компоненты программы, соответствующей одному автомату, достаточно использовать только о д н у внутреннюю переменную вне зависимости от числа выделенных состояний, значность которой равна числу этих состояний. При этом код состояния совпадает с номером соответствующей вершины в графе переходов. Это позволяет ввести в программирование понятие «наблюдаемость» (по одной внутренней переменной для каждой компоненты программы).

При логическом управлении состояния управляющего автомата могут быть выделены «естественно», если первоначально каждое из них однозначно сопоставить с физическим состоянием управляемого объекта. Например, для исправного клапана, который может находиться в четырех состояниях («Закрыт», «Открывается», «Открыт», «Закрывается»), управляющий им автомат Мура первоначально должен строиться также с четырьмя состояниями, каждое из которых поддерживает объект в требуемом состоянии. При этом переход в автомате, выполняемый под воздействием внешнего события (например, нажатие соответствующей кнопки), обеспечивает аналогичный переход в объекте, выполняемый под воздействием значений выходных переменных, формируемых автоматом в новом состоянии. В дальнейшем при необходимости за счет потери указанного однозначного соответствия может ставиться вопрос о минимизации числа состояний автомата, которая в рамках предлагаемого подхода может уменьшить лишь значность, а не число внутренних переменных. Например, для клапана с памятью вместо автомата с четырьмя состояниями может использоваться автомат с тремя состояниями, в котором состояния, соответствующие состояниям объекта «Закрыт» и «Открыт», объединены.

Отметим также, что в рамках предлагаемого подхода для системы, содержащей N автоматов (при ограничении, когда в каждом программном цикле в каждом автомате выполняется не более одно-

го перехода), *N* многозначных переменных позволяют не только закодировать состояния автоматов, но и осуществить их взаимодействие за счет вложенности или вызываемости. При этом у вызываемого ГП на соответствующей дуге выполняется проверка номера состояния, в котором должен находиться вызывающий граф, инициирующий переход по указанной дуге.

Из изложенного следует, что в рамках предлагаемого подхода при необходимости применения бинарных графов их построение должно начинаться с дешифратора состояний, после каждого выхода которого должен размещаться дешифратор входных событий. Построенные таким образом бинарные графы, несмотря на громоздкость, весьма просто понимаются, так как они изоморфны графам переходов, что резко упрощает проверку их функциональной эквивалентности. В то время как для бинарных графов, построенных иначе, проверка эквивалентности с графами переходов (верификация), как было показано в разд. 17.4, весьма трудоемка.

Обратим внимание также на то, что при использовании ЭВМ в контуре управления на дисплее отражаются состояния процесса, а события формируются, например, с помощью клавиатуры. Поэтому для обеспечения хорошей понимаемости управляющих программ принцип их построения должен совпадать с принципом управления, в соответствии с которым Оператор сначала определяет состояние процесса, а затем формирует событие, обеспечивающее изменение этого состояния. Странной является ситуация, в которой управление производится по одному принципу (начиная с анализа состояния), а написание программы, обеспечивающей управление, выполняется на основе прямо противоположного принципа (начиная с анализа события). Это иллюстрируется примером, приведенным в «Приложении 11» в [3].

В заключение раздела отметим, что недостатком рассмотренных в нем принудительного и принудительно-свободного видов кодирования является сложность дешифратора состояний, так как они основаны на применении двоичных переменных.

Принудительно-свободное кодирование обладает и еще одним недостатком: дополнительно вводимые внутренние переменные изображаются в графе переходов на одном уровне с выходными переменными, что отвлекает внимание Заказчика, которого внутренние переменные обычно не интересуют.

17.5.2. Автоматы Мура

Применение модели автомата Мура [3] позволяет внутренние и выходные переменные сделать «разноуровневыми».

Двоичное логарифмическое кодирование. Предположим, что автомат имеет *S* состояний. Присвоим каждому из них код дли-

ны]log S[. Такое кодирование называется двоичным логарифмическим.

На рис. 17.36 приведен граф переходов счетного триггера, в котором использован один из двоично-логарифмических кодов. При этом отметим, что при программной реализации этот код может не быть противогоночным.

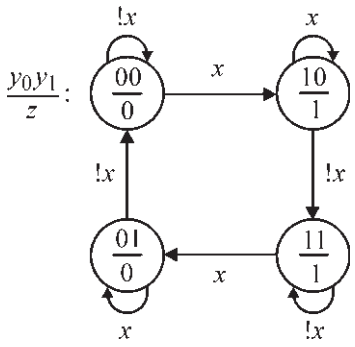


Рис. 17.36

Этому ГП изоморфен СБГ, приведенный на рис. 17.37.

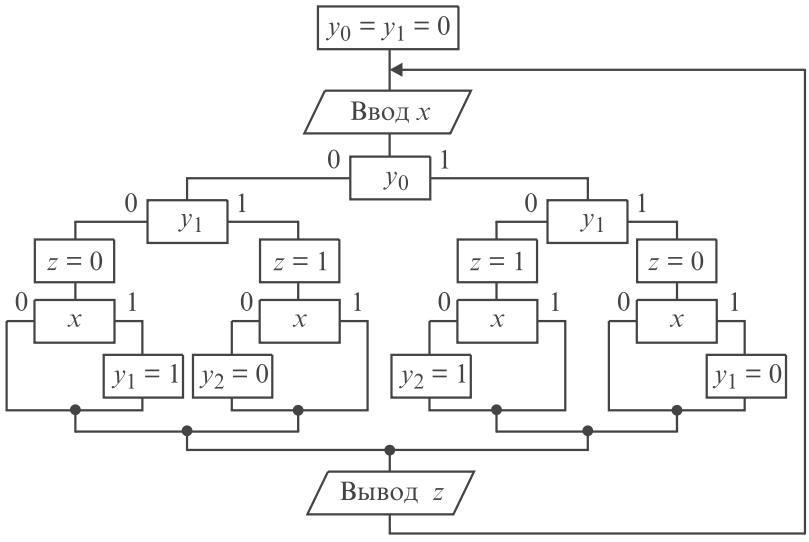


Рис. 17.37

Двоичное кодирование состояний. При таком кодировании каждому состоянию присваивается двоичная переменная, которая принимает единичное значение в этом состоянии и ноль во всех

остальных. На рис. 17.38 приведен граф переходов счетного триггера, в котором применен этот вид кодирования.

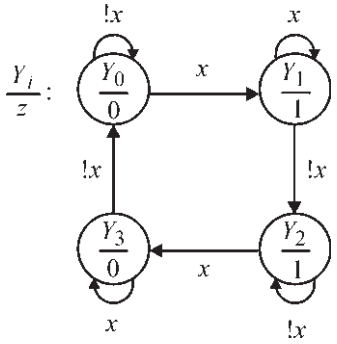


Рис. 17.38

Двоичное кодирование порождает наиболее простые булевы формулы, описывающие поведение автомата, каждая из которых всегда является бесповторной по переменным, кодирующим состояния:

$$Y_{01} = !\,x(Y_3 \vee Y_0);\ Y_{11} = x(Y_0 \vee Y_1);$$
$$Y_{21} = !\,x(Y_1 \vee Y_2);\ Y_3 = x(Y_2 \vee Y_3);$$
$$z = Y_1 \vee Y_2;\ Y_0 = Y_{01};\ Y_1 = Y_{11};\ Y_2 = Y_{21}.$$

Построенному графу переходов изоморфен СБГ на рис. 17.39.

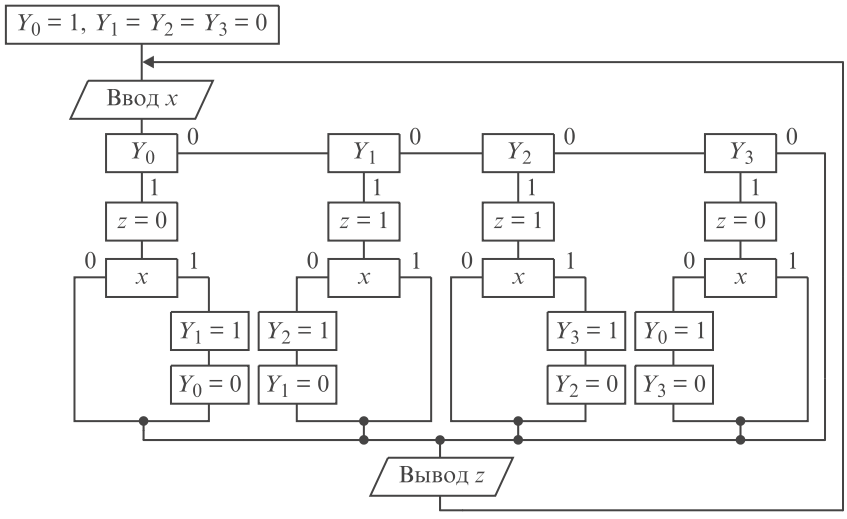


Рис. 17.39

Недостаток этого вида кодирования состоит в необходимости использования S двоичных переменных, каждая из которых требует принудительного сброса.

Многозначное кодирование состояний. При таком кодировании каждому состоянию присваивается одно из значений одной S-значной переменной, которую нет необходимости принудительно сбрасывать.

На рис. 17.29 приведен граф переходов счетного триггера, в котором применен этот вид кодирования. Построенному графу переходов изоморфен СБГ на рис. 17.40.

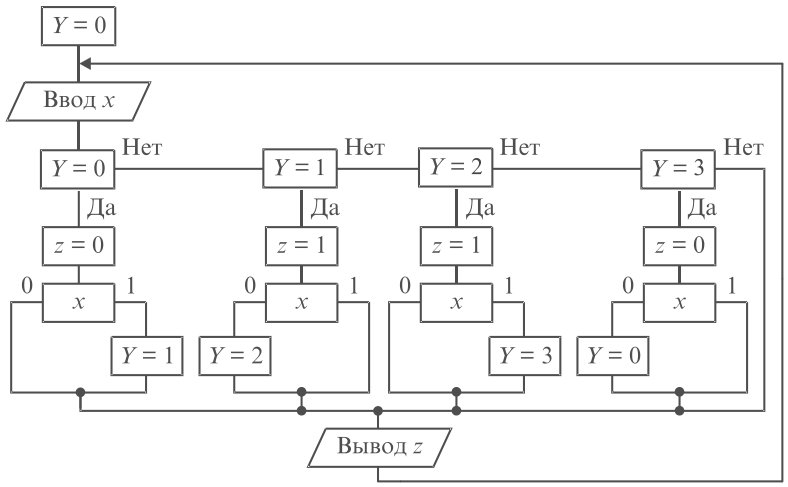


Рис. 17.40

Тело этого СБГ может быть изоморфно реализовано конструкцией `switch` языка СИ:

```
        switch(Y){
case0:  z=0;
        if(x)   Y=1;
        break;
case1:  z=1;
        if(!x)  Y=2;
        break;
case2:  /* z=1; */
        if(x)   Y=3;
        break;
case3:  z=0;
        if(!x)  Y=0;
        break;   }
```

В этой программе неизменяющееся при переходе значение выходной переменной в состоянии 2 сохранено, но закомментировано,

что позволяет, не снижая понятности программы, сократить ее объектный код.

Отметим, что в этой программе в каждой метке `case` происходит присвоение нового значения переменной *Y*, входящей собственно в конструкцию `switch`, что характерно для автоматов с памятью. При этом программа в целом изоморфна графу переходов, что обеспечивает ее хорошую «понимаемость».

В событийно-управляемых программах по значению переменной, соответствующей событию, в общем случае осуществляется вызов процедуры [8], которая вовсе не обязательно должна изменять значения этой переменной (пример *2* Приложения). Однако и в этом случае программы (ввиду наличия внешней обратной связи) могут реализовать автоматы с памятью, поведение которых по тексту или СБГ понять весьма трудно из-за отсутствия изоморфизма с эквивалентным ГП.

Ниже приводится событийно-управляемая программа, реализующая счетный триггер и построенная по СБГ (рис. 17.27):

```
        switch(x){
case0:  if(z) y=1;
        else  y=0;
        break;
case1:  if(y) z=0;
        else  z=1;
        break;   }
```

Сравнение этой программы с предыдущей показывает, что если во второй программе переключатель по переменной *x* является комбинационным (не изменяет значения этой переменной), то в первой из них переключатель по переменной *Y* является последовательностным (изменяет значения этой переменной). По этой причине понять по второй программе, что она реализует автомат с памятью и какой граф переходов ей эквивалентен, существенно труднее, чем по первой программе.

Таким образом, применение конструкции `switch` позволяет, структурируя программу, упростить ее понимание только при предлагаемом подходе к ее построению, что не было отмечено в теории структурного программирования [10, 11]. При этом отметим, что метод Ашкрофта и Манны в этом смысле весьма близок к предлагаемому, но обладает рядом недостатков, отмеченных в [3].

Изложенное, видимо, явилось одной из причин наблюдающегося в настоящее время массового перехода от процедурного к объектно-ориентированному подходу к проектированию сложных программ, при котором поведение классов (объектов) предлагается описывать не с помощью бинарных графов, характерных для процедурного подхода, а с помощью графов переходов, называемых в [9] диаграммами состояний и переходов. Предлагаемый в настоящем разделе метод построения би-

нарных графов может при необходимости сохранить их применение и при объектно-ориентированном подходе.

Структурированный бинарный граф (рис. 17.40) и изоморфная ему программа обладают тем недостатком, что в конце программного цикла выходная переменная *z* всегда сохраняет старое значение, а переменная *Y*, соответствующая состояниям, может принять новое значение. Этого недостатка лишен СБГ, которому изоморфна следующая программа:

```
        switch(Y){
case0:  if(x)   Y=1;
        break;
case1:  if(!x)  Y=2;
        break;
case2:  if(x)   Y=3;
        break;
case3:  if(!x)  Y=0;
        break;   }
        switch(Y){
case0:    z=0;    break;
case1:    z=1;    break;
case2:  /* z=1 */; break;
case3:    z=0;    break;}
```

Программе, построенной по СГБ на рис. 17.40, соответствует структурная модель автомата Мура первого рода, а последней программе — структурная модель автомата Мура второго рода [3, 12].

17.5.3. Автоматы Мили

Для автоматов этого класса [3] могут использоваться те же виды кодирования, что и для автоматов Мура. На рис. 17.41 приведен бинарный граф, изоморфный графу переходов на рис. 17.26.

Тело этого бинарного графа реализуется следующей программой:

```
        switch(P) {
case0:  if(!x1& !x2)  S=0;
        if( x1⊕ x2)  S=1;
        if( x1&  x2) {S=0; P=1;  } break;
case1:  if(!x1& !x2) {S=1; P=0;  }
        if( x1⊕ x2)  S=0;
        if( x1&  x2)  S=1; break;}
```

В рассмотренном бинарном графе (за счет уменьшения изоморфизма с графом переходов) после перестановки последовательно

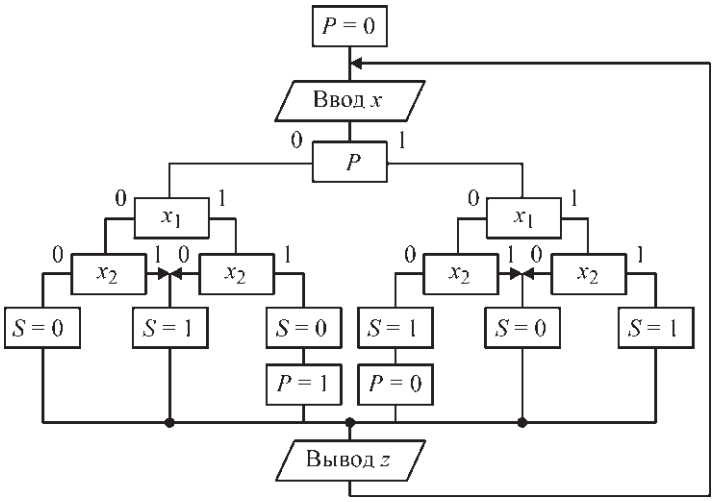


Рис. 17.41

соединенных операторных вершин с пометками S и P число таких вершин может быть сокращено до четырех.

Аналогично можно показать, как по графам переходов строятся бинарные графы для смешанных автоматов и управляющих автоматов, содержащих функциональные элементы задержки [3].

В заключение разд. 17.5 приведем предложенный автором и Б. П. Кузнецовым (НПО «Аврора») структурированный бинарный граф (рис. 17.42), который обеспечивает реализацию взаимо-

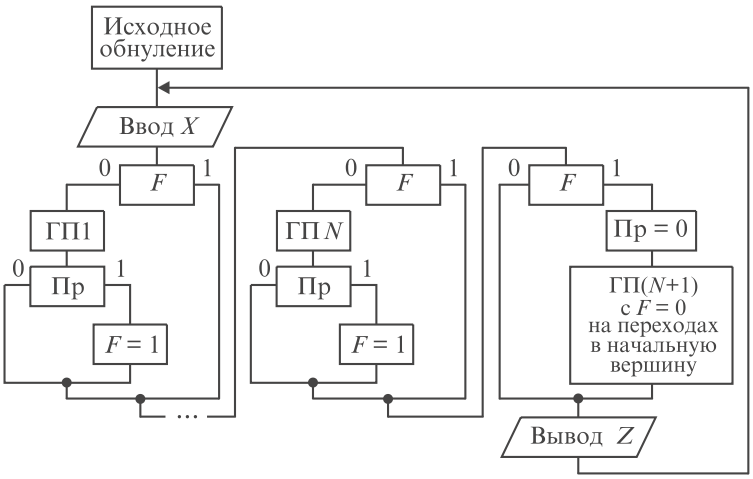


Рис. 17.42

связанной системы из N графов переходов и переход к выполнению $(N + 1)$-го графа переходов при появлении сигнала прерывания (Пр = 1). В последнем графе на переходах в начальную вершину установленный ($F = 1$) ранее флаг должен сбрасываться ($F = 0$).

17.6. Программная реализация графов переходов изоморфными «схемами» из функциональных блоков

Одним из языков программирования программируемых логических контроллеров (ПЛК) является язык функциональных блоков [13]. При использовании этого языка программирование ПЛК и промышленных компьютеров [14, 21] сводится к построению функциональной схемы (ФС) в базисе заданных функциональных блоков.

Если по ГП построить систему булевых формул и реализовать ее функциональной схемой, то она не будет изоморфна ГП.

Ниже предлагаются два метода построения функциональных схем непосредственно по графам переходов, позволяющих устранить этот недостаток. Первый метод применим к ГП с принудительным и принудительно-свободным кодированием состояний, а второй — к ГП с многозначным кодированием состояний [23].

17.6.1. Первый метод

Этот метод, предложенный в [3], строит функциональные схемы, которые содержат три составляющие:

— дешифратор состояний;
— формирователь условий переходов;
— триггеры, фиксирующие коды состояний.

Пример 17.10. Построить ФС для программной реализации графа переходов, в котором используются полные коды (рис. 17.30).

Эта схема приведена на рис. 17.43.

Программная реализация построенной схемы выполняется в порядке, указанном цифрами, расположенными у выходов элементов.

17.6.2. Второй метод

В соответствии со стандартом [13], в библиотеке функциональных блоков при программной реализации функциональных схем рекомендуется наряду с другими блоками иметь логические и цифровые мультиплексоры.

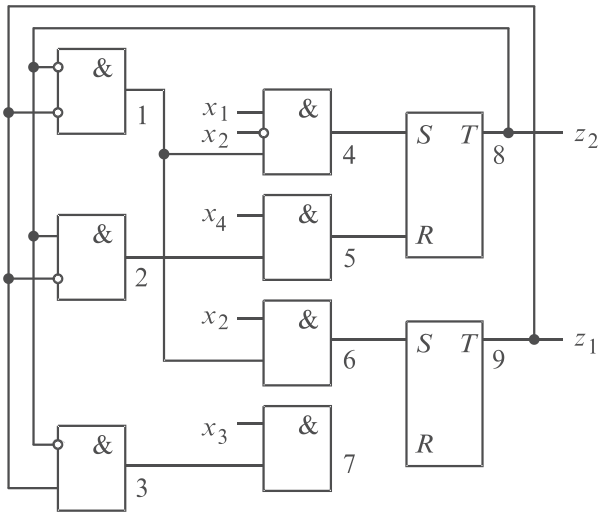


Рис. 17.43

Такие мультиплексоры в отличие от мультиплексоров, реализуемых аппаратно, предназначены для передачи с каждого информационного входа на выход десятичного числа. Выбор информационного входа, как и при аппаратной реализации, выполняется с помощью управляющих входов.

При этом логические мультиплексоры имеют один или несколько управляющих входов, на которые подаются логические переменные, а цифровые мультиплексоры имеют S информационных входов и один управляющий вход, на который подается многозначная переменная, значение которой обеспечивает коммутацию соответствующего информационного входа с выходом.

Если наряду с этими блоками использовать также преобразователь «десятичное число — двоичное число» (Dec — bin), обозначаемый символом «D/b», то появляется возможность непосредственной реализации графов переходов автоматов Мура с многозначным кодированием состояний с помощью изоморфной функциональной схемы, состоящей только из указанных типов блоков [15].

Схемы, которые строятся предлагаемым методом, не содержат триггеров и, по мнению автора, являются при наличии реализуемых ГП наиболее понятными по сравнению с другими разновидностями функциональных схем, реализующих автоматы с памятью.

Пример 17.11. Построить ФС, программно реализующую граф переходов на рис. 17.44.

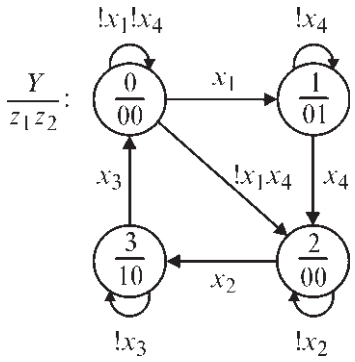


Рис. 17.44

Схема, изоморфная этому графу переходов, приведена на рис. 17.45.

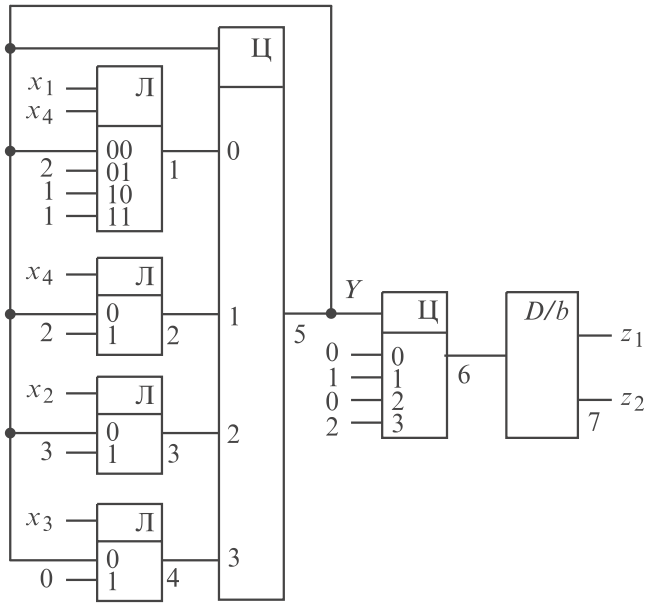


Рис. 17.45

Если число информационных входов логических мультиплексоров ограничено, то в построенной схеме сложный логический мультиплексор может быть заменен схемой из более простых мультиплексоров. На рис. 17.46 приведена схема, в которой может быть заменен элемент с номером 1 в схеме на рис. 17.45.

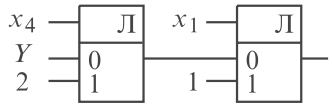


Рис. 17.46

В общем случае функциональные схемы рассматриваемого типа могут быть упрощены (при реализации будут занимать меньший объем памяти), если при их построении наряду с рассмотренными типами блоков применять также логические элементы, реализующие булевы формулы, помечающие дуги, исходящие из вершин ГП. Это целесообразно при выполнении соотношения

$$2^{n_i} > 2^{]\log m_i[},$$

где n_i — суммарное число переменных, помечающих все m_i дуг, исходящих из i-й вершины графа переходов.

Для рассмотренного примера применение логических элементов нецелесообразно, так как в этом случае для всех вершин $n_i =]\log m_i[$.

Если в ГП на рис. 17.44 пометку дуги между вершинами 0 и 2 заменить на $!x_1 x_4 x_5$, то вместо логического мультиплексора «8 в 1», соответствующего нулевой вершине графа переходов, может быть использована одна из двух схем на рис. 17.47.

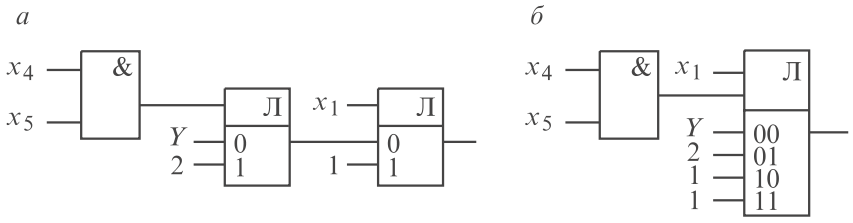


Рис. 17.47

Более подробно (включая реализацию систем графов переходов) рассмотренный метод изложен в [3].

Этот метод был применен в НПО «Аврора» при создании системы логического управления дизель-генератором ДГР-2А 500*500 судна проекта 15640 на базе аппаратуры «Selma-2» фирмы «ABB Stromberg» [16, 17].

Выводы

1. Предложен метод независимых фрагментов, позволяющий строить операторы структуры структурированных бинарных графов (СБГ) для автоматов с памятью. Этот оператор описывает структу-

ру СБГ так же, как булева формула в базисе {&, ∨, !} описывает структуру параллельно-последовательной контактной схемы, но в отличие от булевой формулы, описывающей и функционирование схемы, оператор не описывает функционирование СБГ.

2. Рассмотрен вопрос об использовании канонического и предложенного автором распределительного методов, а также модификации канонического метода для построения бинарных графов для автоматов с памятью.

3. Предложен новый критерий качества бинарных графов («понятность») и введен формальный показатель для его применения.

4. Рассмотрен вопрос о верификации бинарных графов, реализующих автоматы с памятью.

5. Для различных видов кодирования состояний и типов автоматов предложены стандартные структуры СБГ, которые изоморфны графам переходов и конструкции `switch` языка СИ. Такие СБГ в явном виде содержат связи не только по управлению, но и по данным. Они начинаются с дешифратора состояний, а не событий, как это принято в событийном программировании, что делает предлагаемые СБГ понятными. Такой подход связывает структурное программирование с объектно-ориентированным.

6. Предложены методы программной реализации графов переходов изоморфными «схемами» из функциональных блоков, в которые при необходимости легко могут быть внесены изменения.

7. Методы реализации автоматов с памятью в базисах булевых формул, языков инструкций и ассемблеров, лестничных схем, а также алгоритмических языков высокого уровня подробно рассмотрены в монографии автора [3]. В ней также рассмотрены новые подходы к реализации диаграмм «Графсет», используемых при реализации алгоритмов логического управления, содержащих параллельно протекающие процессы. Показаны преимущества систем взаимосвязанных графов переходов с многозначным кодированием вершин в каждом из них по сравнению с этими диаграммами при реализации таких алгоритмов.

8. Вопрос о противогоночном кодировании состояний асинхронных автоматов, которые реализуют простые бинарные графы, вычисляющие системы булевых функций, рассмотрен в [18]. При этом показано, что существуют такие системы функций, для которых противогоночное кодирование состояний является безызбыточным.

9. Исследование переходных процессов в одноконтурных схемах («четные контуры») выполнено В. В. Киселевым (НПО «Аврора») [19] при участии автора [20]. При этом был предложен метод построения функциональных элементов задержки, длительность запаздывания которых может превышать сумму запаздываний элементов, их составляющих. Этот метод описан в [3, 22].

Литература

1. *Баранов С. И.* Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.

2. *Baranov S.* Logic synthesis for control automata. Boston: Kluwer academic publishers, 1994.

3. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

4. *Вирт Н.* Программы = алгоритмы + данные. М.: Мир, 1988.

5. *Миллер Г.* Магическое число семь плюс или минус два: о некоторых пределах нашей способности перерабатывать информацию // Инженерная психология. М.: Прогресс, 1964.

6. *Шалыто А. А.* Использование граф-схем алгоритмов и графов переходов при программной реализации алгоритмов логического управления // Автоматика и телемеханика. 1996. № 6, 7.

7. *Схемы* алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. ГОСТ 19.701-90 (ИСО 5807-85).

8. *Матчо Д., Фолкнер Д.* DELPHI. М.: Бином, 1995.

9. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на С++. М.: Бином; СПб.: Невский диалект, 1998.

10. *Иодан Э.* Структурное проектирование и конструирование программ. М.: Мир, 1979.

11. *Лингер Р., Миллс Х., Уитт С.* Теория и практика структурного программирования. М.: Мир, 1982.

12. *Шалыто А. А., Антипов В. В.* Алгоритмизация и программирование задач логического управления техническими средствами. СПб.: Моринтех, 1996.

13. *International standart IEC 1131-3.* Programmable controllers. Part 3. Programming languages // International Electrotechnical Commision. 1993.

14. *ISaGRAF.* Computer aided software engineering workbench for open PLCs and industrial computers. Version 2.10. User's guide. France: CJ International, 1994.

15. *Кондратьев В. Н., Шалыто А. А.* Использование функциональных схем при программной реализации автоматов // Судостроит. пром-сть. Сер. Автоматика и телемеханика. 1991. Вып.13.

16. *«Селма-2».* Описание функциональных блоков. АББ Стромберг, 1989.

17. *Project 15640.* AS21. DG1. Control. АМИЕ. 95564. 12M. St. Petersburg. ASS «Avrora», 1991.

18. *Сагалович Ю. Л., Шалыто А. А.* Бинарные программы и их реализация асинхронными автоматами // Проблемы передачи информации. 1987. Вып. 1.

19. *Киселев В. В.* Функциональные возможности четных контуров при реализации временных программ // Вопросы судостроения. Сер.Судовая автоматика. 1982. Вып. 27.

20. *Киселев В. В., Шалыто А. А.* Исследование переключательных процессов в «четных» контурах // Проблемы комплексной автоматизации судовых технических средств. Тез. докл. к VII Всесоюз. научно-техн. конф. Л.: ВНТО им. акад. А.Н.Крылова, 1989.

21. *Шакиров С., Биюсов Р., Якубович Б., Журавлев В.* ULTRALOGIC — система подготовки программ для промышленных контроллеров // Современные технологии автоматизации. 1997. № 3.

22. *Киселев В. В., Шалыто А. А.* Исследование переходных процессов в одноконтурных логических схемах // Известия РАН. Теория и системы управления. 1999. № 5.

23. *Шалыто А. А.* Реализация алгоритмов логического управления программами на языке функциональных блоков // Промышленные АСУ и контроллеры. 2000. № 4.