Глава 6

Использование конструкции switch для реализации граф-схем алгоритмов

Излагаемый в настоящей работе подход предполагает использование ГП в качестве языка спецификаций автоматов. Однако весьма часто в качестве такого языка применяются ГСА. Программы, которые строятся по ГСА с помощью конструкций `if`, `then`, `else` (даже без применения конструкций `goto` и меток) и отступов, весьма трудно читаются и понимаются.

Поэтому возникает задача построения по ГСА более читаемых программ.

Остановимся в этой главе на рассмотрении граф-схем второго вида (по классификации ГСА, приведенной в гл. 1).

Решение поставленной задачи для ГСА первого вида изложено в разд. 13.3.

6.1. Реализация автоматных ГСА без внутренних обратных связей и промежуточных переменных

Рассмотрим первоначально класс автоматных ГСА. Будем называть ГСА автоматной, если в ее операторных вершинах допустимо использование только трех типов операторов для выходной переменной y_i:

$$y_i = 0; \quad y_i = 1; \quad y'_i = y_i.$$

Автоматные ГСА можно разделить на два класса: без внутренних обратных связей (ВОС) и с такими связями.

Каждый из этих классов в свою очередь может быть разделен на две разновидности: без внутренних переменных и с этими переменными.

В системах логического управления обычно используются ГСА без ВОС, в то время как (будет показано в гл. 13) внутренние переменные в общем случае весьма целесообразно применять в составе граф-схем с канонической структурой, но нецелесообразно «хаотическое» применение проверок этих переменных, так как при этом затрудняется понимание граф-схем и соответствующих им программ.

ГСА без ВОС и внутренних переменных могут быть разделены на два подкласса: с проверками значений выходных переменных (рис. 4.74) и без таких проверок (рис. 4.75).

Эти проверки могут размещаться «хаотически» (рис. 4.74) или занимать определенные позиции в составе канонической структуры (рис. 4.82).

В ГСА без ВОС соотношения для выходных переменных $y_i' = y_i$ или $z_i' = z_i$ могут указываться явно (рис. 4.75) или умалчиваться (рис. 4.76).

При этом будем предполагать, что если в некоторой операторной вершине какая-либо выходная переменная не упоминается (умалчивается), то ее предыдущее значение в этой вершине сохраняется (рис. 4.84).

Использование умолчаний существенно ухудшает понимание граф-схем и построенных на их основе программ.

Рассмотрим в качестве примера ГСА (рис. 4.84). Эта ГСА при традиционном подходе реализуется весьма простой программой:

```
y1 = 0;      y2 = 0;
M:Ввод x1,   x2, x3;
  if (x1)    y2 = 1;
  if (x2)    y1 = 1;
  if (x3)    y1 = 0; y2 = 0;
  Вывод y1,  y2;
  goto       M.
```

Однако чтение такой ГСА и соответствующей ей программы (определение порядка выполнения операторов), а тем более их понимание (определение в каждом операторе значения каждой выходной переменной) весьма затруднительноы.

Чтение ГСА связано с трассировкой путей при различных значениях входных переменных. При этом в модели ГСА в явном виде не указывается, какие пути необходимо трассировать для получения требуемых результатов.

Понимание ГСА осложняется тем, что при определении значений каждой из не указанных в операторе переменных необходимо знать их предшествующие значения, что в рамках традиционной модели ГСА не обеспечивается и требует введения внемодельного объекта — памяти, в ячейки которой заносятся и из которых читаются вычисленные значения.

Этот недостаток устраняется в модели цифровой вычислительной машины (ЦВМ), в которую наряду с памятью программ входит также и память для запоминания промежуточных результатов, что позволяет ЦВМ (в отличие от человека — с весьма ограниченной оперативной памятью [202]) достаточно просто «читать» и выполнять программы.

В заключение раздела отметим, что используемая в настоящей работе трактовка умолчаний определяется программной реализацией граф-схем и отличается от применяемой, например, в работе [16], ориентированной на аппаратную реализацию ГСА. В этой работе каждая операторная вершина тела граф-схемы считается выходной и в ней указываются обозначения только тех выходных переменных, которые принимают в ней единичные значения, а для всех остальных переменных предполагается, что они принимают в этой вершине нулевые значения и их обозначения умалчиваются.

В ГСА, предназначенных для программной реализации, обычно предполагается, что вывод значений выходных переменных осуществляется только в конце каждой граф-схемы, в то время как в ГП формирование

окончательных значений выходных переменных осуществляется в каждой вершине (автоматы без выходного преобразователя и автоматы Мура) или на каждой дуге (автоматы Мили).

Для построения графа переходов по ГСА сначала определим, применяя метод верификации (разд. 4.3.4), СБФ, соответствующую рассматриваемой граф-схеме.

Для ГСА (рис. 4.84) в разд. 4.3.4 построена СБФ:

$$y_1' = \bar{x}_3(x_2 \vee y_1), \qquad y_2' = \bar{x}_3(x_1 \vee y_2).$$

Эта СБФ может быть запрограммирована непосредственно, однако понимание программы в такой форме еще сложнее, чем понимание ГСА.

Для устранения этого недостатка построим по этой СБФ граф переходов автомата без выходного преобразователя (рис. 4.26), для которого характерны следующие особенности:

— он может использоваться в качестве инструкции по работе оператора, управляющего технологическим процессом, так как в ГП в явной форме указано, какие входные сигналы и в какой последовательности должны подаваться для получения указанных в вершинах графа значений выходных переменных;

— для его анализа не требуется привлечения внемодельных средств;

— все значения выходных переменных в каждой вершине обычно указываются явно (отсутствуют прочерки);

— устойчивость вершин ГП обеспечивается наличием петель, соответствующих запоминанию выходных сигналов без указания конкретных типов элементов памяти; значения этих сигналов, формируемых в некоторой вершине графа, сохраняются, до тех пор пока выполняется условие, помечающее петлю, связанную с этой вершиной;

— все условия перехода из одной вершины ГП в другую указаны на дугах в компактной форме, а именно булевыми формулами (в общем случае — скобочными произвольной глубины);

— локальные изменения алгоритма работы приводят лишь к локальным и (или) упорядоченным изменениям в ГП, что связано с отсутствием зависимости значений выходных переменных от предыстории и с возможностью обеспечения или исключения связи любой вершины с любой из имеющихся или с любой вновь вводимой вершиной; при этом пометки остальных дуг и петли, соответствующих исходной вершине, в общем случае требуют корректировки;

— весьма просто вносятся и некоторые глобальные изменения, такие, например, как введение дополнительных выходов;

— он эквивалентен исходной ГСА в «большом», но не эквивалентен в «малом». Эквивалентность в «большом» следует понимать так, что для рассматриваемого примера и исходного состояния ($y_1 = y_2 = 0$) при $y_1 = y_2 = y_3$ ГСА и ГП формируют на выходе одинаковые значения $y_1 = y_2 = 0$. Неэквивалентность в «малом» следует понимать так, что в ГСА этот переход осуществляется через промежуточные значения 01 и 11, которые, правда, не выдаются на выход:

$$00 \xrightarrow{x_1} 01 \xrightarrow{x_2} 11 \xrightarrow{x_3} 00,$$

в то время как в ГП при этих значениях входных переменных никакие промежуточные значения не формируются. (Аналогичная ситуация имеет место в схемах и носит название «быстрой и медленной тактности» [3]).

Построенный ГП соответствует автомату без выходного преобразователя. Если этому графу сопоставить ГП автомата Мура, дополнительно пронумеровав вершины цифрами от нуля до трех, то, используя конструкцию `switch`, можно получить весьма громоздкую программу, которая, однако, весьма просто читается и понимается ввиду ее полного изоморфизма с ГП и отсутствия умолчаний по всем выходным переменным и переходам:

```
        switch (Y) {
case 0: y1 = 0; y2 = 0;
        if (x̄1 & x̄2 ∨ x3)    {Y = 0; break; }
        if (x1 & x̄2 & x̄3)    {Y = 1; break; }
        if (x1 & x2 & x̄3)    {Y = 2; break; }
        if (x̄1 & x2 & x̄3)    {Y = 3; break; }
case 1: y1 = 0; y2 = 1;
        if (x3)              {Y = 0; break; }
        if (x̄2 & x̄3)         {Y = 1; break; }
        if (x2 & x̄3)         {Y = 2; break; }
case 2: y1 = 1; y2 = 1;
        if (x3)              {Y = 0; break; }
        if (x̄3)              {Y = 2; break; }
case 3: y1 = 1; y2 = 0;
        if (x3)              {Y = 0; break; }
        if (x1 & x̄3)         {Y = 2; break; }
        if (x̄1 & x̄3)         {Y = 3; break; }
                        }.
```

Использование оператора `break` в каждой строке обеспечивает выход из этой программы после выполнения любого из условий, что в общем случае эквивалентно устранению внутренних обратных связей (расцикливание), если они имеются. Это обеспечивает также высокое быстродействие программы.

Текст программы может быть сокращен при сохранении изоморфности с ГП, если перейти от графа переходов (рис. 4.26) к эквивалентному ему графу с прочерками (рис. 6.1).

Однако такой ГП и соответствующая ему программа обладают меньшей понятностью из-за необходимости помнить предыдущие значения выходных переменных.

Из изложенного следует, что при использовании ГП для обеспечения понятного описания требуется построение не любого ГП, а графа, в ко-

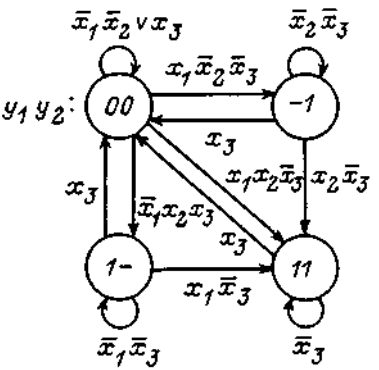

Рис. 6.1

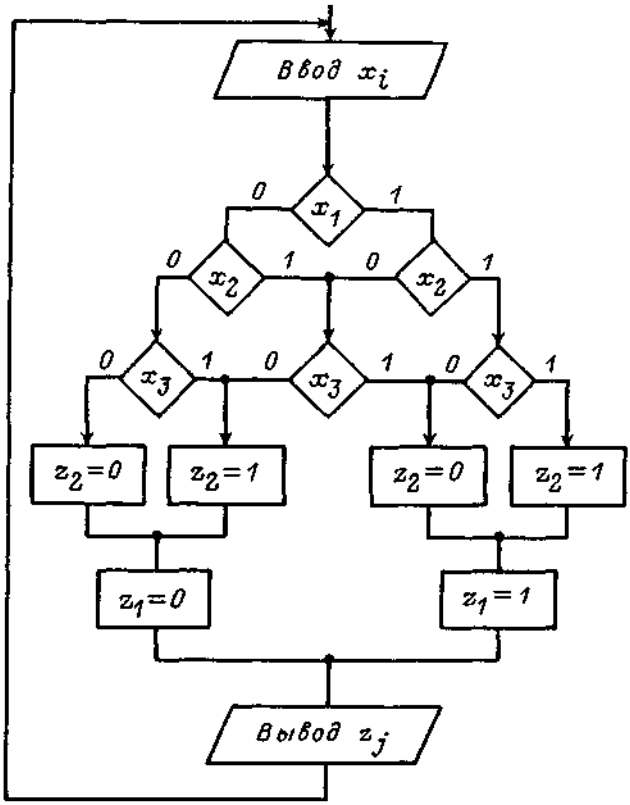

Рис. 6.2

тором зависимость от предыстории отсутствует. Кратко формулируя эту ситуацию, можно утверждать, что «граф графу рознь».

Возвращаясь к сформулированной в начале данной главы задаче, отметим, что изложенный переход от ГСА к читаемой и понятной программе (ГСА—СБФ—ГП без флагов и прочерков — программа без умолчаний) весьма трудоемок, учитывая в особенности тот факт, что для случая m выходных переменных ГП будет содержать 2^m вершин, из которых только некоторые могут являться рабочими.

Изложенное еще раз подтверждает сформулированный в разд. 4.3.3 вывод о том, что применение ГСА, в которых не используется дешифратор состояний, в качестве спецификации для автоматов с памятью нецелесообразно.

На рис. 6.2 в качестве примера приведена ГСА, реализующая одноразрядный сумматор (автомат без памяти), которая построена с помощью модификации канонического метода [144]. Суммарное число условных и операторных вершин в этой ГСА равно 12.

ГСА, построенная для этого автомата с помощью другой модификации канонического метода [84], приведена на рис. 6.3. Вынос вверх двух

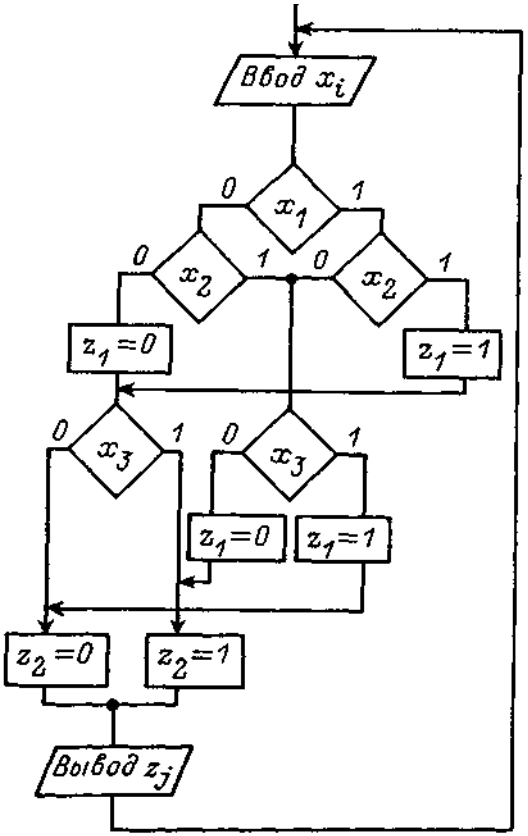

Рис. 6.3

операторных вершин позволил сократить суммарное число условных и операторных вершин в ней до одиннадцати.

На рис. 6.4 и 13.35 приведены ГСА, построенные на базе предыдущих граф-схем, которые реализуют последовательный сумматор (автомат с памятью). Эти ГСА содержат 11 и 10 вершин соответственно.

При этом отметим, что обе эти граф-схемы содержат дешифратор состояний, однако в первой из них он расположен в третьем слое ее тела, а во второй — в первом. Это обеспечивает упрощение и большую «понятность» второй ГСА. Она не полностью изоморфна (формирование нового состояния выполняется раньше, чем формируется значение выходной переменной при переходе в это состояние) графу переходов автомата Мили (рис. 4.113), что позволяет сократить (с 13 до 10) число вершин по сравнению с ГСА, полностью изоморфной этому ГП.

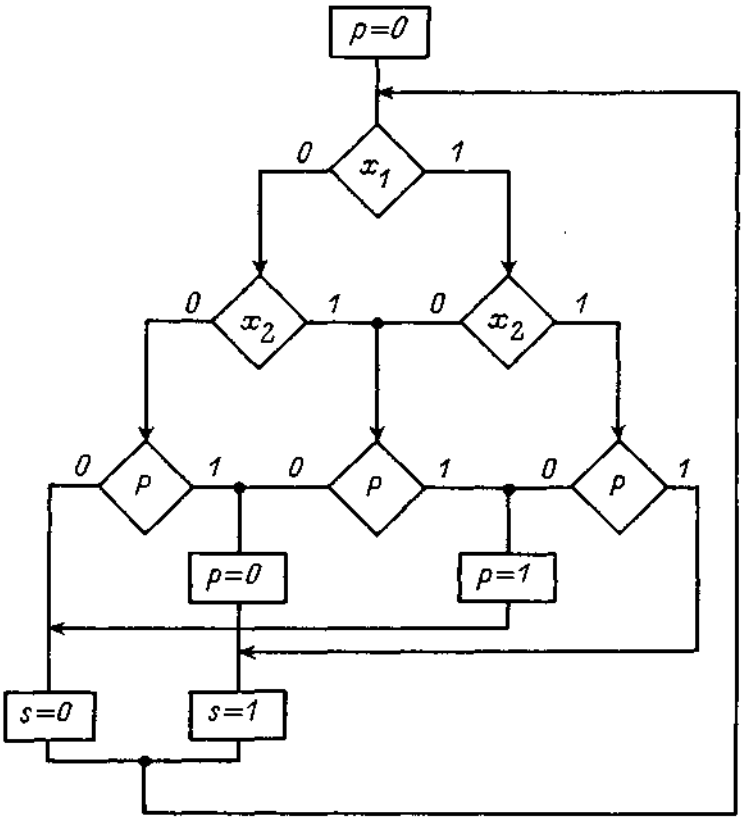

Рис. 6.4

6.2. Реализация автоматных ГСА без внутренних обратных связей при наличии промежуточных переменных

Рассмотрим на примере метод верификации этого подкласса ГСА. Пусть задана ГСА без ВОС с внутренней переменной y_3 и выходными переменными y_1 и y_2 (рис. 6.5). Построим СБФ для этой ГСА, используя подход, изложенный в разд. 4.3.4.

Верификация ГСА осуществляется за семь шагов, указанных римскими цифрами на рис. 6.5:

$$\begin{cases} y_{11} = \bar{x}_4 0 \vee x_4 y_1 = x_4 y_1; \\ y_{12} = \bar{x}_4 y_2 \vee x_4 y_2 = y_2; \\ y_{13} = \bar{x}_4 y_3 \vee x_4 1 = x_4 \vee y_3; \end{cases}$$

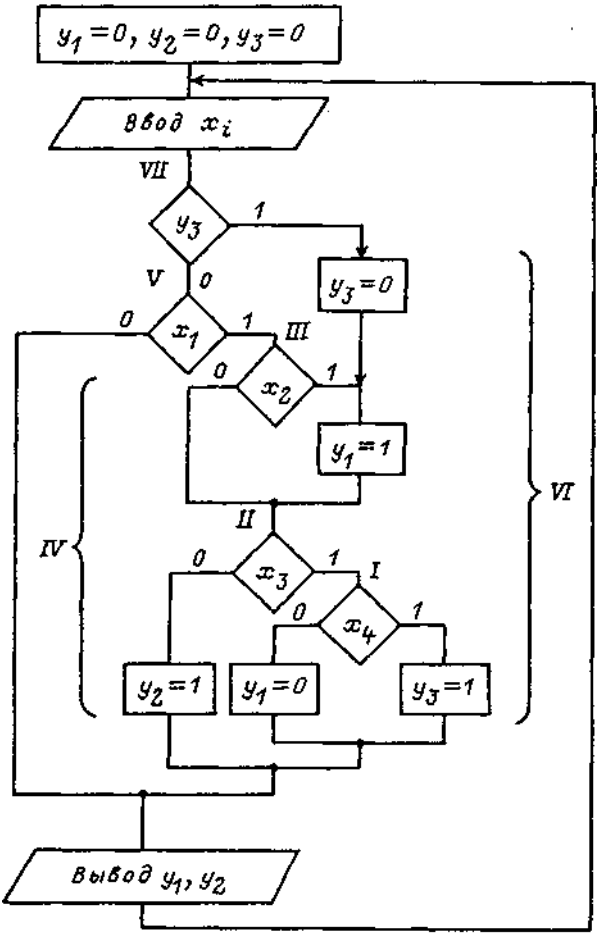

Рис. 6.5

$$\begin{cases} y_{21} = \bar{x}_3 y_1 \vee x_3 y_{11} = (\bar{x}_3 \vee x_4) y_1; \\ y_{22} = \bar{x}_3 1 \vee x_3 y_{12} = \bar{x}_3 \vee y_2; \\ y_{23} = \bar{x}_3 y_3 \vee x_3 y_{13} = x_3 x_4 \vee y_3; \end{cases}$$

$$\begin{cases} y_{31} = \bar{x}_2 y_1 \vee x_2 1 = x_2 \vee y_1; \\ y_{32} = y_2; \\ y_{33} = y_3; \end{cases}$$

$$\begin{cases} y_{41} = y_{21} \leftarrow y_{31} = (\bar{x}_3 \vee x_4)(x_2 \vee y_1); \\ y_{42} = y_{22} \leftarrow y_{32} = \bar{x}_3 \vee y_2; \\ y_{43} = y_{23} \leftarrow y_{33} = x_3x_4 \vee y_3; \end{cases}$$

$$\begin{cases} y_{51} = \bar{x}_1y_1 \vee x_1y_{41} = \bar{x}_1y_1 \vee x_1(x_2 \vee y_1)(\bar{x}_3 \vee x_4); \\ y_{52} = \bar{x}_1y_2 \vee x_1y_{42} = \bar{x}_1y_2 \vee x_1(\bar{x}_3 \vee y_2) = x_1\bar{x}_3 \vee y_2; \\ y_{53} = \bar{x}_1y_3 \vee x_1y_{43} = \bar{x}_1y_3 \vee x_1(x_3x_4 \vee y_3) = x_1x_3x_4 \vee y_3; \end{cases}$$

$$\begin{cases} y_{61} = y_{21} \leftarrow (y_1 = 1) = \bar{x}_3 \vee x_4; \\ y_{62} = y_{22} = \bar{x}_3 \vee y_2; \\ y_{63} = y_{23} \leftarrow (y_3 = 0) = x_3x_4; \end{cases}$$

$$\begin{cases} y_{71} = \bar{y}_3y_{51} \vee y_3y_{61} = \bar{x}_1y_1\bar{y}_3 \vee (x_1(x_2 \vee y_1) \vee y_3)(\bar{x}_3 \vee x_4); \\ y_{72} = \bar{y}_3y_{52} \vee y_3y_{62} = (x_1 \vee y_3)\bar{x}_3 \vee y_2; \\ y_{73} = \bar{y}_3y_{53} \vee y_3y_{63} = (x_1 \vee y_3)x_3x_4. \end{cases}$$

Последней СБФ соответствует либо система из трех взаимосвязанных графов переходов автоматов, каждый из которых имеет два состояния, либо один граф переходов автомата без выходного преобразователя с восемью состояниями, закодированными принудительно-свободно.

В построенном графе переходов переменная y_3 может быть исключена, а каждая вершина отмечена десятичным числом (многозначное кодирование состояний автомата). При этом получается ГП автомата Мура, эквивалентного исходной ГСА.

К рассматриваемому в настоящем разделе классу ГСА относятся также граф-схемы, реализующие последовательный сумматор (рис. 6.4 и 13.35). Граф переходов автомата Мили (рис. 4.113) для этих граф-схем может быть построен либо непосредственно по ГСА, либо по СБФ (разд. 4.4.3).

6.3. Реализация автоматных ГСА с внутренними обратными связями без промежуточных переменных

Пусть задана ГСА с внутренними обратными связями, но без промежуточных переменных (рис. 6.6). Если по этой ГСА должна строиться программа, то она может быть реализована непосредственно. Если же она должна соответствовать процедуре, то для программирования с использованием конструкций `if`, `then`, `else` в ней предварительно должна быть устранена внутренняя обратная связь.

Введем для этого в рассматриваемую ГСА дополнительную (внутреннюю) переменную y_3, значения которой проверяются и устанавливаются в определенных точках преобразованной ГСА (рис. 6.5).

В предыдущем разделе показано, как эта ГСА может быть верифицирована с помощью СБФ, по которой может быть построен ГП автомата Мура.

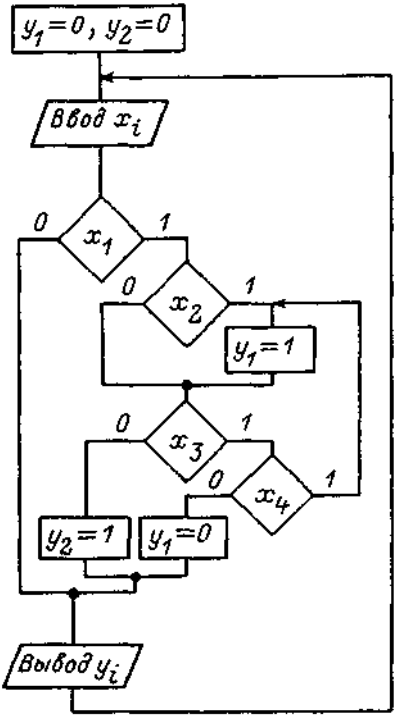

Рис. 6.6

6.4. Реализация автоматных ГСА с внутренними обратными связями и промежуточными переменными

В этом случае в заданной ГСА сначала устраняются внутренние обратные связи за счет введения дополнительных промежуточных переменных (разд. 6.3), а затем полученная ГСА без внутренних обратных связей верифицируется построением СБФ (разд. 6.2), по которой строится ГП автомата Мура.

6.5. Метод структурирования автоматных ГСА

Полученные по ГСА с помощью подходов, изложенных в предыдущих разделах, графы переходов автоматов Мура и Мили могут быть реализованы на основе конструкции `switch`, что обеспечивает построение структурированных программ по неструктурированным граф-схемам.

Изложенный подход может рассматриваться в качестве нового метода структурирования граф-схем алгоритмов, отличного от метода Ашкрофта и Манны [104], так как по каждому графу переходов может быть построена изоморфная ему структурированная ГСА с дешифратором состояний (разд. 13.4).

6.6. Реализация логико-вычислительных ГСА

В общем случае в операторных вершинах граф-схем кроме рассмотренных в разд. 6.1 могут использоваться и любые другие операторы. Для реализации этого класса ГСА предлагается первоначально по заданной граф-схеме построить автоматную ГСА путем замены операторов $z_i = f(X)$ на операторы $z_i = 1$. После этого для автоматной граф-схемы с помощью предлагаемого подхода строится программа на языке СИ, использующая конструкцию `switch`. Осуществляя в этой программе обратную замену введенных операторов: $z_i = 1$ на операторы $z_i = f(X)$, получим хорошо читаемую программу, реализующую исходную ГСА с вычислениями.