Глава 8

Организация взаимодействия в системе графов переходов

Для решения вопроса об организации взаимодействия ГП в системе графов переходов (СГП) рассмотрим задачу о замене исходного ГП без кратных дуг, все вершины которого устойчивы, эквивалентной системой графов переходов, что соответствует задаче декомпозиции исходного ГП на составные части.

При этом рассматриваются три вида декомпозиции: параллельная, последовательная, формульная.

Первые две разновидности декомпозиции являются топологическими, а третья — аналитической. Указанные разновидности целесообразно применять в следующем порядке: сначала параллельную, потом последовательную (если она позволяет увеличить число выделяемых компонент) или формульную, если параллельную декомпозицию провести не удается. Еще одна разновидность декомпозиции рассмотрена в разд. 19.6.

8.1. Параллельная декомпозиция. Головной и вызываемые графы

Пусть исходный ГП декомпозируется параллельно. Будем называть ГП системы, содержащий начальную вершину исходного графа, головным графом переходов, а все остальные графы — вызываемыми графами переходов. При этом вызываемые графы между собой непосредственно не взаимодействуют, а их взаимосвязь осуществляется через головной граф. При этом реализуем следующий принцип взаимодействия головного и вызываемых графов: «запрос—ответ—сброс».

Для обеспечения такого взаимодействия головной граф должен содержать два типа вершин — функциональные и вызывающие, а каждый вызываемый граф — начальную, функциональные и конечные вершины.

Функционирование системы ГП должно происходить следующим образом:

— реализуются переходы между функциональными вершинами головного графа;

— при переходе в вызывающую вершину головной граф передает управление (осуществляет запрос) вызываемому графу. Управление пере-

дается только одному из вызываемых графов, так как декомпозируемый ГП является последовательным по переходам;

— при выполнении условия, соответствующего пометке «запрос», в вызываемом графе осуществляется переход из начальной вершины в одну из функциональных вершин;

— при переходе в конечную вершину вызываемый граф возвращает управление (осуществляет ответ) головному графу;

— при выполнении условия, соответствующего пометке «ответ», головной граф переходит в одну из функциональных или вызывающих вершин, формируя также условия возврата (сброса) вызываемого графа в начальную вершину;

— при выполнении условия, соответствующего пометке «сброс», вызываемый граф переходит в начальную вершину, завершая взаимодействие головного графа с вызываемым.

Изложенный принцип функционирования системы ГП предполагает, что при ее реализации программы, соответствующие ГП, расположены последовательно и в каждом цикле для каждого ГП выполняется не более одного перехода: либо автомат, соответствующий ГП, сохраняет в этом цикле свое состояние, либо переходит в смежное состояние.

Для обеспечения указанной процедуры функционирования декомпозиция исходного ГП должна выполняться следующим образом:

1) в нем выделяется максимально возможное число фрагментов так, чтобы каждый из них (за исключением головного) не содержал более одной вершины, связанной с другими фрагментами. Если такое выделение невозможно, то следует перейти к формульной декомпозиции;

2) каждый выделенный фрагмент изображается отдельно, сохраняя пометку внутренних и исходящих из него дуг;

3) в каждый фрагмент вводятся дополнительные вершины, замещающие выделенные фрагменты, связывая входящие и исходящие дуги таким образом, чтобы каждый из полученных графов отражал структуру исходного ГП;

4) если хотя бы в одном из полученных ГП между двумя вершинами имеются кратные дуги без пометок, то они объединяются в одну;

5) если хотя бы в одном из полученных ГП между двумя вершинами имеется d кратных дуг с пометками, то в каждую из них (за исключением одной) вводится дополнительная вершина;

6) пометим в каждом вызываемом графе дугу, исходящую из начальной вершины (замещающей головной граф), «запросом» — выражением вида $Y_i == j$, где j — номер вершины в головном графе Y_1, из которой вызывается рассматриваемый граф;

7) пометим дугу, исходящую из вызывающей вершины головного графа, «ответом» κ-го вызываемого графа — выражением вида $Y_k == l$, где l — номер конечной вершины κ-го вызываемого графа;

8) пометим дугу между конечной и начальной вершинами κ-го вызываемого графа условием «сброса» — выражением вида $Y_1 == d$, где d — номер вершины, в которую переходит головной граф из вершины, вызывающей κ-й граф.

На рис. 8.1 в качестве примера приведен декомпозируемый ГП, а на рис. 8.2 — система ГП, получаемая в результате его параллельной декомпозиции на две составляющие — головной и вызываемый графы.

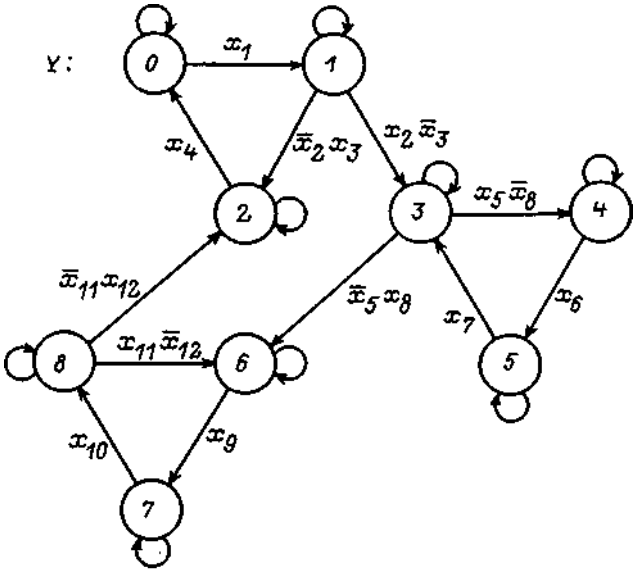

Рис. 8.1

Для доказательства эквивалентности системы ГП и исходного графа построим по этой системе граф достижимых маркировок, в каждой вершине которого записывается состояние каждого ГП, в которое он переходит при определенных входных воздействиях. Переходы между вершинами ГДМ осуществляются под воздействием пометок, указанных на дугах графов переходов системы.

ГДМ описывает поведение системы ГП в целом, отражая все функциональные возможности системы автоматов, так же как один ГП без флагов и умолчаний описывает все функциональные возможности одного автомата.

Построенная указанным способом система ГП обладает следующим свойством: суммарное число вершин (дуг) в графах переходов системы равно числу вершин (дуг) в ГДМ, т. е. суммарная сложность описания системы и сложность ее поведения одинаковы.

Отмеченное свойство является чрезвычайно важным для проверки и тестирования системы графов переходов, так как в общем случае ГДМ может быть существенно сложнее этой системы.

Другими свойствами ГДМ для рассматриваемого случая являются следующие: пометки всех вершин различны; пометки двух соседних вершин отличаются только в одной компоненте; в каждой вершине не более одной компоненты соответствует вершине исходного графа, а остальные компоненты — дополнительным вершинам, введенным в систему в ходе декомпозиции.

Отметим также, что если исходный ГП допускает построение различных систем с одинаковым числом графов, то сложность всех этих систем одинакова.

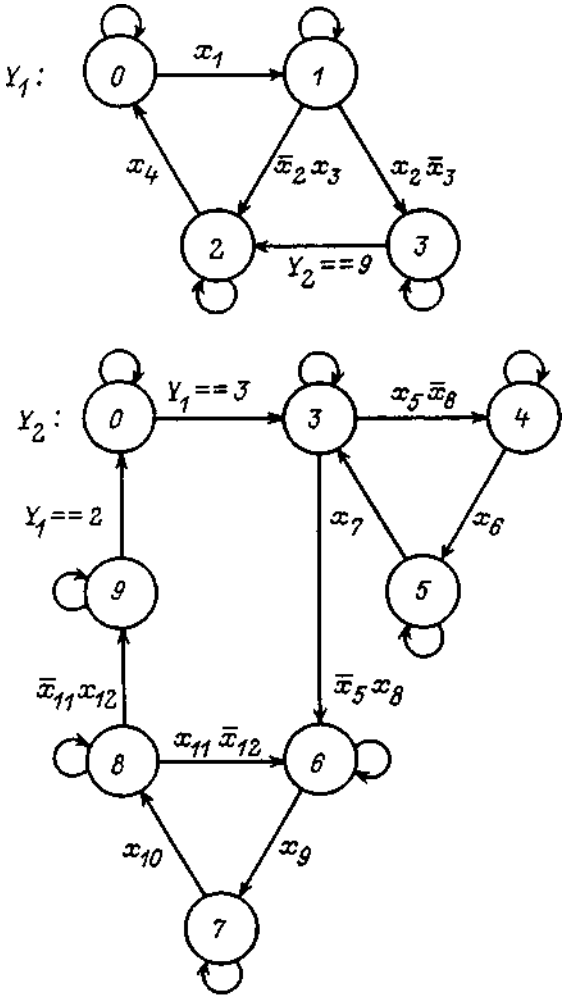

Рис. 8.2

Рис. 8.2

На рис. 8.3 приведен ГДМ для рассматриваемого примера. При этом каждой дуге в этом графе соответствует определенная дуга в одном из ГП системы, что свидетельствует о том, что сложность ГДМ, определяемая числом дуг, равна суммарной сложности графов переходов системы.

Если в ГДМ исключить дуги с пометками $Y_a == b$ и вершины, из которых эти дуги исходят, а также изменить пометку оставшихся вершин, сохранив в каждой из них только «рабочую» компоненту, то в случае, если декомпозиция выполнена корректно, должен получиться исходный ГП.

Рассмотренный метод декомпозиции определяет такую организацию системы ГП, в которой имеется головной и вызываемые графы, сложность

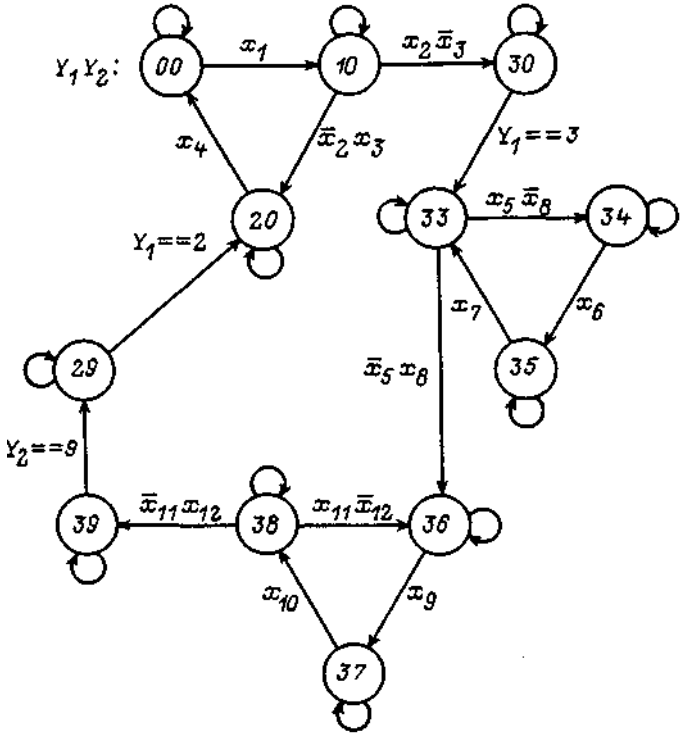

Рис. 8.3

поведения которых совпадает со сложностью их описания. Кроме того, предлагаемый подход позволяет для оформления подпрограмм, их вызова и возврата в головную программу не использовать соответствующие конструкции языка, а применять последовательно расположенные операторы switch, что резко упрощает программирование.

Для иллюстрации изложенного приведем в качестве примера программу, реализующую систему ГП (рис. 8.2):

```
          switch (Y1) {
case 0: if (x1)         Y1 = 1;
        break;
case 1: if (x̄2 & x3)    Y1 = 2;
        if (x2 & x̄3)    Y1 = 3;
        break;
case 2: if (x4)         Y1 = 0;
        break;
case 3: if (Y2 == 9)    Y1 = 2;
        break;
                        }.
```

```
            switch (Y2) {
case 0:  if (Y1 == 3)  Y2 = 3;
         break;
         . . .
case 9:  if (Y1 == 2)  Y2 = 0;
         break;
                       }.
```

Для рассмотренной системы ГП можно отказаться от взаимодействия графов по принципу «запрос—ответ—сброс» и перейти к взаимодействию по принципу «запрос—ответ», исключив конечную вершину вызываемого графа и изменив пометку «ответ» в головном графе (рис. 8.4). При этом

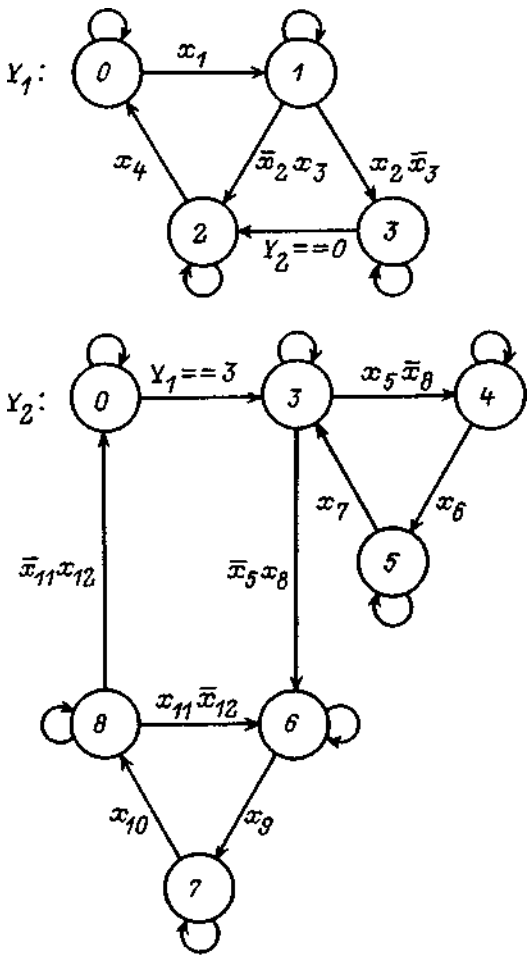

Рис. 8.4

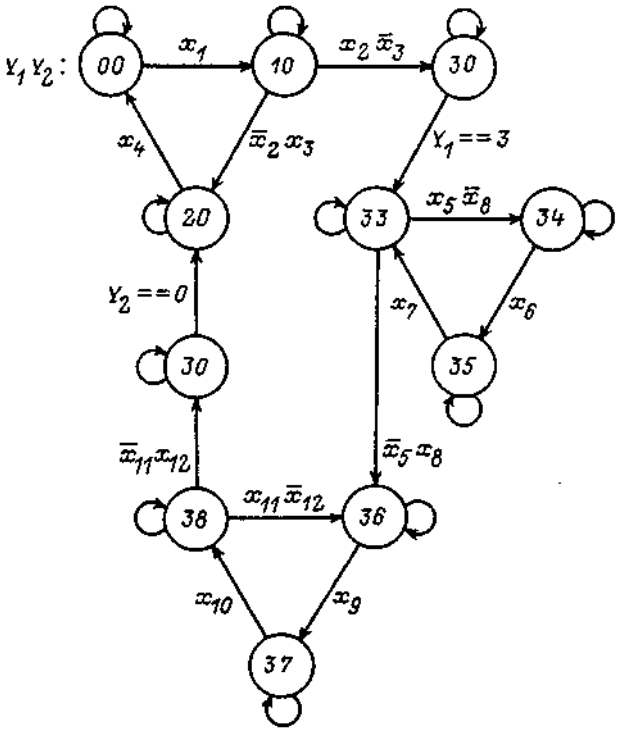

Рис. 8.5

упрощается также ГДМ, в котором, однако, появляются вершины с одинаковыми номерами (рис. 8.5).

Рассмотренный ГП (рис. 8.1) с помощью предложенного подхода может быть параллельно декомпозирован и на три составляющие (рис. 8.6). В ГДМ в этом случае все вершины имеют различную пометку.

Если вызываемые ГП имеют однотипную структуру (рис. 8.6), то они могут быть объединены в настраиваемый вызываемый ГП (рис. 8.7).

Настройка осуществляется указанием номера вершины, в которой находится головной ГП. В этом случае при одной настройке вызываемый граф работает с одним набором входных переменных, а при второй настройке — с другим. Таким образом обеспечивается возможность работы вызываемого графа с различными наборами фактических параметров.

Из изложенного следует, что рассмотренный метод параллельной декомпозиции может быть дополнен этапом минимизации числа служебных вершин в вызываемых графах за счет перехода к организации взаимодействия графов по принципу «запрос—ответ». Так как при параллельной декомпозиции вызываемые графы между собой непосредственно не взаимодействуют, то в каждом из них начальная и конечная вершины

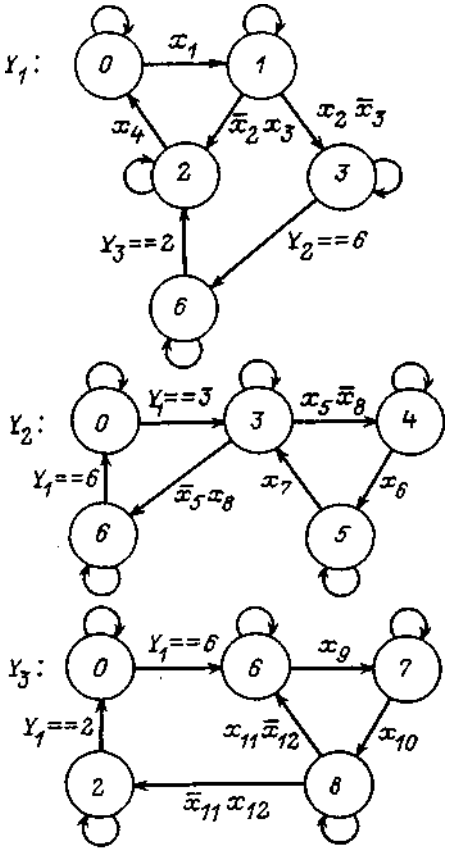

Рис. 8.6

могут быть совмещены, а пометки «ответ» в головном графе — изменены (рис. 8.8).

Объединяя вызываемые графы (рис. 8.8) в настраиваемый граф, можно получить более эффективную структуру по сравнению с представленной на рис. 8.7.

В примере (рис. 8.1) каждый фрагмент, являющийся телом вызываемого графа, имеет по одному входу из головного графа. Если фрагмент имеет несколько входов, соединенных только с одной его вершиной, то декомпозиция выполняется так же, как в рассмотренном примере. Наличие в исходном ГП хотя бы одного фрагмента с входами, подсоединенными к различным его вершинам, при параллельной декомпозиции недопустимо.

Таким образом, наиболее просто параллельно декомпозируется структурированный ГП, содержащий фрагменты, имеющие один вход и один выход, аналогично, как это имеет место в структурном программировании [103, 104].

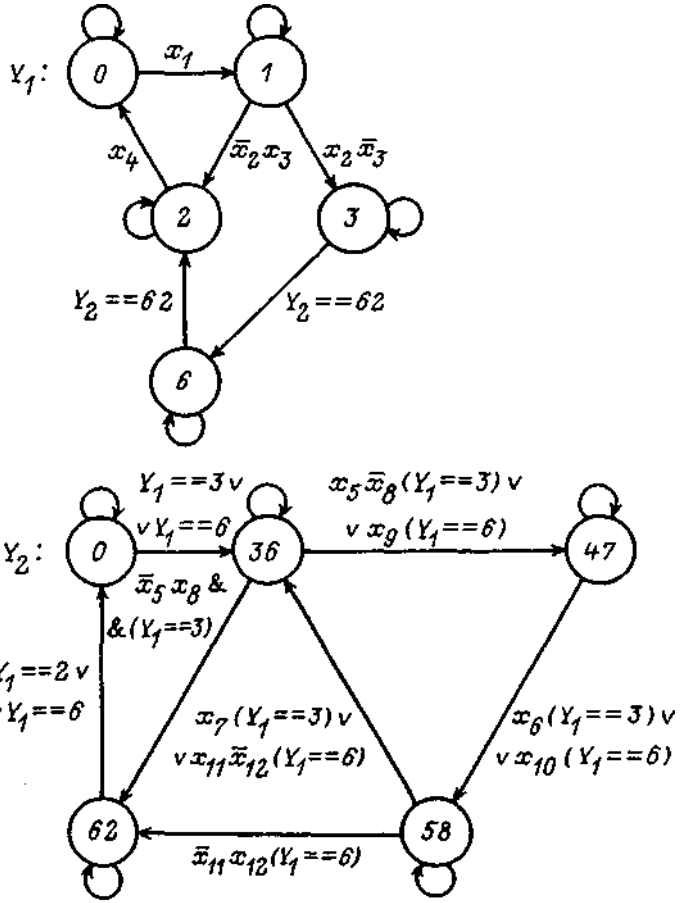

Рис. 8.7

8.2. Последовательная декомпозиция

Изложим, как выполняется последовательная декомпозиция, которая позволяет строить многоуровневые вызываемые структуры, и как обеспечить ее реализацию при наличии во фрагментах нескольких «входных» вершин.

Последовательная декомпозиция сводится к параллельной, выполняемой многократно. При этом на первом шаге исходный ГП декомпозируется на головной и вызываемый графы. На следующем шаге вызываемый граф, если это возможно, параллельно декомпозируется на головной и вызываемый графы второго уровня. После этого делается попытка декомпозировать новый вызываемый граф и т. д.

Графы переходов в построенной системе, как и при параллельной декомпозиции, взаимодействуют друг с другом по принципу «запрос—

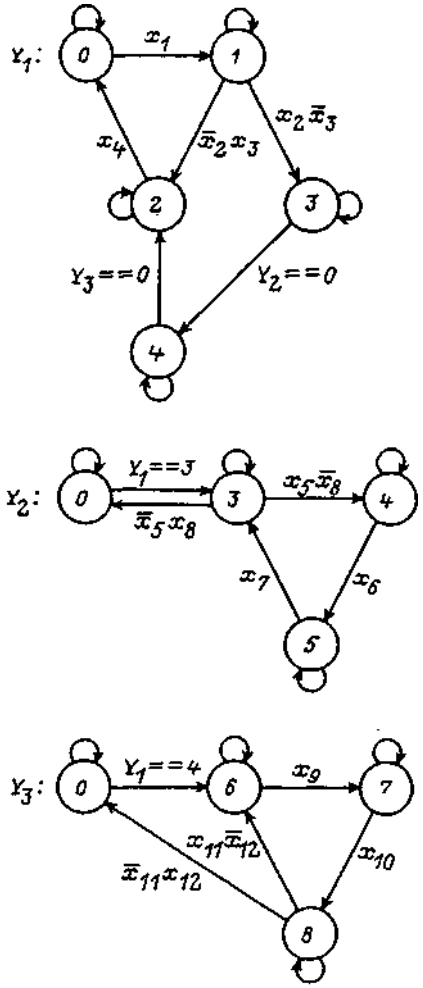

Рис. 8.8

Рис. 8.8

ответ—сброс». При этом допускается общий сброс всех вызываемых графов системы.

Выполним в качестве примера последовательную декомпозицию ГП (рис. 8.1). Пусть на первой стадии граф декомпозируется на головной и вызываемый графы, как показано на рис. 8.4. Выполняя декомпозицию второго графа, получим новую систему ГП, состоящую из трех компонент (рис. 8.9).

Если сравнить системы, состоящие из трех ГП, построенные с помощью параллельной (рис. 8.8) и последовательной (рис. 8.9) декомпозиций, можно утверждать, что их сложность, как по числу вершин, так и по числу дуг, одинакова, однако вызываемые графы, полученные путем

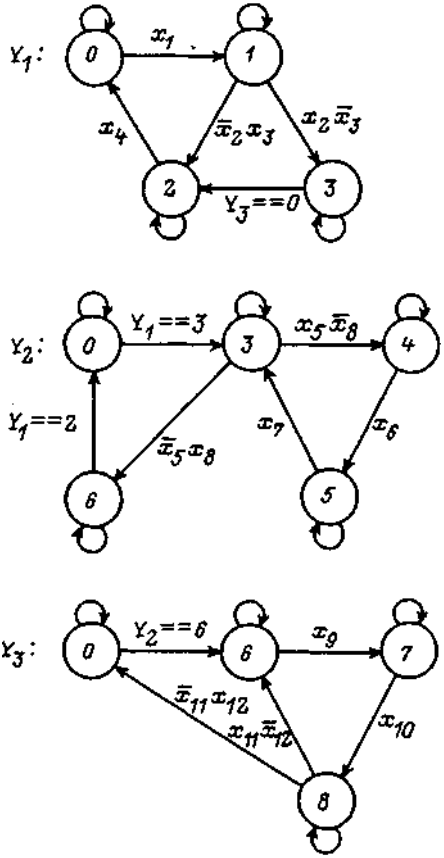

Рис. 8.9

Рис. 8.9

последовательной декомпозиции, не могут быть объединены в настраиваемый граф.

ГДМ, получаемый для системы ГП, построенной методом последовательной декомпозиции (так же как и при параллельной), имеет число вершин (дуг), равное суммарному числу вершин (дуг) в графах переходов системы.

В рассмотренном примере разделить исходный граф на три составляющие удается как с помощью параллельной, так и последовательной декомпозиции.

На рис. 8.10 приведен ГП, который с помощью последовательной декомпозиции удается разделить на три составляющие (рис. 8.11), в то время как при параллельной декомпозиции это невозможно, так как второй фрагмент, образованный вершинами 3, 4, 5, имеет две «входные» вершины с номерами 3 и 4.

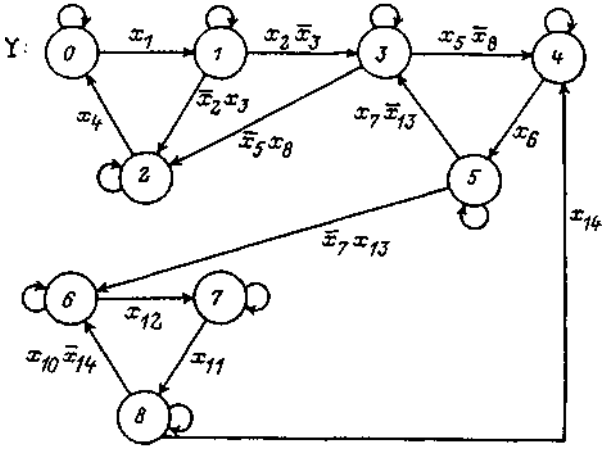

Рис. 8.10

Из изложенного следует, что рассмотренные варианты взаимодействия графов переходов в системе обеспечивают весьма простое, обозримое и предсказуемое поведение системы в целом.

Такое поведение сохраняется и для случая параллельно-последовательной декомпозиции, при которой строятся головной и несколько вызываемых графов, а в дальнейшем один или несколько вызываемых графов декомпозируются дальше.

Число служебных вершин в системе ГП с минимальным числом вершин (конечные вершины в вызываемых графах отсутствуют) определяется соотношением

$$N_д = 2(M - 1),$$

где M — число ГП в системе; $(M - 1)$ — суммарное число дополнительных начальных вершин в графах переходов системы и $(M - 1)$ — суммарное число дополнительных конечных вершин.

При этом общее число вершин в системе

$$N = N_и + N_д = N_к + 2(M-1),$$

где $N_и$ — число вершин в исходном графе.

Число дополнительных дуг в системе

$$D_д \leqslant 2(M-1),$$

а общее число дуг (без петель)

$$D = D_и + D_д \leqslant D_и + 2(M-1),$$

где $D_и$ — число дуг в исходном ГП.

В ГДМ, построенном по «минимальной» системе ГП, $N_{гдм} = N$; $D_{гдм} = D$.

Так как для ГП (рис. 8.1) $N_и = 9$, $D_и = 12$, а число ГП в системе $M - 2$, то для системы ГП (рис. 8.4) и ГДМ (рис. 8.5) $N = N_{гдм} = 11$, $D = D_{гдм} = 14$.

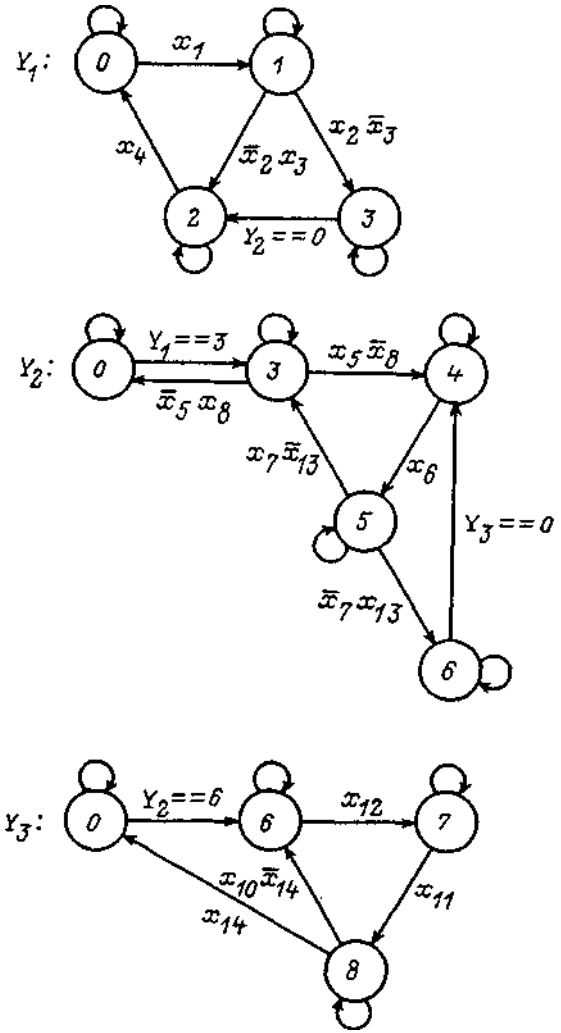

Рис. 8.11

Для систем ГП (рис. 8.8, 8.9) $N = 13$, $D = 16$.

Для ГП (рис. 8.10) $N_и$ - 9, $D_и = 13$, в то время как для системы ГП с $M = 3$ (рис. 8.11) $N = 13$, $D = 17$.

8.3. Организация циклических структур

Изложенный принцип организации взаимодействия графов переходов позволяет реализовать циклические структуры. Рассмотрим два типа циклических конструкций: конструкцию «`do while`» (например, нали-

вать воду из ведер в бочку до тех пор, пока последняя не заполнится) и конструкцию «for» (например, налить в бочку заданное число ведер). Отметим, что полное название первой из этих конструкций «do statement while not finished» — «выполнять оператор до тех пор, пока не возникнет условие завершения».

На рис. 8.12 приведен пример циклического выполнения фрагмента, состоящего из вершин 3, 4, 5, до тех пор пока выполняется условие x_2. На рис. 8.13 приведен пример трехкратного выполнения этого фрагмента. Во втором случае имеется возможность в каждом цикле проводить вычисления не только над новыми значениями одних и тех же переменных, но и над значениями новых переменных, так же как и для настраиваемых вызываемых графов переходов.

Таким образом, предложенный подход позволяет для программирования циклических структур, так же как и подпрограмм, не использовать соответствующие конструкции языка, а применять только записанные последовательно операторы switch.

Для первого класса циклических структур число вершин в ГДМ и суммарное число вершин в головном и вызываемом графах совпадают, а для второго класса — число вершин в ГДМ превышает суммарное число вершин в графах системы.

Рассмотрим счетчиковые структуры, при декомпозиции которых суммарное число вершин в графах системы может быть меньше (и существенно), чем в исходном графе (рис. 8.14), содержащем $W = n + 1$ вершин.

Декомпозируем этот граф на N компонент так, что каждая из них содержит начальную вершину, а для остальных вершин справедливо соотношение $n = \prod_{i=1}^{N} m_i$, где m_i,- — число вершин (без начальной) в i-м графе системы. При этом суммарное число вершин в системе $B = N + \sum_{i=1}^{N} m_i$. Приведем минимальные значения B для различных n:

$n = 1 \div 7$	$B = n + 1$;	$n = 8 = 4 \times 2$	$B = 8$;
$n = 9 = 3 \times 3$	$B = 8$;	$n = 10 = 5 \times 2$	$B = 9$;
$n = 12 = 4 \times 3$	$B = 9$;	$n = 15 = 5 \times 3$	$B = 10$;
$n = 16 = 4 \times 4$	$B = 10$;	$n = 18 = 6 \times 3 = 3 \times 3 \times 2$	$B = 11$;
$n = 20 = 5 \times 4$	$B = 11$;	$n = 21 = 7 \times 3$	$B = 12$;
$n = 24 = 6 \times 4 = 4 \times 3 \times 2$	$B = 12$;	$n = 25 = 5 \times 5$	$B = 12$;
$n = 27 = 3 \times 3 \times 3$	$B = 12$;	$n = 28 = 7 \times 4$	$B = 13$;
$n = 30 = 5 \times 3 \times 2$	$B = 13$;	$n = 32 = 4 \times 4 \times 2$	$B = 13$;
$n = 36 = 4 \times 3 \times 3$	$B = 13$;	$n = 48 = 4 \times 4 \times 3$	$B = 14$;
$n = 64 = 4 \times 4 \times 4$	$B = 15$;	$n = 10^6$	$B = 66$.

На рис. 8.15 приведена система ГП для случая $n = 8$, из анализа которой следует, что сокращение числа вершин в ней по сравнению с единым ГП (рис. 8.14) достигается за счет усложнения пометок на дугах.

Из изложенного можно заключить также, что для счетчиковых структур декомпозиция проводилась не на топологической основе, а с помощью некоторого стандартного приема.

В заключение раздела отметим, что, например, переменная в цикле `for` может рассматриваться в качестве многозначной переменной, кодирующей состояния соответствующего автомата.

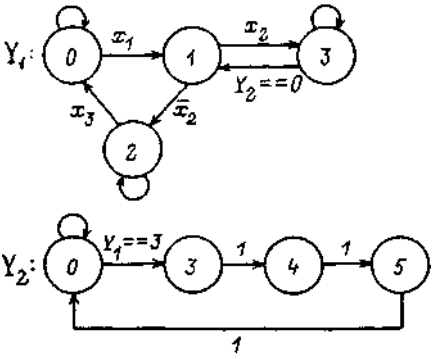

Рис. 8.12

При этом если максимальное значение этой переменной сравнительно невелико (например, не превышает десяти), то как управляющая (ациклическая, нерегулярная), так и операционная (циклическая, регулярная) части реализуемого алгоритма могут быть описаны одинаково — графами переходов соответствующих автоматов с явным заданием их состояний в виде вершин графов, каждый из которых может быть реализован конструкцией `switch`.

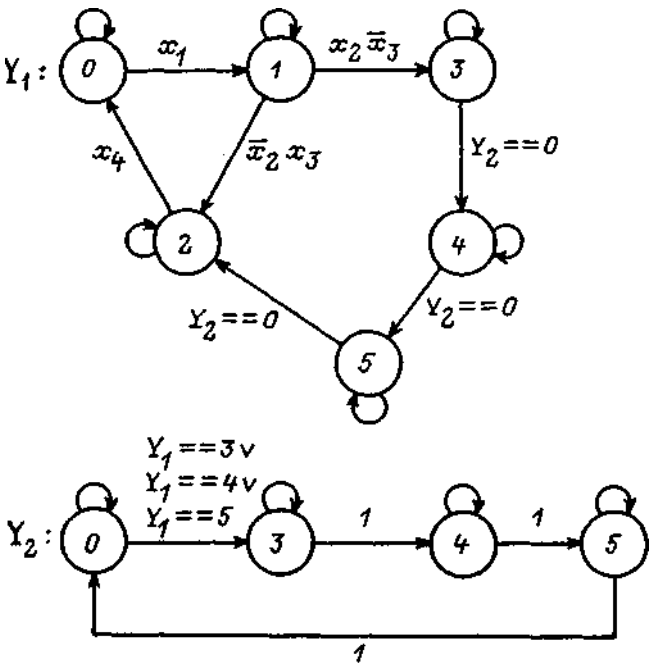

Рис. 8.13

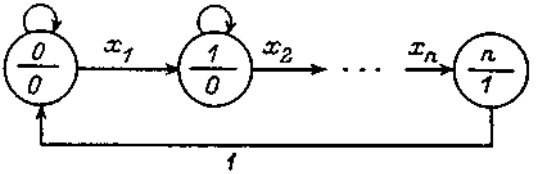

Рис. 8.14

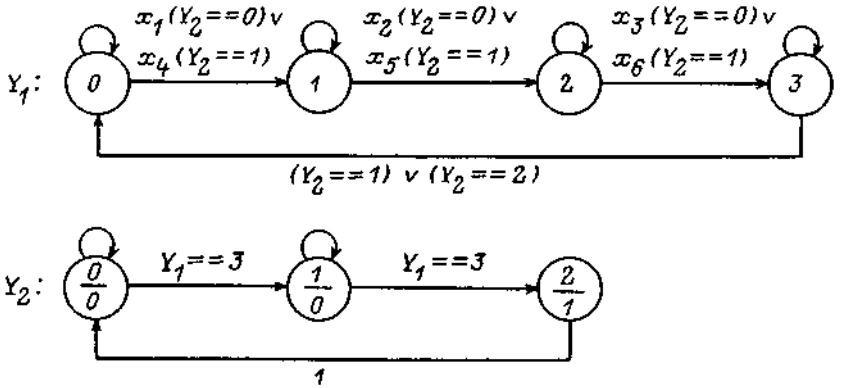

Рис. 8.15

При сравнительно большом значении числа состояний в описании циклических фрагментов алгоритмов применение граф-схем (за счет неявного использования понятия «состояние», что достигается расширением базиса операций, применяемых в операторных вершинах, например за счет введения арифметической операции «сложение») по сравнению с применением графов переходов позволяет обеспечить компактное описание реализуемого фрагмента алгоритма.

Так, например, если одновходовой счетчик (по модулю *n)* числа появлений потенциальной переменной *x* описывается (без использования декомпозиции) графом переходов смешанного автомата «автомат Мили с флагом — автомат без выхода» с *n* вершинами, то при применении ГСА этот счетчик реализуется внутри программного цикла следующей компактной структурированной программой:

```
if   (x)
     {x = 0;
       if (Y == m)     Y = 0;
       else            Y = Y + 1;- }.
```

8.4. Формульная декомпозиция

Как отмечалось в начале данной главы, в случае если топологическая декомпозиция невыполнима, то может быть проведена формульная декомпозиция, суть которой состоит в построении по исходному графу системы булевых формул, по которой в свою очередь строится система ГП (разд. 4.4.7).

При этом необходимо отметить, что обычно если исходный ГП обладает большой связностью вершин, то графы переходов, получающиеся в результате декомпозиции, либо не взаимодействуют между собой (независимы, параллельны), либо их взаимодействие отличается от принципа «запрос—ответ—сброс».

Табл. 4.19 задает сильносвязанный ГП, который формульным методом может быть декомпозирован на два независимых (параллельных) графа (рис. 4.147).

ГП на рис. 4.145 декомпозируется на два параллельных ГП, связанных между собой лишь по входным переменным (рис. 4.146).

Более интересным является ГП на рис. 4.142. Его формульная декомпозиция приводит к ГП, которые связаны между собой не только по входным переменным, но и по переменной состояния (рис. 4.143). Однако в этом случае переход во втором графе выполняется не по вычисленному (новому) значению переменной состояния первого графа, а по предыдущему ее значению, что делает анализ системы ГП весьма трудным. Ситуация резко упрощается, если полученную СБФ удается преобразовать так, что вместо зависимости $y_2' = f_1(y_1)$ имеет место зависимость $y_2' = f_2(y_1')$. Построенная по этой СБФ система ГП анализируется весьма просто, так как становится ясно, что переход во втором графе осуществляется после завершения перехода в вершину «1» в первом графе (рис. 4.144). В этом случае можно утверждать, что имеет место последовательная работа ГП при включении и возможен параллелизм в их работе при отключении.

Простота анализа системы (рис. 4.144) объясняется тем, что для нее весьма легко строится ГДМ, в котором вместо фрагмента «00, x_1, 11», как это имеет место в графе переходов на рис. 4.142, может быть выделен фрагмент «00, x_1, 10, y_1', 11». Таким образом, можно утверждать, что и при формульной декомпозиции полученная система ГП эквивалентна исходному графу лишь в «большом» и отличается от последнего в «малом» ввиду наличия в ГДМ внутренних переменных.

8.5. Объединение графов переходов в систему

Рассмотренный вопрос об организации взаимодействия графов в системе позволяет не только проводить декомпозицию имеющегося ГП, но осуществлять композицию графов переходов, что чрезвычайно важно для практики проектирования.

Более того, из изложенного следует, что при использовании принципов «запрос—ответ—сброс» и «запрос—ответ» возможно иерархическое проектирование «сверху вниз», при котором на первом этапе детально

проектируется лишь головной граф, а каждый из вызываемых содержит начальную и конечную(ые) вершины, а вместо всех его функциональных вершин применяется одна вершина — заглушка. Уже по этой системе ГП может быть написана программа-прототип для проверки правильности взаимодействия графов в системе, а после обеспечения нормального функционирования прототипа возможен переход к детальному проектированию отдельных вызываемых графов, в том числе и параллельно разными разработчиками.

Отдельные ГП, являющиеся стандартными модулями, путем добавления в каждый из них начальной, а возможно, и конечной(ых) вершин всегда могут быть преобразованы в вызываемые ГП для их применения в составе конкретной системы.

Система ГП в отличие от ГСА, использующей подпрограммы, является не «картинкой», для превращения которой в программу требуются дополнительные интеллектуальные усилия, а рабочим проектом, по которому изготовление программы может выполняться формально. Изоморфизм между текстом программы (использующей только конструкции `switch`) и системой ГП позволяет получать программу по этой системе, так же как осуществляется сборка контактной схемы по соответствующему чертежу.

Если проект программы детально разработан и промоделирован, то даже при применении для программирования языков низкого уровня, например ассемблера, особых проблем обычно не возникает, так как в этом случае приходится не проектировать программу в базисе этого языка, а осуществлять с помощью формализованных методов стандартную реализацию проекта в указанном базисе.

При использовании ГП возможен и другой подход к проектированию «сверху вниз». При этом сначала строится укрупненный ГП, каждая вершина в котором соответствует, например, некоторому режиму управления. По этому ГП может быть написана программа-прототип для моделирования на ПЭВМ.

После этого каждая вершина ГП может быть детализирована до получения вершин, соответствующих состояниям автомата Мура. Недостаток этого подхода состоит в том, что фрагменты ГП сами по себе не являются графами переходов (ГП замкнуты и не содержат входящих и исходящих дуг), что не позволяет для каждого из них написать программу и отладить ее автономно. При этом отметим, что, например, для функциональных схем такой подход возможен, так как фрагменты в них также являются схемами, содержащими входы и выходы.

Этого недостатка лишены вложенные графы переходов, рассмотренные в разд. 12.4.

8.6. Содержательная декомпозиция

Изложенные выше вопросы относятся к формальной организации ГП в системе. Полученные результаты можно использовать и при содержательной декомпозиции алгоритма управления, выполняемой на ранних этапах проектирования, называемых архитектурным проектированием.

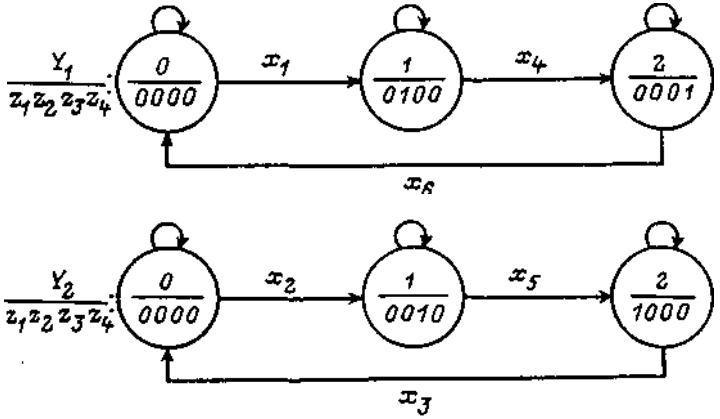

Рис. 8.16

Рассмотрим основные разновидности структурной организации автоматов — централизованную, децентрализованную и иерархическую структуры.

При централизованном управлении (рис. 5.41) проектируется единый ГП (рис. 5.43) для объекта управления в целом.

Для обеспечения децентрализованного управления по режимам декомпозируем ГП (рис. 5.43) на две части: (0, 1, 2) и (3, 4, 5). Соединяя дугой начальную и конечную вершины в каждом графе и проводя новую нумерацию вершин во втором графе, получим желаемую систему ГП, декомпозированную по режимам (рис. 8.16). При этом первый граф обеспечивает режим открытия клапанов, а второй — режим их закрытия. Однако в этой системе ввиду независимости графов имеется возможность подачи противоречивых входных воздействий (сигналов x_1 и x_2 одновременно), что может быть устранено взаимными блокировками.

Промежуточным вариантом между децентрализованным и иерархическим управлением является декомпозиция алгоритма одновременно по режимам и объектам без применения координирующего автомата. При этом первый граф должен обеспечить оба режима («открытие» и «закрытие») для первого клапана, а второй — оба режима для второго клапана. Если функционально оба графа являются равноценными, что предполагает децентрализованное управление, то при программной реализации из-за порядка расположения один из них является как бы головным, а второй — вызываемым, что характерно для иерархического управления. Кроме декомпозиции по объектам, как будет показано ниже, в данном случае имеет место декомпозиция по общим режимам объектов, так как сигнал на открытие клапанов поступает в первый граф, а сигнал на их закрытие — во второй. Взаимодействие графов осуществляется за счет обмена данными, соответствующими номерам определенных состояний.

При декомпозиции исходного графа используем результаты, приведенные в предыдущем разделе. При этом разобьем ГП (рис. 5.43) на два фрагмента (0, 1, 5) и (2, 3, 4). Добавляя к первому фрагменту вершину-

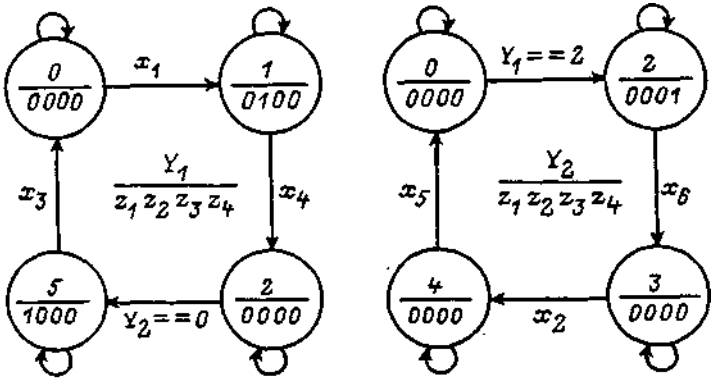

Рис. 8.17

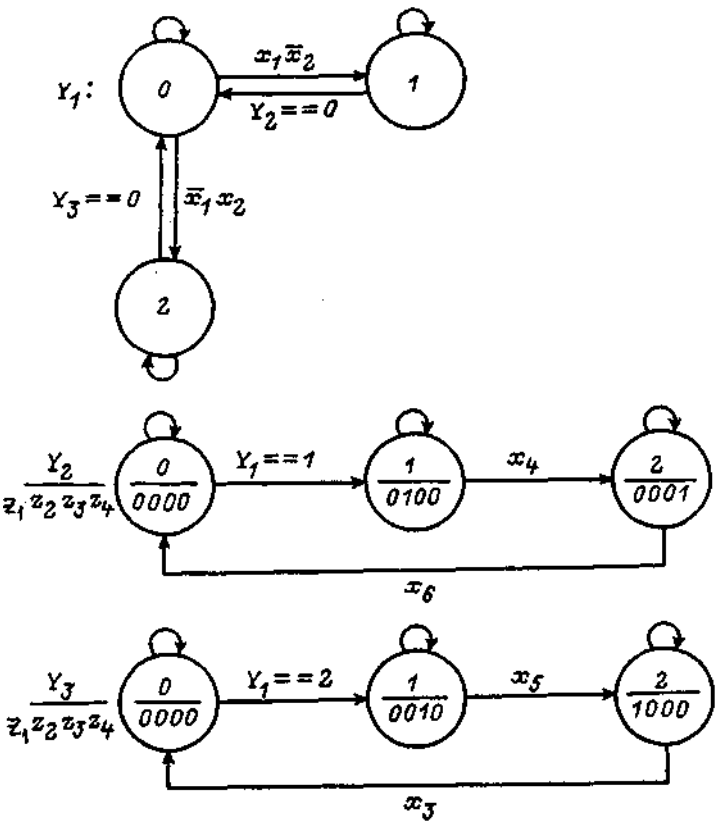

Рис. 8.18

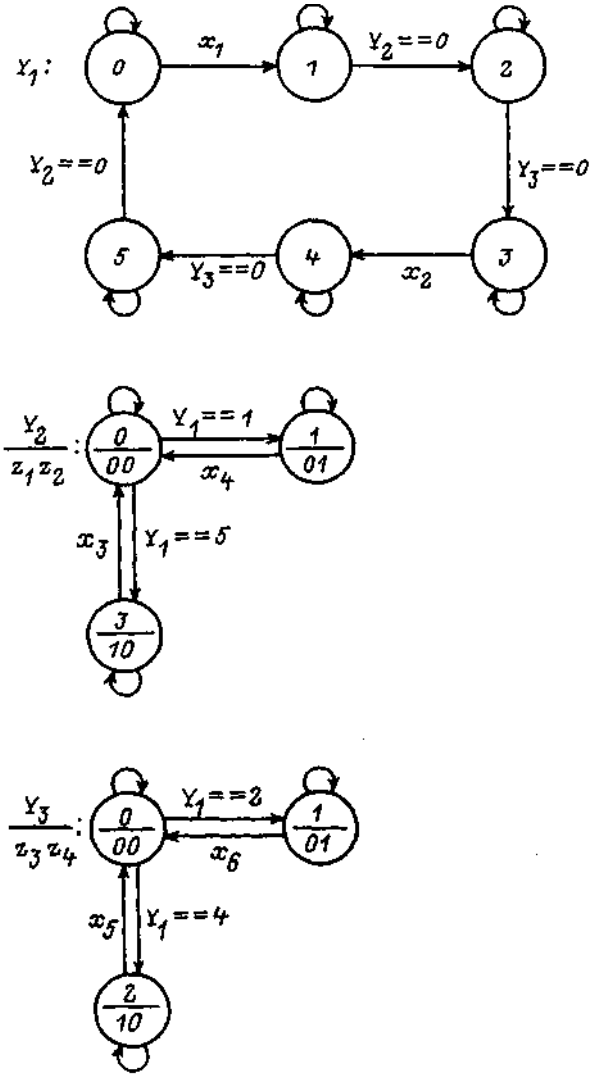

Рис. 8.19

заместитель второго графа (вершина с номером два и нулевыми выходами) и соединяя ее (с вершиной номер пять) дугой с пометкой $Y_2 == 0$, получим первый граф переходов. Введя начальную вершину с нулевыми выходами во второй фрагмент и соединяя ее (с вершиной номер два этого фрагмента) дугой с пометкой $Y_1 == 2$, получим второй граф переходов (рис. 8.17). Исключив в первом графе третью и четвертую выходные переменные, а во втором — первую и вторую и проводя перенумерацию вершин, получим

систему ГП (рис. 5.52), которой соответствует схема связи на рис. 5.51, определяющая декомпозицию по объектам.

При иерархическом управлении должен быть построен координирующий автомат, обеспечивающий взаимодействие автоматов более низких уровней.

При декомпозиции графа переходов (рис. 5.42) по режимам головной граф выбирает один из двух вызываемых графов, первый из которых обеспечивает открытие клапанов, а второй — их закрытие (рис. 8.18).

Другой подход к иерархическому управлению состоит в том, что головной граф осуществляет выбор режима, а первый (второй) вызываемый граф осуществляет управление первым (вторым) клапаном (рис. 5.56 и 8.19).

В заключение раздела отметим, что рассмотренный подход к организации системы графов переходов базируется на двух основных принципах:

— взаимодействие графов осуществляется по данным (номерам состояний);

— при реализации в каждом программном цикле производится опрос каждого графа, в котором выполняется не более одного перехода.

Если требуется, чтобы, например, один из вызываемых графов не опрашивался в каждом программном цикле, то он должен быть связан с головным графом по управлению, т. е. входить в него. Более подробно этот вопрос рассмотрен в разд. 12.4.

8.7. Взаимодействие процессов

Рассмотрим три класса задач о взаимодействии процессов.

Первый класс задач относится к прерываниям процессов (переключение активности процессов) с сохранением состояний прерванных процессов. При этом требуется обеспечить возможность при появлении сигнала, инициирующего некоторый процесс, прервать предыдущий с возможностью его продолжения при появлении сигнала, инициирующего прерванный процесс.

Второй класс задач относится к другому типу прерываний. При этом один из процессов в определенном состоянии при заданных условиях может прервать один или несколько других процессов, переведя их в фиксированные состояния. Автоматы, реализующие такие процессы, будем называть «автоматы с перелетами».

Третий класс задач связан с выбором процессов по инициирующим сигналам. При этом выбранный процесс не может быть прерван и продолжается до конца.

8.7.1. Переключение активности процессов

Комбинационное переключение активности процессов. Пусть имеются два двоичных сигнала: p_1 и p_2. При этом требуется, чтобы при $P_1 = P_2 = 0$ был активен (т. е. мог изменять состояния под воздействием входных переменных x_1.) процесс Y_4, а процессы Y_5 и Y_6 были пассивными (т. е. сохраняли бы свои состояния). При $p_1 = 1$, $p_2 = 0$ активен процесс Y_5, а остальные — пассивны. При $p_1 = 0$, $p_2 = 1$ активен процесс Y_6, а

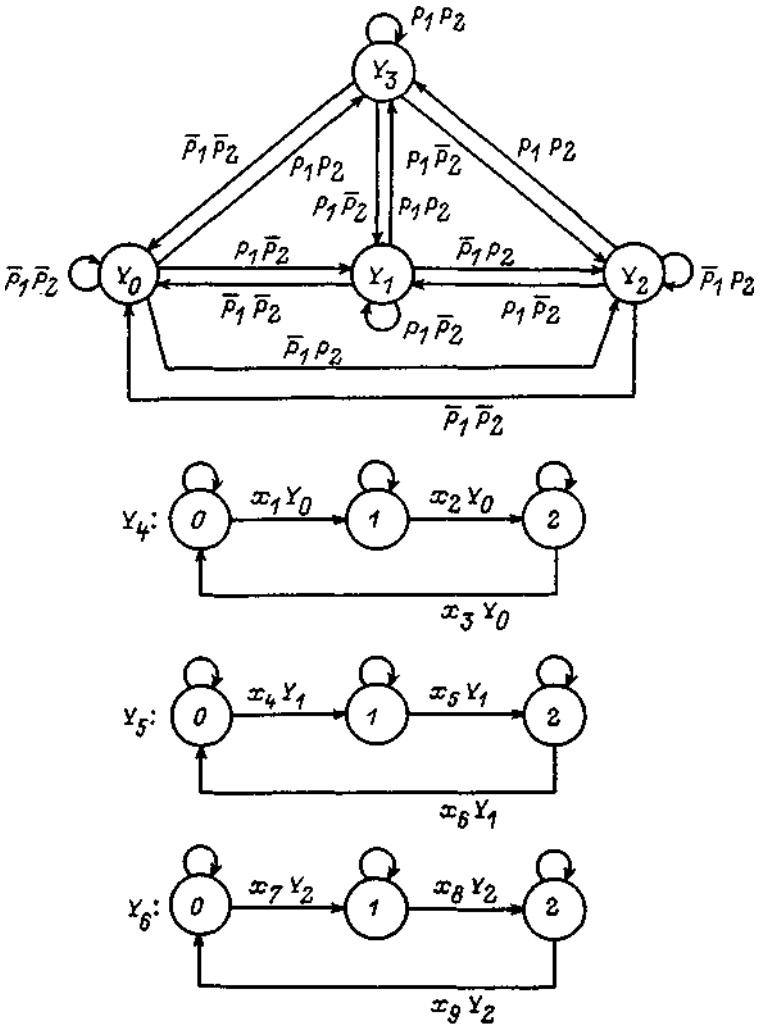

Рис. 8.20

Рис. 8.20

остальные — пассивны. При $p_1 = 1$, $p_2 = 1$ все три процесса сохраняют свои состояния (режим дезактивации).

Из изложенного следует, что управляющий (головной) ГП в этом случае должен быть полным, так как допустим любой порядок смены указанных ситуаций в системе. Поэтому указанный алгоритм может быть реализован СВГП, приведенной на рис. 8.20.

Головной ГП в этой системе, реализующий автомат с памятью при двоичном кодировании состояний, является весьма сложным как по структуре, так и по описанию его СБФ. Если, используя переменные $Y_0 — Y_3$, перейти к двоичному логарифмическому кодированию и постро-

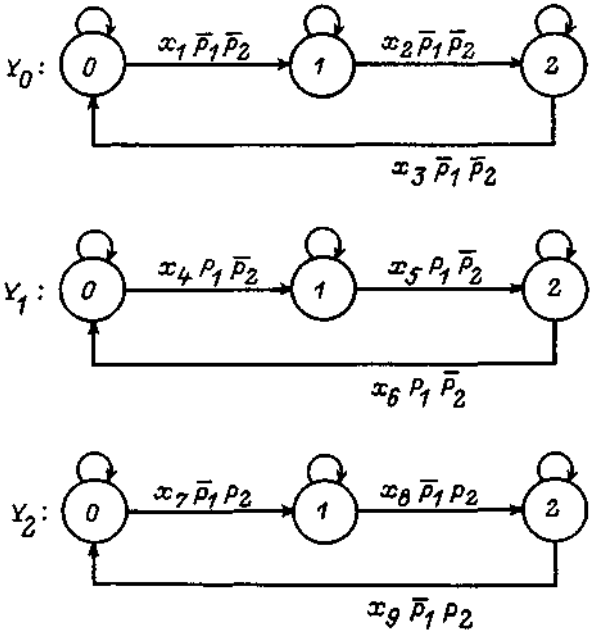

Рис. 8.21

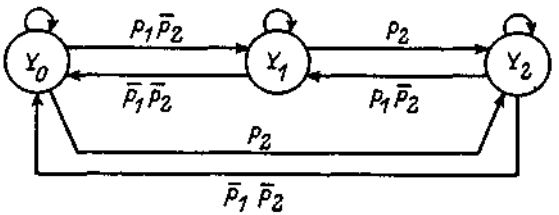

Рис. 8.22

ить ГП с 16 вершинами, в котором головной граф переходов сохраняется в виде ядра нового ГП с точностью до изменения пометок вершин ($Y_0 - Y_0 \& \bar{Y}_1 \& \bar{Y}_2 \& \bar{Y}_3$; $Y_1 - \bar{Y}_0 \& Y_1 \& \bar{Y}_2 \& \bar{Y}_3$; $Y_2 - \bar{Y}_0 \& \bar{Y}_1 \& Y_2 \& \bar{Y}_3$; $Y_3 - \bar{Y}_0 \& \bar{Y}_1 \& \bar{Y}_2 \& Y_3$), а остальные вершины не соединены между собой, а связаны лишь с вершинами ядра дугами с пометками: $\bar{p}_1 \& \bar{p}_2$ (для связей с вершиной «0»), $p_1 \& \bar{p}_2$ (для связей с вершиной « 1 $\bar{p}_1 \& p_2$1 л я связей с вершиной «2»), $p_1 \& p_2$ (для связей с вершиной «3»), то такой граф также будет решать поставленную задачу. Однако этот ГП реализует уже не автомат с памятью, а комбинационную схему, которая описывается простой СБФ:

$$Y_0 = \bar{p}_1 \& \bar{p}_2;\ Y_1 = p_1 \& \bar{p}_2;\ Y_2 = \bar{p}_1 \& p_2;\ Y_3 = p_1 \& p_2.$$

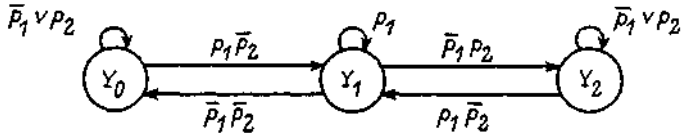

Рис. 8.23

Именно этой СБФ и может быть заменен головной ГП на рис. 8.20. При этом заданный алгоритм может быть реализован следующей программой на языке СИ:

$Y_0 = \bar{p}_1 \,\&\, \bar{p}_2;\ Y_1 = p_1 \,\&\, \bar{p}_2;\ Y_2 = \bar{p}_1 \,\&\, p_2;\ Y_3 = p_1 \,\&\, p_2;$

```
      switch (Y4)   {
case 0: if (x1 & Y0)  Y4 = 1; break;
case 1: if (x2 & Y0)  Y4 = 2; break;
case 2: if (x3 & Y0)  Y4 = 0; break; }
  ...
      switch (Y6)   {
case 0: if (x7 & Y2)  Y6 = 1; break;
case 1: if (x8 & Y2)  Y6 = 2; break;
case 2: if (x9 & Y2)  Y6 = 0; break; }.
```

Решение рассматриваемой задачи может быть еще более упрощено, если головной ГП исключить, а инициирующие сигналы перенести в графы переходов процессов (рис. 8.21).

Если в предыдущей задаче изменить условие так, что при $p_2 = 1$ активным становится ГП с пометкой Y_2, то головной граф упрощается (рис. 8.22).

Если от этого ГП, описывающего автомат с памятью, перейти к другому ГП с восемью вершинами, реализующему комбинационную схему, то головной ГП может быть заменен СБФ: $Y_0 = \bar{p}_1 \,\&\, \bar{p}_2;\ Y_1 = p_1 \,\&\, \bar{p}_2;\ Y_2 = p_2$. Как и в предыдущем случае, головной ГП может быть и вовсе исключен.

Последовательностное переключение активности процессов. Предположим, что порядок смены ситуаций, определяемых инициирующими входными воздействиями, в отличие от предыдущих двух случаев не произволен, а строго фиксирован. В этом случае управляющий ГП описывает автомат с памятью, который не может быть комбинационной схемой.

Предположим, что в рассматриваемом примере смена ситуаций задается управляющим ГП, приведенным на рис. 8.23. Этому графу может быть сопоставлена диаграмма (рис. 8.24), на

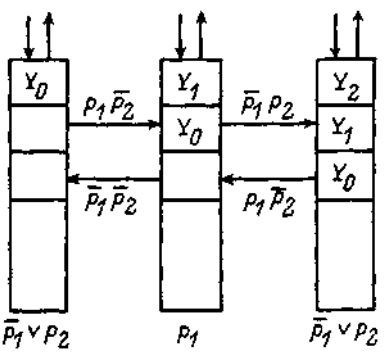

Рис. 8.24

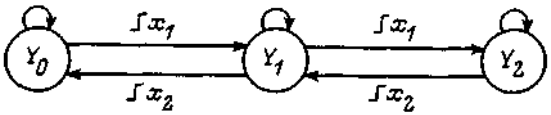

Рис. 8.25

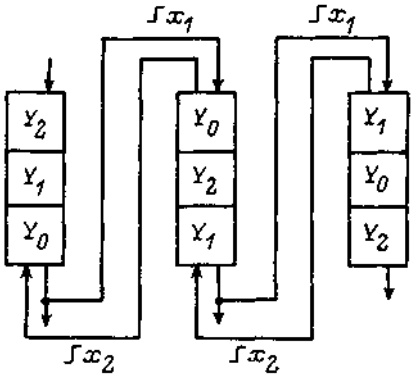

Рис. 8.26

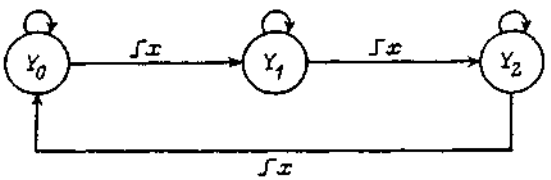

Рис. 8.27

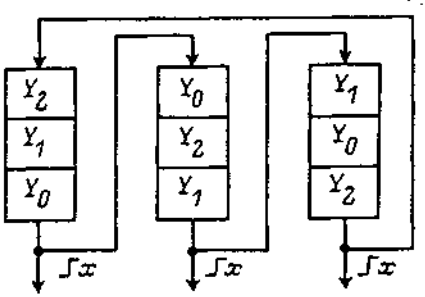

Рис. 8.28

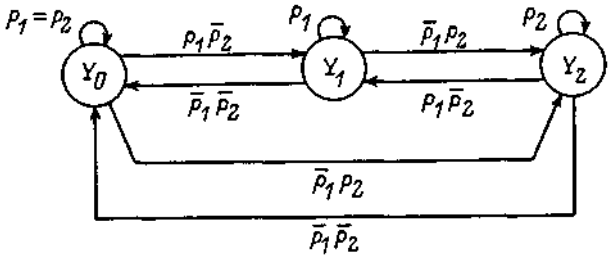

Рис. 8.29

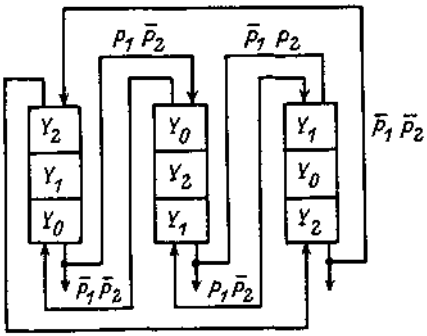

Рис. 8.30

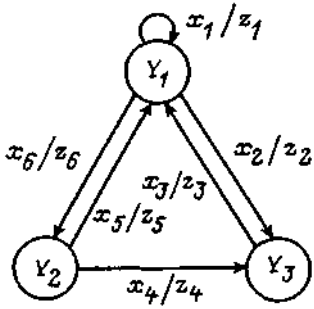

Рис. 8.31

которой указано содержимое стека в разных состояниях, т. е. используется конструкция LIFO (Last In First Out). При этом содержимое верхушки стека моделирует состояния автомата.

На рис. 8.25 приведен управляющий ГП, использующий две импульсные переменные x_1, и x_2, первая из которых увеличивает номер вызываемого графа, а вторая — его уменьшает. Этому ГП может быть сопоставлена диаграмма (рис. 8.26), на которой указано содержимое реверсивного сдвигового регистра в разных состояниях. При этом содержимое дна регистра моделирует состояния автомата.

Управляющему ГП (рис. 8.27) может быть сопоставлена диаграмма (рис. 8.28), на которой указано содержимое сдвигового регистра в разных состояниях, т. е. используется конструкция FIFO (First In First Out). При этом содержимое регистра моделирует состояния автомата.

Управляющему ГП (рис. 8.29) может быть сопоставлена диаграмма (рис. 8.30), на которой указано содержимое кольцевого реверсивного сдвигового регистра в разных состояниях. Граф переходов (рис. 8.29) осуществляет произвольное переключение активности процессов без приоритетов и режима дезактивации и поэтому в отличие от графа на рис. 8.20 и 8.22 не может быть заменен графом переходов, соответствующим комбинационной схеме.

Отметим также, что ГП могут быть весьма просто представлены в виде нелинейных связных списков. На рис. 8.31 в качестве примера приведен

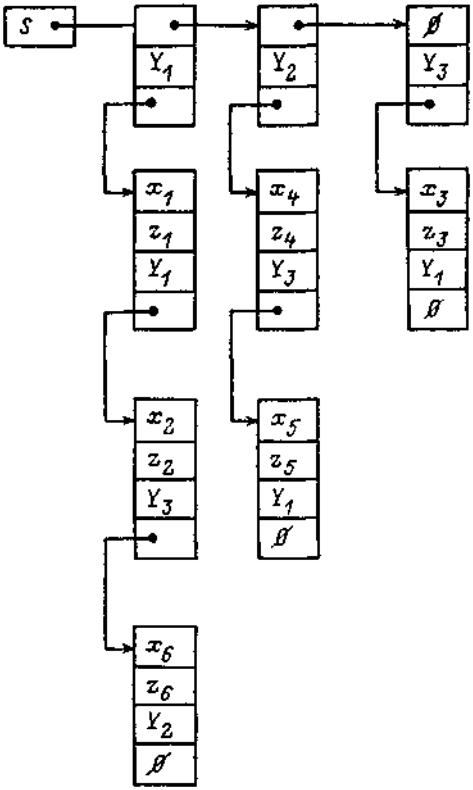

Рис. 8.32

ГП автомата Мили, для обеспечения корректности которого его входные переменные должны быть переобозначены, а на рис. 8.32 — его реализация в виде нелинейного двухсвязного списка [240].

8.7.2. Графы переходов с «перелетами»

Для ГП характерно, что переходы в каждом из них осуществляются только по дугам. Для системы вызываемых ГП взаимодействие между ними осуществляется не введением дуг между графами (связи по управлению), а за счет соответствующих пометок их дуг (связи по данным).

Если же такие связи, например между двумя графами, необходимы, то они рассматриваются обычно как один граф.

Если допустить связи по управлению между ГП, не рассматривая при этом их как одну компоненту, то в управляемом графе появляются переходы между его вершинами, происходящие не по дугам графа. Поэтому такие графы можно назвать «графами переходов с перелетами». Эти графы описывают прерывания без сохранения предыдущего состояния.

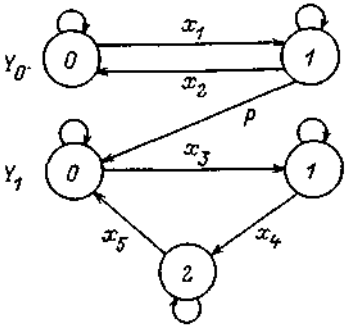

Рис. 8.33

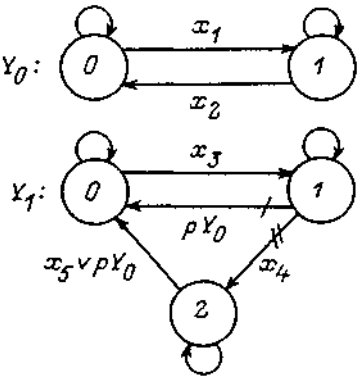

Рис. 8.34

На рис. 8.33 в качестве примера приведены два графа переходов, связанных между собой по управлению. Связь между ними следует понимать в том смысле, что при переходе первого графа в первую вершину и при $p = 1$ второй граф остается в нулевой вершине или «перелетает» в нее, вне зависимости от того, в какой вершине он находился до этого. Условие «перелета»: $(Y_0 == 1)$ & $(p = 1)$.

В первом графе дуги, исходящие из первой вершины, не ортогонализованы, что позволяет при $x_2 = p = 1$ выполнить параллельно два события: в первом графе перейти в нулевую вершину, а во втором — остаться в нулевой вершине или «перелететь» в нее. Эти графы реализуются следующей программой на языке СИ:

```
      switch (Y0) {
case 0: if (x1)     Y0 = 1; break;
case 1: if (x2)     Y0 = 0;
        if (p)      Y1 = 0; break; }
      switch (Y1) {
case 0: if (x3)     Y1 = 1; break;
case 1: if (x4)     Y1 = 2; break;
case 2: if (x5)     Y1 = 0; break; }.
```

Рассмотренная задача может быть также решена с помощью системы взаимосвязанных ГП, в которой графы не связаны между собой по управлению (дополнительными дугами), а связаны лишь по данным (без использования таких дуг) (рис. 8.34).

Противоречивость во втором графе устранена расстановкой приоритетов. Эти графы реализуются более сложной программой по сравнению с приведенной выше. В данном случае она будет иметь то же число строк, что и предыдущая, но два условия в ней являются более сложными.

Наиболее естественной является связь между графами по данным в случае, когда в одном из них на дуге или в вершине формируется номер состояния, в которое должен «перелететь» другой граф. При этом во втором графе пометки о состояниях первого из них не используются.

8.7.3. Выбор процессов

Простейший выбор. В графе переходов (рис. 8.35) осуществляется выбор (и, возможно, многократная реализация) одного из трех процессов или не выбор ни одного из них.

Выбор, определяемый автоматом. Предположим, что при $p_1 = p_2 = 0$ должен однократно выполниться первый процесс. После его окончания при $p_1 = 1$, $p_2 = 0$ должен также однократно выполняться второй процесс. После его завершения при $p_1 = 0$, $p_2 = 0$ снова однократно выполняется первый процесс, а при $p_1 = p_2 = 1$ — третий. После однократного выполнения этого процесса при $p_1 = 1$, $p_2 = 0$ опять должен выполняться второй процесс. Этот алгоритм реализуется системой взаимосвязанных графов переходов (рис. 8.36), соответствующих автоматам Мили с флагами.

В заключение раздела отметим, что в следующей главе приведен пример использования прерываний, трактуемых более традиционно, а пример применения «реальных» прерываний от клавиатуры рассмотрен в Приложении 6.

В этом приложении приведены два примера программ, моделирующих автомат управления клапаном с памятью, с вводом входных переменных с помощью клавиатуры ПЭВМ.

Поведение этого автомата задается графом переходов Мура с приоритетами, отличающимся от графа переходов автомата без выходного преобразователя, приведенного на рис. 5.38, в частности тем, что пометка дуги $x_1 \bar{x}_4$ в нем заменена на пометку x_1. При этом непротиворечивость в вершине «0» обеспечивается тем, что контроллер клавиатуры в каждый момент времени обрабатывает нажатие только одной клавиши. При

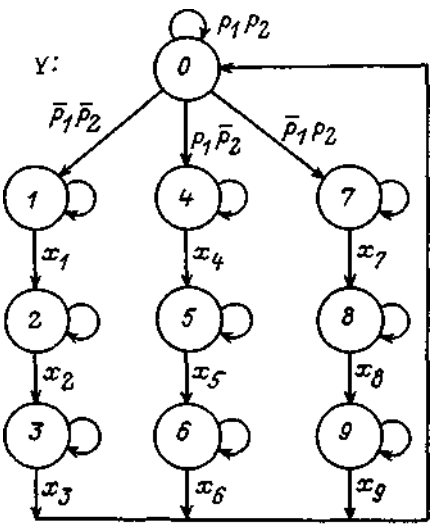

Рис. 8.35

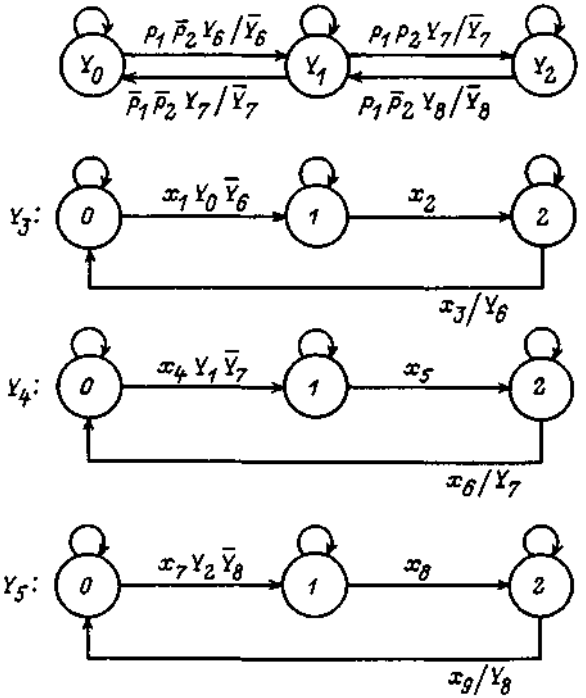

Рис. 8.36

нажатии двух клавиш одновременно контроллер формирует сигналы в порядке возрастания кодов, присвоенных клавишам. Поэтому при одновременном нажатии клавиш 1 и 4 осуществляются два перехода по траектории «0—1—2», а при таком же нажатии клавиш 2 и 3 — по траектории «2—3—0».

В первой программе нажатая клавиша определяется в результате циклического опроса клавиатуры с последующим считыванием кода нажатой клавиши, а во второй — по прерыванию, возникающему при нажатии клавиши. Особенность этих программ состоит в том, что в каждой из них операционная и функциональная части разделены.