Глава 12

Язык «Графсет» и графы переходов

12.1. Язык «Графсет». Основные понятия

Наиболее известным из современных языков спецификации и программирования для решения задач логического управления является язык «Графсет» [59] и его модификации, широко применяемые в различных типах программируемых логических контроллеров [260].

Название этого языка на французском «Grafcet» связано с начальными буквами выражения «de Graphe de Commande des Etapes et Transitions», которое на русский язык переводится следующим образом: «граф команд с этапами и переходами».

Основными достоинствами этого языка являются возможность описания параллельно-последовательных процессов в наглядной и компактной форме и автоматическое получение по этому описанию (при наличии транслятора) управляющих программ.

Теоретической основой языка «Графсет» являются сети Петри [145, 146], которые рассмотрены в гл. 11.

При этом для целей управления используются сильно связанные сети Петри, в которых из любой вершины существует путь вдоль дуг в любую вершину, а исходная маркировка определяется только одной фишкой, располагаемой в начальной вершине.

В [145] приведен метод реализации автоматов Мили с помощью сетей Петри. При этом входные и выходные символы, а также состояния автомата моделируются позициями, а переходы моделируют события, состоящие в обеспечении маркировки позиций, соответствующих следующему состоянию и значениям выходных переменных, на основе маркировки позиций, соответствующих настоящему состоянию и значениям входов. Достоинство такой модели состоит в том, что она позволяет весьма просто осуществить композицию таких моделей, однако ее громоздкость, связанная с представлением значений входов и выходов позициями, практически исключает применение традиционных сетей Петри для реализации автоматов.

Ситуация с практическим использованием резко изменяется, если применять не традиционные, а помеченные сети Петри, в которых маркированная позиция соответствует выполнению некоторого этапа алгоритма, а срабатывание перехода (при выполнении условия, которым

помечен переход) обеспечивает передачу одного или нескольких маркеров из предыдущих позиций в следующие.

Именно помеченные сети Петри и легли в основу языка «Графсет», при создании которого были частично изменены и расширены графические средства базового языка.

Диаграмму на языке «Графсет» образуют следующие основные элементы [59]:

— квадрат, изображающий этап, выполняемый в одной из ветвей процесса;

— прямоугольник, связанный горизонтальной линией с этапом и предназначенный для записи выходных воздействий;

— полочка-переход, разделяющий два последовательных этапа;

— булева формула (преемственность), связанная с переходом и определяющая условия его прохождения;

— расходимость и сходимость по ИЛИ, изображаемые одиночными горизонтальными линиями и служащие для отображения оператора «Выбор»;

— расходимость и сходимость по И, изображаемые двойными горизонтальными линиями и служащиедля отображения запуска и завершения параллельных процессов и синхронизации этапов;

— одиночные вертикальные линии, осуществляющие связь переходов с этапами и наоборот.

Диаграмма в целом должна быть замкнутой. Ее функционирование состоит в передаче активности (меток) от этапа к этапу (в том числе и параллельно) при выполнении условий преемственности.

12.2. Реализация языка «Графсет»

Для программной реализации процесса, описанного на языке «Графсет», необходимо для каждого фрагмента диаграммы выбрать соответствующую алгоритмическую модель.

В качестве такой модели для отображения *i-го* этапа в [59] предложено использовать двоичную переменную, описываемую как *S*- или *R*-триггер. В дальнейшем для определенности применяется описание переменной с помощью *R*-триггера, для которого $Y_i^н = \bar{R}\,(S \vee Y_i)$ (индекс «н» обозначает новое значение переменной). При этом соотношения для записи *R* и *S* зависят от конфигурации фрагмента, включающего предыдущий Y_{i-1} (предыдущие), настоящий Y_i и следующий Y_{i+1} (следующие) этапы и условия переходов X_{i-1} и X_i, являющиеся булевыми формулами от входных переменных.

Таким образом, основная идея этого метода программной реализации диаграммы, построенной в базисе языка «Графсет» (диаграмма «Графсет»), состоит в записи эквивалентной ей системы булевых формул.

Исторически [13, 14, 95, 147—150] сначала решалась задача об асинхронной модульной схемной реализации алгоритмов, заданных с помощью сети Петри или диаграммы «Графсет». При этом для обеспече-

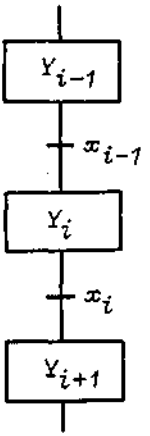


Рис. 12.1

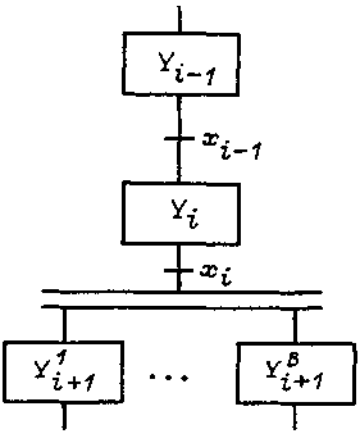


Рис. 12.2

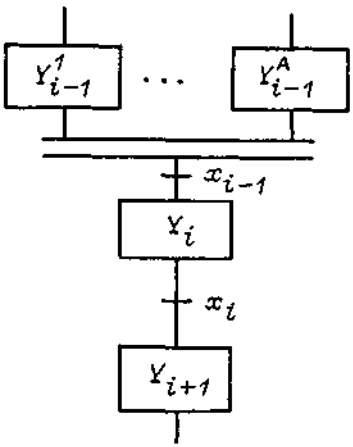


Рис. 12.3

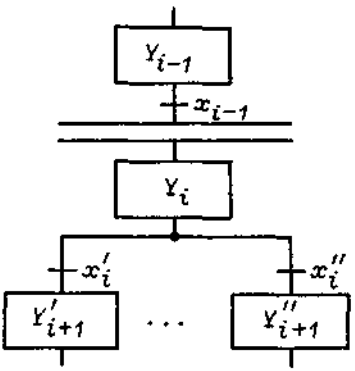


Рис. 12.4

ния правильного функционирования схемы использовалась следующая дисциплина передачи активности между соседними модулями: если после актиния правильного функционирования схемы использовалась следующая дисциплина передачи активности между соседними модулями: если после активизации *i*-го модуля должен быть активизирован *(i* + 1)-й модуль, то он активизируется, и лишь после этого снимается активность *i*-го модуля.

I

Рис. 12.3

Эта дисциплина взаимодействия модулей была сохранена [59] и при программной реализации. В табл. 12.1 для основных типовых фрагментов диаграмм «Графсет» (рис. 12.1—12.5) приведены булевы формулы, обеспечивающие реализацию (первый метод) рассмотренной выше дисциплины взаимодействия модулей (этапов).

На рис. 12.6 в качестве примера приведена диаграмма «Графсет», реализующая алгоритм логического управления, в которой могут протекать параллельные процессы.

В силу того что рассматриваемая диаграмма содержит 8 этапов и 3 выхода, то СБФ содержит 11 булевых формул. Структура формул,

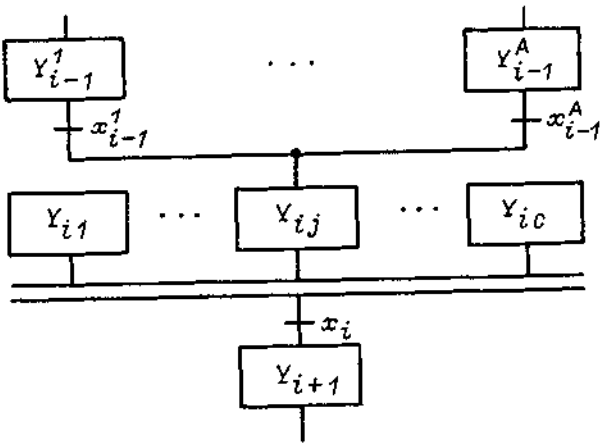


Рис. 12.5

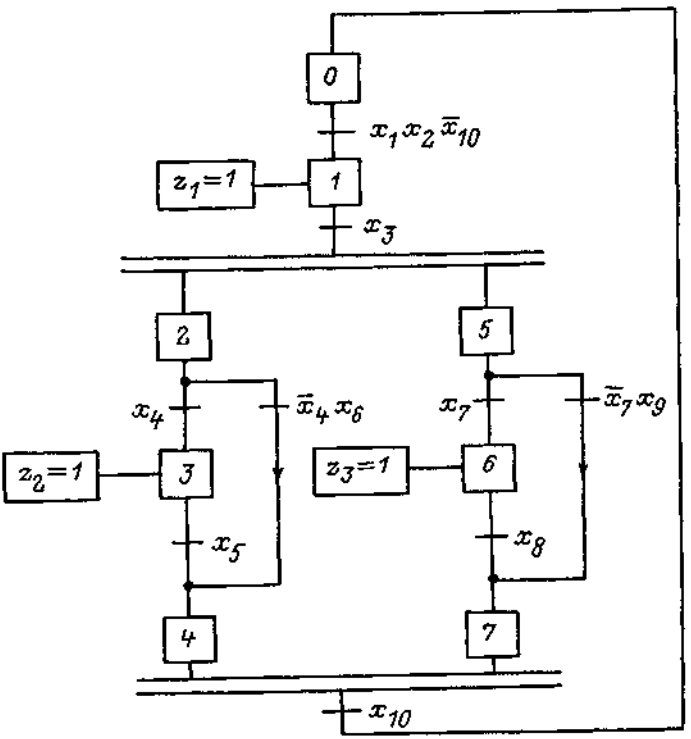


Рис. 12.6

Таблица 12.1

Номер фрагмента	Формула
1	$Y_i^{\text{к}} = \overline{Y}_{i+1}(x_{i-1} Y_{i-1} \vee Y_i)$
2	$Y_i^{\text{к}} = Y_{i+1}^{1} \vee \ldots \vee Y_{i+1}^{\text{в}}(x_{i-1} Y_{i-1} \vee Y_i)$
3	$Y_i^{\text{н}} = \overline{Y}_{i+1}(x_{i-1} Y_{i-1}^{1} \ldots Y_{i-1}^{\text{а}} \vee Y_i)$
4	$Y_i^{\text{к}} = Y'_{i+1} \vee \ldots \vee Y''_{i+1}(x_{i-1} Y_{i-1} \vee Y_i)$
5	$Y_{ij}^{\text{н}} = \overline{Y}_{i+1}(x_{i-1}^{1} Y_{i-1}^{1} \vee \ldots \vee x_{i-1}^{\text{а}} Y_{i-1}^{\text{а}} \vee Y_{ij})$

описывающих этапы, определяется по диаграмме (рис. 12.6) в соответствии с табл. 12.1:

$Y_0 = 1;$

M: $Y_{01} = \overline{Y}_1 (x_{10} Y_4 Y_7 \vee Y_0);$

$Y_{11} = \overline{Y_2 \vee Y_5}\,(x_1 x_2 \bar{x}_{10} Y_0 \vee Y_1);$

$Y_{21} = \overline{Y_3 \vee Y_4}\,(x_3 Y_1 \vee Y_2);$

$Y_{31} = \overline{Y}_4 (x_4 Y_2 \vee Y_3);$

$Y_{41} = \overline{Y}_0 (\bar{x}_4 x_6 Y_2 \vee x_5 Y_3 \vee Y_4);$

$Y_{51} = \overline{Y_6 \vee Y_7}\,(x_3 Y_1 \vee Y_5);$

$Y_{61} = \overline{\overline{Y}_7}\,(x_7 Y_5 \vee Y_6);$

$Y_7 = \overline{\overline{Y}_0}\,(\bar{x}_7 x_9 Y_5 \vee x_8 Y_6 \vee Y_7);$

$z_1 = Y_{11}; \quad z_2 = Y_{31}; \quad z_3 = Y_{61};$
$Y_0 = Y_{01}; \quad Y_1 = Y_{11}; \quad Y_2 = Y_{21}; \quad Y_3 = Y_{31}; \quad Y_4 = Y_{41};$
$Y_5 = Y_{51}; \quad Y_6 = Y_{61};$
goto M.

В построенной системе применяются переобозначения переменных Y_i, для того чтобы в течение одного программного цикла исключить возможность использования новых значений этих переменных.

Основной недостаток диаграмм «Графсет» состоит в том, что для отражения параллелизма в диаграмме приходится применять двоичное кодирование вершин, соответствующих этапам. При этом для диаграммы, содержащей V этапов, должно использоваться V двоичных переменных Y_i, а с учетом переобозначений число двоичных переменных, предназначенных для кодирования вершин, увеличивается до величины $2V - 1$.

Число двоичных переменных при операторной реализации формул полученной системы может увеличиваться еще больше.

Это приводит к существенному расходу битовых ячеек оперативной памяти, число которых в ПЛК обычно незначительно.

В качестве примера приведенная выше СБФ была реализована на языке инструкций ALPro (гл. 14). Число команд в программе — 81, а число битовых ячеек оперативной памяти — 16.

Необходимо отметить, что поведение диаграммы в целом не изменится, если в выражении под инверсией в формуле второго типа (табл. 12.1) знаки дизъюнкции заменить на знаки конъюнкции. Отсюда следует, что в построенной выше системе вторая формула может быть записана следующим образом:

$$Y_{11} = \overline{Y_2 Y_5}\,(x_1\, x_2\, \overline{x_{10}}\, Y_0 \vee Y_1).$$

Возвращаясь собственно к языку «Графсет», отметим, что в известных автору версиях программного обеспечения для ПЛК, использующих этот язык, отсутствовала возможность проведения отладки программ на ПЭВМ, а для этой цели требуется применять собственно ПЛК, что не позволяет проводить отладку программы до поставки управляющей аппаратуры.

Наличие системы булевых формул и оболочки, обеспечивающей возможность устанавливать требуемые значения входных переменных и наблюдать значения всех внутренних и выходных переменных, позволяет моделировать поведение рассматриваемой схемы на ПЭВМ. При этом значения тех внутренних переменных, которым соответствуют активизированные этапы, равны единице, а для остальных — нулю.

Как отмечалось выше, при использовании формул из табл. 12.1 сохраняется тот недостаток модульных схем, что после срабатывания перехода активность «пройденного» этапа снимается не мгновенно, а лишь на следующем цикле вычислений. Это приводит к тому, что в некоторые моменты времени активными являются также и те этапы, которые по логике работы диаграммы должны быть сброшены. При этом весьма велико и число шагов, требующихся для полного обхода диаграммы при фиксированных значениях входных переменных.

Табл. 12.2 описывает функционирование (изменение активности этапов) диаграммы (рис. 12.6) при ее однократном обходе для случая применения СБФ, приведенной выше.

Таблиа 12.2

Условия	Номер шага	Номер вершины							
		0	1	2	3	4	5	6	7
$x_1 = 1$	0	1	0	0	0	0	0	0	0
$x_2 = 1$	1	1	1	0	0	0	0	0	0
$x_3 = 1$	2	0	1	1	0	0	1	0	0
$x_4 = 0$	3	0	0	1	0	1	1	0	1
$x_6 = 1$ $x_7 = 0$ $x_9 = 1$ $x_{10} = 0$	4	0	0	0	0	1	0	0	1
$x_{10} = 1$	5	1	0	0	0	0	0	0	0

Таблица 12.3

Номер фрагмента	Формула
1	$Y_i^{\text{н}} = \overline{x_i Y_i^{\text{н}}}\,(x_{i-1}\,Y_{i-1} \vee Y_i)$
2	$Y_i^{\text{н}} = \overline{x_i \overline{Y_i^{\text{н}}}}\,(x_{i-1}\,Y_{i-1} \vee Y_i)$
3	$Y_i^{\text{н}} = \overline{x_i \overline{Y_i^{\text{н}}}}\,(Y_{i-1}^{1} \,\&\, \ldots \,\&\, Y_{i-1}^{A} \vee Y_i)$
4	$Y_i^{\text{н}} = \overline{(x_i' \vee \ldots \vee x_i'')\,\overline{Y_i^{\text{н}}}}\,(x_{i-1}\,Y_{i-1} \vee Y_i)$
5	$Y_{ij}^{\text{н}} = \overline{\overline{Y_{ij}^{\text{н}}}\,\overline{Y_{i1} \ldots Y_{i(j-1)} Y_{i(j+1)} \ldots \overline{Y_{ic}}}}\,(x_{i-1}^{1}\,Y_{i-1}^{1} \vee \ldots \vee x_{i-1}^{A}\,Y_{i-1}^{A} \vee Y_{ij})$

Еще один недостаток рассмотренного описания состоит в том, что если в диаграмме имеются кольца, состоящие из двух этапов, то они должны быть предварительно преобразованы введением дополнительных этапов.

Для устранения указанных недостатков автором предлагается использовать (второй метод) другое описание рассмотренных фрагментов (табл. 12.3).

При этом обеспечивается мгновенный сброс «пройденного» этапа, повышается быстродействие, а число активных этапов в диаграмме соответствует требуемому их количеству.

При применении этих соотношений система булевых формул для примера, приведенного на рис. 12.6, приобретает вид:

$Y_0 = 1;$

M: $Y_{01} = \overline{x_1 x_2 \overline{x}_{10} Y_{01}}\,(x_{10} Y_4 Y_7 \vee Y_0);$

$Y_{11} = \overline{x_3 Y_{11}}\,(x_1 x_2 \overline{x}_{10} Y_0 \vee Y_1);$

$Y_{21} = \overline{Y_{21}}\,(x_3 Y_1 \vee Y_2);$

$Y_{31} = \overline{x_5 Y_{31}}\,(x_4 Y_2 \vee Y_3);$

$Y_{41} = \overline{Y_{41} Y_7}\,(\overline{x}_4 x_6 Y_2 \vee x_5 Y_3 \vee Y_4);$

$Y_{51} = \overline{Y_{51}}\,(x_3 Y_1 \vee Y_5);$

$Y_{61} = \overline{x_8 Y_{61}}\,(x_7 Y_5 \vee Y_6);$

$Y_7 = \overline{Y_4 Y_7}\,(\overline{x}_7 x_9 Y_5 \vee x_8 Y_6 \vee Y_7);$

$z_1 = Y_{11};\quad z_2 = Y_{31};\quad z_3 = Y_{61};$
$Y_0 = Y_{01};\quad Y_1 = Y_{11};\quad Y_2 = Y_{21};\quad Y_3 = Y_{31};$
$Y_4 = Y_{41};\quad Y_5 = Y_{51};\quad Y_6 = Y_{61};$

goto M.

Отметим, что при программной реализации этой системы на языке инструкций ALPro число команд в программе по сравнению с предыдущим вариантом увеличивается.

Таблица 12.4

Условия	Номер шага	Номер вершины							
		0	1	2	3	4	5	6	7
$x_1 = 1$	0	1	0	0	0	0	0	0	0
$x_2 = 1$	1	0	1	0	0	0	0	0	0
$x_3 = 1$	2	0	0	1	0	0	1	0	0
$x_4 = 0$ $x_6 = 1$ $x_7 = 0$ $x_9 = 1$ $x_{10} = 0$	3	0	0	0	0	1	0	0	1
$x_{10} = 1$	4	1	0	0	0	0	0	0	0

Поведение диаграммы в этом случае для одного ее обхода описывается в табл. 12.4.

Таким образом, описание фрагментов, предложенное автором, позволяет моделировать на ПЭВМ поведение диаграмм «Графсет» по быстродействию более эффективно по сравнению с [59].

Однако этот подход, являющийся эволюционным развитием метода, изложенного в [59], весьма специфичен. Предложенное описание во многом определяется графическим представлением диаграмм «Графсет», и в частности отсутствием изображения петель, каждая из которых позволяет отобразить «длительное» пребывание в этапе. При этом должны выполняться условия, являющиеся логическим дополнением дизъюнкции условий, указанных на других дугах, исходящих из рассматриваемого этапа. Наличие необходимых петель в диаграммах «Графсет» предполагается по умолчанию.

Диаграммы «Графсет» могут быть весьма просто и естественно описаны СБФ (третий метод), если использовать:

— двоичное кодирование (разд. 3.3) этапов;

— условия длительного пребывания в этапах, отсутствующие в явном виде в диаграммах;

— метод записи формул (разд. 4.1.3), расширенный учетом условий выполнения и завершения параллельных процессов.

При этом если имеются параллельные процессы, завершающиеся этапами Y_i, Y_j, Y_k, то условие продолжения i-го этапа имеет вид $Y_i\overline{Y_j Y_k} = 1$, а условием завершения всех процессов является равенство $Y_i Y_j Y_k = 1$.

Применяя этот метод, построим СБФ для диаграммы на рис. 12.6:

$$Y_0 = 1;$$

$$\text{M: } Y_{01} = x_1 x_2 \overline{x_{10}} Y_0 \vee x_{10} Y_4 Y_7;$$

$$Y_{11} = x_1 x_2 \overline{x_{10}} Y_0 \vee \overline{x_3} Y_1;$$

$Y_{21} = x_3 Y_1;$

$Y_{31} = x_4 Y_2 \vee \bar{x}_5 Y_3;$

$Y_{41} = \bar{x}_4 x_6 Y_2 \vee x_5 Y_3 \vee Y_4 \overline{Y_7};$

$Y_{51} = x_3 Y_1;$

$Y_{61} = x_7 Y_5 \vee x_8 Y_6;$

$Y_7 = \bar{x}_7 x_9 Y_5 \vee x_8 Y_6 \vee \overline{Y_4} Y_7;$

$z_1 = Y_{11}; \quad z_2 = Y_{31}; \quad z_3 = Y_{61};$
$Y_0 = Y_{01}; \quad Y_1 = Y_{11}; \quad Y_2 = Y_{21}; \quad Y_3 = Y_{31};$
$Y_4 = Y_{41}; \quad Y_5 = Y_{51}; \quad Y_6 = Y_{61};$
goto M.

Эта система, так же как и предыдущая, обеспечивает однофазный режим работы: за один цикл осуществляется снятие активности одних и возбуждение других этапов.

Число команд в программе, построенной по этой системе, практически совпадает с их количеством, полученным выше для первой СБФ.

Трудоемкость построения программы и число команд в ней могут быть снижены, если в качестве описания диаграммы «Графсет» применять не систему булевых формул, а систему секвенций [39, 174], дополненную переобозначениями двоичных переменных, кодирующих этапы (четвертый метод).

Для рассматриваемого примера может быть использовано следующее описание диаграммы:

$Y_0 = 1;$

M: $Y_0 x_1 x_2 \bar{x}_{10} \quad \vdash \overline{Y_0} Y_{11};$
$Y_1 (z_1 \bar{z}_2 \bar{z}_3) x_3 \quad \vdash \overline{Y_1} Y_{21} Y_{51};$
$Y_2 x_4 \quad \vdash \overline{Y_2} Y_{31};$
$Y_2 \bar{x}_4 x_6 \quad \vdash \overline{Y_2} Y_{41};$
$Y_3 (\bar{z}_1 z_2 \bar{z}_3) x_5 \quad \vdash \overline{Y_3} Y_{41};$
$Y_5 x_7 \quad \vdash \overline{Y_5} Y_{61};$
$Y_5 \bar{x}_7 x_9 \quad \vdash \overline{Y_5} Y_{71};$
$Y_6 (\bar{z}_1 \bar{z}_2 z_3) x_8 \quad \vdash \overline{Y_6} Y_{71};$
$Y_4 Y_7 x_{10} \quad \vdash \overline{Y_4} \overline{Y_7} Y_0;$
$Y_1 = Y_{11}; \quad Y_2 = Y_{21}; \quad Y_3 = Y_{31};$
$Y_4 = Y_{41}; \quad Y_5 = Y_{51}; \quad Y_6 = Y_{61}; \quad Y_7 = Y_{71};$
goto M,

где $\vdash$ — знак секвенции.

Эта система на языке инструкций ALPro (при использовании в качестве основной инструкции IF) изоморфно реализуется программой, содержащей 71 команду и использующей 14 битовых ячеек оперативной памяти.

В заключение раздела отметим, что все рассмотренные методы реализации применяются к каждой диаграмме «Графсет» как к единой компоненте и для отражения параллелизма в ней применяют двоичное кодирование вершин, соответствующих этапам.

Это приводит к сравнительно большому числу команд в программах и значительному расходу битовых ячеек оперативной памяти, что ограничивает применение языка «Графсет» для ПЛК с небольшими ресурсами.

Эти недостатки устраняются с помощью подхода, излагаемого в следующем разделе.

12.3. Реализация параллельных процессов системой графов переходов

Как было отмечено выше, появление языка «Графсет» было вызвано необходимостью эффективного отображения параллельно-последовательных процессов.

Этот язык получил большее распространение, чем, например, такие языки, как параллельные ГСА [4] или язык параллельных алгоритмов логического управления [154], однако, по мнению автора, и в использовании этого языка для рассматриваемого класса задач нет необходимости.

Это мнение базируется на принципе Оккама, по которому «не следует размножать сущности без необходимости» [287].

Исторически первой «сущностью» в области формализации задач логического управления являются графы переходов. Ниже излагается подход, позволяющий применять графы переходов для описания параллельных процессов.

По мнению автора, тот факт, что этот язык не используется для указанной цели, объясняется предубеждением, состоящим в том, что графы переходов не позволяют описывать параллельные процессы. Последнее обстоятельство, по-видимому, связано с определением абстрактного автомата (и соответствующего ему графа переходов), для которого характерно, что его входные (выходные) символы поступают (формируются) последовательно, а для состояний запрещен «параллелизм».

Однако уже в структурном автомате (и в соответствующем графе переходов) допускается прием параллельных входных и формирование параллельных выходных сигналов. При этом смена состояний, так же как и в абстрактном автомате, осуществляется последовательно. Однако даже эта модель, пусть и недостаточно эффективно, позволяет описывать параллельные процессы. Доказательством этого служит тот факт, что если, например, заданы два графа переходов, не имеющих общих переменных, то по ним формально всегда может быть построен единый граф, эквивалентный заданным, который последователен по состояниям, но параллелен по входам и выходам. Таким образом, единый граф переходов может реализовать параллельные процессы.

Эффективность реализации параллельных процессов (сокращение числа вершин в графах) повышается при переходе от единого ГП к СВГП.

При этом наряду с сохранением параллелизма по входам и выходам появляется возможность отражения параллелизма по состояниям.

Условием запуска параллельных процессов является пометка дуг вызываемых графов номером вершины головного графа, из которой вызываемые графы должны запускаться, а условием их окончания — пометка дуги головного графа, являющаяся конъюнкцией номеров конечных вершин всех вызываемых графов.

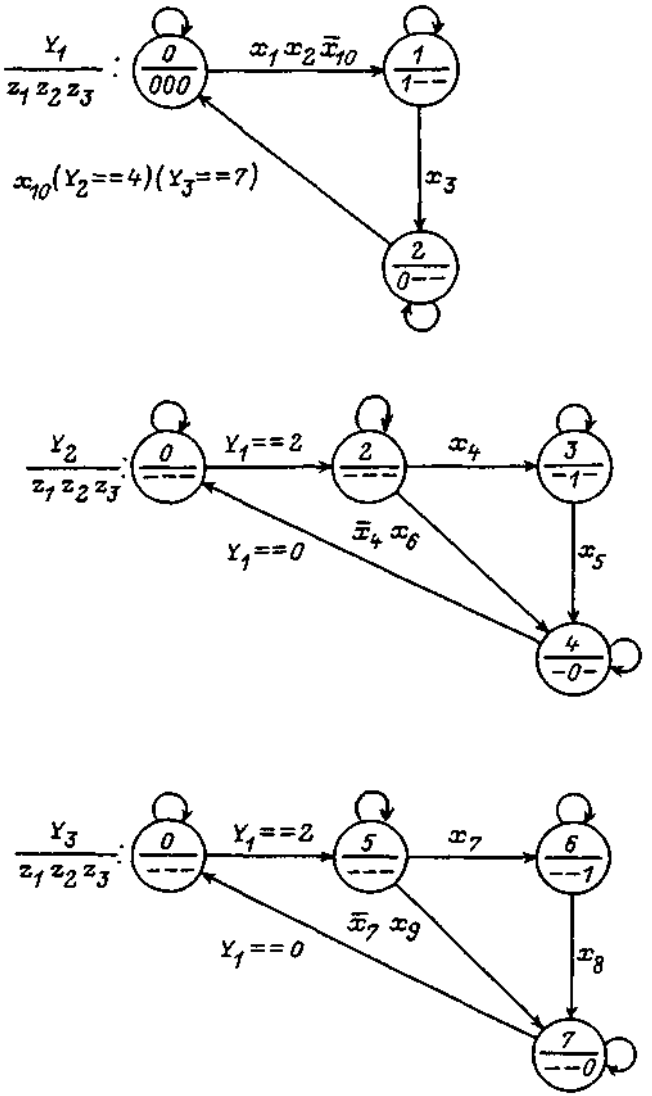


Рис. 12.7

На рис. 12.7 приведена СВГП, эквивалентная диаграмме «Графсет» (рис. 12.6).

Возможность отображения параллельных процессов с помощью системы графов переходов позволяет и в этом случае применять при программировании конструкции `switch` и обеспечить решение проблемы изоморфизма между спецификацией и текстом программы без введения новых «сущностей».

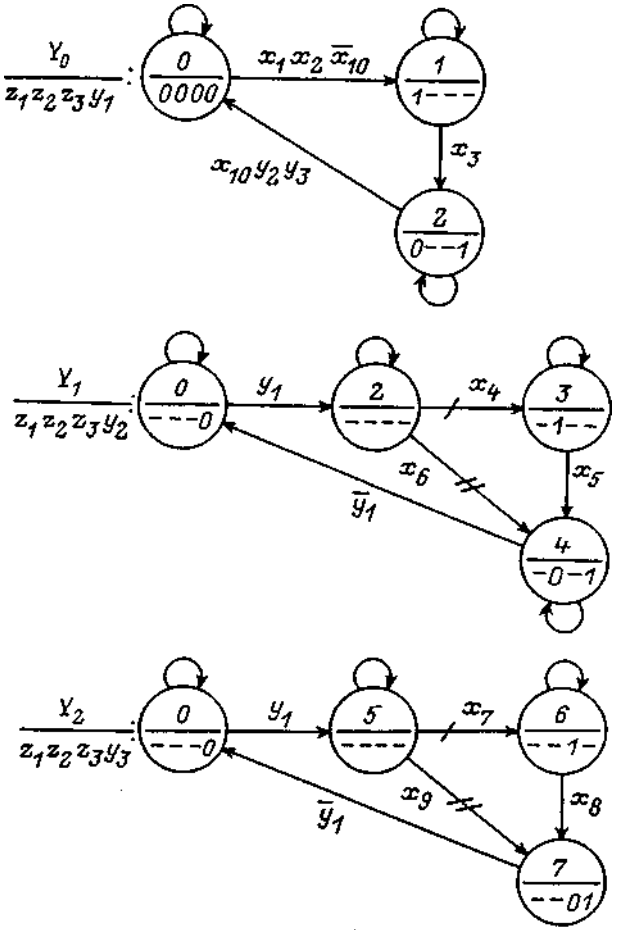


Рис. 12.8

Полученная система ГП может быть реализована программой на языке инструкций ALPro с использованием трех шаговых регистров (трех многозначных переменных Y_0, Y_1 Y_2).

Для сокращения длины программы СВГП (рис. 12.7) должна быть преобразована в СВГП (рис. 12.8), отличающуюся от первоначальной: введением трех битовых переменных — y_1, y_2, y_3, упрощающих в применяемом языке описание обменов между графами переходов системы; использованием приоритетов вместо ортогонализации.

Это позволяет реализовать заданный алгоритм программой на языке инструкций ALPro, состоящей из 53 команд. При этом применяются всего лишь шесть внутренних переменных, которые не требуется переобозначать.

Изложенный подход можно рассматривать также и как решение задачи декомпозиции диаграмм «Графсет» на составляющие, описанные, правда, в других терминах.

Анализ функциональных возможностей полученной системы графов переходов может выполняться с помощью построения графа достижимых маркировок, используемого при необходимости в качестве сертификационного теста.

Если рассматриваемые процессы не имеют общих переменных, то они могут проверяться независимо, что резко снижает трудоемкость проверки. Однако такая независимая проверка возможна лишь в том случае, когда имеется полная уверенность в правильности реализуемого алгоритма в части независимости процессов.

Пусть, например, требуется обеспечить возможность независимого открытия и закрытия клапанов продувания и вентиляции цистерны и независимая проверка на объекте позволила убедиться в этом. Если в графе достижимых маркировок (тесте) отсутствует проверка их совместного открытия, то при испытаниях не будет выявлен тот факт, что совместное открытие клапанов запрещено, и система будет принята для эксплуатации со скрытой ошибкой.

Таким образом, вопрос о снижении размерности графа достижимых маркировок может быть положительно решен только при полной информации об объекте управления.

В заключение раздела отметим, что кроме рассмотренного варианта в ряде случаев бывает целесообразно использовать и другие методы реализации параллельных процессов, базирующиеся, например, на формульной записи алгоритмов.

Применение многоразрядных (битовых) логических операций (многоразрядность ЭВМ) позволяет одновременно (параллельно) вычислять:

— одну булеву формулу на всех входных наборах, размещая все значения каждой переменной в одном слове. Например, формула $y = (x_1 \vee x_2)\ x_3$ может быть вычислена одновременно на всех восьми наборах с помощью двух многоразрядных логических операций при следующем задании значений входных переменных: x_1 — 00001111, x_2 — 00110011, x_3 — 01010101;

— однотипные булевы формулы (формулы одной структуры) на любом входном наборе, размещая соответствующие переменные разных формул в одном слове. Например, система формул $y_1 = (x_1 \vee x_2)\ x_3$ и $y_2 = (\ x_4 \vee x_5\ )\ x_6$ может быть вычислена на любом входном наборе с помощью двух многоразрядных логических операций при следующем размещении переменных по словам: x_1x_4; x_2x_5; x_3x_6.

Параллельные вычисления различных булевых формул на любом входном наборе могут осуществляться по арифметическому полиному, эквивалентному этим формулам [82]. Например, система булевых формул

$$y_1 = x_1x_2; \quad y_2 = x_1 \vee x_2; \quad y_3 = x_1 \oplus x_2$$

может быть вычислена оператором $\mathrm{bin}_3\ (3x_1 + 3x_2)$, где $\mathrm{bin}_3\ a$ — двоичное трехразрядное представление десятичного числа a.

В заключение раздела отметим, что существуют микроконтроллеры, системы команд которых допускают логические операции между битами одного слова [136], что упрощает последовательное вычисление булевых формул.

12.4. Простые и расширенные диаграммы «Графсет». Вызываемые, расширенные, вложенные и иерархические автоматы

Рассмотренные в предыдущих разделах диаграммы «Графсет» могут быть названы простыми, так как в них в прямоугольниках, связанных с этапами, выходным переменным присваиваются только константы. Такие диаграммы позволяют изоморфно реализовать графы переходов автоматов Мура. Однако вопрос о реализации этими диаграммами графов переходов автоматов других классов остается открытым.

Ниже на примерах показано, что реализация таких графов переходов требует расширения языка «Графсет», допустимого стандартом IEC 1131—3 [260]. При этом вводятся расширенные диаграммы «Графсет», в которых в прямоугольниках кроме присваиваний выходным переменным констант допустимы также присваивания более сложных выражений.

Рассмотрение расширенных диаграмм без параллелизма позволяет ввести новый класс автоматов — «расширенные автоматы, являющиеся частным случаем более широкого класса автоматов — вложенных автоматов, которые в свою очередь являются подклассом еще более широкого класса — иерархических автоматов.

В первых трех из рассматриваемых ниже примеров предполагается, что имеются три входных устройства (кнопки без памяти x_1, x_2, x_3) и выходное устройство z, воздействующее на объект управления, называемый в дальнейшем «замок».

Пример 12.1. Построить управляющее устройство, которое осуществляет открытие «замка» только при нажатии кнопок в следующей последовательности: $x_1 \to x_3 \to x_2$, а его закрытие — при отпускании кнопки, нажатой последней.

Рассмотрим, как ограничения на дисциплину подачи входных переменных влияют на сложность автоматов, реализующих заданное описание.

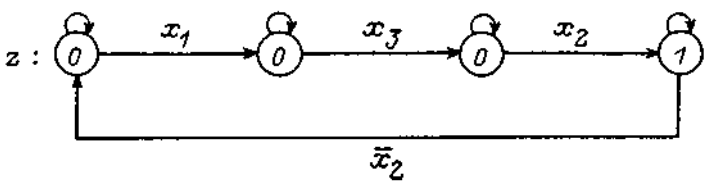


Рис. 12.9

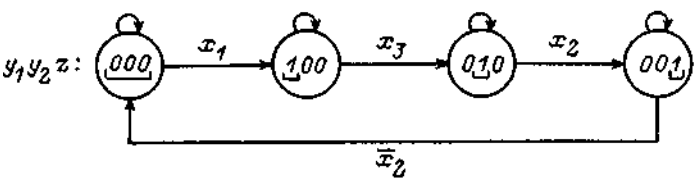


Рис. 12.10

1. Кнопки не могут быть нажаты параллельно и невозможно существование параллельно нажатых кнопок. Это ограничение имеет место, например, в случае, когда кнопки расположены весьма далеко друг от друга.

Построим по приведенному описанию кольцевой граф переходов автомата без выходного преобразователя с принудительным кодированием состояний (рис. 12.9). Этот граф не может быть реализован, так как первые три его вершины помечены одинаково.

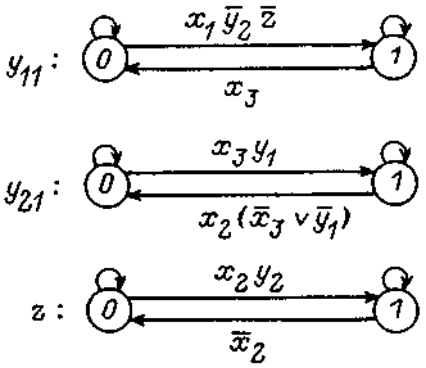


Рис. 12.11

Для обеспечения реализуемости графа используем принудительно-свободное кодирование, выбрав в качестве примера разметку вершин, приведенную на рис. 12.10.

Для построения системы булевых формул, описывающей этот граф, выберем фрагменты пометок, различающие вершины. Указанные фрагменты на рис. 12.10 отмечены скобками. При этом

$$y_{11} = x_1 \,\&\, \bar{y}_1 \,\&\, \bar{y}_2 \,\&\, \bar{z} \vee \bar{x}_3 \,\&\, y_1;$$
$$y_{21} = x_3 \,\&\, y_1 \vee \bar{x}_2 \,\&\, y_2;$$
$$z = x_2 \,\&\, (y_2 \vee z).$$

Построенная модель в свою очередь может быть реализована системой взаимосвязанных графов переходов (рис. 12.11).

Если последовательный характер приведенного описания в одном графе переходов (рис. 12.10) отражается весьма наглядно, то для системы графов, располагаемых в пространстве параллельно (рис. 12.11), последовательный характер описания определяется только наличием в пометках их дуг символов переменных состояний других графов, что весьма ненаглядно.

Если говорить об изоморфной реализации спецификации, задаваемой одной компонентой, с помощью программы, например на языке СИ, то в рамках предлагаемой технологии наиболее целесообразно применять граф переходов автомата Мура с многозначным кодированием состояний (рис. 12.12).

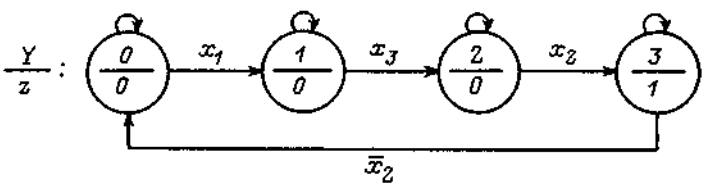


Рис. 12.12

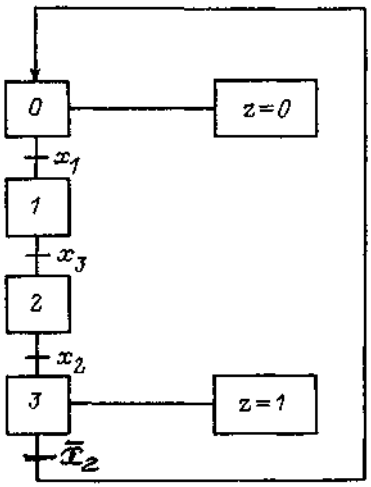


Рис. 12.13

Построим по этому графу переходов последовательную простую диаграмму «Графсет», в которой неизменяющиеся значения выходной переменной z умалчиваются (рис. 12.13).

Обратим внимание на тот факт, что так как в этой диаграмме нет параллельных ветвей, то для различения вершин в ней, так же как и в графе переходов, может быть использовано многозначное кодирование, что в [59] даже не обсуждалось.

2. Кнопки не могут быть нажаты параллельно, но допустимо существование параллельно нажатых кнопок. Это возможно в тех случаях, когда кнопки нажимаются последовательно, но ранее нажатые кнопки могут не отпускаться.

Менее строгая дисциплина подачи входных воздействий по сравнению с предыдущим случаем приводит к усложнению автомата, реализующего заданное описание. При этом в кольцевом графе переходов (рис. 12.12) и (или) последовательной простой диаграмме «Графсет» (рис. 12.13) должны быть выполнены следующие изменения: x_1 заменяется на $x_1 \,\&\, \bar{x}_2 \,\&\, \bar{x}_3$; x_3 — на $\bar{x}_2 \,\&\, x_3$.

3. Кнопки могут быть нажаты параллельно и могут оставаться параллельно нажатыми.

Требования к дисциплине подачи входных воздействий еще более снизились, и поэтому, для того чтобы автомат реагировал только на заданную входную последовательность, он должен быть еще более сложным. При этом в графе (рис. 12.12) и в диаграмме (рис. 12.13) должны быть выполнены следующие изменения: x_1 заменяется на $x_1 \,\&\, \bar{x}_2 \,\&\, \bar{x}_3$; x_3 — на $\bar{x}_1 \,\&\, \bar{x}_2 \,\&\, x_3$; x_2 — на $\bar{x}_1 \,\&\, x_2 \,\&\, \bar{x}_3$.

Пример 12.2. Построить управляющее устройство, которое осуществляет открытие «замка» при последовательном нажатии кнопок в любом порядке, а его закрытие — в случае, когда все три кнопки отпущены.

Ограничение на дисциплину подачи входных переменных: кнопки не могут быть нажаты параллельно и невозможно существование параллельно нажатых кнопок после их последовательного нажатия.

Для решения сформулированной задачи построим ветвящийся граф переходов автомата Мура с многозначным кодированием состояний (рис. 12.14). Ветвящаяся простая диаграмма «Графсет», изоморфная этому графу, приведена на рис. 12.15. В этих моделях в каждый момент времени активной может быть только одна вершина, т. е. в них отсутствуют параллельные процессы.

Пример 12.3. Построить управляющее устройство, осуществляющее открытие «замка» при последовательном нажатии кнопок в любом поряд-

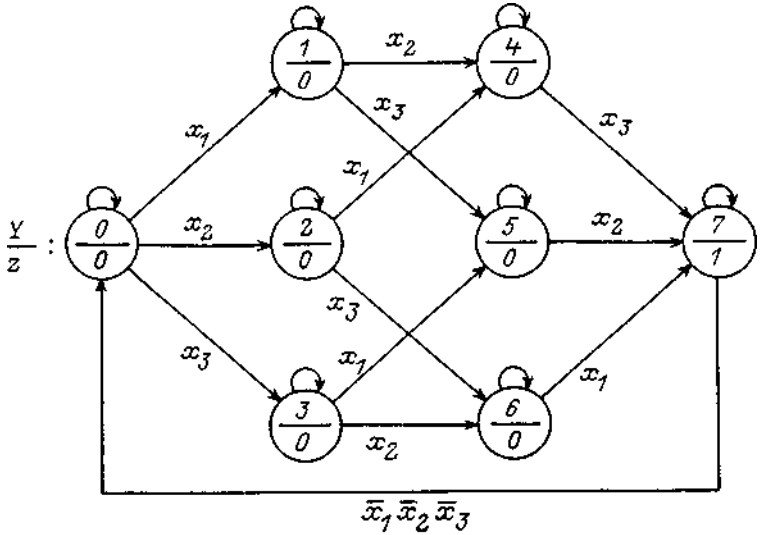


Рис. 12.14

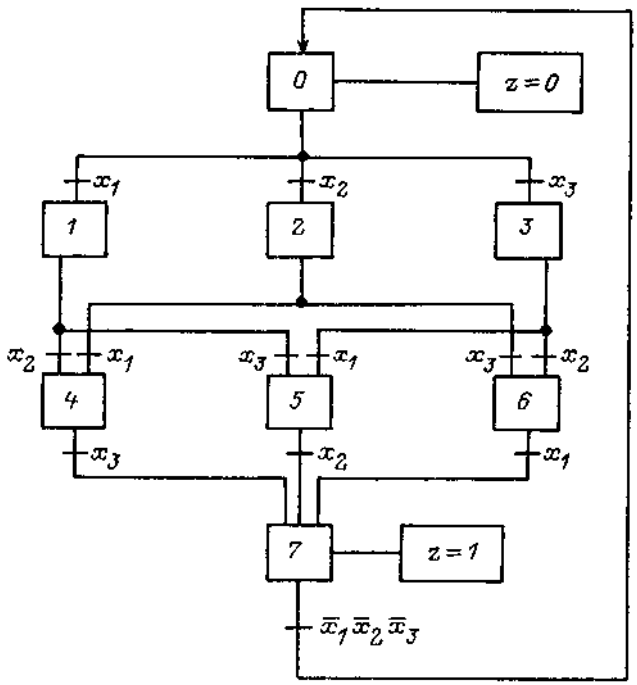


Рис. 12.15

ке и при параллельном их нажатии в любом сочетании, а его закрытие — в случае, когда все три кнопки отпущены.

Реализуем заданное описание с помощью различного числа автоматов.

1. Один автомат.

Если по заданному описанию построить планарный граф переходов автомата Мура с многозначным кодированием состояний, то он будет содержать 14 вершин, 14 петель и 26 дуг. Если в этом графе объединить эквивалентные вершины, то получится непланарный граф с 8 вершинами, 8 петлями и 18 дугами (рис. 12.16).

Приведенная реализация громоздка и недостаточно наглядно отражает возможный параллелизм процессов, имеющийся в задании.

2. Три автомата и одна булева формула.

На рис. 12.17 приведены три графа переходов, каждый из которых соответствует автомату без выходного преобразователя, и булева формула, образованная из символов, присвоенных состояниям автоматов.

Из рассмотрения этих графов следует, что в данном случае имеют место параллельные процессы, инициируемые тремя событиями — нажатием кнопок x_1, x_2 и x_3. Но так как формирование единичного значения выходной переменной z «не привязано» к какому-либо одному состоянию, то в данном случае непосредственное построение диаграммы «Графсет»

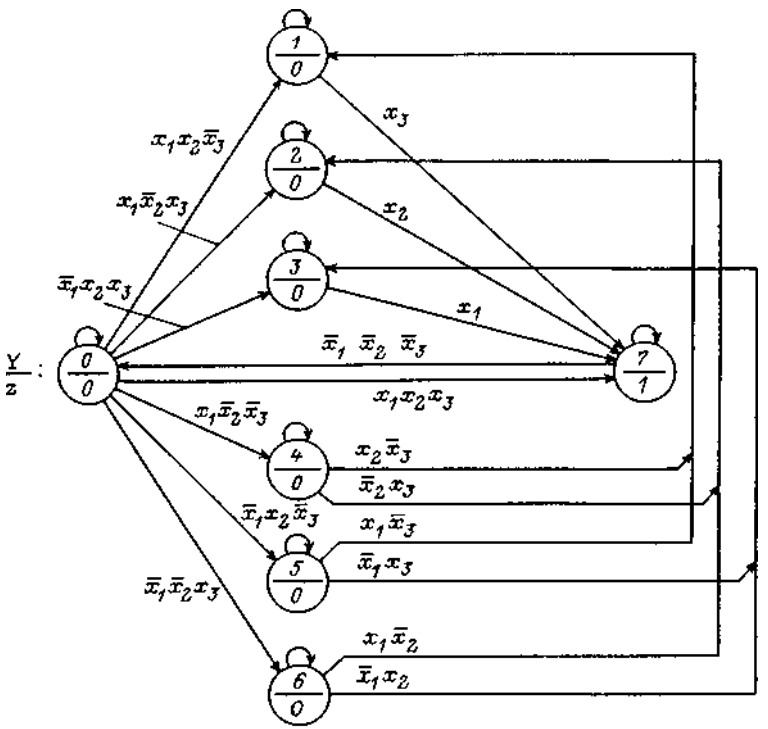


Рис. 12.16

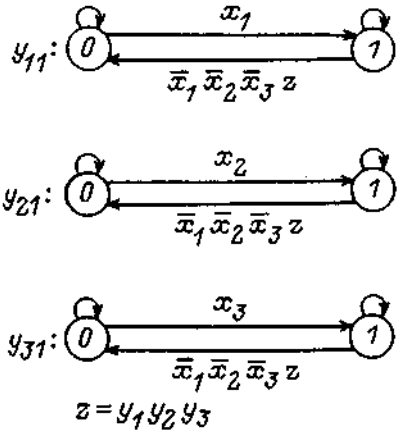


Рис. 12.17

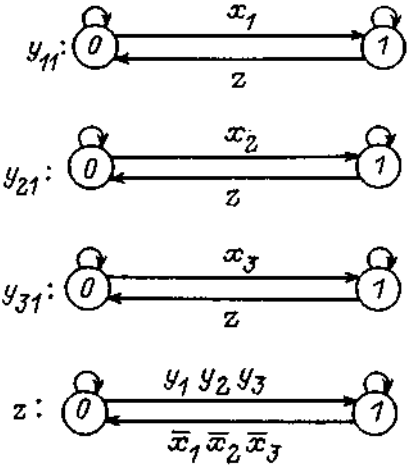


Рис. 12.18

по приведенной модели невозможно: система взаимосвязанных автоматов обладает более широкими изобразительными возможностями по сравнению с диаграммами «Графсет».

3. Четыре автомата.

Реализуем в данном случае автоматом без выходного преобразователя не только функции формирования двоичных внутренних переменных y_{11}, y_{21}, y_{31}, но и функцию выходов z (рис. 12.18).

Построим на базе приведенной системы взаимосвязанных графов переходов простую диаграмму «Графсет», обладающую параллельно-последовательной структурой (рис. 12.19).

Параллельный фрагмент этой структуры может обеспечить одновременную активность более чем одной вершины, что невозможно в одном графе переходов, а ее последовательный фрагмент — активность только одной вершины.

В приведенной диаграмме из-за параллелизма процессов не может быть использовано многозначное кодирование, а применяется двоичное кодирование этапов. Обратим внимание на тот факт, что если в системе графов переходов (рис. 12.18) используются только три внутренние двоичные переменные — y_{11}, y_{21}, y_{31}, то в диаграмме «Графсет» таких переменных восемь — Y_0, Y_1,..., Y_1

Поэтому и число формул, обеспечивающих установку «единиц» и их поддержание для всех внутренних переменных, равно трем и восьми соответственно.

Интересным является тот факт, что если в системе графов две вершины одного графа кодируются одной двоичной внутренней переменной (рис. 12.18), то в диаграмме «Графсет» из-за необходимости отражения параллелизма в одной компоненте ее фрагмент, в некотором смысле эквивалентный графу переходов с двумя вершинами, приходится кодировать двумя такими переменными (рис. 12.19).

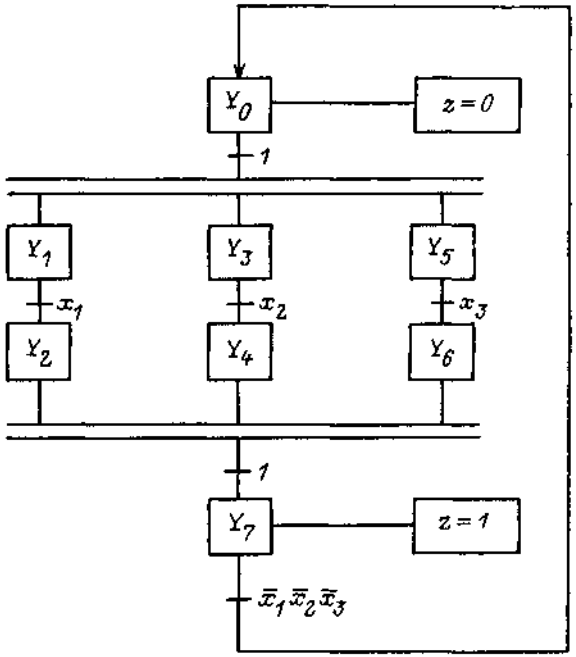


Рис. 12.19

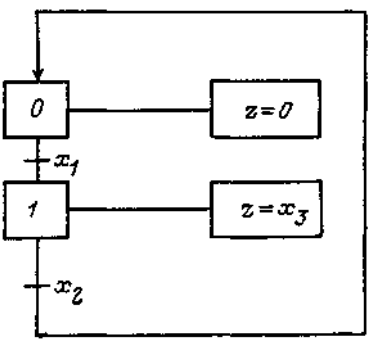


Рис. 12.20

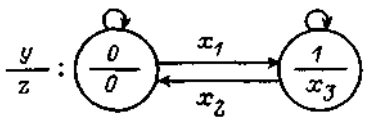


Рис. 12.21

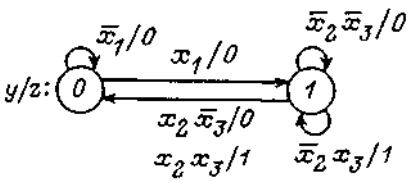


Рис. 12.22

Из-за необходимости переобозначения внутренних переменных разница в их числе при переходе от моделей к программным реализациям еще более увеличивается.

Заметим также и то, что заданное описание может быть реализовано с помощью только одной внутренней многозначной переменной для одного графа переходов (рис. 12.16) или одной простой диаграммы «Графсет» без параллелизма, изоморфно построенной по этому графу переходов, которые, однако, в данном случае весьма ненаглядны.

Из изложенного следует, что более наглядное отображение параллельно-последовательного процесса с помощью диаграммы «Графсет» (рис. 12.19) по сравнению с его отображением с помощью графов переходов (рис. 12.16—12.18) приводит к весьма ненаглядной реализации, требующей к тому же больших внутренних ресурсов используемого управляющего вычислительного устройства. Если таким устройством является ПЛК, то последнее замечание становится чрезвычайно существенным.

Процессы, каждый из которых является чисто последовательным, протекающие чисто параллельно (без использования каких-либо общих переменных), изображены в виде двух компонент на рис. 4.147.

Некоторые версии языка «Графсет» допускают в прямоугольниках запись не только присваиваний выходным переменным констант 0 и 1, но и выражений, в том числе булевых. Будем называть эти версии языка и соответственно диаграммы расширенными.

Определим, с одной стороны, каким типам автоматов соответствуют такие диаграммы без параллелизма, а с другой — как по графу переходов автомата, отличного от автомата Мура, построить расширенную диаграмму.

Эти вопросы в документации по программированию контроллеров даже не упоминаются, однако, как будет показано ниже, их решение может существенно упростить программную реализацию.

Пример 12.4. На рис. 12.20 приведена последовательная расширенная диаграмма «Графсет». Построить графы переходов автоматов, которые эквивалентны этой диаграмме.

Из рассмотрения заданной диаграммы следует, что можно ввести новый класс автоматов, которые назовем расширенными автоматами. Для таких автоматов характерно, что в графах переходов, описывающих их поведение, в каждой вершине наряду с кодом состояния через дробь с ним для каждой выходной переменной в общем случае указывается также и булево выражение, которое может быть и константой.

На рис. 12.21 приведен граф переходов расширенного автомата, изоморфный расширенной диаграмме (рис. 12.20). Ниже будет показано, что расширенным автоматам могут соответствовать классические автоматы разных типов. Покажем, что в данном случае расширенные диаграмма и автомат эквивалентны автомату Мили.

Построим предварительно систему булевых формул, описывающую эти модели. Для этого воспользуемся правилами построения формул, применяемыми для автоматов Мура:

$$y_1 = x_1 \,\&\, \bar{y} \vee \bar{x}_2 \,\&\, y;$$
$$z = 0 \,\&\, \bar{y} \vee x_3 \,\&\, y = x_3 \,\&\, y. \qquad (12.1)$$

Получим по этой системе кодированную таблицу переходов и выходов (таблицу истинности), от которой перейдем к графу переходов автомата Мили (рис. 12.22).

Если указанные преобразования выполнить в обратном порядке, то по графу переходов автомата Мили могут быть построены расширенные автомат и диаграмма.

Этот переход является весьма целесообразным, так как позволяет упростить программную реализацию практически на любом языке программирования.

Так, при использовании языка профаммирования СИ вместо программы

```
        switch (y) {
case 0: if (x̄1)          z = 0;
        if (x1)          {z = 0; y = 1;}
        break;
case 1: if (x̄2 & x̄3)      z = 0;
        if (x̄2 & x3)      z = 1;
        if (x2 & x̄3)     {z = 0; y = 0;}
        if (x2 & x3)     {z = 1; y = 0;}
        break;},
```

изоморфной графу переходов автомата Мили (рис. 12.22), можно построить существенно более простую профамму по эквивалентному графу переходов расширенного автомата (рис. 12.21):

```
        switch (y) {
case 0: z = 0;
        if (x1)          y = 1;
        break;
case 1: z = x3;
        if (x2)          y = 0;
        break;}.
```

Эта реализация определяет правило чтения графов переходов расширенных автоматов: «Когда автомат находится в устойчивой вершине, формируется значение выхода, вычисляемое в этой вершине; при переходе автомата в смежную вершину формируются номер „новой" вершины и значение выхода, вычисленное в „старой" вершине».

Обратим внимание также и на тот факт, что написание программы непосредственно по системе булевых формул (12.1), дополненной соотношением $y = y_1$, приводит к еще более простой реализации, которая эквивалентна, но однако не изоморфна приведенным графам переходов.

Возвращаясь к рассмотрению графа переходов расширенного автомата (рис. 12.21), можно утверждать, что он является многофункциональным логическим модулем, настраиваемым кратковременными потенциальными переменными. Для обеспечения возможности реализации любой одной из двух ($F = 2$) булевых формул ($z = 0$; $z = x_3$) этот ГП имеет два ($s = 2$) состояния и две ($N = 2$) настроечные переменные — x_1 и x_2.

В общем случае модуль, который при кратковременной настройке может длительное время (до смены настройки) реализовывать любую из F булевых формул и переходить после этого к реализации любой другой из этих формул, должен быть расширенным автоматом, граф переходов которого является полным и содержит $s = F$ вершин и $N = F$ настроечных переменных, помечающих дуги графа.

При этом обратим внимание на тот факт, что многофункциональные логические модули, настраиваемые длительной подачей переменных [39],

которые позволяют реализовать любую из F булевых формул и непосредственно переходить к реализации любой другой из этих формул, являются автоматами без памяти с $N =]\log_2 F[$ настроечными переменными.

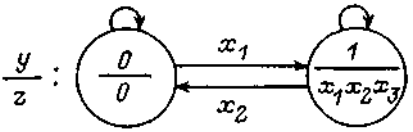


Рис. 12.23

Число настроечных переменных можно уменьшить до одной, если:

— фиксировать порядок реализации формул;

— разрешить ($N = F$)-кратную подачу (снятие) единичного значения настроечной переменной;

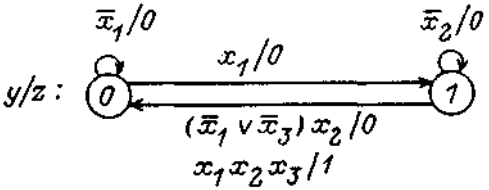


Рис. 12.24

— использовать кольцевой граф переходов расширенного автомата с $s = 2F$ вершинами, в котором в двух соседних вершинах, связанных между собой дугой, помеченной инверсией настроечной переменной, реализуется одна и та же формула.

В предыдущем примере переменные, помечающие переходы и вершины в графе, были различными. Рассмотрим пример, в котором эти переменные хотя бы частично совпадают.

Пример 12.5. По графу переходов расширенного автомата (рис. 12.23) построить граф переходов классического автомата.

Записав по заданному графу систему булевых формул и построив по ней кодированную таблицу переходов и выходов, перейдем к графу переходов, который соответствует автомату Мили (рис. 12.24).

Из рассмотрения этого графа следует, что расширенный автомат в данном случае не может применяться в качестве многофункционального логического модуля в традиционном его понимании, так как он реализует одну из заданных формул ($z = x_1 \,\&\, x_2 \,\&\, x_3$) лишь кратковременно, при переходе из первой вершины в нулевую.

Пример 12.6. По графу переходов автомата Мили последовательного сумматора (рис. 4.113) построить граф переходов эквивалентного ему расширенного автомата.

По заданному графу переходов автомата Мили построим систему булевых формул, в которой обе формулы записаны в форме разложения Шеннона по переменной переноса p:

$$p_1 = x_1 \,\&\, x_2 \,\&\, \bar{p} \vee (x_1 \vee x_2) \,\&\, p;$$
$$s = (x_1 \oplus x_2) \,\&\, \bar{p} \vee (\bar{x}_1 \oplus x_2) \,\&\, p.$$

Непосредственно по этим формулам может быть построен искомый граф переходов (рис. 12.25), который более компактен, но менее понятен по сравнению с исходным графом. Поэтому если исходный граф переходов целесообразно использовать в качестве спецификации решаемой

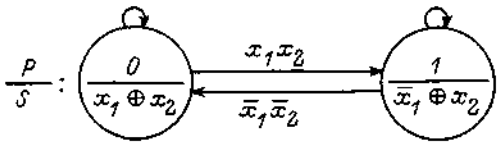


Рис. 12.25

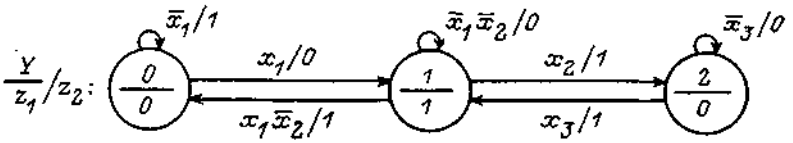


Рис. 12.26

задачи, то полученный граф может применяться в качестве спецификации на программирование.

Построенный граф реализуется конструкцией switch:

```
        switch (p)  {
case 0: s = x1 ⊕ x2;
        if (x1 & x2) p = 1;
        break;
case 1: s = x̄1 ⊕ x2;
        if (x̄1 & x̄2) p = 0;
        break;       },
```

существенно более эффективно по сравнению с такой же реализацией исходного автомата Мили (разд. 5.1.3).

Реализация по построенной системе булевых формул, дополненной соотношением $p = p_1$, даже без проведения минимизации еще более эффективна, но и еще более ненаглядна.

Использование расширенного автомата обеспечивает компромисс между понятностью модели и сложностью ее реализации.

Пример 12.7. По графу переходов смешанного автомата (рис. 12.26) построить граф переходов эквивалентного ему расширенного автомата.

Для выходной переменной z_2 определим условия формирования ее единичного значения:

$$z_2 = x_1 \& (Y==0) \vee x_1 \& x_2 \& (Y==1) \vee x_2 \& (Y==1) \vee x_3 \& (Y==2) =$$
$$= x_1 \& (Y==0) \vee (x_1 \vee x_2) \& (Y==1) \vee x_3 \& (Y==2).$$

Сохраняя структуру автомата Мура по переменной z_1 и учитывая найденные условия формирования переменной z_2, построим искомый граф переходов (рис. 12.27).

Пример 12.8. По графу переходов автомата Мура (рис. 12.28) построить граф переходов расширенного автомата.

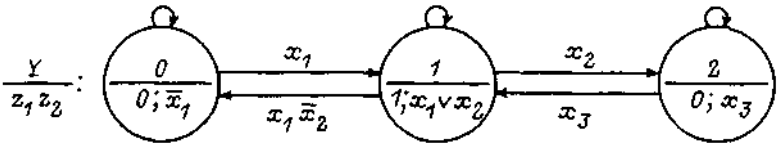


Рис. 12.27

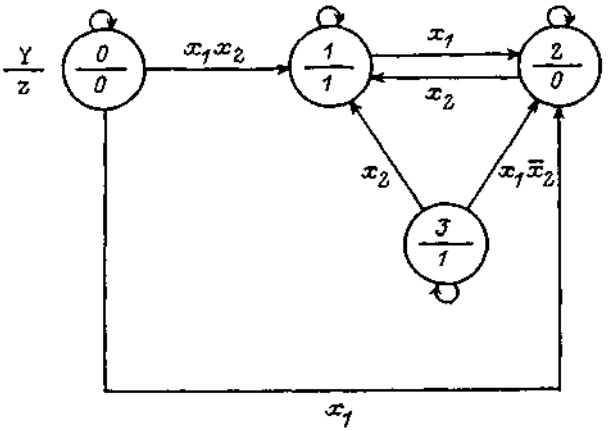


Рис. 12.28

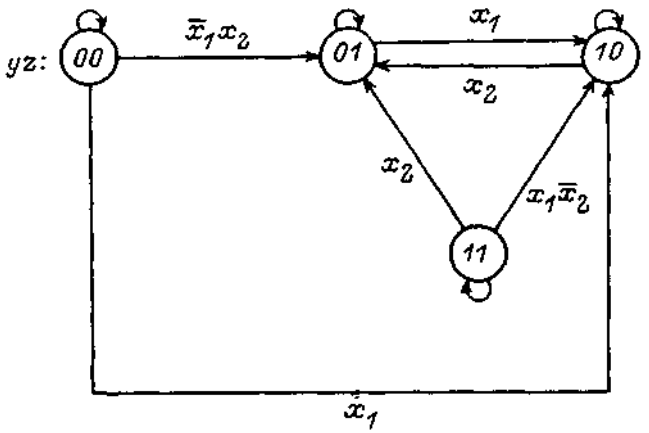


Рис. 12.29

Преобразуем заданный граф переходов в граф переходов автомата без выходного преобразователя с принудительно-свободным кодированием состояний. В данном случае достаточно использовать только одну внутреннюю переменную y (рис. 12.29).

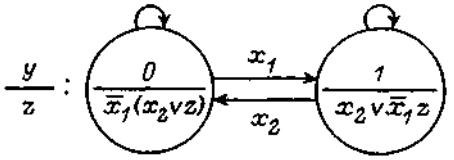


Рис. 12.30

По этому графу переходов запишем систему булевых формул, каждую их которых представим в форме разложения Шеннона по внутренней переменной y:

$$y_1 = x_1 \,\&\, \bar{y} \vee \bar{x}_2 \,\&\, y;$$
$$z = \bar{x}_1 \,\&\, (x_2 \vee z) \,\&\, \bar{y} \vee (x_2 \vee \bar{x}_1 \,\&\, z) \,\&\, y.$$

Непосредственно по этой системе формул может быть построен искомый граф переходов (рис. 12.30).

Из рассмотренного примера следует, что:

— существуют автоматы Мура и автоматы без выходного преобразователя, для которых при переходе к расширенным автоматам число состояний сокращается;

— существуют автоматы Мура, которые при реализации расширенной диаграммой «Графсет» содержат меньшее число этапов по сравнению с изоморфной заданному автомату простой диаграммой «Графсет»;

— существуют автоматы Мура, которые имеют более сложную программную реализацию по сравнению с реализацией эквивалентных им расширенных автоматов.

Программа, изоморфная графу переходов автомата Мура (рис. 12.28), имеет вид:

```
          switch (Y) {
case 0:   z = 0;
          if (~x1 & x2)      Y = 1;
          if (x1)            Y = 2;
          break;
case 1:   z = 1;
          if (x1)            Y = 2;
          break;
case 2:   z = 0;
          if (x2)            Y = 1;
          break;
case 3:   z = 1;
          if (x2)            Y = 1;
          if (x1 & ~x2)      Y = 2;
          break;},
```

в то время как программа, построенная по графу переходов эквивалентного расширенного автомата (рис. 12.30), существенно проще:

```
        switch (y) {
case 0: z = x̄1 & (x2 ∨ z);
        if (x1)        y = 1;
        break;
case 1: z = x2 ∨ x̄1 & z;
        if (x2)        y = 0;
        break;       }.
```

Рассмотренный подкласс расширенных автоматов по форме изображения может быть назван расширенными автоматами Мура. Естественно, что могут существовать также и другие подклассы этого класса, например расширенные автоматы Мили и расширенные смешанные автоматы.

Пример 12.9. По графу переходов автомата Мили (рис. 12.31) построить эквивалентный ему граф переходов расширенного автомата Мили.

Заполним по заданному графу кодированную таблицу переходов и выходов, а по ней запишем систему булевых формул, каждая из которых представлена в форме разложения Шеннона по внутренней переменной *y:*

$$y_1 = x_1 \,\&\, \bar{y} \vee \bar{x}_2 \,\&\, y;$$

$$z = x_1 \,\&\, x_3 \,\&\, x_4 \,\&\, \bar{y} \vee x_2 \,\&\, (x_3 \vee x_4) \,\&\, y.$$

По этой системе может быть построен граф переходов расширенного автомата Мура с двумя вершинами, в котором нулевая вершина помечена выражением $x_1 \,\&\, x_3 \,\&\, x_4$, а первая — выражением $x_2 \,\&\, (x_3 \vee x_4)$. Построенная система может быть реализована также и графом переходов расширенного автомата Мили (рис. 12.32). Построенный граф менее понятен, чем исходный, но имеет более простую программную реализацию.

Графы переходов расширенных автоматов могут быть изображены и в другой форме. Применение этой формы позволяет рассматривать расширенные автоматы как частный случай более широкого класса автоматов — вложенных автоматов. При этом если расширенным автоматам соответствуютдвухуровневые вложенные автоматы -, то сами вложенные автоматы могут иметь любое число уровней вложений.

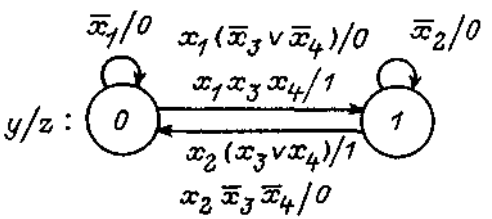


Рис. 12.31

Так, например, граф переходов расширенного автомата Мура (рис. 12.30) может быть изображен в виде двухуровневой структуры, образованной тремя графами переходов, из которых верхний соответствует *JK*-триггеру, левый нижний — *R*-триггеру, а правый нижний — *S*-триггеру (рис. 12.33).

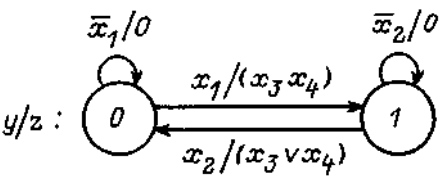


Рис. 12.32

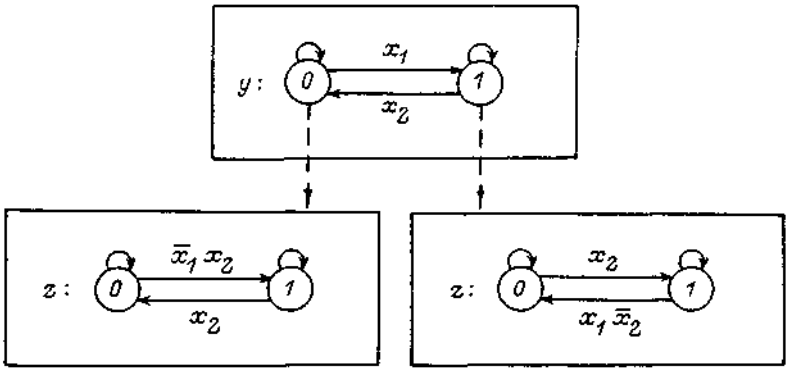


Рис. 12.33

Если для вложенных автоматов Мура пунктирные связи исходят из соответствующих вершин, то для вложенных автоматов Мили они исходят от соответствующих дуг, в то время как для вложенных смешанных автоматов — из вершин и дуг одновременно.

При этом если для вложенных автоматов Мура граф нижнего уровня может выполняться весьма долго (пока «верхний» граф переходов находится в соответствующей вершине), то для вложенных автоматов Мили в «нижнем» графе не может быть выполнено более одного перехода при каждом обращении к нему.

Использование моделей вложенных автоматов приводит к моделям программной реализации, основанным на применении вложенных конструкций switch. При этом все автоматы, входящие в каждую такую модель, рассматриваются как единое целое и являются одной программной компонентой.

Приведем программу, изоморфно реализующую двухуровневую структуру, представленную на рис. 12.33:

```
                switch (y) {
case 0: {       switch (z)  {
        case 0: if (~x1 & x2) z = 1;
                break;
        case 1: if (x2)       z = 0;
                break;        }
        }
                if (x1)       y = 1;
                break;
case 1: {       switch (z)  {
        case 0: if (x2)       z = 1;
                break;
        case 1: if (x1 & ~x2) z = 0;
                break;        }
        }
                if (x2)       y = 0;
                break;     }.
```

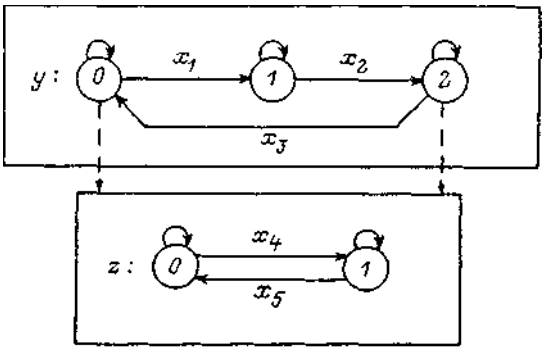


Рис. 12.34

Применительно к вложенным автоматам Мура один и тот же граф может быть связан с различными вершинами как одного графа более высокого уровня, так и с вершинами разных графов, в том числе и принадлежащих разным уровням иерархии. При этом вкладываемый граф в модели может либо повторяться столько раз, во сколько вершин он вложен, либо существовать только в одном экземпляре, но выполняться во всех указанных вершинах.

В первом случае реализация может выполняться только с помощью вложенных конструкций `switch`, а во втором — должны применяться также и процедуры.

На рис. 12.34 в качестве примера приведены вложенные автоматы, нижний из которых существует только в одном экземпляре, но выполняется два раза — в нулевом и втором состояниях автомата верхнего уровня. Приведем профамму, изоморфную этим графам переходов (рис. 12.34):

```
void proc_z();
void main ()
  {     switch (y)  {
case 0: proc_z();
        if (x1)       y = 1;
        break;
case 1: if (x2)       y = 2;
        break;
case 2: proc_z();
        if (x3)       y = 0;
        break;        } }
void proc_z()
  {     switch (z)  {
case 0: if (x4)       z = 1;
        break;
case 1: if (x5)       z = 0;
        break;        } }.
```

Для проверки этой профаммы по графам переходов вложенных автоматов (рис. 12.34) построим граф достижимых маркировок (рис. 12.35), отражающий все функциональные (поведенческие) возмож-

ности рассматриваемой совокупности автоматов при $x_i x_j = 0$, без построения и анализа которого говорить о правильности реализации исходного задания не приходится.

По графу достижимых маркировок построим сертификационные тесты, каждый из которых соответствует одному контуру в графе:

1. $00 \xrightarrow{x_1\bar{x}_4} 10 \xrightarrow{x_2} 20 \xrightarrow{x_3\bar{x}_4} 00;$
2. $00 \xrightarrow{x_1\bar{x}_4} 10 \xrightarrow{x_2} 20 \xrightarrow{\bar{x}_3x_4} 21 \xrightarrow{x_3\bar{x}_5} 20 \xrightarrow{x_3\bar{x}_4} 00;$
3. $00 \xrightarrow{x_1\bar{x}_4} 10 \xrightarrow{x_2} 20 \xrightarrow{\bar{x}_3x_4} 21 \xrightarrow{x_3\bar{x}_5} 01 \xrightarrow{\bar{x}_1x_5} 00;$
4. $00 \xrightarrow{\bar{x}_1x_4} 01 \xrightarrow{\bar{x}_1x_5} 00;$
5. $00 \xrightarrow{\bar{x}_1x_4} 01 \xrightarrow{x_1\bar{x}_5} 11 \xrightarrow{x_2} 21 \xrightarrow{\bar{x}_3x_5} 20 \xrightarrow{x_3\bar{x}_4} 00;$
6. $00 \xrightarrow{\bar{x}_1x_4} 01 \xrightarrow{x_1\bar{x}_5} 11 \xrightarrow{x_2} 21 \xrightarrow{x_3\bar{x}_5} 01 \xrightarrow{\bar{x}_1x_5} 00.$

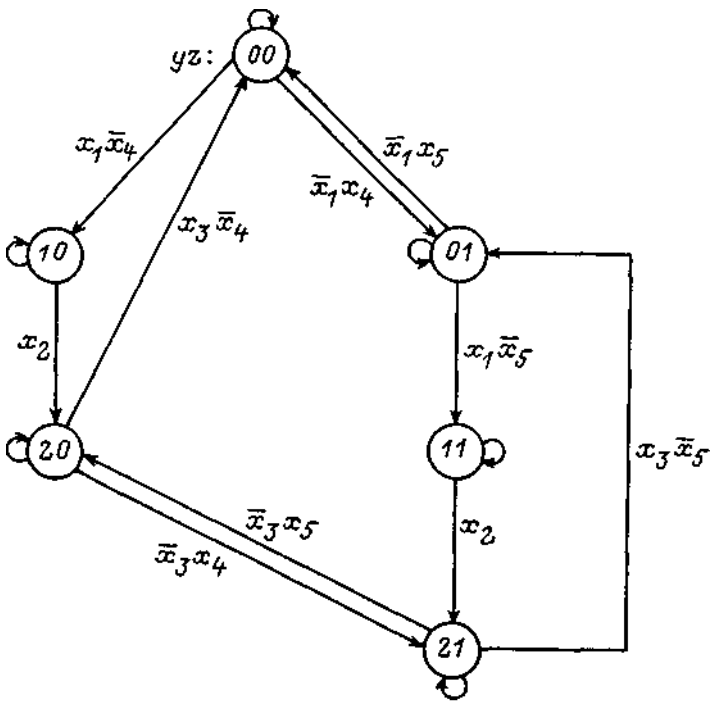


Рис. 12.35

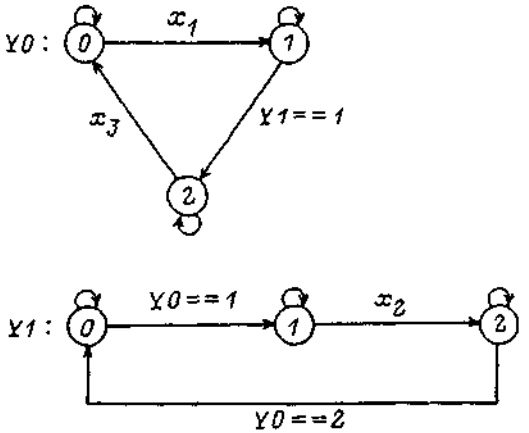


Рис. 12.36

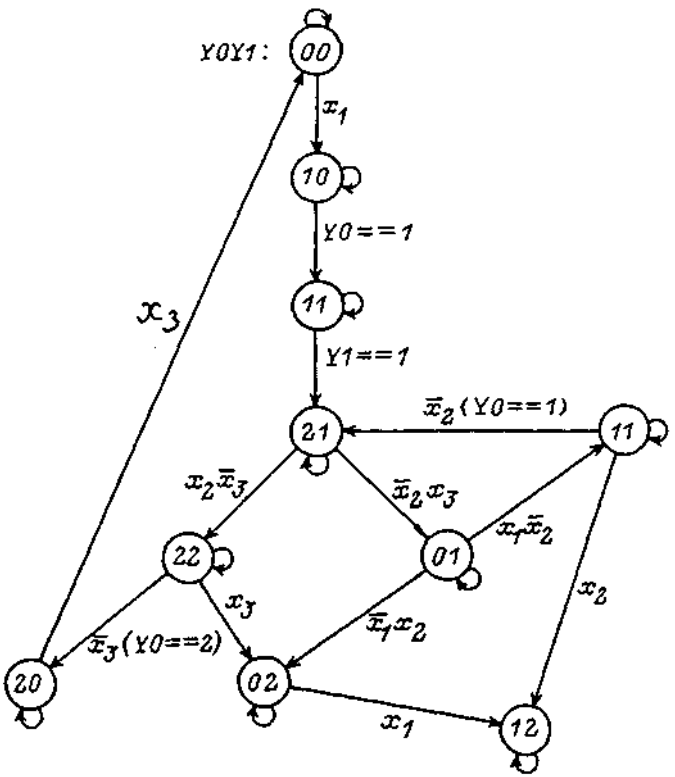


Рис. 12.37

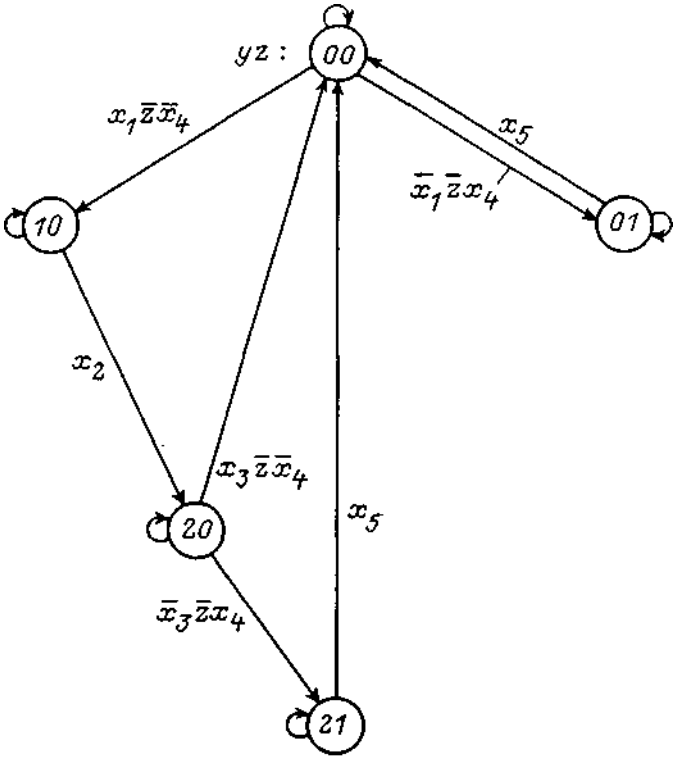


Рис. 12.38

Из сказанного следует, что вложенные автоматы в свою очередь могут рассматриваться как подкласс класса иерархических автоматов.

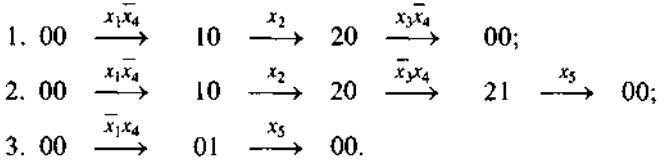
Другим подклассом этого класса являются автоматы, рассмотренные в гл. 8, которые могут быть названы вызываемыми. Они отличаются от вложенных автоматов как механизмом взаимодействия, так и реализацией (не связанные по управлению конструкции switch, обменивающиеся данными — в основном номерами состояний).

Пример вызываемых автоматов приведен на рис. 12.36. Граф достижимых маркировок, определяющий совокупное поведение вызываемых графов Y0 и Y1 при $x_i x_j = 0$, изображен на рис. 12.37.

Из анализа этого графа, в частности, следует, что при Y0 = = 1 и Y1 == 2 имеет место «смертельное объятие» — дедлок [14], соответствующий тупиковой вершине «12» в графе.

В общем случае возможно совместное использование рассмотренных механизмов взаимодействия графов. Так, если в примере, приведенном на рис. 12.34, ввести дополнительное ограничение — начатый процесс, описываемый нижним графом, обязательно должен быть закончен (не может быть прерван), то в верхнем графе должны быть сделаны следующие замены: x_1 на x_1 & $\bar{z}$; x_3 на x_3 & $\bar{z}$

Введенное ограничение существенно упрощает граф достижимых маркировок (рис. 12.38) по сравнению со случаем, когда это ограничение не используется (12.35). При этом уменьшается и число сертификационных тестов для соответствующей программы, которая получается из предыдущей при выполнении указанных выше замен:



Построение ГДМ позволяет ответить на вопрос: что является понятием «состояние» для системы взаимосвязанных автоматов?

Ответ на этот вопрос состоит в следующем: состоянию такой системы в некоторый момент времени соответствует (но необязательно определяет это состояние) кортеж значений многозначных переменных, каждое из которых соответствует номеру состояния одного из автоматов системы в этот момент времени.

Каждому состоянию системы автоматов сопоставляется одна из вершин ГДМ, построенного по СВГП. Поэтому число состояний в системе взаимосвязанных автоматов равно числу вершин в соответствующем ей ГДМ. При этом, так как вершины ГДМ могут быть помечены одинаковыми кортежами значений переменных, число состояний системы автоматов может превышать число различных кортежей, порождаемых СВГП. Наличие одинаковых кортежей в ГДМ может являться необходимым (но недостаточным) условием для минимизации числа состояний в системе автоматов.

Рассмотренные в настоящем разделе модели представляют интерес при построении систем с многоуровневой разрешающей способностью (Multiresolutional Systems) [278, 279] и позволяют улучшить познавание (cognitive) [263] и понимание (perception) человеком процессов, происходящих в сложных системах логического управления.