Глава 13

Применение граф-схем алгоритмов и графов переходов при программной реализации

На практике, видимо в связи с традициями в области вычислительной техники, при программной реализации широкое использование в качестве языка спецификаций получили блок-схемы, называемые также граф-схемами, или просто схемами [97, 98], что закреплено соответствующими стандартами [203, 245]. При этом граф-схемы, предназначенные для описания алгоритмов, были названы граф-схемами алгоритмов (ГСА).

Однако в этих стандартах определяются лишь правила изображения граф-схем и отсутствуют требования (кроме изобразительных) к их построению для обеспечения возможности легкого их понимания (чтения).

Необходимо отметить, что и в научной литературе недостаточно рассматривались свойства граф-схем, затрудняющие их применение в качестве языка общения для указанного класса задач. Так, в [103, 104] были предложены методы построения граф-схем, структурная организация которых улучшает их понимание. При этом авторы этих работ считали, что если граф-схемы построены только из вложенных базовых управляющих конструкций без использования операторов `goto`, то это решает проблему их легкого понимания, и не учитывали ряда других факторов, затрудняющих чтение, и в частности умолчания значений некоторых переменных.

Однако в последнее время специалисты по проектированию программ [226] обратили внимание на тот факт, что только структурное проектирование указанной проблемы не решает, и предложили объектно-ориентированный подход к проектированию программ, в рамках которого были введены понятия «объект» и его «состояние» и рекомендовано применение графов переходов для описания динамики реализуемых процессов. Однако, кроме одного примера использования таких графов, эта работа не содержит теоретического обоснования требований к их построению, обеспечивающих, в частности, простоту понимания программ.

В настоящем разделе разрабатываются требования к построению понимаемых граф-схем и графов переходов и предлагаются методы построения понимаемых ГП по ГСА и понимаемых ГСА по ГП.

13.1. Граф-схемы алгоритмов. Основные проблемы

Будем рассматривать автоматные граф-схемы, в которых в операторных вершинах формируются или сохраняются единичные и нулевые значения переменных. Автоматные граф-схемы разобьем на два класса: граф-схемы алгоритмов и граф-схемы программ.

Будем различать ГСА по следующим основным признакам:

— наличие внутренних обратных связей;

— используемые типы переменных;

— наличие дешифратора состояний;

— наличие умолчаний и неоднозначно задаваемых переменных;

— наличие переменных, которые могут неоднократно изменяться за один проход граф-схемы;

— место выдачи значений выходных переменных.

Применяя некоторые из этих признаков и не претендуя на полноту классификации, разделим ГСА на четыре подкласса:

— ГСА с внутренними обратными связями (ВОС) и выдачей значений выходных переменных в любой операторной вершине, обозначаемые ГСА1;

— ГСА без ВОС и дешифратора состояний, в которых выдача значений выходных переменных выполняется в «конце» тела граф-схемы, обозначаемые ГСА2;

— ГСА без ВОС, учитывающие особенности используемых управляющих конструкций языка программирования, обозначаемые ГСАЗ;

— ГСА без ВОС, но с дешифратором состояний и выдачей значений выходных переменных в «конце» тела граф-схемы, которые не содержат выходных переменных, неоднократно изменяющихся за один проход граф-схемы, обозначаемые ГСА4.

При этом отметим, что внешняя обратная связь в управляющих алгоритмах и программах существует всегда (режим сканирования в ПЛК).

Отметим также, что если в вычислительном устройстве (ВУ) реализуется алгоритм управления в виде одной компоненты (задачи), то ГСА1, используя ГСАЗ, могут при наличии соответствующих управляющих конструкций изоморфно отражаться в граф-схемы программ.

Если же алгоритм управления реализуется в вычислительном устройстве в виде нескольких компонент, то для ГСА1 отсутствует возможность изоморфного отражения в граф-схему программы. Это объясняется тем, что в задачах логического управления при использовании ГСА1 при определенных условиях на длительное время может наступать зацикливание по внутренним обратным связям, что исключает возможность перехода к реализации других компонент алгоритма управления до выхода из цикла. Зацикливание может происходить, например, пока «не нажата кнопка», «не сработал сигнализатор», «не кончилась задержка времени» или «вал не совершил нескольких оборотов».

На рис. 13.1 в качестве примера приведена схема связей управляющего автомата с объектом управления, состоящим из трех клапанов (Кл1, Кл2, КлЗ) и двигателя (Д). При этом управляющий автомат декомпозирован на автомат и функциональные элементы задержки. На рис. 13.2 для рассмат-

риваемого примера приведена ГСА1 с пятью ВОС [230]. Эта граф-схема реализует либо управляющий автомат в целом (функциональные элементы задержки ФЭ31 и ФЭ32 реализуются в третьей и четвертой операторных вершинах), либо только автомат (временные переменные t_1 и t_2 рассматриваются как обращения к процедуре, реализующей функциональные элементы задержки), а двоичные переменные T_1, и T_2 сигнализируют о срабатывании этих элементов.

Эта граф-схема содержательно неполна, так как в ней не указано, в каких операторных вершинах сбрасываются выходные z_i (i = 1,…,4) и временные t_j (j = 1, 2) переменные. Поэтому она не может применяться в качестве формальной спецификации задачи и требует доопределения. Обращаясь к принципам работы одновходовых исполнительных механизмов, используемых в рассматриваемом объекте управления, можно утверждать, что они не обладают памятью и поэтому в ГСА1 (рис. 13.2) по умолчанию предполагается, что в тех операторных вершинах, в которых переменная не указана, ее значение равно нулю.

Однако предположение о том, что если в операторных вершинах отсутствует некоторая выходная или временная переменная, то ее значение равно нулю, справедливо далеко не всегда, так как более часто при применении граф-схем считают, что умалчиваемые переменные сохраняют предыдущее значение. Такие граф-схемы также могут быть использованы для вычислений, так как вычислительное устройство помнит предыдущие значения всех переменных, сохраняющиеся во внешней по отношению к граф-схемам памяти, но они весьма трудно понимаются человеком, которому сложно помнить предысторию, особенно по нескольким переменным одновременно [202]. При этом отметим, что граф-схемы предназначены в основном для отражения связей по управлению и в существенно меньшей степени — по данным [98, 246]. Таким образом, граф-схемы с умалчиваемыми значениями переменных нецелесообразно применять в качестве языка общения.

Поэтому качественной можно считать только такую спецификацию, в которой кроме связей по управлению в максимальной степени отражены также и данные [246]. Отсутствие в явном виде некоторых данных в граф-схеме не позволяет применять ее также и в качестве теста для проверки построенной по ней программы. Более того, если программа строится по граф-схеме эвристически, то даже при отсутствии умолчаний с помощью последней можно проверить лишь то, что программа реализует заданную граф-схему, но невозможно установить, что она не делает ничего дополнительно.

Возвращаясь к описанию особенностей различных подклассов ГСА, отметим, что для ГСА1 характерно, что в них выдача значений выходных переменных осуществляется не в конце граф-схемы, а в любых операторных вершинах (рис. 13.2).

ГСА1 могут быть построены так, что в условных вершинах применяются только входные переменные типов X и T, а в операторных — выходные переменные типов z и t. Наличие в ГСА1 только тех переменных, которые упоминаются в алгоритме управления, является важным достоинством этого подкласса граф-схем.

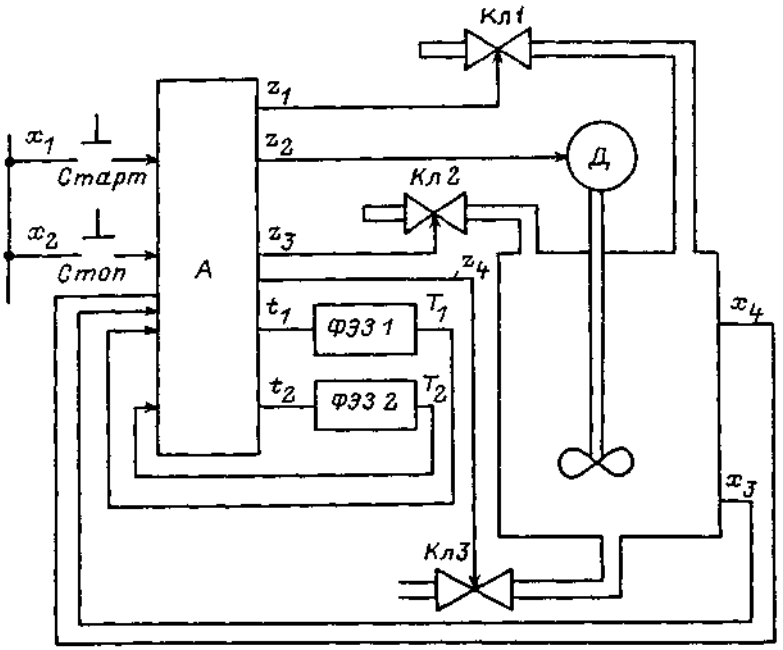

Рис. 13.1

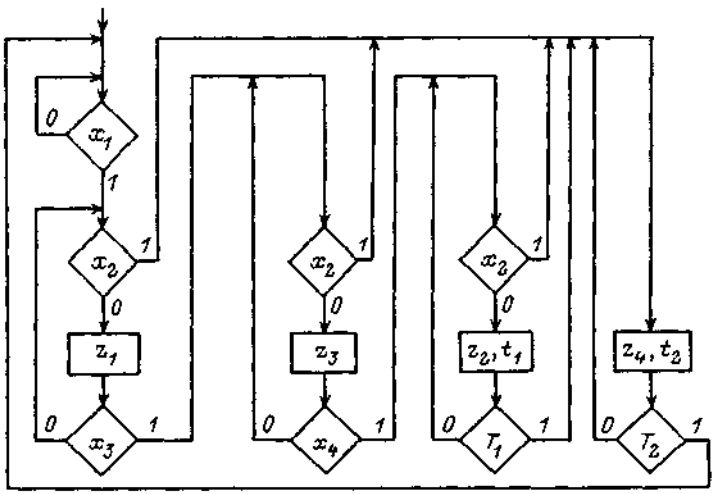

Рис. 13.2

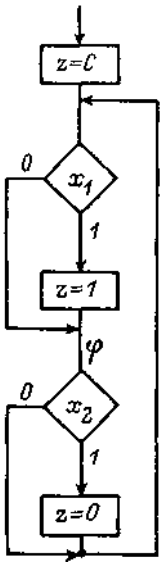

Рис. 13.3

В ГСА2 крайне редко используются только те переменные, которые указаны в алгоритме управления, и не применяются «лишние» переменные. В качестве такого примера на рис. 13.3 приведена ГСА2, реализующая *R*-триггер, в которой используются только переменные, определенные в алгоритме его функционирования: x_1 — переменная установки триггера; x_2 — переменная сброса триггера; z — выходная переменная. Даже эта ГСА (без дополнительных переменных) весьма трудно понимается, так как выдача результатов в ней происходит в конце тела программы и поэтому при $x_1 = x_2 = 1$ значения z вычисляются дважды (пересчитываются), а кроме того, в ней при $x_1 = x_2 = 0$ имеется «пустой» от операторных вершин путь, при прохождении которого сохраняется предыдущее значение z, что выполняется за счет запоминания во внешней относительно граф-схемы ячейке памяти значений z, указанных в операторных вершинах. Из рассмотренного примера следует, что если в ГСА2 существует хотя бы один путь, при прохождении которого не устанавливается ни нулевое, ни единичное значение хотя бы одной выходной переменной, то такая граф-схема реализует последовательностный автомат.

Как отмечалось выше, ГСА2 без «лишних» переменных встречаются крайне редко. В общем случае в этом подклассе ГСА в условных вершинах наряду с входными переменными типов *X* и *T* проверяются также и значения выходных переменных *Z*, и отсутствующие в алгоритме управления промежуточные (внутренние) переменные *Y*, которые устанавливаются и сбрасываются наряду с выходными переменными в операторных вершинах. В силу того что обычно применяются битовые переменные *y*, ГСА2 в этом случае весьма громоздки. Кроме того, они обычно неупорядоченны по структуре, так как порядок расположения вершин в граф-схемах стандартами не оговаривается. ГСА2 весьма трудно понимаются и по причине умолчаний сохраняющихся значений переменных *Y* и *Z*.

ГСА2 также трудно понимать в случаях, если значения переменных зависят от предыстории, а именно от значений соответствующих переменных, указанных в ранее расположенных операторных вершинах, но их особенно трудно понимать, если значения переменных не только зависят, но и изменяются в зависимости от предыстории, т. е. зависят от путей, по которым можно попасть в рассматриваемую операторную вершину. В последнем случае возникают также большие проблемы с безошибочным внесением изменений в граф-схемы.

Необходимо отметить, что если для Программиста проверки переменных *Y* и *Z* вполне естественны, а для Разработчика объяснимы, то для Заказчика, Технолога, Оператора и Контролера их наличие неприемлемо, так как, например, Заказчик в техническом задании обычно не просит устанавливать какую-либо внутреннюю переменную равной единице. Особые сложности могут возникнуть при чтении в тех случаях, когда

проверяемые переменные Y и Z одной ГСА2 могут изменяться с помощью других компонент алгоритма, реализуемых в том же вычислительном устройстве.

Из изложенного следует, что если ГСА1 и ГСА2 без проверки переменных Y и Z в условных вершинах отражают только семантику реализуемого алгоритма, то ГСА2 с такими проверками представляют собой алгоритм реализации заданного алгоритма управления в вычислительном устройстве, что делает нецелесообразным использование таких граф-схем в качестве языка общения.

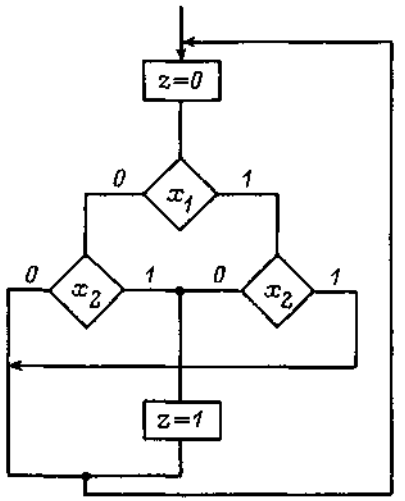

Рис. 13.4

Проблемы с построением ГСА1 и ГСА2 на изложенном не заканчиваются, так как для безошибочного перехода к граф-схеме программы и возможности проведения экспертизы (для ответственных объектов управления) желательно по исходной ГСА построить ГСА3, учитывающую основные свойства управляющих конструкций применяемого языка программирования. Например, в большинстве ПЛК для команд IF допустимы переходы только вперед и запрещены возвраты назад. Другая принципиальная особенность команд IF многих ПЛК состоит в том, что, во-первых, они одноадресны, а во-вторых, обладают тем свойством, что если некоторая команда IF при невыполнении условия передает управление на команду (метку) CONT, то тогда и все размещаемые между этими командами другие команды IF при невыполнении условий также должны передавать управление на использованную метку CONT [230]. При этом необходимо отметить, что команда безусловного перехода STOP позволяет реализовать только внешнюю обратную связь.

В этих условиях ГСА3 должна быть линеаризованной и структурированной только с помощью управляющих конструкций «последовательное соединение» и «неполный выбор». В тех случаях, когда ГСА2 в исходном виде обладает такой структурой (например, граф-схема на рис. 13.3), необходимость в построении ГСА3 отпадает. Однако если, например, ГСА2, реализующая однотактный автомат, описываемый одной из булевых формул $z = \bar{x}_1 \& x_2 \vee x_1 \& \bar{x}_2 = x_1 \oplus x_2$, имеет «плоскостную» (рис. 13.4), а не «линейную» структуру, то для перехода к граф-схеме программы и тексту программы по ГСА2 целесообразно предварительно построить ГСА3 (рис. 13.5). Из рассмотрения последней ГС следует, что сложность ее понимания по сравнению с ГСА2 (рис. 13.4) увеличилась (даже несмотря на то что в этих граф-схемах не применяются проверки переменных Y и Z), так как при прохождении любого пути в ГСА3 входные переменные приходится проверять неоднократно. При этом отметим, что если программирование булевых формул проводить не в бинарной, а в операторной форме (непосредственно по формуле), то

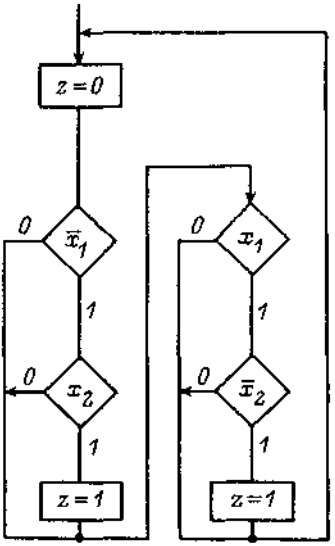

Рис. 13.5

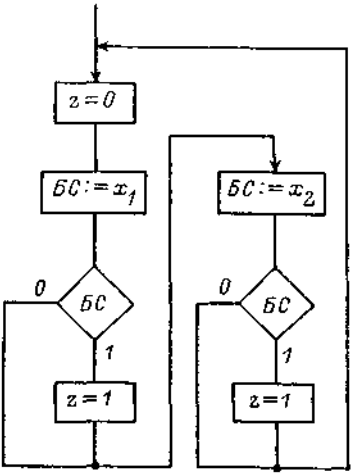

Рис. 13.6

ГСА2 всегда будет иметь линейную структуру, что, правда, в общем случае приводит к снижению быстродействия и требует использования промежуточных переменных.

В вычислительных устройствах, в которых ввод входных переменных может осуществляться по мере необходимости, как это обычно имеет место в некоторых типах ПЛК, эти переменные за один проход граф-схемы могут не только неоднократно проверяться, но и изменять свои значения, что может приводить к нарушению функционального соответствия, задаваемого спецификацией (таблицей истинности), которое по аналогии с аппаратной реализацией может быть названо «риском» программной реализации. Для уменьшения степени риска для таких ПЛК должны применяться буферизация или максимально бесповторная реализация.

Таким образом, если в ГСА1 отражается только семантика реализуемого алгоритма, в ГСА2 кроме семантики учитывается также и возможность реализации алгоритма, состоящего из нескольких компонент, в одном вычислительном устройстве, то в ГСА3, кроме того, отражается специфика используемых управляющих конструкций применяемого языка программирования. Однако ГСА3 еще значительно отличаются от граф-схем программ, так как в последних должна отражаться также и семантика всех применяемых команд или операторов используемого языка программирования. При этом в граф-схемах программ появляются упоминания о таких архитектурных особенностях вычислительного устройства, которые в исходной ГСА не применяются. Например, на рис. 13.6 приведена граф-схема программы, реализующая ГСА2 (рис. 13.3), в которой символом БС обозначен битовый сумматор ПЛК. Из изложенного следует, что граф-схемы программ, а тем более тексты программ не должны использоваться в качестве языка общения.

Поэтому только ГСА1 без умолчаний могут претендовать на роль языка общения, однако при их программировании возникают проблемы, связанные с их расцикливанием, линеаризацией и структурированием. Видимо, по этой причине на практике обычно в лучшем случае строят ГСА2 и, минуя построение других типов граф-схем, неформально пишут текст программы.

При этом возникают проблемы с выбором тестов и доказательством правильности программы, так как при таком подходе из-за отсутствия «ясной» спецификации не удается обсудить с заинтересованными Специалистами, правильно ли понята поставленная задача, и проблема «правильности» перекладывается на испытания, при которых можно обнаружить, что что-то делается неправильно, но весьма трудно установить, что программа в случае ошибок в ней может делать что-либо, не заданное спецификацией.

По мнению автора, именно возможность досконального обсуждения со Специалистами правильности построения спецификации является основной предпосылкой для качественного прохождения всех этапов жизненного цикла систем логического управления.

Указанные проблемы усугубляются тем, что в практике проектирования систем рассматриваемого класса обычно методика проверки функционирования представляет собой таблицу, содержащую два столбца, в первом из которых указываются значения входных переменных, а во втором — значения выходов. Однако такая проверка, естественная для однотактных автоматов, некорректна для последовательностных задач. Создание же методики, учитывающей также значения всех внутренних, а ряде случаев и предшествующие значения выходных переменных, при неупорядоченной структуре граф-схемы проблематично из-за большой размерности.

По мнению автора, весь этот клубок проблем во многом связан с тем, что в граф-схемах алгоритмов и собственно в программах обычно не используется понятие «внутреннее состояние» компоненты в целом, называемое «состояние», а применяются лишь отдельные битовые переменные, косвенно его характеризующие. Введение этого понятия позволяет переходить от граф-схем к графам переходов и обратно и использовать последние в качестве языка общения и спецификации.

В заключение раздела отметим, что автоматные ГСА могут быть двух типов — неветвящиеся (последовательные) и ветвящиеся.

Изложенное в настоящем разделе относится ко второму типу граф-схем, так как если алгоритм логического управления описать с помощью системы булевых формул, то он всегда может быть реализован последовательной ГСА с одной обратной связью, которая является внешней.

Однако ГСА этого типа ненаглядны, трудно читаются и понимаются, и поэтому их нецелесообразно применять для спецификации алгоритмов.

В разд. 13.4 будет показано, что произвольный граф переходов всегда может быть реализован ветвящейся ГСА только с одной (внешней) обратной связью, если при ее построении использовать понятие «состояние».

Граф-схемы, которые строятся с помощью предлагаемого подхода, могут быть названы граф-схемами с дешифратором состояний.

При этом отметим, что если заданный граф переходов не содержит флагов и умолчаний значений выходных переменных, то соответствующая ему ГСА с дешифратором состояний будет хорошо «понимаемой». В граф-схемах этого класса при некоторых видах кодирования допустимы умолчания неизменяющихся компонент вектора следующего состояния, так как они всегда могут быть определены в результате анализа всех компонент вектора текущего состояния, указанных в явном виде в ветвящейся структуре, являющейся дешифратором состояний.

По определению Н. Вирта, «программы — это алгоритмы плюс данные» [246]. Связи по управлению в алгоритмах весьма эффективно описываются граф-схемами, в то время как связи по данным, находящимся в ячейках оперативной памяти, в этих структурах представляются не столь наглядно.

Граф-схемы с дешифратором состояний позволяют весьма эффективно наряду со связями по управлению отразить также и связи по такой разновидности данных, как внутренние переменные.

Граф-схема с дешифратором состояний изоморфна соответствующим ей графу переходов и конструкции switch, и поэтому если не используются флаги и умолчания значений выходных переменных, то она может применяться в качестве спецификации как алгоритмов, так и программ.

При этом отметим, что графы переходов в качестве языка спецификаций алгоритмов применять обычно более целесообразно даже по сравнению с ГСА, содержащими дешифратор состояний, например ввиду их большей однородности и компактности, что в свою очередь обеспечивает их лучшую обозримость и понятность.

Однако в отдельных случаях, например по причине отсутствия стандартов, применить графы переходов в качестве языка спецификации не удается и приходится использовать ГСА, построение которых определено стандартом [245]. При этом целесообразно строить ГСА не произвольной структуры, а с дешифратором состояний, соответствующую выбранному типу автомата и виду кодирования его состояний.

Простота некоторых ГСА без дешифратора состояний создает иллюзию простоты их поведения. Такие ГСА весьма трудно анализировать, так как предшествующие значения внутренних, а возможно, и выходных переменных проверяющему приходится «держать» в голове. Поэтому для анализа поведения граф-схем без дешифратора состояний целесообразно проводить верификацию, состоящую в построении эквивалентного графа переходов.

Продемонстрируем это на примерах.

Рассмотрим первоначально «идеальную» ГСА (рис. 13.3), которая является структурированной, не содержит внутренних переменных и проверок выходной переменной. Покажем, что эта ГСА реализует автомат с памятью.

Структура этой граф-схемы позволяет проводить ее верификацию в направлении «сверху вниз».

При этом первый блок реализует формулу

$$\varphi = z' \,\&\, \overline{x}_1 \vee z'' \,\&\, x_1 = z' \,\&\, \overline{x}_1 \vee 1 \,\&\, x_1 = x_1 \vee z,$$

а второй — формулу

$$z = \varphi' \,\&\, \bar{x}_2 \vee \varphi'' \,\&\, x_2 = \varphi \,\&\, \bar{x}_2 \vee 0 \,\&\, x_2 = \bar{x}_2 \vee \varphi.$$

Выполняя подстановку первой формулы во вторую, получим формулу, верифицирующую заданную ГСА: $z = \bar{x}_2 \,\&\, (x_1 \vee z)$. Из рассмотрения этой формулы следует, что в данном случае при одних и тех же значениях входных переменных формируются различные значения выходной переменной, т. е. реализуется автомат с памятью, содержащий два состояния.

Переходя к рассмотрению второго примера,выполним в рассмотренной ГСА следующие замены: в первой (начальной) операторной вершине вместо $z = C$ запишем $y = C_1$; $z = C_2$; во второй — вместо $z = 1$ запишем $y = 1$; в третьей — вместо $z = 0$ запишем $z = 1$; в первой условной вершине вместо x_1 запишем x, а во второй — вместо x_2 запишем y; вместо пометки φ введем пометки y_1 и z_1.

Рассматривая новую ГСА также сверху вниз, получим две системы формул:

$$\begin{cases} y_1 = y' \,\&\, \bar{x} \vee y'' \,\&\, x = y \,\&\, \bar{x} \vee 1 \,\&\, x = x \vee y; \\ z_1 = z' \,\&\, \bar{x} \vee z'' \,\&\, x = z \,\&\, \bar{x} \vee z \,\&\, x = z; \end{cases}$$

$$\begin{cases} y = y_1' \,\&\, \bar{y} \vee y_1'' \,\&\, y = y_1 \,\&\, \bar{y} \vee y_1 \,\&\, y = y_1; \\ z = z_1' \,\&\, \bar{y} \vee z_1'' \,\&\, y = z_1 \,\&\, \bar{y} \vee 1 \,\&\, y = y \vee z_1. \end{cases}$$

Выполняя подстановку первой системы во вторую, построим систему формул, верифицирующую рассматриваемую ГСА:

$$y_1 = x \vee y; \quad z_1 = y \vee z$$

Заполним по этой системе кодированную таблицу переходов и выходов (таблицу истинности) и построим по ней граф переходов автомата без выходного преобразователя, который будет содержать четыре состояния. Этот граф переходов может быть описан матрицей переходов (табл. 13.1).

Таблица 13.1

yz.	00	01	11	01
00	$\bar{x}$	x	—	—
10	—	—	1	—
11	—	—	1	—
01	—	—	x	$\bar{x}$

Из анализа этой матрицы следует, что рассматриваемая ГСА для целей управления непригодна, так как содержит две начальные вершины («00» и «01»), одну неустойчивую («10») и одну тупиковую («10») вершины.

Из приведенной матрицы следует также, что эта ГСА при $C_1 = 0$, $C_2 = 0$ реализует полный подграф графа переходов, содержащий три вершины: «00», «10», «11», а при $C_1 = 0$, $C_2 = 1$ — полный подграф, состоящий из двух вершин («01», «11»).

Обратим внимание на тот факт, что неустойчивое состояние «10» соответствует «мгновенному» состоянию ячеек оперативной памяти ПЭВМ, которое ненаблюдаемо на «выходе» граф-схемы с одним оператором «Вывод», располагаемым в конце ее «тела», в то время как остальные состояния граф-схемы на ее «выходе» наблюдаемы.

Таким образом, переменные y и z с учетом внешней обратной связи порождают четыре состояния ГСА, в которой понятие «состояние» в явном виде не используется.

Граф-схема в третьем примере отличается от предыдущей тем, что в ее теле в первой операторной вершине применяются пометки $y = 0$, $z = 1$, а в нижней — $z = 0$.

Выполняя и в этом случае верификацию в направлении «сверху вниз», построим две системы формул:

$$\begin{cases} y = y' \,\&\, \bar{x} \vee y'' \,\&\, x = y \,\&\, \bar{x} \vee 0 \,\&\, x = \bar{x}\, y; \\ z = z' \,\&\, \bar{x} \vee z'' \,\&\, x = z \,\&\, \bar{x} \vee 1 \,\&\, x = x \vee z; \end{cases}$$

$$\begin{cases} y = y_1' \,\&\, \bar{y} \vee y_1'' \,\&\, y = y_1 \,\&\, \bar{y} \vee y_1 \,\&\, y = y_1; \\ z = z_1' \,\&\, \bar{y} \vee z_1'' \,\&\, y = z_1 \,\&\, \bar{y} \vee z_1'' \,\&\, y. \end{cases}$$

Выполняя подстановку первой системы формул во вторую и учитывая, что во втором блоке ГСА $z = 0$, окончательно получим:

$$y_1 = \bar{x}\, y;$$
$$z_1 = (x \vee z)\, \bar{y} \vee (x \vee 0) y = x \vee \bar{y} z.$$

Этой системе формул также соответствует граф переходов автомата без выходного преобразователя с четырьмя вершинами, из которых одна неустойчивая («11»), а одна тупиковая («01»).

Методы верификации граф-схем, структура которых отличается от рассмотренных, изложены в разд. 4.3.4.

Указанные методы верификации являются аналитическими. Изложим также универсальный метод построения графа переходов по ГСА, основанный на умозрительном трассировании путей в ней. Этот метод состоит в том, что в начальной вершине вместо переменных C_i подставляются константы 0 и 1, формируя все возможные коды, каждый из которых соответствует одной из вершин графа переходов, эквивалентного рассматриваемой ГСА, а затем для каждой из этих вершин по граф-схеме определяются условия перехода в смежную вершину графа переходов, в качестве которой может выступать и сама вершина, из которой выполняется переход.

Например, для ГСА, рассмотренной во втором примере, граф переходов может быть построен следующим образом. Полагая в начальной вершине $y = 0$, $z = 0$, определим, что при $x = 0$ состояние сохраняется. При $y = 0$, $z = 0$, $x = 1$ выполняется переход в неустойчивое состояние $y = 1$, $z = 0$. При $y = 0$, $z = 1$, $x = 0$ состояние сохраняется, а при $y = 0$, $z = 1$, $x = 1$ выполняется переход в состояние $y = 1$, $z = 1$. При $y = 1$, $z = 0$, $x = 0$ или $x = 1$ — следующее состояние $y = 1$, $z = 1$, а при $y = 1$, $z = 1$, $x = 0$ или $x = 1$ состояние сохраняется. Таким образом строится граф переходов, описываемый табл. 13.1.

Из изложенного следует, что анализ поведения ГСА целесообразно проводить не по ней непосредственно, так как при этом не гарантируется полнота проверки, а по эквивалентному ей графу переходов, в котором все состояния и все переходы между ними записаны явно. Это позволяет определить все функциональные возможности рассматриваемой граф-схемы.

Следовательно, граф переходов, эквивалентный ГСА, можно рассматривать в качестве «портрета» всех ее функциональных возможностей, по которому можно проанализировать все переходы в граф-схеме.

Отметим, что в настоящее время типичная тенденция в практическом программировании состоит в том, что программист обычно не использует никакие графические модели, а непосредственно на экране ПЭВМ пишет текст программы, а затем методом «проб и ошибок» эта программа доводится им до «правдоподобного» поведения. При этом в большинстве случаев, добившись такого поведения, программист не знает всех функциональных возможностей программы, так как ее верификацию обычно не проводит. Такие модели несомненно потребовались бы при анализе и (или) синтезе программ, если бы речь шла не о «показательном», а о «доказательном» программировании, которое должно использоваться при управлении ответственными технологическими объектами.

На основе изложенного выше подхода появляются предпосылки для верификации текстов по крайней мере «автоматных» программ. При этом по тексту программы предлагается строить эквивалентную ей ГСА, которая и должна верифицироваться, как изложено выше.

Пусть, например, задана следующая программа:

```
      y = C1; z = C2;
M:    if (x)   y = 1;
      if (y)   z = 1;
      goto M.
```

Эта программа изоморфна граф-схеме, рассмотренной во втором примере, верификация которой позволяет сделать вывод о том, что рассматриваемая программа непригодна для целей управления.

Для корректировки программы необходимо: построить «желательный» граф переходов с принудительно-свободным кодированием его вершин; заполнить по нему кодированную таблицу переходов и выходов; с помощью модифицированного метода Блоха (разд. 4.3.2) построить ГСА; реализовать полученную граф-схему программой.

При этом отметим, что при оптимизации граф-схем некорректно применение операции «вынос вверх операторной вершины» в случае,

когда обозначения переменной в этой вершине и в соответствующей ей условной вершине совпадают.

Из изложенного следует, что для построения «правильной» программы либо ее анализ (при эвристическом синтезе и отсутствии ошибок в программе), либо ее синтез, либо анализ и синтез (при эвристическом первоначальном синтезе и наличии ошибок в программе) должны выполняться формально.

13.2. Графы переходов. Расширение понятий

Определение 1. ГП, в котором в каждой вершине кроме кодов состояний явно указаны также и значения каждой выходной переменной, назовем «графом переходов автомата Мура с явным заданием значений всех выходных переменных».

В этом случае значения выходов соответствуют номеру вершины и не зависят от предыстории. По этой причине такой граф переходов просто читается и в него легко вносятся изменения.

Определение 2. ГП, в котором в вершинах кроме явно определенных значений одних переменных используются также неявно, но однозначно определенные значения других переменных, назовем «графом переходов автомата Мура с неявным заданием значений выходных переменных».

Неявное задание переменной в вершине отображается прочерком, который в данном случае может быть заменен только одним значением булевой переменной. Эта переменная в рассматриваемой вершине сохраняет то значение, которое присваивается ей во всех «смежных» вершинах. Связь между двумя вершинами может быть непосредственной с помощью входящей в вершину дуги или транзитной через другие вершины, в которых вместо значений этой переменной применяются прочерки.

Графы переходов этого типа читаются несколько хуже, чем ГП автоматов Мура, по определению *1,* однако их целесообразно использовать для сокращения объема памяти программ. Такие графы могут быть изображены так, что в каждой вершине в явном виде указаны значения каждой выходной переменной, а не изменяющиеся относительно «смежных» вершин значения этих переменных отмечаются перечеркиванием.

Определение 3. ГП, в котором в вершинах кроме явно определенных значений одних переменных применяются также неявно и неоднозначно заданные значения других переменных, назовем «графом переходов автомата Мура с неоднозначным заданием значений выходных переменных».

Графы переходов этого класса могут содержать для одной и той же задачи меньшее число вершин, чем ГП автоматов Мура двух других указанных выше разновидностей, так как в этом случае в вершине вместо одного значения умалчиваемой переменной могут формироваться различные значения той же переменной. Это порождает в этой вершине различные наборы значений выходных переменных и обеспечивает эквивалентность одной такой вершины нескольким, полностью определенным.

Это преимущество с точки зрения компактности описания и сокращения длины программы порождает одновременно и главный недостаток

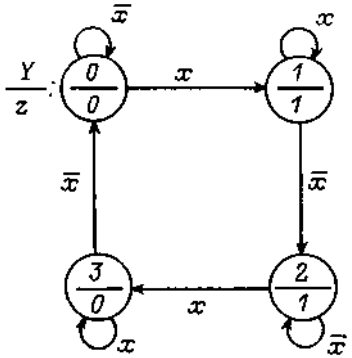

Рис. 13.7

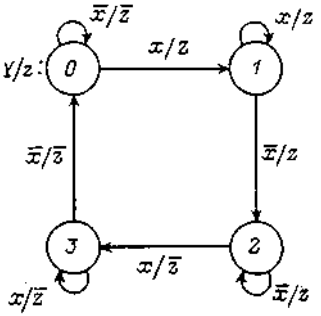

Рис. 13.8

графов переходов этого типа — трудность их чтения и понимания. Именно по этой причине такие графы не рекомендуется использовать в качестве языка общения.

Спецификацию задач рассматриваемого класса целесообразно производить с помощью указанных выше первых двух моделей ГП автоматов Мура, а в случае применения для спецификации ГСА1 третьей модели ее целесообразно преобразовать к одной из этих двух моделей.

Это, естественно, не исключает использования графов переходов для других классов автоматов. Аналогичная классификация может быть предложена для ГП автоматов Мили, в которых значения выходных переменных формируются не в вершинах графов, а на их дугах. Во многих задачах логического управления переход от модели автомата Мура к модели автомата Мили не уменьшает числа состояний (вершин ГП). На рис. 13.7 в качестве примера приведен ГП автомата Мура, реализующий счетный триггер, а на рис. 13.8 эта же задача описана с помощью ГП автомата Мили. В других случаях трансформация графа переходов автомата Мура или автомата без выходного преобразователя в ГП автомата Мили позволяет сократить число вершин. На рис. 4.114 приведен ГП автомата без выходного преобразователя с четырьмя вершинами, реализующий последовательный сумматор, а на рис. 4.113 этот алгоритм описан ГП автомата Мили с двумя вершинами.

При этом необходимо отметить, что если для автоматов Мура и автоматов без выходного преобразователя с однозначным заданием значений выходов число состояний равно числу комбинаций этих значений (среди которых учитываются повторяющиеся), а для автоматов, в которых все комбинации различны, число состояний равно числу различных комбинаций, то уже для автоматов этих классов с неоднозначным их заданием понятие «состояние» становится менее связанным со значениями выходов и поэтому является более абстрактным и менее понятным. Это положение усугубляется для автоматов Мили, и поэтому для автоматов этого класса с неоднозначными значениями выходов в [120] вместо понятия «состояние» было введено понятие «ситуация», а вместо термина «граф переходов» — термин «граф переключений». Однако, несмотря на то что граф переключений может содержать меньшее число

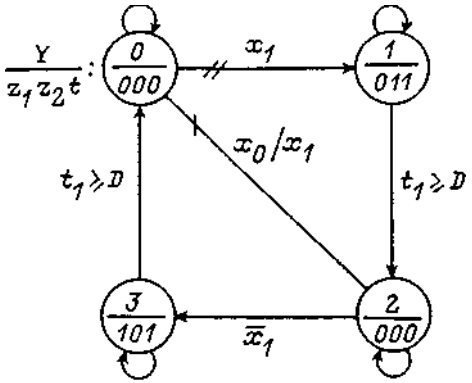

Рис. 13.9

вершин, чем эквивалентный граф переходов, применение графов переключений в качестве языка общения нецелесообразно ввиду сложности их понимания.

Аналогичная ситуация с числом состояний из-за умолчаний значений переменных может иметь место и для такого типа автоматов, как смешанные автоматы: «автоматы без выходного преобразователя—автоматы Мили» (СФ1) и «автоматы Мура—Мили» (СА2), а также для автоматов всех перечисленных классов с флагами, в которых одна и та же переменная может использоваться в одном и том же графе переходов как в качестве входной, так и выходной переменной.

В качестве флага, например, могут применяться переменные, если они находятся во внешних относительно автомата ячейках памяти, значения которых им проверяются, и могут быть изменены не только внешним источником информации, например кнопкой, но и собственно автоматом. В графе переходов автомата СА2 (рис. 13.9) в качестве флага используется переменная x_1, значения которой, сохраняемые во внешней относительно автомата битовой ячейке памяти, могут быть изменены в устойчивых состояниях автомата кнопкой или собственно автоматом на переходе «0—2», инициируемом входной переменной x_0. Значения переменной x_1 не только формируются автоматом, но и проверяются им на переходах «0—1» и «2—3».

Более типично применение флагов как дополнительно введенных переменных, воздействующих на ячейки памяти, содержимое которых не может быть изменено от других источников информации. В случае, когда флаги вводятся в автомат без выходного преобразователя, эти ячейки могут рассматриваться как принадлежность автомата наряду с ячейками, сохраняющими значения переменных, различающих состояния.

Для автоматов Мура, Мили и смешанных автоматов флагами являются дополнительно вводимые переменные *F,* значения которых запоминаются в ячейках памяти, внешних относительно рассматриваемого автомата. Если при использовании многозначного кодирования состояний автоматов любой сложности, принадлежащих этим классам, достаточно иметь в каждом из них только одну промежуточную переменную, то при введении флагов во внешней относительно автомата среде появляются дополнительные ячейки памяти, в том числе и многозначные. При этом номер следующего состояния определяется не только номером рассматриваемого состояния и входным воздействием, как это имеет место в автоматах без флагов, но и зависит от глубокой предыстории. Таким образом, для таких автоматов не только значения выходов, но и значения состояний могут зависеть и изменяться от предыстории, что, естественно, резко усложняет их понимание.

На рис. 9.7 приведен не содержащий ни одной устойчивой вершины ГП автомата Мура с десятью вершинами. Число вершин в этом графе удается сократить до семи за счет введения двоичного флага *F1,* формируемого в этом случае только автоматом, и неоднозначности значений флага в некоторых вершинах ГП (рис. 9.9).

Дальнейшее сокращение числа вершин в этом графе обеспечивается введением дополнительного многозначного флага *F2* и применением умолчаний его значений (рис. 9.10).

Использование ГП с неоднозначным заданием значений выходных переменных и (или) ГП с флагами не позволяет в полной мере называть вершины таких графов состояниями, в отличие от ГП без флагов и умолчаний. Состояниями автоматов, описываемых ГП указанных разновидностей, являются вершины в построенном для каждого из них графе достижимых маркировок. Например, для ГП, приведенных на рис. 9.9 и 9.10, содержащих 7 и 5 вершин, соответствующий им ГДМ (рис. 9.7) состоит из 10 вершин, и поэтому можно утверждать, что соответствующие этим графам автоматы содержат по 10 состояний.

Аналогичная ситуация имеет место в диаграммах «Графсет» и графах переключений [120].

13.3. Метод построения читаемых графов переходов по ГСА с обратными связями

Изложение метода выполним на примере. Пусть задана граф-схема алгоритма с ВОС (ГСА1) (рис. 13.10) и требуется понять эту граф-схему.

Сложность понимания этой граф-схемы определяется отсутствием обозначений некоторых выходных переменных в операторных вершинах, для каждой из которых по умолчанию предполагается сохранение предыдущего ее значения.

Если закодировать числами (числа в кружках) точки начала и конца (если последняя имеется), а также точки, следующие за операторными вершинами, и определить пути в ГСА между смежными точками [16], то можно построить граф переходов автомата Мили с неоднозначными значениями выходных переменных (рис. 13.11), число вершин в котором равно количеству точек, введенных в ГСА. Этот граф переходов может быть эффективно запрограммирован, например на языке СИ, но понять (из-за умолчаний некоторых значений выходных переменных), как он функционирует и соответственно как функционирует ГСА1, весьма сложно.

Поэтому построим по ГСА1 граф переходов автомата Мура, который по принципам построения должен пониматься лучше. Для этого закодируем числами (без кружков) каждую операторную вершину и определим все пути между смежными вершинами [16]. Используя эту информацию, построим граф переходов автомата Мура с неоднозначными значениями выходных переменных, содержащий пять вершин, закодированных многозначными (десятичными) числами (рис. 13.12). Этот граф понимается лучше, чем предыдущий, но также весьма сложно, так как и он содержит умолчания.

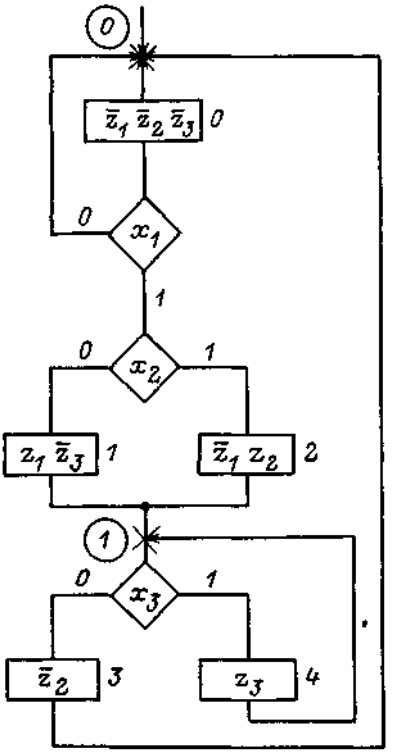

Рис. 13.10

Анализируя значения, формируемые при прохождении различных путей в этом графе, построим по результатам анализа граф переходов автомата Мура без умолчаний с девятью вершинами (рис. 13.13). Так как в новом графе в каждой паре вершин — (41, 341) и (1, 31) — формируются одинаковые значения выходных переменных, то вторые вершины в этих парах вместе с дугами, помеченными единицами (безусловные переходы), могут быть исключены. При этом первые вершины этих пар соединяются с нулевой вершиной. С этой же вершиной соединяется вершина «2», так как вершина «32», в которой формируются те же значения выходных переменных, что и в нулевой вершине, может быть исключена. Получающийся при этом ГП автомата Мура содержит шесть вершин (рис. 13.14). Этот граф «абсолютно» понятен, так как он компактен и при его чтении не требуется помнить предысторию. При этом необходимо отметить, что его структура в отличие от других графов (рис. 13.11, 13.12) существенно отличается от исходной ГСА.

Полученный граф переходов полон и непротиворечив, однако содержит, как и ГСА1, два генерирующих контура: «0 = 1» и «0 = 2», устраняя которые, например введением переменной x_3 в пометки дуг «0—1» и «0—2», получим корректный граф (рис. 13.15). Обратим при этом внимание на тот факт, что в граф переходов указанные изменения были внесены весьма просто, без корректировки его структуры, в то время как для ГСА1 это потребовало бы существенных изменений схемы, что может приводить к ошибкам.

Из изложенного следует, что именно граф переходов на рис. 13.15, а не ГСА, должен использоваться в качестве «предмета обсуждения» с Заказчиком и являться спецификацией на программирование при отсутствии жестких ограничений на объем памяти программ. Приведенный граф в отличие от ГСА1 описывает поведение автомата в явной и понятной форме и содержит достаточно информации, чтобы

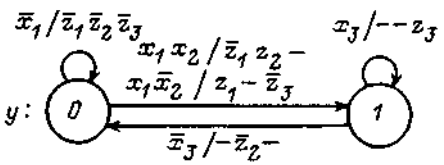

Рис. 13.11

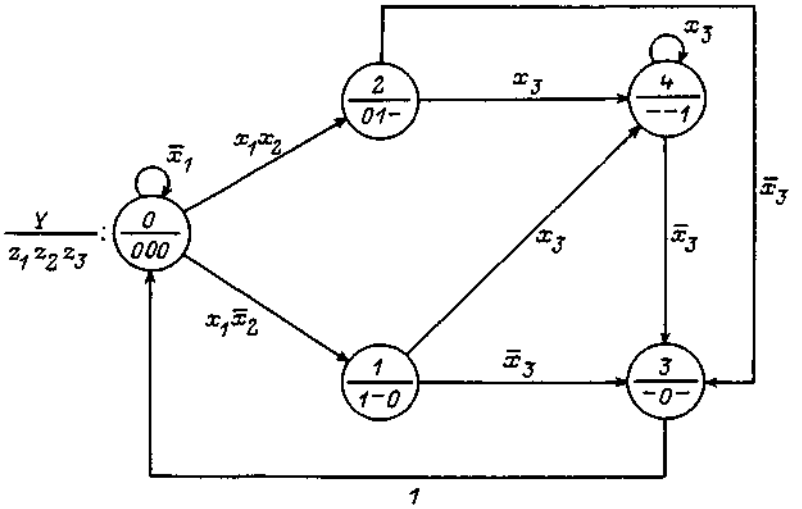

Рис. 13.12

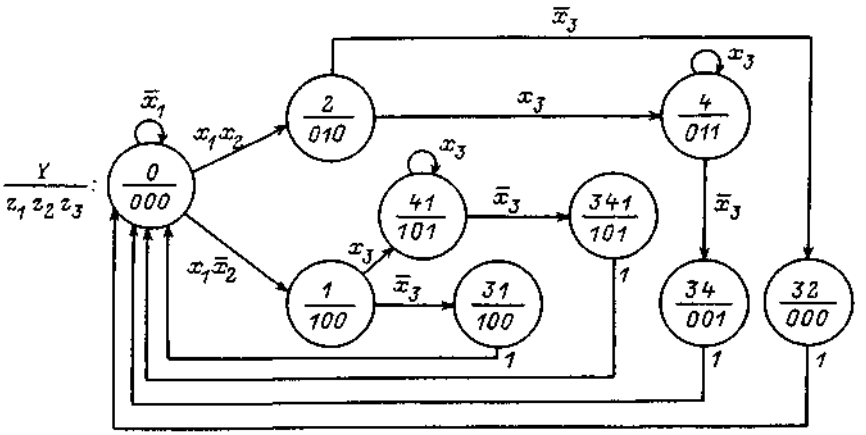

Рис. 13.13

быть формально реализованным с помощью различных алгоритмических моделей (СБФ, ФС, ГСА4 и т. д.).

Каждая из этих алгоритмических моделей в свою очередь может быть изоморфно отражена программной моделью, однако степень изоморфизма последней с графом переходов различна. Действительно, если переход к тексту программы, например на языке СИ, производится с помощью СБФ, то в зависимости от применяемого метода построения этой системы степень ее изоморфизма с ГП будет существенно различной. С другой стороны, в состав указанного языка входит такая управляющая конструкция, как `switch`, при применении которой удается обеспечить изоморфизм между графом переходов и текстом программы, что позволяет сравнительно просто понимать не только спецификации (при соответствующем их построении), но и собственно тексты программ.

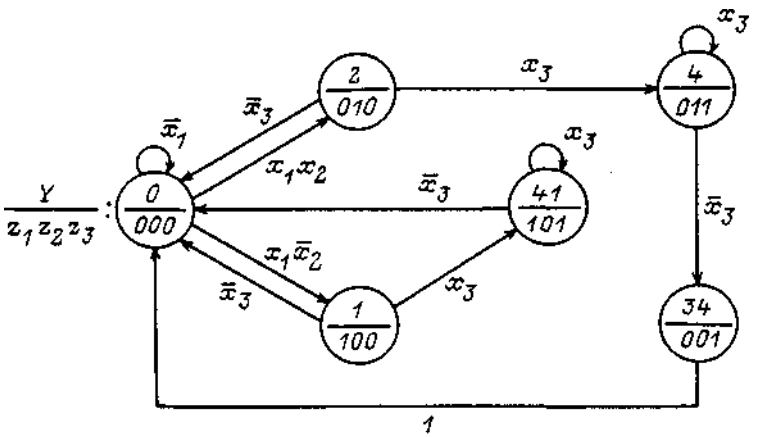

Рис. 13.14

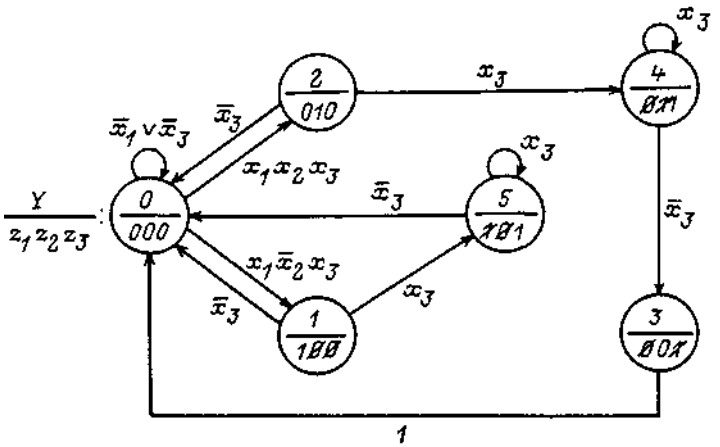

Рис. 13.15

В заключение раздела отметим, что если для построения некоторых алгоритмических и программных моделей, например системы булевых формул или функциональной схемы, вся информация, приведенная на рис. 13.15, необходима, то при использовании управляющей конструкции switch в графе переходов пометки петель могут умалчиваться (предполагая, что в каждой вершине обеспечивается логическая полнота пометок дуг, исходящих из нее), а вместо ортогонализации этих пометок могут расставляться приоритеты, указываемые на дугах штрихами, число которых тем меньше, чем выше номер приоритета (рис. 13.16). Это в общем случае позволяет резко упростить граф переходов и уменьшить объем программы, практически без ухудшения ее понимаемости.

Необходимо, отметить также, что если для ГСА первого вида (по классификации, приведенной в гл. 1) рассмотренный переход к ГП

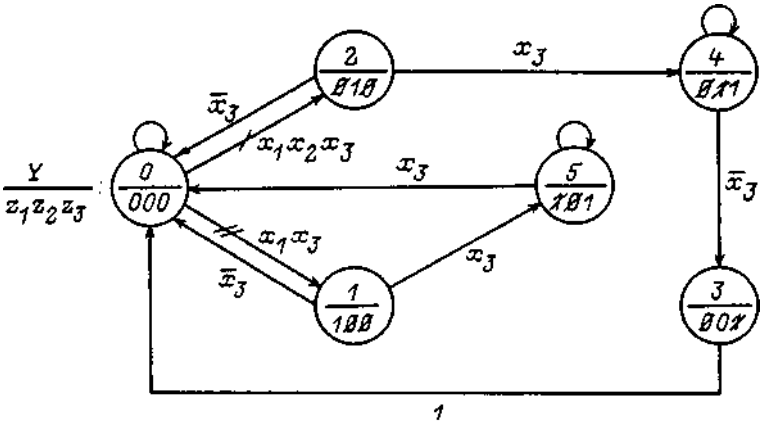

Рис. 13.16

правомочен, то для ГСА второго вида, в которых не применяется дешифратор состояний, он некорректен, так как в таких ГСА значения выходных переменных выводятся только в одной точке, а при построении ГП предполагается, что они выводятся в каждой вершине (для автоматов без выходного преобразователя и автоматов Мура) или на каждой дуге (для автоматов Мили).

13.4. Построение читаемых ГСА без внутренних обратных связей по графам переходов без умолчаний

В разд. 13.1 было показано, что «плохо организованные» ГСА без внутренних обратных связей (ГСА2) весьма трудно читаются. Покажем, какой структурой должны обладать ГСА этого класса, чтобы устранить указанный недостаток.

13.4.1. Реализация автоматов без выходного преобразователя с принудительным кодированием состояний

Пусть задан граф переходов автомата без выходного преобразователя с принудительным кодированием состояний (рис. 13.17), в котором при импульсных переменных x_1 и x_2 генерирующие контуры отсутствуют. Кодирование названо принудительным, так как коды, определяющие состояния, совпадают со значениями выходных переменных, формируемых в соответствующих состояниях (вершинах). Этот вид кодирования использован, ввиду того что в рассматриваемом графе комбинации значений выходных переменных во всех вершинах различны. Этот граф переходов предлагается реализовать ГСА4 (рис. 13.18), тело которой состоит из трех слоев. Первый из них содержит условные вершины, помеченные переменными z_1 и z_2 и является дешифратором состояний. Второй слой образован условными вершинами, помеченными входными

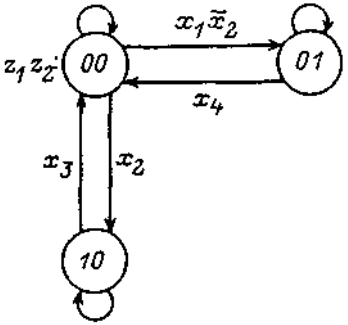

Рис. 13.17

переменными, и реализует функции переходов автомата. Третий слой содержит операторные вершины, в которых указаны значения выходов, совпадающие в данном случае с кодами следующих состояний.

Несмотря на то что в операторных вершинах этой граф-схемы применяются умолчания, она достаточно хорошо понимается, так как в этом случае в отличие от ГСА2 нет необходимости помнить сохраняемые значения, а их можно определить, «поднимаясь» вверх по соответствующему пути в граф-схеме. По этой же причине в рассматриваемой ГС используется правая условная вершина с пометкой z_2, которая при необходимости может быть исключена.

Понимаемость этой граф-схеме придает также и то, что она обладает в отличие, например, от ГСА2 (рис. 13.2) глубиной, определяемой только одним переходом в исходном графе переходов. Кроме того, в этой ГС за один проход в отличие от граф-схемы на рис. 13.3 не приходится неоднократно пересчитывать значения ни одной выходной переменной, а в отличие от граф-схемы на рис. 13.5 — многократно проверять значения одной и той же входной переменной. Эта граф-схема не только структурирована в традиционном понимании, но и хорошо организована в том смысле, что в ней условные вершины с пометками Z и X не перемешаны между собой и с операторными вершинами.

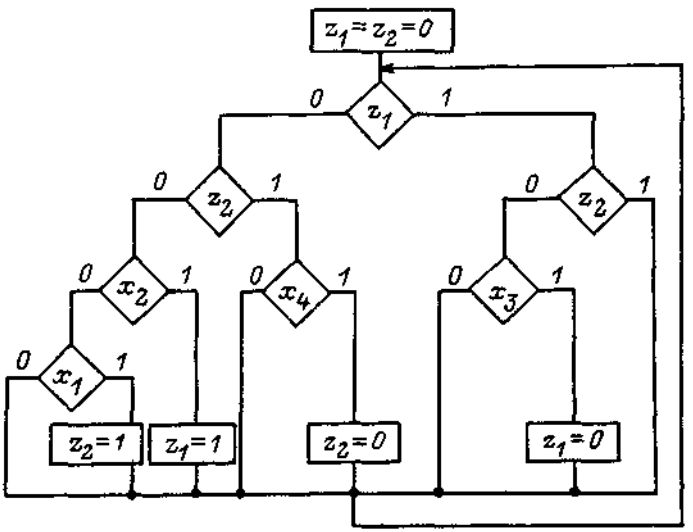

Рис. 13.18

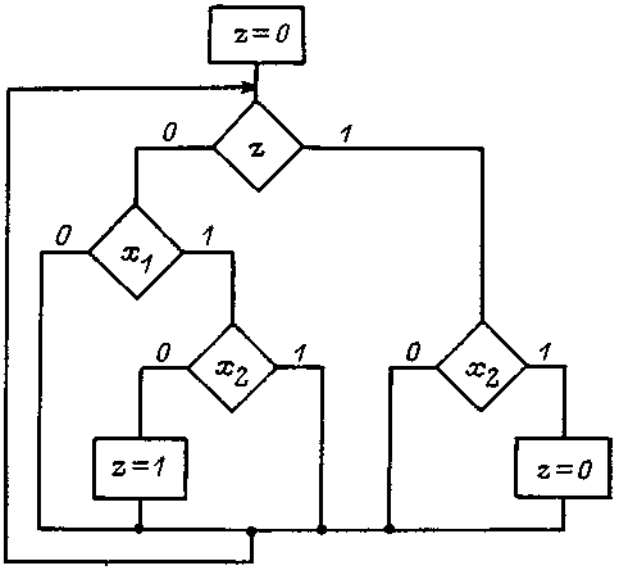

Рис. 13.19

Такая организация ГСА («настоящее состояние—условие перехода—следующее состояние») соответствует нормальному человеческому поведению, при котором, например, просыпаясь утром, человек сначала определяет свое внутреннее состояние («жив—мертв», «здоров—болен») и только потом «опрашивает» значения входных переменных (например, тепло или холодно на улице), а затем в зависимости от значений этих переменных переходит в следующее состояние. Ясно, что в этом случае поведение без учета внутреннего состояния, только по значениям входных переменных, будет выглядеть странным.

Однако при алгоритмизации с помощью ГСА без внутренних обратных связей понятие «состояние» обычно в явном виде не применяется и граф-схему строят начиная с опроса входных переменных и в зависимости от их значений формируют те или иные значения выходных, а возможно, и дополнительно вводимых внутренних переменных, которые определяют состояния косвенным образом.

Например, несмотря на то что ГСА2 (рис. 13.3), реализующая *R*-триггер, идеально соответствует принципам структурного программирования, она обладает двумя недостатками, затрудняющими ее чтение: пересчет значения z при $x_1 = x_2 = 1$ и отсутствие в модели указания в явном виде значения z при $x_1 = x_2 = 0$. Этих недостатков лишена ГСА4 с дешифратором состояний (рис. 13.19), которая, несмотря на большую сложность по сравнению с ГСА2, читает-

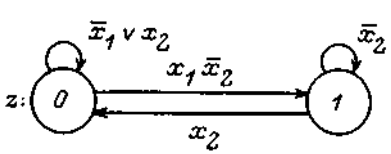

Рис. 13.20

ся проще. Простоту чтения и компактность изображения совмещает в себе граф переходов (рис. 13.20), который при программировании с помощью конструкции `switch` может быть еще более упрощен (умолчание пометок петель).

13.4.2. Реализация автоматов без выходного преобразователя с принудительно-свободным кодированием состояний

Пусть требуется реализовать счетный триггер, поведение которого описывается графом переходов (рис. 13.21). Для различения вершин этого графа используем принудительно-свободное кодирование состояний автомата, введя отсутствующую в алгоритме его функционирования внутреннюю переменную y (рис. 13.22). Название этого вида кодирования определяется тем, что значения части разрядов кода принудительно задаются значениями выходных переменных z_i, а значения остальных разрядов, обозначаемых переменными y_j, могут быть выбраны при программной реализации свободно. На рис. 13.23 приведена ГСА4, эквивалентная этому графу переходов.

Для рассматриваемой задачи ГСА может быть значительно упрощена за счет потери изоморфизма с графом переходов (рис. 13.22). Для этого запишем переходы и условия, при которых они происходят:

$$
\begin{array}{llcll}
1.\ z=0 & y=0 & \rightarrow & z=1 \quad y=0 & \text{при } x=1; \\
2.\ z=1 & y=0 & \rightarrow & z=1 \quad y=1 & \text{при } x=0; \\
3.\ z=1 & y=1 & \rightarrow & z=0 \quad y=1 & \text{при } x=1; \\
4.\ z=0 & y=1 & \rightarrow & z=0 \quad y=0 & \text{при } x=0.
\end{array}
$$

Исключим в кодах, определяющих вершины графа переходов, некоторые компоненты:

$$
\begin{array}{llcll}
1'. & y=0 & \rightarrow & z=1 & \text{при } x=1; \\
2'.\ z=1 & & \rightarrow & y=1 & \text{при } x=0; \\
3'. & y=1 & \rightarrow & z=0 & \text{при } x=1; \\
4'.\ z=0 & & \rightarrow & y=0 & \text{при } x=0.
\end{array}
$$

Выполненное упрощение корректно, так как новые соотношения непротиворечивы. Из изложенного следует, что в данном примере для различения вершин могут применяться не только все переменные, помечающие вершины, как это было показано выше, но и другой подход — для каждой вершины может быть выбрана лишь одна помечающая ее переменная, которая совместно со значением входной переменной, помечающей исходящую из этой вершины дугу, обеспечит корректный переход в графе переходов.

В этом случае вершины ГП различаются следующими кортежами значений переменных: $y = 0$, $x = 0$; $z = 1$, $x = 1$; $y = 1$, $x = 0$; $z = 0$, $x = 1$.

Следовательно, при таком подходе каждая вершина ГП характеризуется не кортежем значений переменных z и y, определяющим состояние автомата, а лишь его отдельной компонентой, рассматриваемой совместно со значением входной переменной [270].

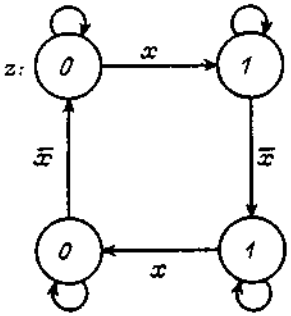

Рис. 13.21

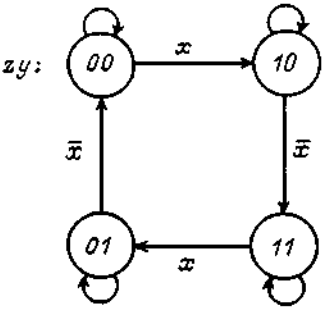

Рис. 13.22

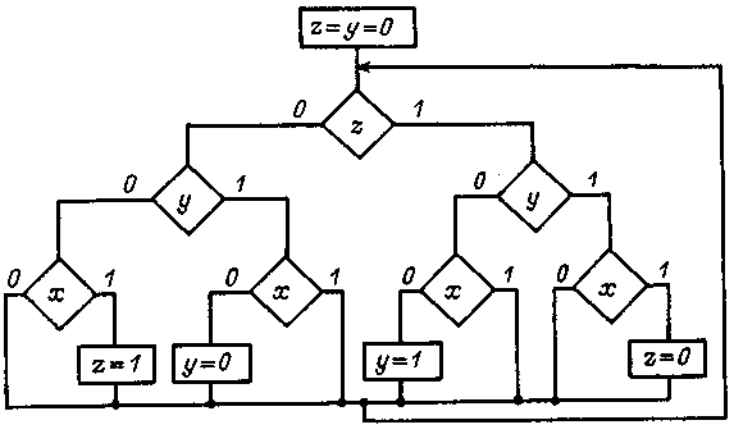

Рис. 13.23

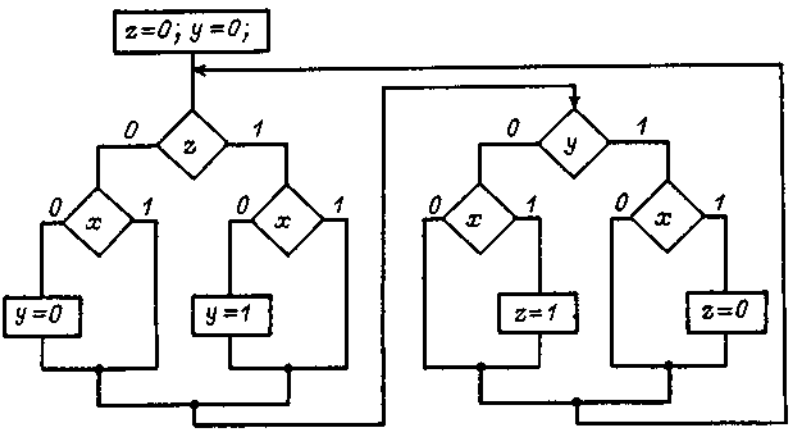

Рис. 13.24

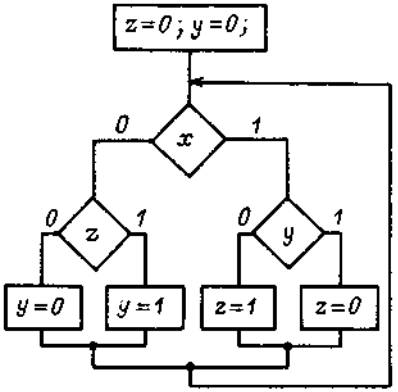

Рис. 13.25

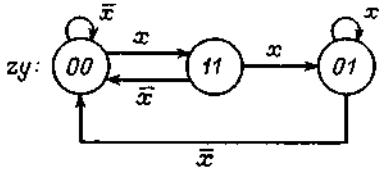

Рис. 13.26

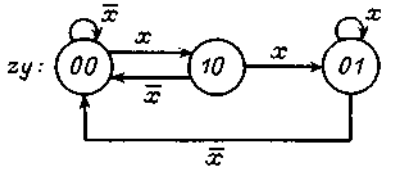

Рис. 13.27

Используя этот подход, объединим соотношения 2' и 4' по переменной z, а соотношения 1' и 3' по переменной y и построим на их основе ГСА (рис. 13.24), содержащую меньшее число вершин по сравнению с предыдущей граф-схемой. В построенной ГСА порядок расположения блоков, соединенных последовательно, может быть изменен.

Дальнейшее упрощение реализации счетного триггера достигается при применении входной переменной x для объединения между собой соотношений 2'—4' и 1'—3' (рис. 13.25). Эта же ГСА может быть построена формально по кодированной таблице переходов и выходов, используя канонический метод Блоха [18] при начальном порядке переменных x, y, z.

Полученная ГСА еще менее связана с состояниями автомата в целом и ориентирована на использование отдельных переменных.

Эта ГСА характерна для событийно-управляемого программирования, так как она начинается с дешифратора значений входного события, каждое из которых совместно со значением выходной (внутренней) переменной приводит к формированию определенного значения внутренней (выходной) переменной. ГСА (рис. 13.25) потеряла изоморфизм с ГП (рис. 13.22), и поэтому понять ее функционирование достаточно сложно.

При автоматном программировании, предлагаемом в настоящей работе, построение ГСА начинается с формирования дешифратора состояний, на каждом выходе которого устанавливается дешифратор значений входного события (рис. 13.23).

Такая ГСА, несмотря на громоздкость, весьма просто понимается, так как она изоморфна ГП (рис. 13.22).

Можно показать, что граф переходов на рис. 13.26 может быть реализован ГСА без проверки значений переменной z, в то время как граф переходов на рис. 13.27, отличающийся от предыдущего лишь кодированием вершин, не может быть реализован ГСА, в которой отсутствуют проверки всех переменных, используемых в ГП.

В заключение раздела отметим, что зависимость состояний автомата от значений выхода z в рассмотренных ГСА (рис. 13.18, 13.19 и 13.23) может создать трудности при условии, что эти значения могут изменяться

в других компонентах алгоритма управления. Поэтому сделаем независимыми состояния автомата от его выходных значений, перейдя к модели автомата Мура.

13.4.3. Реализация автоматов Мура с двоичным логарифмическим кодированием состояний

На рис. 13.28 приведен ГП автомата Мура, реализующий счетный триггер, для рассматриваемого варианта кодирования, а на рис. 13.29 — соответствующая ему ГСА. Тело этой ГСА является четырехслойным («дешифратор состояний—формирование значений выходов—реализация функций переходов—формирование следующего состояния»). Недостатки этой ГСА состоят в значительной глубине пирамидального дешифратора и большом числе вновь вводимых переменных y_j, где $0 < j < \lceil \log_2 s \rceil - 1$; s — число состояний автомата. Первый из этих недостатков устраняется в разд. 13.4.4, а оба недостатка — в разд. 13.4.5. ГСА на рис. 13.29 может быть упрощена за счет подъема операторных вершин типа $z = 0$ и $z = 1$ «вверх».

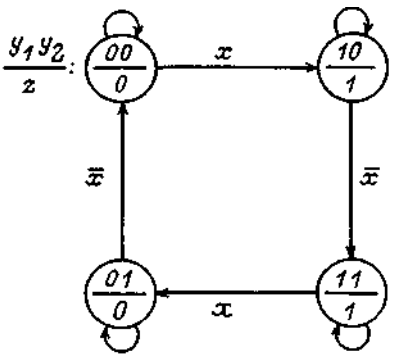

Рис. 13.28

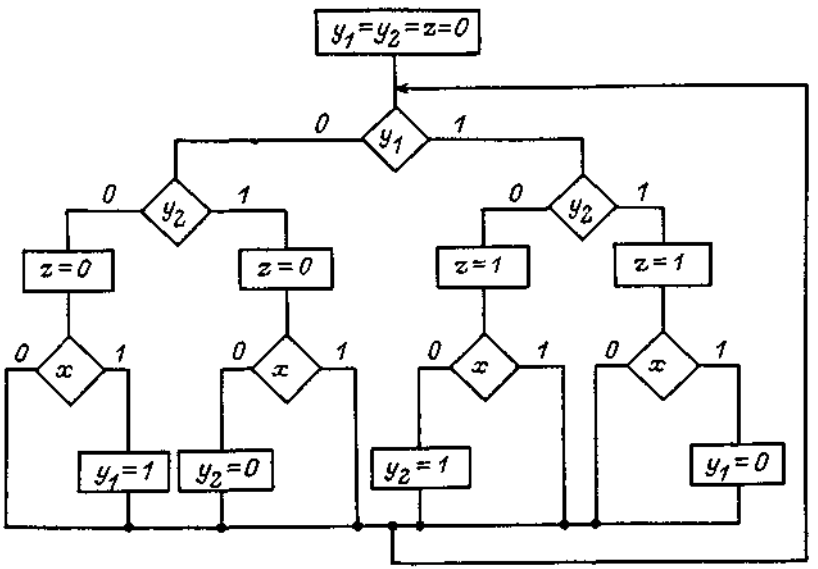

Рис. 13.29

13.4.4. Реализация автоматов Мура с двоичным кодированием состояний

Такой вариант кодирования вершин ГП автомата Мура (рис. 13.30), названный двоичным (в j-й вершине графа переходов только битовая переменная y_j принимает значение, равное единице, а в остальных вершинах ее значение равно нулю), позволяет использовать в ГСА (рис. 13.31) линейный дешифратор состояний вместо пирамидального, который применялся в ГСА, соответствующих автоматам Мура при других видах кодирования. Недостаток ГСА в этом случае состоит в еще большем (по сравнению с предыдущим случаем) числе вновь вводимых битовых переменных y_j ($0 < j < s$ - 1), каждую из которых приходится не только устанавливать, но и, к сожалению, принудительно сбрасывать.

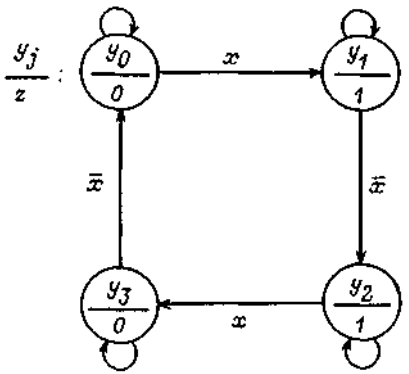

Рис. 13.30

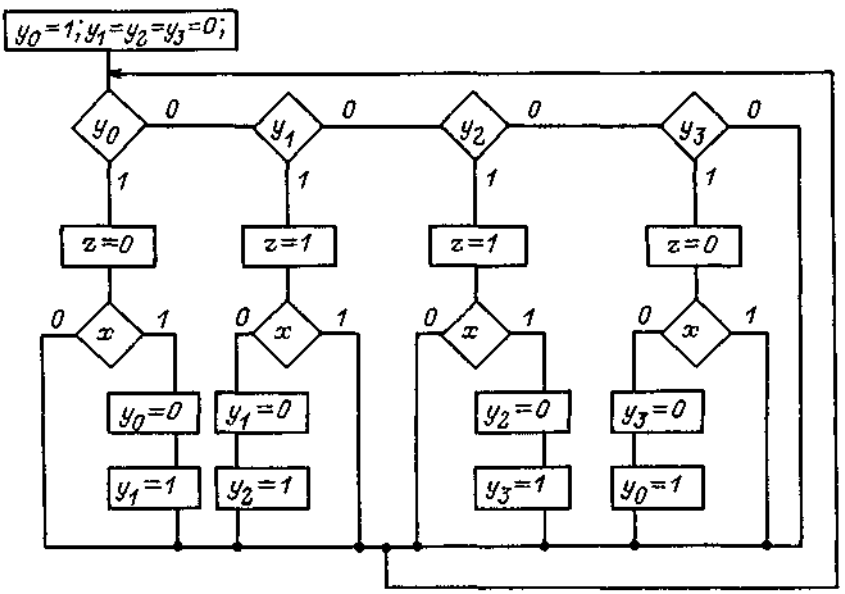

Рис. 13.31

13.4.5. Реализация автоматов Мура с многозначным кодированием состояний

Все недостатки предыдущих вариантов кодирования устраняются при переходе к многозначному кодированию состояний автоматов. При этом для автоматов Мура, Мили и смешанных автоматов, реализуемых в виде одной компоненты, для кодирования состояний достаточно иметь одну переменную Y значности s, которую не требуется принудительно сбрасывать, так как это происходит автоматически при переходе от ее одного значения к другому. При этом для реализации N-компонентного алгоритма управления (реализуемого N графами переходов) с помощью указанных классов автоматов требуется N многозначных переменных Y_j, где $j = 0, ..., N - 1$. Этот вариант кодирования, практически не реализуемый при аппаратной реализации автоматов, по мнению автора, при отсутствии жестких ограничений на объем памяти программ и возможности работы с многозначными переменными должен стать основным при программной реализации автоматов.

На рис. 13.32 в качестве примера приведена ГСА для рассматриваемого варианта кодирования, построенная по ГП автомата Мура счетного триггера (рис. 13.7). В этой граф-схеме за счет ухудшения читаемости условная вершина с пометкой $Y = 3$ и вторая операторная вершина с пометкой $z = 1$ могут быть исключены.

ГСА аналогичной структуры может быть построена для любого ГП автомата Мура. При этом в отличие от графа переходов такая ГСА всегда может быть планарной. Применение всего лишь одной промежуточной переменной при «хорошей» организации структуры делает использование ГСА этого типа весьма привлекательным. Единственный недостаток такого типа структур помимо громоздкости (если умалчивание значений выходных переменных исключено) состоит в том, что визуальное обнаружение генерирующих контуров становится весьма затруднительным, так как связь в таких ГСА между фрагментами осуществляется не непо-

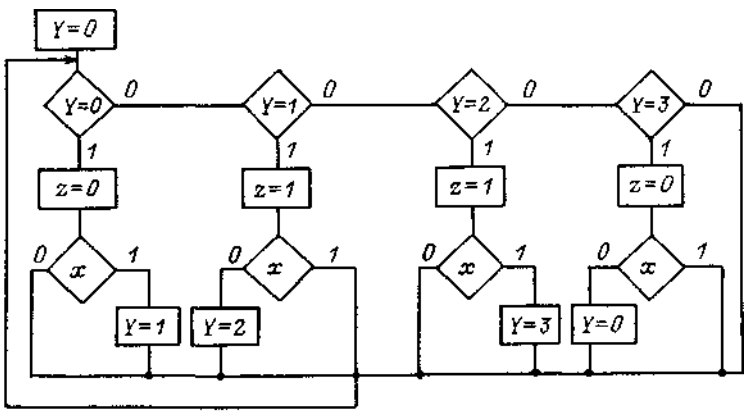

Рис. 13.32

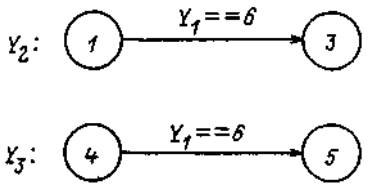

Рис. 13.33

средственно, как в графах переходов, а по данным (значениям переменной Y).

Из изложенного следует, что если в качестве языка общения по каким-либо причинам выбрана ГСА без внутренних обратных связей, то в этом случае целесообразно применять граф-схему с дешифратором многозначно закодированных состояний.

При использовании многозначного кодирования открывается весьма эффективная в изобразительном отношении дополнительная возможность обеспечения связей между компонентами алгоритма управления (в том числе и при реализации параллельных процессов) за счет обмена десятичными значениями применяемых при взаимодействии номеров состояний компонент (возможность взаимодействия по входным, выходным и дополнительно вводимым внутренним переменным сохраняется). Пусть, например, требуется, чтобы вторая компонента алгоритма (автомат Y_2) перешла из первого состояния в третье, а третья компонента (автомат Y_3) — из четвертого состояния в пятое при условии, что первая компонента (автомат Y_1) находится в шестом состоянии. Тогда без введения дополнительных переменных этот параллельный процесс весьма наглядно отражается фрагментом СВГП, представленным на рис. 13.33.

13.4.6. Реализация автоматов Мили с многозначным кодированием состояний

В автоматах Мили, так же как и в автоматах Мура, могут использоваться различные виды кодирования состояний, однако, исходя из изложенного, рассмотрим только многозначное кодирование. На рис. 13.8 приведен ГП автомата Мили, реализующий счетный триггер при принятом

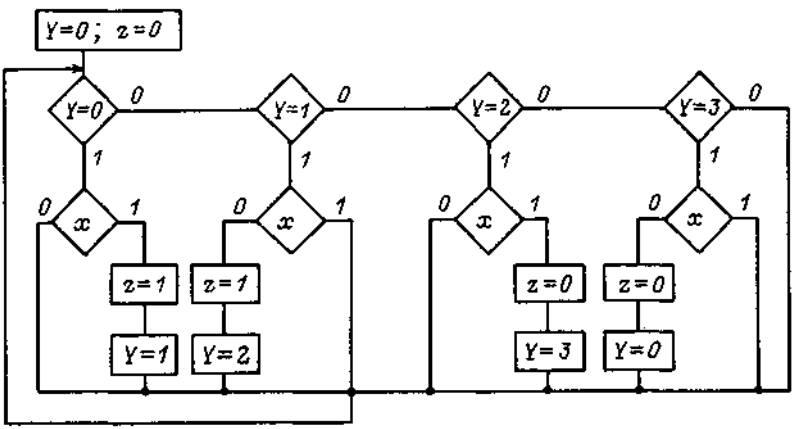

Рис. 13.34

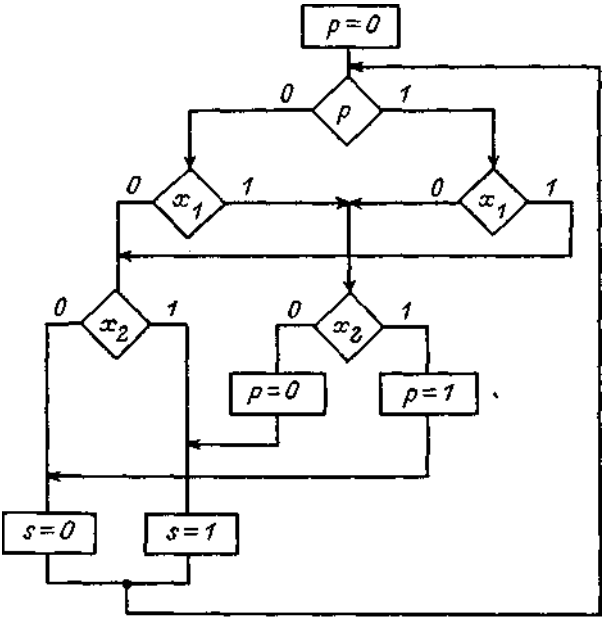

Рис. 13.35

варианте кодирования, а на рис. 13.34 — соответствующая ГСА4. В этой граф-схеме могут быть исключены условная вершина с пометкой $Y = 3$ и вторые операторные вершины с пометками $z = 1$ и $z = 0$. Отличие этой граф-схемы от ГС автомата Мура состоит в размещении слоя операторных вершин, помеченных значениями выходной переменной, не перед дешифраторами значений входных переменных, а после них.

Второй пример реализации этого класса автоматов приведен на рис. 13.35 для последовательностного сумматора (рис. 4.113). В этой граф-схеме для минимизации числа вершин слой значений выходной переменной расположен после слоя со значениями следующего состояния автомата. Тело этой граф-схемы, состоящее всего лишь из девяти вершин, построено с помощью модификации метода Блоха [84].

13.4.7. Реализация смешанных автоматов с многозначным кодированием состояний

На рис. 13.36 изображено тело ГСА, реализующей С-автомат с флагом, представленный на рис. 13.9. Эта граф-схема является пятислойной: «дешифратор состояний—формирование значений выходных переменных—реализация функций переходов—формирование значений флага—формирование следующего состояния». В приведенной граф-схеме переменная x_1 определяет содержимое счетного триггера (другой компоненты алгоритма управления), который также управляется и от другого источ-

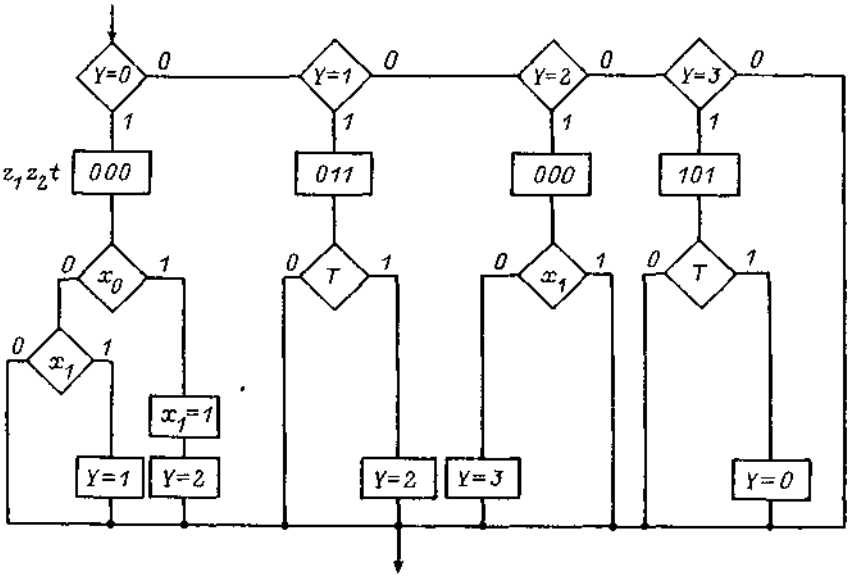

Рис. 13.36

ника информации — кнопки. В этой ГСА в отличие от графа переходов противоречивость устранена ортогонализацией и применяется булев признак T вместо неравенства $t' \geqslant D$.

При этом необходимо отметить, что если при использовании системы взаимосвязанных графов переходов каждая компонента замкнута, то для граф-схем, реализуемых в ПЛК, замыкание выполняется для алгоритма управления в целом.

13.5. Сравнение предлагаемого подхода с методом построения структурированных ГСА по Ашкрофту и Манне

Универсальный метод построения структурированных граф-схем алгоритмов (СГСА) был предложен Ашкрофтом и Манной [104]. Метод по неструктурированной ГСА1 или ГСА2 строит структурированную ГСА4 с многозначным кодированием состояний.

Однако, по мнению автора настоящей работы, этот метод обладает рядом недостатков, не позволяющих получать хорошо читаемые ГСА, а именно:

— в явном виде не используется понятие «состояние»;

— не делается различия по типам автоматов и типам применяемых переменных;

— используется только один тип кодирования фрагментов неструктурированной ГСА при объединении в каноническую структуру;

— из-за связи фрагментов СГСА по данным визуально весьма сложно обнаружить генерирующие «контуры»;

— глубина СГСА не ограничена одним переходом, что может не позволить использовать значения некоторых ее внутренних переменных в других компонентах алгоритма;

— при применении в качестве исходной ГСА1 в ней не устраняется неоднозначность значений выходов;

— при использовании в качестве исходной ГСА2 в ней не только не устраняется неоднозначность значений выходов, но, кроме того, она может содержать большое число битовых промежуточных и флаговых переменных (в том числе с умалчиваемыми значениями), которые не исчезают при структурировании и к которым добавляется еще одна многозначная переменная, соответствующая номерам выделенных структурированных фрагментов;

— человек привыкает и понимает исходную форму представления алгоритма управления и обычно психологически не готов к работе с другими формами его представления;

— метод ориентирован на использование языков высокого уровня.

Так как указанный метод предназначен для построения структурированных ГСА, то поэтому если исходная ГСА уже структурирована, к ней этот метод применяться не должен. По этой причине если, например, в качестве исходной задана СГСА2 (рис. 13.3), то, исходя из изложенного, она не должна подвергаться дальнейшим преобразованиям. Однако эта ГСА «плохо» читается, и вместо нее для целей общения целесообразно использовать ГСА (рис. 13.19) или ГП (рис. 13.20).

Из философии настоящей работы следует, что в целом идея, допускающая построение исходной неструктурированной ГСА, применяемая в структурном программировании, порочна, так как, по мнению автора, незачем сначала строить неструктурированную ГСА, чтобы потом ее структурировать. Для автоматов с памятью следует либо сразу строить структурированную ГСА с дешифратором состояний для принятого варианта кодирования, как предложено в настоящей работе, либо, что более целесообразно, проводить исходное описание в виде графа переходов для выбранного типа автомата, в котором по возможности должны отсутствовать флаги и умалчиваемые значения выходных переменных и который изоморфно реализуется, например с помощью управляющей конструкции switch языка СИ, производящей одновременно расцикливание и структурирование и обеспечивающей доступ к любому значению многозначной переменной, кодирующей состояния автомата. Такой подход, как будет показано в гл. 14, применим и для языков низкого уровня, например языков инструкций ПЛК.

Введение понятия «состояние» делает каждую компоненту программы «наблюдаемой» изнутри (с помощью одной многозначной переменной), а не только по входам и выходам. В получаемые программы легко могут быть внесены изменения. Они достаточно просто проверяются (в качестве теста в этом случае может использоваться ГП) и ввиду изоморфизма с первичным описанием легко верифицируются.

Графы переходов по форме изображения являются в общем случае «существенно плоскостными», что позволяет отражать алгоритмы управления в более естественной форме, чем в виде граф-схемы или диаграммы «Графсет», которые по стандартам обычно изображаются в форме, при-

ближающейся к «линейной», в направлении сверху вниз. Такая форма представления соответствует последовательному книжному изображению и чтению текстов, но не соответствует плоскостному (параллельному) изображению картин, более удобных для восприятия человеком. При этом естественно, что графы переходов должны быть в максимальной степени планарными. Они являются более ранней «сущностью» теории алгоритмизации (теории автоматов) по сравнению с другими, рассмотренными в настоящей работе, и поэтому в соответствии с принципом Оккама (разд. 12.3) можно утверждать, что для многих задач логического управления необходимость использования новых «сущностей» не наступает.

При этом отметим также, что графы переходов в отличие от таблиц переходов, оперирующих с минтермами и содержащих обычно большое число пустых клеток, существенно более обозримы и практически применимы для задач большой размерности. Основное достоинство графов переходов, позволяющее описывать задачи такой размерности, состоит в том, что если граф ортогонализован, то переход между двумя вершинами определяется не всеми входными переменными, упоминаемыми в пометках всех дуг графа, как это имеет место при использовании таблиц переходов, а только теми входными переменными, которые помечают дугу между указанной парой вершин, — свойство локальности описания.

При устранении противоречий не ортогонализацией, а расстановкой приоритетов умолчание обозначений некоторых входных переменных ухудшает читаемость графов переходов, так как для дуг с меньшими приоритетами, исходящих из некоторой вершины, не удается сразу определить (прочесть) перечень всех переменных, от которых зависит каждый переход по этим дугам. Для определения указанного перечня в этом случае приходится анализировать пометки всех дуг, исходящих из рассматриваемой вершины, а значит, локальность описания уменьшается.

По мнению автора, при современном уровне развития элементной базы при программной реализации задач логического управления ответственными объектами основным критерием построения программ должна являться их хорошая понимаемость и как следствие безошибочность, а остальные критерии должны быть вспомогательными (хотя для ПЛК их ограниченные ресурсы могут определять выбор и других критериев в процессе алгоритмизации). Поэтому автор надеется, что настоящая работа позволит графам переходов занять подобающее им место при программной реализации рассматриваемого класса задач.

Предлагаемый подход может быть использован не только для реализации логико-временных, но и для логико-вычислительных алгоритмов. На рис. 11.29 в качестве примера приведена ГСА1, реализующая алгоритм определения наибольшего общего делителя двух целых положительных чисел M и N (алгоритм Евклида). Эта ГСА является не автоматной, а логико-вычислительной. Она может программироваться с помощью подхода, предлагаемого в настоящей работе по графу переходов, описывающему логико-вычислительный процесс и являющемуся автоматом Мили с флагами (рис. 11.30). Необходимо отметить, что этот граф существенно компактнее, чем соответствующая граф-схема алгоритма.

13.6. Программирование графов переходов и ГСА с многозначным кодированием состояний в базисе языков высокого уровня

Использование многозначного кодирования позволяет для ГСА4, так же как и для графа переходов, изоморфно переходить к тексту программы, минуя построение промежуточных граф-схем. При этом, несмотря на то что графы переходов содержат петли и контуры (негенерирующие), их не приходится принудительно расцикливать и структурировать, а эти проблемы решаются при программировании выбором соответствующей управляющей конструкции.

Наиболее просто это достигается при применении такой управляющей конструкции, как например оператор switch языка СИ [225]. Приведем в качестве примера программу, использующую этот оператор для реализации ГП автомата Мура (рис. 13.7) и ГСА4 (рис. 13.32):

```
            switch (Y) {
case 0:     z = 0;
            if(x)       Y = 1;
            break;
case 1:     z = 1;
            if(x̄)       Y = 2;
            break;
case 2: /*z = 1;*/
            if(x)       Y = 3;
            break;
case 3:     z = 0;
            if(x̄)       Y = 0;
            break;        }.
```

Приведенная программа обладает замечательным свойством: она с гарантией делает только то, что описывает граф переходов, и не делает ничего лишнего, что легко проверяется сверкой текста программы с ГП, так как при безошибочном переходе от графа к программе должен быть обеспечен их полный изоморфизм. Этим свойством обладают не все алгоритмические модели. Так, например, если автомат с s состояниями реализуется системой m булевых формул ($m =]\log_2 s[$), а $s \neq 2^m$, то эта система реализует другой автомат с числом состояний 2^m, в котором исходный автомат является лишь ядром построенного, а переходы из новых ($2^m - s$) состояний в заданные зависят от принятого варианта доопределения.

В приведенной программе в состоянии «2» (case 2) значение выходной переменной, не изменяющееся при переходе из предыдущего состояния, закомментировано. Это, с одной стороны, уменьшает объем памяти, а с другой — сохраняет хорошую читаемость программы. В этой программе каждый оператор break осуществляет передачу управления за закрывающуюся фигурную скобку, что при невыполнении условия в операторе if сохраняет предыдущее состояние, а при его выполнении — не позволяет выполнить более одного перехода в графе переходов. При этом для СВГП

за один программный цикл в каждом графе реализуется не более одного перехода. Это делает доступным любое состояние рассматриваемого автомата для других компонент алгоритма управления вне зависимости от значений входных переменных в отличие, например, от ГСА2 (рис. 13.3), в которой при $x_1 = x_2 = 1$ значение $z = 1$ для других компонент недоступно.

Так, например, если первая компонента алгоритма реализована указанным образом и значение переменной Y_1 равное шести, доступно другим компонентам, то фрагменту СВГП (рис. 13.33) соответствует следующий фрагмент программы, использующий значение шестого состояния первой компоненты в качестве условия перехода в двух других компонентах:

```
        switch (Y2) {
        ...
case 1: if(Y1 = = 6)  Y2 = 3;
        ...          }
        switch (Y3) {
        ...
case 4: if(Y1 = = 6)  Y3 = 5;
        ...          }.
```

Из изложенного следует весьма неприятный факт, состоящий в том, что особенности программной реализации влияют не только на чтение текста программы, но и на чтение спецификации — ГП или СВ ГП. Для обеспечения универсальности, в том числе и для графов переходов с неустойчивыми вершинами, целесообразно считать, что в каждом графе за один проход выполняется не более одного перехода, что на программном уровне поддерживается указанным образом.

Приведем теперь пример программы, построенной по ГП автоматами (рис. 13.8) и ГСА4 (рис. 13.34), предполагая, что в начале $Y = 0$; $z = 0$:

```
        switch (Y){
case 0: if(x)       {  z = 1;   Y = 1;}
        break;
case 1: if(x̄)       {/*z = 1;*/ Y = 2;}
        break;
case 2: if(x)       {  z = 0;   Y = 3;}
        break;
case 3: if(x̄)       {/*z = 0;*/ Y = 0;}
        break;      }.
```

С помощью управляющей конструкции `switch` также эффективно реализуются и С-автоматы.

Из изложенного следует, что граф переходов, во-первых, может одновременно применяться в качестве спецификации как алгоритма управления, так и программы, а во-вторых, с помощью управляющей конструкции `switch` он изоморфно отражается в текст структурированной программы, которая может быть наблюдаемой не только по двоичным входам и выходам, но и, что самое главное, по десятичным номерам состояний. При этом для проверки программы на дисплей для каждого

графа переходов АМ, АМИ и СА достаточно вывести только одну десятичную внутреннюю переменную (сигнатуру). Это позволяет при исследовании поведения одного автомата в динамике следить только за значениями одной переменной (предварительно определив, например для автомата Мура, соответствие между номером состояния и значениями выходных переменных в этом состоянии), а не за многими двоичными внутренними и выходными переменными, как это делается при традиционном подходе.

Это позволяет ввести в программирование понятие «наблюдаемость» по аналогии с понятием «измеримость» в теории линейных систем (гл. 1).

Получаемые таким образом программы обладают высокой модификационной способностью (в некотором смысле легко управляемы) и хорошо читаются при условии, что значения выходных переменных задаются однозначно, так как в этом случае они не зависят от глубокой предыстории. Графы переходов автоматов с флагами также изоморфно отражаются в тексты программ с прмощью конструкции switch, однако зависимость следующего состояния от предыстории резко уменьшает их модификационную способность.

Отметим также, что основная причина простоты внесения изменений в программы, построенные указанным образом по графам переходов, состоит в том, что в ГП вершины связаны между собой непосредственно в отличие от граф-схем, в которых операторные вершины в общем случае соединены через условные вершины, и поэтому изменение в одном переходе оказывает существенное влияние на структуру граф-схемы.

При использовании предлагаемого подхода верификация программы может производиться сверкой ее текста с графом переходов. Для однокомпонентных алгоритмов граф переходов без флагов и умолчаний может применяться в качестве теста для сертификации программы, а для многокомпонентных алгоритмов (по аналогии с сетями Петри) по СВГП должен строиться граф достижимых маркировок (как, впрочем, и для ГП в общем случае), позволяющий исследовать все функциональные возможности системы и представить ее поведение одним графом переходов. Для независимых параллельных процессов строить единый граф переходов нет необходимости, так как их реализации могут проверяться по отдельности.

13.7. Программирование ГСА с внутренними обратными связями в базисе языков высокого уровня

Подход, излагаемый в настоящей работе, позволяет предложить метод программирования граф-схем алгоритмов с внутренними обратными связями, которые в многозадачном режиме использования применяемого вычислительного устройства не реализуются традиционным путем без предварительного расцикливания.

Суть предлагаемого метода состоит в том, что по заданной ГСА строится эквивалентный ей граф переходов, который затем изоморфно отражается в текст программы с помощью конструкции switch.

Пусть, например, задана ГСА с внутренними обратными связями (рис. 13.10). По этой ГСА может быть построен ГП автомата Мили

(рис. 13.11) или ГП автомата Мура (рис. 13.12), каждый из которых может быть однозначно реализован конструкцией `switch`. Несмотря на то что получающиеся при этом программы весьма компактны, эти графы переходов содержат генерирующие контуры и их чтение весьма затруднительно из-за неоднозначности значений выходных переменных. Эти недостатки отсутствуют в программе, построенной по преобразованному ГП автомата Мура (рис. 13.16):

```
          switch (Y) {
case 0:   z1 = 0;  z2 = 0;  z3 = 0;
          if(x1&x3)          Y = 1;
          if(x1&x2&x3)       Y = 2;
          break;
case 1:   z1 = 1;/*z2 = 0;  z3 = 0;*/
          if(x̄3)             Y = 0;
          if(x3)             Y = 1; break;
case 2: /*z1 = 0;*/z2 = 1;/*z3 = 0;*/
          if(x̄3)             Y = 0;
          if(x3)             Y = 4;
case 3: /*z1 = 0;*/z2 = 0;/*z3 = 1;*/
                             Y = 0; break;
case 4: /*z1 = 0;  z2 = 1;*/z3 = 1;
          if(x̄3)             Y = 3;
          break;
case 5: /*z1 = 1;  z2 = 0;*/z3 = 1;
          if(x̄3)             Y = 0;
          break;     }.
```

Число сзрок в этой программе равно $s + d + 1$, где d — число дуг (включая петли) в графе переходов. В метке `case 0` оператор `if`, соответствующий дуге с большим приоритетом, исходящей из вершины «0», размещается ниже оператора if, соответствующего дуге с меньшим приоритетом. При такой реализации проблемы расцикливания и структурирования исходной ГСА решаются автоматически в ходе построения программы. При этом отметим, что построение графа переходов по ГСА с внутренними обратными связями не является необходимым. Для написания программы в этом случае достаточно выполнить многозначное кодирование операторных вершин заданной ГСА (для построения программы, соответствующей автомату Мура) или точек, следующих за операторными вершинами (для построения программы, соответствующей автомату Мили).

Быстродействие предыдущей программы может быть повышено при усложнении ее структуры:

```
          switch (Y) {
case 0: z1 = 0;  z2 = 0; z3 = 0;
          if(x1&x2&x3)     {Y = 2; break; }
          if(x1&x3)         Y = 1;
          break;
```

```
case 1:    z1 = 1;/*z2 = 0;   z3 = 0;*/
           if(x̄3)              Y = 0;
           if(x3)              Y = 1; break;
case 2: /*z1 = 0;*/z2 = 1;/*z3 = 0;*/
           if(x̄3)              Y = 0;
           if(x3)              Y = 4; break;
case 3: /*z1 = 0;*/z2 = 0;/*z3 = 1;*/
                               Y = 0; break;
case 4: /*z1 = 0;  z2 = 1;*/z3 = 1;
           if(x̄3)              Y = 3;
           break;
case 5: /*z1 = 1;  z2 = 0;*/z3 = 1;
           if(x̄3)              Y = 0;
           break;      }.
```

В этой программе фрагмент, соответствующий дуге с большим приоритетом, исходящей из нулевой вершины, располагается выше фрагмента, соответствующего дуге с меньшим приоритетом, исходящей из той же вершины.

В качестве второго примера рассмотрим ГСА с внутренними обратными связями (рис. 13.2) и построим эквивалентный ей ГП автомата Мура (рис. 13.37). В этом графе, входящем в состав управляющего автомата,

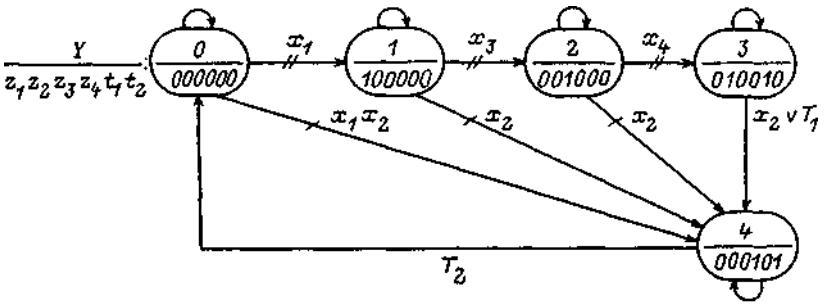

Рис. 13.37

каждой единице на позициях t_1 и t_2 соответствует обращение, например к процедуре `_time (i, D)`, где D — время задержки *i*-го ФЭЗ. Этот граф переходов, так же как и предыдущий, реализуется с помощью конструкции `switch`.

13.8. Программная реализация ГСА

Пусть задан граф переходов (рис. 4.15), реализующий *R*-триггер, в котором отсутствуют вершины с входящими и выходящими (за исключением петель) дугами, помеченными неортогональными условиями. В этом графе по умолчанию считается, что пометки дуг обеспечивают полноту вершин.

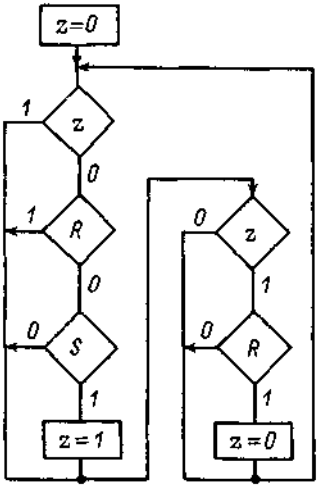

Рис. 13.38

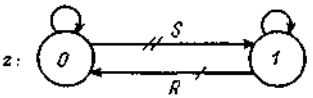

Рис. 13.39

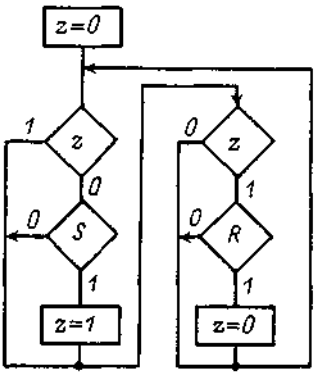

Рис. 13.40

Построим по этому графу переходов ГСА (рис. 4.72) с внутренними обратными связями. Для этой ГСА характерно, что ввод переменных происходит в каждой условной вершине, а вывод — в каждой операторной.

Такая ГСА не может быть реализована вычислительным устройством, в котором считывание выполняется только в конце программного цикла, а ввод информации — либо только в конце, либо только в начале цикла. Кроме того, если в вычислительном устройстве реализуется более одной ГСА, причем по крайней мере одна из них содержит хотя бы одну внутреннюю обратную связь, то до выхода из цикла, который может наступить не скоро, выполнить другие ГСА невозможно. Поэтому построим по графу переходов (рис. 4.15) граф-схему алгоритмов без внутренних обратных связей (рис. 4.83).

Для этой граф-схемы характерно, во-первых, наличие дешифратора состояний, во-вторых, разделение условных и операторных вершин и, в-третьих, то что каждый путь в ней соответствует одному переходу в графе переходов. Эта ГСА является алгоритмом реализации алгоритма управления с помощью вычислительного устройства.

Если необходимо учесть особенности системы команд используемого языка программирования, то ГСА должна еще один раз быть преобразована. На рис. 13.38 приведена линеаризованная и структурированная ГСА, число вычислительных блоков в которой равно числу дуг (без петель) в графе переходов (рис. 4.15). В этой ГСА условные и операторные вершины «перемешаны». Так как в графе перехода неортогонализованные условия на входящих и исходящих дугах в каждой вершине отсутствуют, а за программный цикл значения входных переменных не изменяются, то эта ГСА за один такой цикл не может реализовать более одного перехода в ГП.

Рассмотрим граф переходов (рис. 13.39), в котором входящие и исходящие дуги не ортогонализованы. В этом графе на дугах расставлены приоритеты, которые будут учитываться при построении ГСА. При этом, так же как и при расстановке приоритетов дуг, исходящих из одной вершины, используется следующая символика: чем выше приоритет, тем меньше штрихов изображается на дуге.

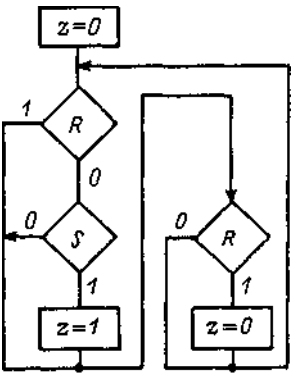

Рис. 13.41

Этот граф переходов некорректно реализует R-триггер, так как при его чтении предполагается, что в течение одного программного цикла реализуется не более одного перехода в графе. Поэтому при $R = S = 1$ считается, что на выходе кратковременно будет появляться единичный сигнал ($z = 1$), что не должно происходить в R-тригтере.

Несмотря на это, построим по этому графу переходов линеаризованную и структурированную ГСА (рис. 13.40), в которой блок, соответствующий дуге с высшим приоритетом, расположен ниже блока, соответствующего дуге с более низким приоритетом. Выполним верификацию этой ГСА, построив по ней кодированную таблицу переходов, а по ней в свою очередь граф переходов (рис. 4.15). Из рассмотрения этого графа следует, что построенная по недостаточно корректной спецификации граф-схема реализует требуемый алгоритм.

Вариант реализации, рассмотренный последним, является весьма опасным, так как в течение одного программного цикла может происходить фильтрация промежуточных значений переменных. Именно по этой причине и пришлось выполнять верификацию ГСА.

Приведенные выше ГСА построены непосредственно по графам переходов. Если использовать существенно более трудоемкие табличные методы и перебор вариантов [18, 19], то для задач небольшой размерности в ряде случаев могут быть найдены более эффективные реализации. На рис. 4.76 приведена граф-схема алгоритма, построенная с помощью метода Блоха [18] по кодированной таблице переходов с порядком переменных $R, S, z(y)$, а на рис. 13.41 приведена соответствующая линеаризованная граф-схема.

Еще более трудоемкий метод — модифицированный метод Блоха [84] строит по кодированной таблице переходов при порядке переменных S, R, $z(y)$ непосредственно линеаризованную граф-схему алгоритма (рис. 4.79). Эта ГСА является простейшей для рассматриваемого автомата в классе линеаризованных граф-схем с условными вершинами.

Если условные вершины не использовать, то ГСА для R-триггера при реализации по булевой формуле имеет линейную структуру (рис. 4.73).

Рассмотренный пример обладает тремя специфическими особенностями:

— для кодирования состояний не требуется применения промежуточных переменных типа «Y»;

— имеется возможность ортогонализации входящих и исходящих дуг;

— число дуг (без петель) и число вершин совпадают.

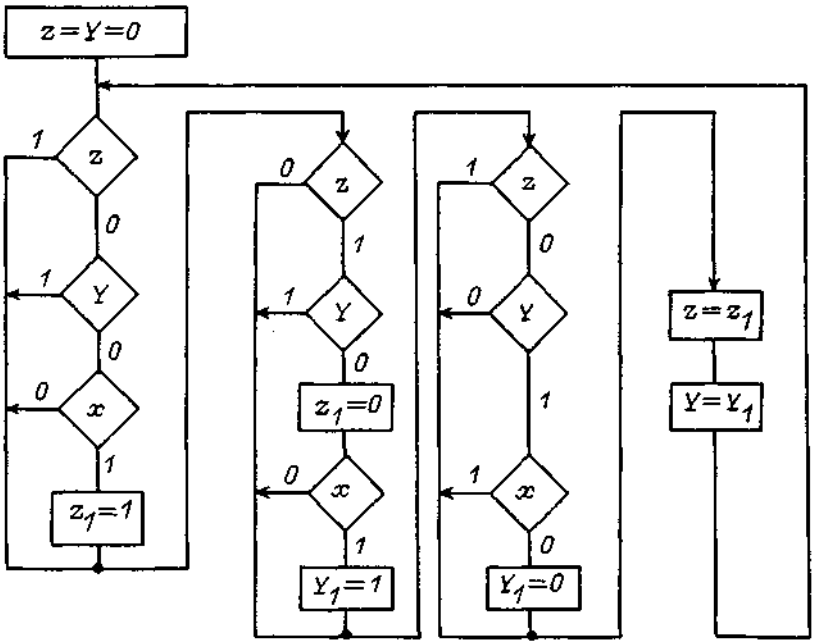

Рис. 13.42

Рис. 13.42

Рассмотрим более общий случай (рис. 13.27), в котором указанные ограничения сняты. Этот граф переходов при реализации линеаризованной ГСА (ЛГСА) требует переобозначения переменных, так как в противном случае при $z = Y = 0$ и $x = 1$ значения $z = 1$, $Y = 0$ будут отфильтрованы. В этом случае, как и в предыдущем, ЛГСА может быть построена по переходам (дугам), но так как число вершин в графе переходов меньше числа дуг, то более целесообразно построить ЛГСА по вершинам (рис. 13.42).

Приведенная ГСА может быть упрощена, если вместо полных кодов состояний использовать неполные коды. При этом могут быть исключены условные вершины с пометкой Y во втором и третьем вычислительных блоках в ГСА на рис. 13.42.

Последний пример обладает спецификой: в каждой вершине все исходящие дуги были ортогонализованы. Если в заданном графе переходов (рис. 13.43) вместо ортогонализации на исходящих из вершины дугах расставлены приоритеты, то при построении ЛГСА (рис. 13.44) по вершинам и «неполном» кодировании состояний (рис. 13.43) они должны учитываться на основе правила: чем выше приоритет дуги, тем ниже соответствующий подблок располагается в ЛГСА.

Предложенный подход позволяет формально получать граф-схемы алгоритмов, эквивалентные по поведению и изоморфные по структуре с заданными графами переходов.

Интересные возможности при построении ГСА открываются при реализации счетного триггера, поведение которого задается графом пере-

ходов (рис. 13.22). Если этот граф реализовать не непосредственно, а по кодированной таблице переходов и выходов, то появляется возможность построения ГСА (рис. 13.25), в которой применяется минимально возможное число условных вершин z и y, определяющих состояния автомата и порождающих совместно с условной вершиной x правильные переходы.

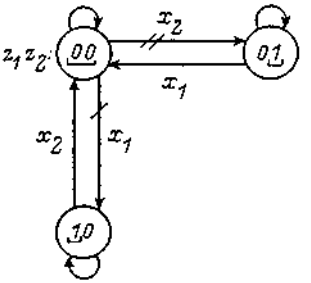

Рис. 13.43

Последний ГП при использовании многозначного кодирования приведен на рис. 4.124. Этот граф реализуется структурированной плоскостной ГСА на рис. 13.32, которая по структуре совпадает с канонической граф-схемой, предложенной Ашкрофтом и Манной [104]. Однако эти авторы предлагают строить структурированную граф-схему по неструктурированной, что весьма трудоемко, в то время как в настоящей работе ее предлагается строить непосредственно по графу переходов, что существенно проще.

Полученная ГСА весьма эффективно «линеаризуется». Получающаяся ЛГСА (рис. 13.45) изоморфна графу переходов на рис. 4.124. Применение переобозначения переменных, необходимое в общем случае, в данном примере не требуется (рис. 13.46), так как за один программный цикл значение входной переменной не изменяется, а в каждой вершине входящая и выходящая дуги имеют ортогональные пометки.

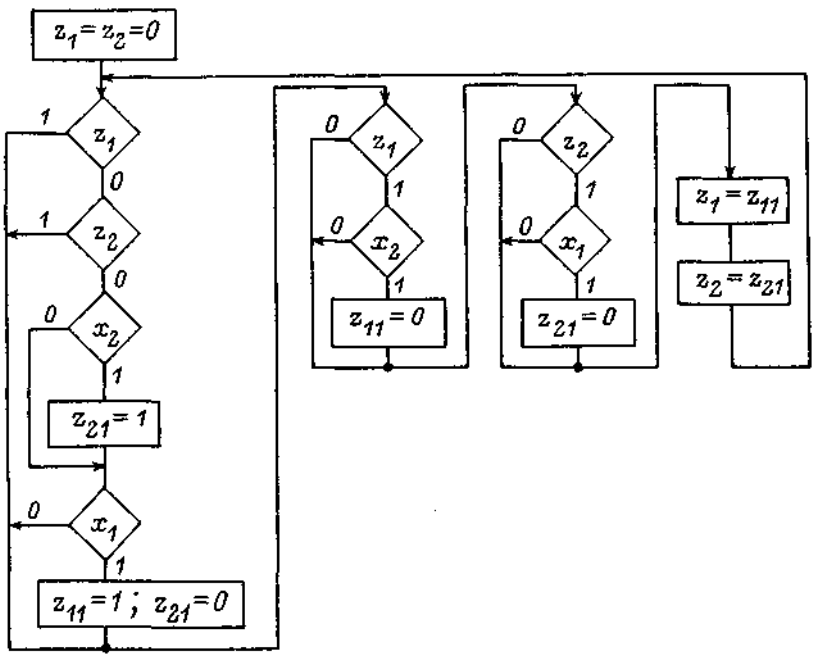

Рис. 13.44

Для ГП автоматов Мили операторные вершины, в которых присваиваются значения выходных переменных, располагаются в ГСА после проверки значений входных переменных, а для графов переходов С-автоматов эти вершины размещаются как до, так и после проверки значений входных переменных.

Учитывая изоморфизм между графом переходов и линеаризованной ГСА, удается отказаться от построения последней, отражая структуру графа переходов непосредственно в тексте программы.

В заключение раздела отметим, что вопрос о построении линеаризованной ГСА, в которой не могут применяться двухадресные условные переходы, решается по-разному в зависимости от того, должна ли быть эта граф-схема структурированной или нет.

Если линеаризованная ГСА не должна быть структурированной, то преобразование «плоскостной» ГСА в «линейную» сводится к изоморфному изображению исходной ГСА без увеличения числа условных и операторных вершин. Решение этой задачи не является единственным. При этом число вершин, соответствующих безусловным переходам, зависит, в частности, от того, допустимы ли в линеаризованной ГСА «возвраты назад».

Построение структурированной и линеаризованной ГСА (СЛГСА) по неструктурированной или даже структурированной (с использованием управляющей конструкции «полный выбор») плоскостной ГСА связано с введением избыточных условных и операторных вершин.

Рассмотрим вопрос о выборе числа дополнительных вершин в СЛГСА в случае использования в качестве спецификации графа переходов.

При этом возможны два подхода:

— построение по графу переходов плоскостной структурированной ГСА и линеаризация последней;

— непосредственное построение СЛГСА по графу переходов.

При применении первого подхода, используя изложенное в настоящей главе, по графу переходов может быть построена плоскостная структурированная ГСА. Линеаризация этой ГСА проводится с помощью только двух управляющих конструкций («последовательное соединение блоков» и «неполный выбор») следующим образом:

— перечисляются в плоскостной ГСА все «значащие пути», проходящие от корня ГСА до операторных вершин, соединенных с ее выходом;

— последовательно соединяются блоки, каждый из которых образован из вершин ГСА, соответствующих одному «значащему» пути;

— вводятся в операторные вершины блоков новые обозначения переменных, кодирующих состояния в графе переходов;

— последовательно подключаются к уже построенной структуре новые блоки, в которых осуществляется присвоение значений новых переменных переменным, кодирующим состояния.

Построение СЛГСА по «значащим» путям плоскостной ГСА приводит обычно к большей избыточности, и поэтому более целесообразно использовать второй подход.

При этом отметим, что СЛГСА на рис. 13.45 построена с помощью второго подхода непосредственно по графу переходов на рис. 4.124, а не за счет линеаризации плоскостной ГСА (рис. 13.32), эквивалентной этому графу.

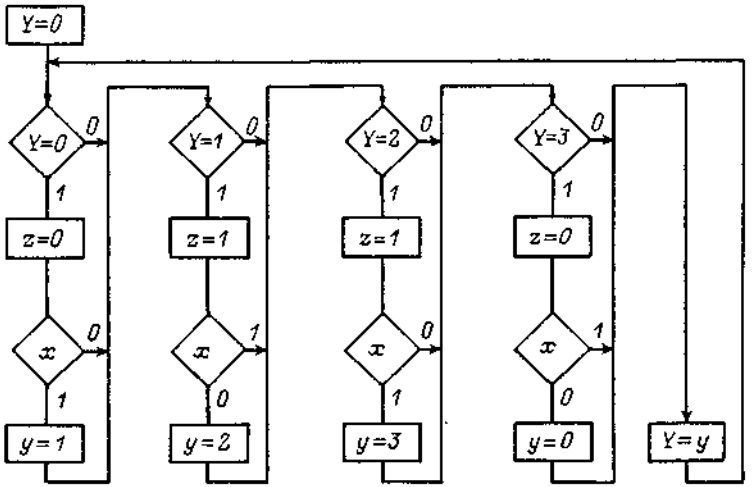

Рис. 13.45

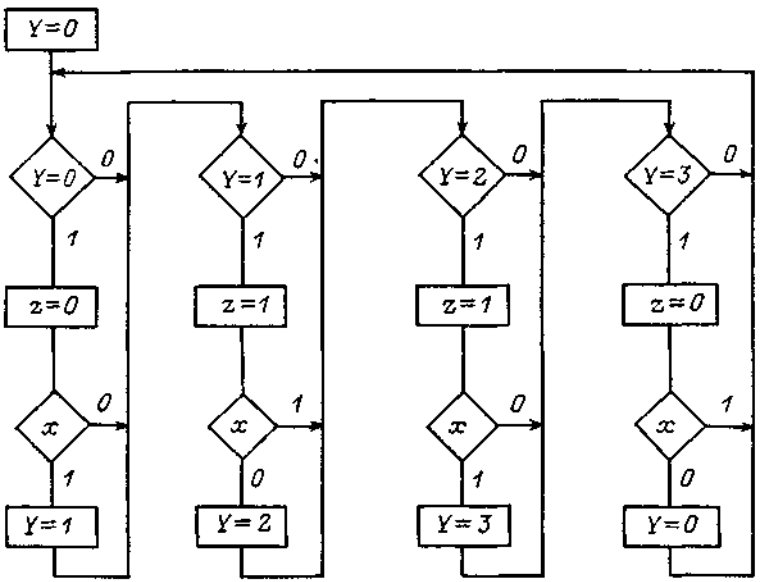

Рис. 13.46

Рис. 13.46

Применение второго подхода позволило построить СЛГСА практически безызбыточно по сравнению с плоскостной ГСА, что невозможно при использовании первого подхода.

Сравнивая структуру граф-схем на рис. 13.32 и 13.45, отметим, что в первом случае применяется децентрализованный (единый) дешифратор состояний, а во втором — распределенный, что и обеспечивает эффективность применения второго подхода.

Если при использовании многозначного кодирования второй подход приводит к однозначному решению задачи, то при принудительном или принудительно-свободном кодировании число вершин в СЛГСА зависит от выбора (при построении распределенного дешифратора состояний) в каждой вершине графа переходов того числа компонент, которое отличает (идентифицирует) ее от всех остальных вершин графов переходов. При этом если критерием качества построения СЛГСА является ее читаемость, то каждый фрагмент распределенного дешифратора должен содержать все компоненты, помечающие соответствующую вершину графа, в то время как для минимизации числа вершин в структурированной ЛГСА для построения ее распределенного дешифратора в каждой вершине графа должны выбираться лишь те компоненты вектора состояния, которые отличают это состояние от любого другого.

Так, если СЛГСА на рис. 13.42, построенная по графу переходов (рис. 13.27), содержит распределенный дешифратор полных кодов состояний, то дешифратор состояний в СЛГСА на рис. 13.44 использует неполные коды, помечающие вершины графа переходов на рис. 13.43, обеспечивая сокращение числа вершин в этой граф-схеме.

Приведем еще один пример применения неполных кодов при построении СЛГСА. Предположим, что в графе переходов каждая из пяти вершин закодирована следующим образом:

```
1 – 0  0  0  0  0  0
2 – 1  0  0  1  0  0
3 – 0  0  0  0  0  1
4 – 0  1  0  0  1  0
5 – 0  1  1  0  0  0
```

Если в этих векторах при помощи скобок выделить определяющие значения следующим образом:

```
1 – (0  0  0)  0  0 (0)
2 – (1  0  0)  1  0  0
3 – (0  0  0)  0  0 (1)
4 –  0 (1  0)  0  1  0
5 –  0  1 (1)  0  0  0,
```

то в этом случае могут использоваться как распределенный (с 14 вершинами), так и централизованный (с суммарной длиной путей в нем, равной 14) дешифраторы состояний. При этом последний реализуется следующим образом:

```
if(z3)  then  5;
if(z2)  then  4;
if(z1)  then  2;
if(z6)  then  3
        else  1.
```

При линеаризации этот дешифратор порождает в СЛГСА число вершин, равное суммарному числу путей в нем.

При выделении определяющих значений:

```
1 — (0  0) 0  0 0 (0)
2 — (1) 0  0  1 0  0
3 — (0  0) 0  0 0 (1)
4 — (0  1  0) 0 1  0
5 — (0  1  1) 0 0  0
```

— также могут быть построены как распределенный (с 13 вершинами), так и централизованный (с суммарной длиной путей в нем, равной 13) дешифраторы состояний, последний из которых реализуется следующим образом:

```
if(z1)  then 2;
if(z2)  then
            if(z3)  then 5
                    else 4;
        else
            if(z6)  then 3
                    else 1.
```

При выделении определяющих значений:

```
1 — (0  0) 0  0 0 (0)
2 — (1) 0  0  1 0  0
3 —  0  0  0  0 0 (1)
4 —  0 (1) 0  0 1  0
5 —  0  1 (1) 0 0  0
```

— единый дешифратор не может быть построен, а распределенный дешифратор содержит лишь 7 вершин.

Из изложенного следует, что построение СЛГСА непосредственно по графу переходов является более предпочтительным по сравнению с ее построением на основе линеаризации плоскостной граф-схемы алгоритма.