Глава 9

Иерархия моделей автоматов

В предыдущих разделах настоящей работы показано, что в моделях одного автомата для фиксации его внутренних состояний достаточно иметь одну многозначную ячейку памяти «П» (рис. 9.1).

При этом необходимо отметить, что граф переходов описывает алгоритм работы автомата в целом, в то время как при программной реализации конструкция switch (переключатель) описывает только комбинационную часть автомата, а функция запоминания реализуется одной из ячеек памяти применяемого вычислительного устройства, которая считается принадлежностью автомата.

В этой ячейке фиксируются значения внутренней переменной *Y,* применяемой в конструкции switch. На рис. 9.1 значение этой переменной в настоящий момент времени обозначено символом *Y,* а в следующий — *Y '.*

При использовании одного автомата в качестве управляющего устройства на его вход подаются двоичные переменные от источников информации, а на его выходе формируются двоичные переменные, подаваемые на приемники информации.

Отметим, что при программной реализации алгоритмов кроме памяти для внутренней переменной необходима также память для запоминания входных и выходных переменных. Однако эта разновидность применяемой памяти в модель автомата не вводится, так как она в некотором смысле имеет не алгоритмическую (особенно по входным переменным), а технологическую природу, определяемую принципом последовательной (циклической) обработки информации.

Действительно, из идеи построения графов переходов следует, что автомат, находясь в некоторой вершине, может принять все входные переменные, помечающие дуги, исходящие из этой вершины, причем эти переменные могут быть любой длительности. Однако при реализации нескольких, даже независимых автоматов на одном процессоре, если не использовать память указанного типа, значения входных переменных, длительность которых меньше времени цикла выполнения программы, могут быть потеряны.

При использовании системы взаимосвязанных автоматов в качестве модели алгоритма алфавит входных переменных по отношению к одиночному автомату должен быть расширен. Кроме собственных переменных, применяемых в таком автомате, к его входным переменным должны быть

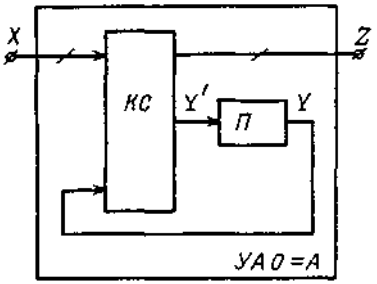

Рис. 9.1

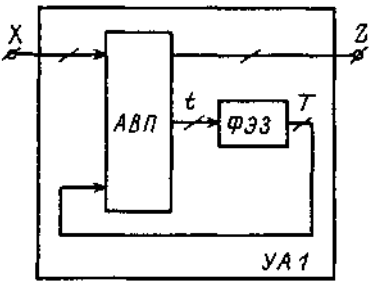

Рис. 9.2

отнесены также двоичные выходные, а самое главное — многозначные внутренние переменные других автоматов, передаваемые в рассматриваемый автомат.

Автоматы этого класса могут быть названы связанными автоматами.

Для реализации временных алгоритмов в гл. 2 было введено понятие управляющего автомата, состоящего из автомата, поведение которого описывается графом переходов, и внешних по отношению к этому автомату функциональных элементов задержки (рис. 9.2).

При этом алфавит выходных и входных переменных автомата, входящего в состав управляющего автомата, по сравнению с предыдущим случаем расширяется за счет двоичных переменных, воздействующих на ФЭЗ, и двоичных переменных, формируемых при срабатывании этих элементов, соответственно.

Автоматы рассмотренного класса могут быть названы автоматами с временными переменными (АВП). В этом определении наличие остальных типов переменных умалчивается, так как считается, что они естественным образом входят в понятие «автомат».

Для отличия управляющих автоматов рассмотренного типа от других типов таких автоматов, описываемых ниже, будем называть их управляющими автоматами первого типа (УА1). При этом собственно автомат (рис. 9.1) можно называть управляющим автоматом нулевого типа (УАО).

Следующей алгоритмической моделью является система взаимосвязанных управляющих автоматов первого типа, в которой автоматы, входящие в их состав, могут использовать все типы перечисленных выше входных и выходных переменных.

В ряде случаев бывает целесообразным применять другую модель автомата, отличающуюся входным и выходным алфавитами от модели одиночного автомата. Расширение этих алфавитов осуществляется за счет введения вектора многозначных переменных *F,* называемых флагами, которые несут информацию о предыдущих состояниях автомата.

Автоматы этого класса будем называть автоматами с флагами (АФ). Автомат этого класса взаимодействует с внешними по отношению к нему ячейками многозначной памяти (П1), в которые записывается информация о предыдущих состояниях автомата, образуя вместе с ними

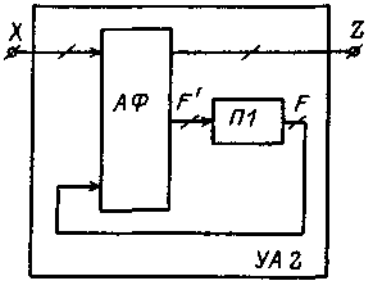

Рис. 9.3

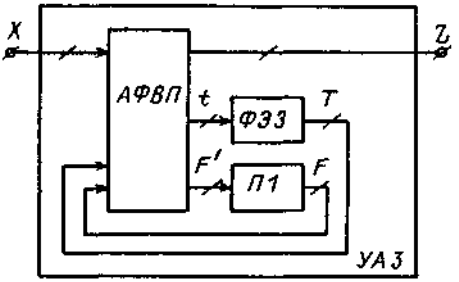

Рис. 9.4

управляющий автомат второго типа (УА2). На рис. 9.3 флаговым переменным F соответствуют символы F и F', обозначающие значения этих переменных в настоящий и следующий моменты времени.

Флаговые переменные по отношению к переменной Y, определяющей состояния автомата, являются вспомогательными, что находит свое отражение в ГП, где значения переменной Y нумеруют его вершины, а флаговые переменные используются наравне с другими в качестве входных и выходных.

Как и другие разновидности автоматов, рассмотренные выше, автоматы с флагами могут быть как автоматами Мура, так и автоматами Мили. Использование моделей автоматов Мура и Мили определяется классом решаемых задач.

Автоматы Мура наиболее естественным образом описывают алгоритмы логического управления, в то время как автоматы Мили обычно более удобны при вычислениях. Это, в частности, объясняется тем, что в модели автомата Мура значения выходных переменных формируются в вершине ГП, что обеспечивает возможность сохранения этих значений в течение всего времени, пока автомат находится в рассматриваемом состоянии, в то время как в модели автомата Мили значения этих переменных формируются на переходе из одного состояния в другое.

Кроме моделей автоматов Мура и Мили в ряде случаев бывает целесообразным применение смешанной модели С-автомата, позволяющего использовать достоинства каждой из этих моделей [16].

Для автомата с флагами характерно, что его следующее состояние может зависеть не только от состояния и значений входных переменных в настоящий момент времени, но и от более глубокой предыстории.

При этом необходимо отметить, что если при изменении значений входных переменных имеется возможность наблюдать кроме значений выходных переменных еще и значения внутренней и N флаговых переменных, то наблюдателю не удается выяснить, какая модель в данном случае применяется — автомат с N флагами или $N + 1$ взаимосвязанных автоматов без флагов. Поэтому автомат с N флагами может рассматриваться как единый автомат, «склеенный» из $N + 1$ взаимосвязанных автоматов без флагов.

Задача декомпозиции автомата с флагами на систему взаимосвязанных автоматов может быть решена аналитически, правда, за счет перехода от

многозначного кодирования к двоичному, записи булевой формулы для каждой внутренней (а также выходной) переменной и построения системы графов переходов по этим формулам.

При переходе от модели автомата без флагов к модели автомата с флагами сложность описания обычно уменьшается, правда, за счет усложнения поведения, для анализа которого необходимо строить граф достижимых маркировок, состояния которого определяются значениями внутренней и флаговых переменных. Из изложенного следует, что замена моделей может использоваться для оптимизации памяти и времени вычисления программ, реализующих эти модели. При этом отметим, что если переход от автомата Мура к автомату Мура с флагами или автомату Мили не увеличивает числа состояний, то переход к системе автоматов без флагов может привести как к увеличению этого показателя, так и к его уменьшению.

Следующей моделью является система взаимосвязанных управляющих автоматов второго типа. В этой модели применяются связанные автоматы с флагами.

В управляющем автомате третьего типа (УАЗ) автомат взаимодействует с ФЭЗ и элементами дополнительной памяти (рис. 9.4). Будем называть такой автомат автоматом с флагами и временными переменными (АФВП).

Наиболее мощной по изобразительным средствам моделью для решения рассматриваемого класса задач является система взаимосвязанных управляющих автоматов третьего типа. В этой модели используются связанные автоматы с флагами и временными переменными.

Проиллюстрируем применение некоторых из перечисленных моделей на примере программной реализации вычислителя с прерываниями, трактуемыми,как описано ниже.

Предположим, что основным режимом вычислителя является умножение четырех переменных — d_1, d_2, d_3, d_4,, выполняемое следующим образом: $R_1 = d_1 * d_2$; $R_1 = R_1 * d_3$; $R_1 = R_1 * d_4$. При этом считается, что вычисление каждого из этих выражений неделимо.

Процесс умножения может быть прерван ($X = 1$) с запоминанием вычисленного результата. При этом вычислитель переходит в режим однократной обработки прерывания — сложению тех же переменных ($R_2 = d_1 + d_2 + d_3 + d_4$), по завершению которого вне зависимости от наличия сигнала прерывания (маскирование прерывания) вычислитель продолжает выполнение основной программы, используя вычисленный на момент прерывания результат. После завершения основного режима вычисленные значения R_1 и R_2 сбрасываются и начинается новый цикл работы.

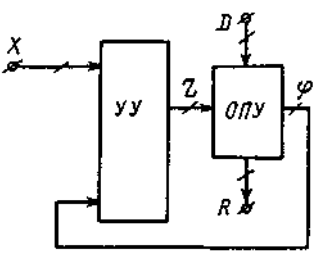

Рис. 9.5

Реализацию указанного алгоритма будем выполнять применяя основную структурную схему (рис. 9.5), предложенную в теории построения вычислителей [7]. При этом считается, что управляющее устройство (УУ) формирует управляющие сигналы Z,

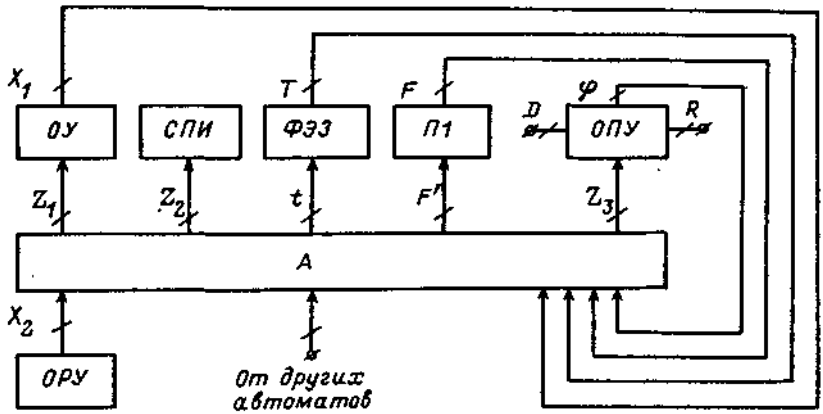

Рис. 9.6

под воздействием которых операционное устройство (ОПУ) выполняет соответствующие микрооперации для получения результата R над данными D. При этом ОПУ вырабатывает значения двоичных переменных φ для УУ.

Обобщая модели, рассмотренные в настоящем разделе и в гл. 2, приведем схему связей ядра логико-вычислительной системы управления с объектом управления (рис. 9.6).

Возвращаясь к модели на рис. 9.5, сопоставим для рассматриваемого примера управляющие переменные и микрооперации следующим образом:

$$z_1 : R_1 = d_1 * d_2; \quad z_2 : R_1 = R_1 * d_3; \quad z_3 : R_1 = R_1 * d_4;$$

$$z_4 : R_2 = d_1 + d_2 + d_3 + d_4; \quad z_5 : R_1 = 0; \quad z_6 : R_2 = 0.$$

Решение поставленной задачи может быть выполнено в три этапа: разработка формальной спецификации, описывающей функционирование управляющего «устройства»; программная реализация спецификации; введение в программу вместо управляющих переменных микроопераций. Разработку спецификации будем выполнять с помощью графов переходов.

Используем первоначально в качестве модели автомат Мура с десятью вершинами (рис. 9.7). При этом предполагается, что приведенные в вершине выходные переменные принимают в ней значения, равные единице, в то время как остальные выходные переменные — нулевые значения.

Число состояний может быть сокращено до семи при переходе к модели, состоящей из двух взаимосвязанных автоматов Мура (рис. 9.8). При этом первый автомат обеспечивает выполнение основного режима, а второй — вспомогательных.

То же число вершин, но в одном ГП может быть обеспечено при применении модели автомата Мура с флагом F (рис. 9.9). Установка флага фиксирует факт выполнения вычислений по прерыванию, а информация о значении флага используется для обеспечения однократности этих

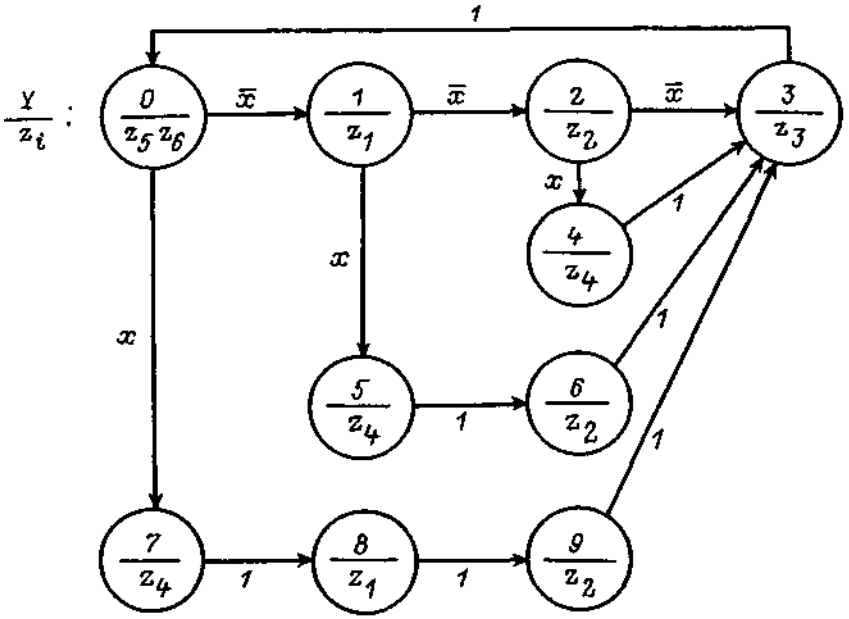

Рис. 9.7

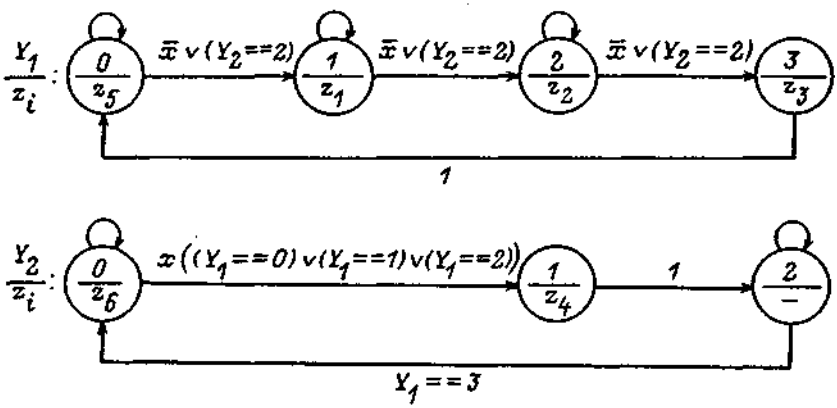

Рис. 9.8

вычислений. Переменная Y определяет номера вершин графа переходов и поэтому является для автомата основной, в то время как флаговая переменная F— вспомогательной. При этом по умолчанию предполагается, что если в вершине переменная F не упоминается, то она сохраняет последнее из значений, присвоенных этой переменной на пути в рассматриваемую вершину.

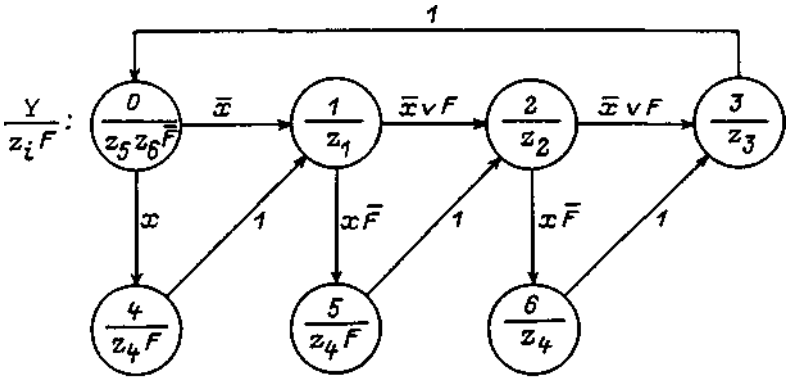

Рис. 9.9

Наличие флага обеспечивает возможность перехода из одной вершины ГП в разные его вершины при одном и том же значении входной переменной, что невозможно в традиционной модели автомата. В этом случае имеет место разновидность зависимости перехода автомата в следующее состояние от предыстории.

При реализации ГП автомата с флагами с помощью конструкции switch необходимо учесть некоторые особенности, которые отсутствовали при реализации автоматов без флагов.

Действительно, если для вершины с номером «1» графа переходов на рис. 9.9 записать конструкцию case следующим образом (без указания соотношений для выходных переменных):

```
case 1: if (x & F̄)    {Y = 5; F = 1; }
        if (x̄ ∨ F)     Y = 2;
        break;
```

то при исходных значениях $x = 1$ и $F = 0$ автомат с флагом из первого состояния перейдет вместо пятого во второе состояние, так как во втором условии проверяется переменная F, значение которой вырабатывается в первой строке и обеспечивает выполнение этого условия. При этом отметим, что эта ошибка появляется, несмотря на то что условия в строках ортогональны. Ошибку можно устранить тремя способами:

— введением дополнительного оператора break в первую строку

```
case 1: if (x & F̄)   {Y = 5; F = 1; break;}
        if (x̄ ∨ F)    Y = 2;
        break;
```

```
case 1: if (x & F̄)   {Y = 5; C = 1; }
        if (x̄ ∨ F)    Y = 2;
        break;
...
} F = C;
```

овой переменной

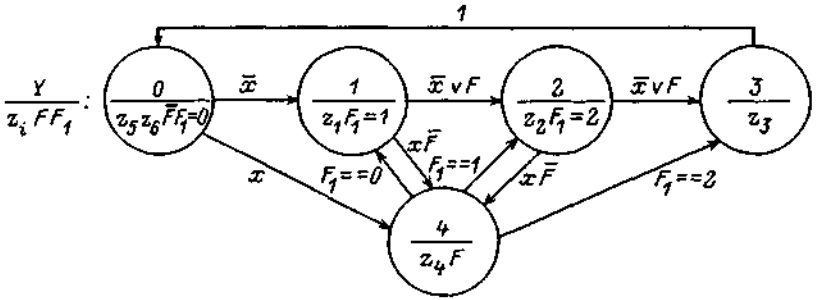

Рис. 9.10

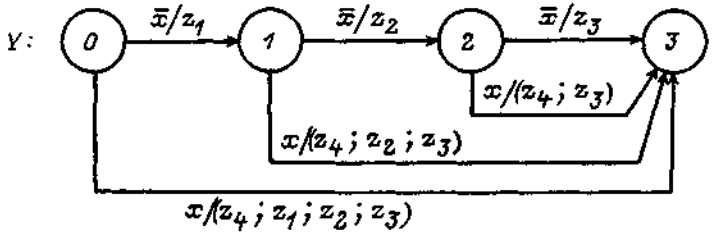

Рис. 9.11

— изменением порядка проверок

```
case 1: if   (x̄ ∨F)      Y = 2;
        if   (x & F̄)    {Y = 5; F = 1; }
        break;
```

Приоритет использования этих способов обратен порядку их перечисления.

Дальнейшее сокращение числа состояний автомата достигается при использовании модели автомата Мура с двумя флагами: F и F_1 (рис. 9.10). При этом если первый флаг двоичный, то второй — многозначный (в данном случае троичный). Первый флаг используется для той же цели, что и в предыдущем примере, а установка второго флага фиксирует номер вершины, находясь в которой автомат принимает сигнал прерывания и применяется для определения номера «точки возврата».

В рамках использования моделей автомата Мура дальнейшее сокращение числа состояний невозможно.

Это удается осуществить при переходе к модели автомата Мили с четырьмя состояниями, в которой на дугах в качестве пометок применяются символы не только входных, но и через дробь с ними выходных переменных (рис. 9.11).

Переход к системе из двух взаимосвязанных автоматов Мили увеличивает суммарное число состояний автоматов до шести при упрощении пометок выходных переменных (рис. 9.12).

Построение одного автомата Мили с флагом F вновь сокращает число состояний автомата до четырех (рис. 9.13) и изменяет пометки дуг.

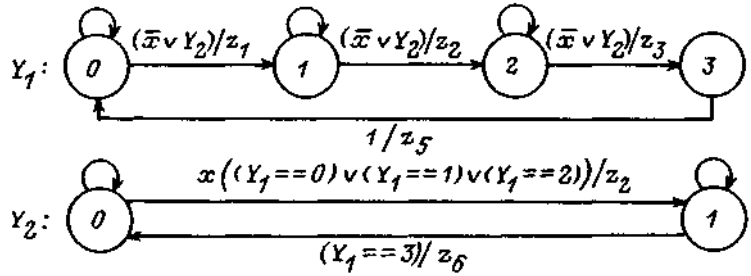

Рис. 9.12

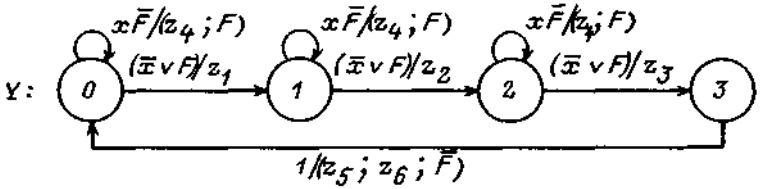

Рис. 9.13

После выбора одной из рассмотренных моделей и реализации ее с помощью конструкций switch может быть выполнена замена соотношений для выходных переменных соответствующими микрооперациями.

Рассмотрим еще один пример, который для своего эффективного решения может потребовать использования моделей, отличных от описанных выше.

Пусть требуется построить автомат, который при поступлении входного сигнала $(x = 1)$ формирует сигнал $t = 1$, обеспечивающий выполнение микрооперации $d := d + 1$. После того как выполнится условие $d \geqslant L$, формируются выходной сигнал $z = 1$ и сигнал $t = 0$, который инициирует микрооперацию $d = 0$. При снятии входного сигнала $(x = 0)$ в любой момент времени выходные сигналы автомата исчезают ($z = 0$; $t = 0$).

Наиболее естественным образом эта задача решается при декомпозиции логико-вычислительного устройства на автомат без выходного преобразователя, описываемый ГП (рис. 9.14), и операционное устройство, реализующее микрооперации, инициируемые автоматом, и формирующее

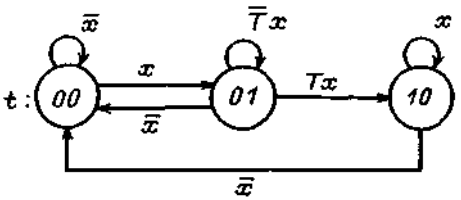

Рис. 9.14

значения дополнительной переменной *T,* нулевое значение которой определяет выполнение первой микрооперации, а ее единичное значение — второй.

При этом автомат описывается системой булевых формул, при построении первой из которых используются неполные коды по переменным *Z* и *V.*

```
z1  = x &(T & t ∨ z);
t   = x & z̄ & (T̄ ∨ t̄);
z   = z1,
```

а логико-вычислительное «устройство» в целом — с помощью системы булевых формул и условных выражений вида:

```
z1  = x & (T & t ∨ z);
t   = x & z̄ & (T̄ ∨ t̄);
z   = z1;
if (t)      d = d + 1;
else        d = 0;
if(d ≥ L)   T = 1;
else        T = 0.
```

Сложность программной реализации может быть снижена при применении вместо рассмотренного нового графа — «графа переходов с неравенствами», который получается из исходного при следующих заменах: $!T$ на $d < L$; T на $d \geqslant L$. Этот ГП соответствует менее жесткой по сравнению с предыдущим случаем декомпозиции логико-вычислительного устройства, приводящей к замене булевых формул на логические и исключению второго условного выражения:

```
z1  = x & ((d ≥ L) & t ∨ z);
t   = x & z̄ & ((d < L) ∨ t̄);
z   = z1;
if (t)        d = d + 1;
else          d = 0;
```

Дальнейшее упрощение программной реализации достигается при полном отказе от декомпозиции, т. е. при написании программы непосредственно по граф-схеме алгоритма (рис. 9.15).

При этом одновременно описываются автомат и операционное устройство, что усложняет чтение такой спецификации. Это объясняется тем, что соответствующие этой граф-схеме графы переходов для автоматов Мура (рис. 9.16) и Мили (рис. 9.17) содержат неустойчивые состояния и флаги, а выходные значения автоматов зависят от предыстории.

При этом отметим, что если программы, построенные по графам переходов (даже с неустойчивыми состояниями), могут быть написаны так, что в них за один программный цикл реализуется не более одного перехода, то при программировании по произвольно построенной ГСА один путь, проходимый за цикл, может содержать более одного перехода в эквивалентном графе переходов, что, повышая быстродействие, осложняет чтение программы и не позволяет в случае необходимости использовать промежуточные результаты вычислений в других программах.

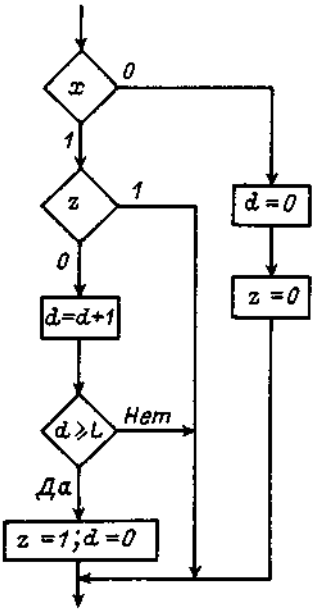

Рис. 9.15

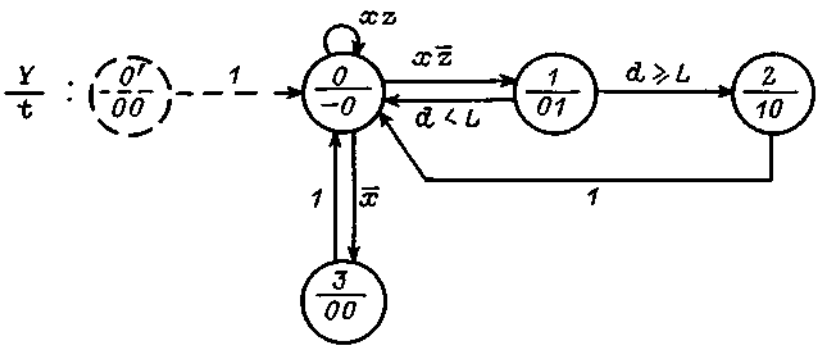

Рис. 9.16

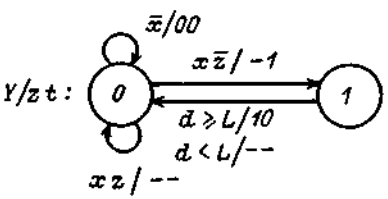

Рис. 9.17

Отметим также, что если программирование проводить не непосредственно по ГСА, а на основе декомпозиционного подхода с применением, например, графа переходов (рис. 9.16), то при двоичном кодировании состояний получается описание более сложное, чем в предыдущем случае:

```
Y01 = x & z & Y0 ∨ (d < L) & Y1 ∨ Y2 ∨ Y3;
Y11 = x & z̄ & Y0;
Y21 = (d ⩾ L) & Y1;
Y3  = x̄ & Y0;
z1  = Y2 ∨ z & (Y0 ∨ Y1);
t   = Y1 ∨ z & Y0;
if(t)       d = d + 1;
else        d = 0;
Y0  = Y01; Y1 = Y11; Y2 = Y21; z = z1.
```

Начальная установка в этом случае имеет вид:

```
Y0 = 1; Y1 = 0; Y2 = 0; Y3 = 0; z = 0; d = 0.
```

Обратим внимание на то, что пятая и шестая формулы этой системы кроме установки единичных и нулевых значений переменных г и г обеспечивают также сохранение их предыдущих значений.

Из рассмотрения графов переходов (рис. 9.16 и 9.17), построенных по ГСА, следует также, что использование этих графов в качестве спецификаций автомата вместо исходно построенного ГП (рис. 9.14) нецелесообразно, так как при этом не обеспечивается основное требование к первичному описанию — его понятность.

Рассмотренные примеры еще раз подтверждают тот факт, что предложенный в настоящей работе подход к применению определенным образом построенных графов переходов в качестве языка спецификаций позволяет выражать концепции решения задач логического управления непосредственно в понятной даже для неспециалиста форме. Это чрезвычайно существенно, так как «язык формирует наш способ мышления и определяет, о чем мы можем мыслить эффективно» [225].

Отказ от использования языка графов переходов целесообразен лишь в тех случаях, когда при его применении не удается удовлетворить ограничения по объему памяти и быстродействию.

Некоторые идеи о взаимодействии автоматов с внешней памятью и с другими автоматами совпадают с идеями, используемыми в объектно-ориентированном программировании [226].