Глава 5

Программная реализация управляющих автоматов и моделей объектов управления

5.1. Программные модели автоматов

В предыдущей главе рассмотрены различные алгоритмические модели автоматов. После выбора алгоритмической модели начинается этап собственно программной реализации, первая стадия которой — выбор языка программирования. Предположим, что в качестве такого языка выбран язык СИ, весьма удобный при реализации СБФ, так как он наряду с логическими операторами содержит также и поразрядные логические операторы. Ниже будет показано, что этот язык также удобен и для реализации ГП.

Под программной моделью будем понимать программу и совокупность операторов, используемых в ней при реализации алгоритмической модели в рамках выбранного языка программирования.

Так, например, если в качестве алгоритмической модели выбрана СБФ, то одна программная модель может базироваться на поразрядных логических операторах И, ИЛИ, НЕРАВНОЗНАЧНОСТЬ, ДОПОЛНЕНИЕ, а другая — на логических операторах И, ИЛИ, НЕ.

Аналогичная ситуация возникает также и при использовании ГСА в зависимости от того, применяется ли оператор условия или условный оператор.

По аналогии с алгоритмическими моделями существуют и программные модели второго порядка — модели реализации в базисе языка ассемблера. Так, например, при использовании в записи булевой формулы поразрядных операторов трансляция осуществляется в операторные ассемблерные программы, а при использовании логических операторов — в бинарные ассемблерные программы [88].

Автором совместно с В. Н. Кондратьевым выполнено сравнение по памяти команд и быстродействию более 60 программ в базисе языка СИ, реализующих (разными методами и операторами) функцию «голосование два и более из трех», и более 25 программ, реализующих СБФ, описывающую одноразрядный сумматор. При этом необходимо отметить, что полученные результаты для одного и того же алгоритма могут отличаться вплоть до десятков раз (Приложения 1, 2).

Из изложенного в разд. 2.3 следует, что в качестве языка спецификаций автоматов с памятью наиболее целесообразно использовать ГП. При переходе к алгоритмическим моделям графы переходов естественно сохранить. Поэтому и в качестве программной модели автоматов с памятью целесообразно выбрать такую конструкцию, которая позволяет изоморфно отразить в ней ГП. Такой моделью является конструкция switch языка СИ, используемая в виде аналогов в большинстве языков программирования высокого уровня.

В рамках применения конструкции switch в свою очередь возможна многовариантность, определяемая числом таких конструкций (одна или две), выбранной структурной моделью автомата, учитывающей в том числе и его род, и способом реализации булевых формул.

Вопрос о независимой или совместной реализации булевых формул, входящих в конструкцию switch, может решаться как при выборе структурной модели автомата, так и при построении его программной модели.

В любом случае эти формулы могут быть реализованы и оптимизированы либо независимо в каждом операторе if, либо совместно для всех этих операторов после выноса формул, содержащих более одной буквы, из конструкции switch.

Оптимизация формул может выполняться и в рамках каждой метки case[1] за счет изменения порядка проверки. Так, например, эквивалентны конструкции:

```
case 0:if (x1)       Y = 1; case 0:if (x2) Y = 3;
       if (x̄1 & x2) Y = 3;         if (x1) Y = 1;
       break;                     break;
```

булевы формулы в которых обладают разной сложностью.

5.1.1. Применение конструкции switch

Изоморфизм ГП и текста программы при использовании конструкции switch при отсутствии жестких ограничений на объем памяти и быстродействие является в большинстве случаев определяющим для применения пентады (состояние—независимость от глубокой предыстории—система взаимосвязанных ГП—-многозначное кодирование—конструкция switch) в качестве основы технологии программной реализации алгоритмов логического управления по крайней мере на стадии их моделирования.

Использование указанной пентады позволяет обеспечить также и структурированность программ. Это является принципиальным положением, так как вопрос о структурировании программ весьма сложен. В [103] в качестве наилучшего средства структуризации предлагается использовать алгоритмическую конструкцию, близкую к конструкции switch, однако не непосредственно, как в настоящей работе, а с целью объединения отдельных фрагментов в исходной неструктурированной ГСА. Предлагаемый в настоящей работе подход позволяет отказаться от

[1] case (англ.) — случай.

построения произвольной ГСА с последующим ее структурированием и обеспечивает возможность непосредственного вложения ГП в конструкцию switch.

Отметим также, что конструкция switch в отличие от системы булевых формул не может породить состояния, отсутствующие в реализуемом ГП, а в отличие от ГСА, в которых обычно используется двоичное кодирование внутренних переменных, обеспечивает за счет применения только одной многозначной внутренней переменной существенно более компактную запись программ.

При использовании конструкции switch резко улучшается «читаемость» программ, так как нет необходимости разбираться с большим числом двоичных внутренних переменных, отсутствующих в исходном задании.

Может сложиться впечатление, что конструкция switch является только более компактной формой записи ГСА, однако это не совсем так, ввиду того что кроме перечисленных выше особенностей этой конструкции она задает фиксированную дисциплину описания процесса управления: настоящее состояние; проверка условий для выбора следующего состояния; назначение следующего состояния. Место выдачи значений выходных переменных в этой последовательности зависит от вида используемой для реализации структурной модели автомата.

Конструкции switch в отличие от традиционных граф-схем соответствуют «короткие» ГСА, в каждой из которых за один проход реализуется не более одного перехода в эквивалентном ей ГП.

При этом отметим, что обычно при использовании ГСА стандартная структура не фиксируется: имеется возможность ее построения, как начиная с опроса внутренних или выходных переменных (состояний автомата), так и с опроса входных переменных (событий), причем вторая возможность реализуется чаще [79], ввиду того что понятие «событие» при программной реализации алгоритмов логического управления в настоящее время более традиционно по сравнению с понятием «состояние», являющимся в предлагаемой технологии основным.

Однако, по мнению автора, предлагаемая дисциплина описания процесса управления является более рациональной, так как в большей степени соответствует «житейской логике», по которой, просыпаясь утром, человек обычно сначала определяет свое внутреннее состояние, а лишь потом интересуется параметрами внешней среды.

5.1.2. Использование конструкции switch при реализации автоматов Мура

Пусть автомат Мура задан ГП на рис. 4.34.

Применим первоначально модель автомата Мура первого рода (рис. 3.6). Реализуем эту модель, применяя две конструкции switch, первая из которых соответствует комбинационной схеме КС2, а вторая — комбинационной схеме КС1, охваченной обратной связью:

```
      switch (Y)      {
case 0: z1=0;   z2=0;    break;
case 1: z1=0;   z2=1;    break;
case 2: z1=1;   z2=0;    break;}
      switch (Y)      {
case 0: if  (x̄1 ⊕ x2)     Y=0;
        if  (x1 & x̄2)     Y=1;
        if  (x̄1 & x2)     Y=2; break;
case 1: if  (x3)          Y=0;
        if  (x̄3)          Y=1; break;
case 2: if  (x3)          Y=0;
        if  (x̄3)          Y=2; break;}.
```

Отметим, что в этой и последующих программах основного текста книги для упрощения используется обозначение инверсии, отличающееся от принятых в языке СИ.

Текст этой программы может быть упрощен за счет исключения по второй конструкции `switch` строк, соответствующих сохранению предыдущего состояния:

```
        switch (Y)  {
case 0: z1=0; z2=0;   break;
case 1: z1=0; z2=1;   break;
case 2: z1=1; z2=0;   break;}
        switch (Y)  {
case 0: if  (x1 & x̄2)   Y=1;
        if  (x̄1 & x2)   Y=2;
        break;
case 1: if  (x3)        Y=0;
        break;
case 2: if  (x3)        Y=0;
        break;}.
```

В рассмотренной реализации петли, имеющиеся в ГП, в явном виде во второй конструкции `switch` не отражаются, а заменяются операторами `break`, располагаемыми в отдельной строке после операторов `if` в каждой метке `case`. При этом число строк в этой конструкции, как и в предыдущей реализации, равно числу дуг в ГП.

Имеется возможность объединить обе конструкции `switch` в одну конструкцию, образовав обобщенную программную модель автомата Мура первого рода, аналог которой отсутствует в теории автоматов:

```
        switch (Y)   {
case 0:   z1=0;  z2=0;
        if  (x1 & x̄2)   Y=1;
        if  (x̄1 & x2)   Y=2;
        break;
case 1: /*z1=0;*/z2=1;
        if  (x3)        Y=0;
        break;
```

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

Эта модель полностью изоморфна ГП, по которому она построена. В приведенной программе не изменяющиеся при переходе значения выходных переменных закомментированы и в объектный код не транслируются.

Недостаток рассмотренных моделей состоит в том, что при их применении значения выходных переменных запаздывают на «такт» (программный цикл) относительно значений переменной, кодирующей состояния.

Этого недостатка лишены модели, приведенные ниже.

Если в качестве структурной модели для рассматриваемого примера выбрать автомат Мура второго рода (рис. 3.7), то ей соответствует программная модель, использующая две конструкции `switch`:

```
          switch   (Y) {
case 0: if (x1 & x̄2)     Y=1;
          if (x̄1 & x2)     Y=2;
          break;
case 1: if (x3)           Y=0;
          break;
case 2: if (x3)
          break;}
          switch   (Y) {
case 0: z1=0;   z2=0; break;
case 1: z1=0;   z2=1; break;
case 2: z1=1;   z2=0; break;}.
```

Так же как и для автоматов Мура первого рода, и в этом случае имеется возможность объединить обе конструкции `switch` в одну, образовав обобщенную программную модель автомата Мура второго рода, аналог которой также отсутствует в теории автоматов:

```
          switch (Y) {
case 0: if (x1 & x̄2)  {Y=1; z1=0; z2=1;}
          if (x̄1 & x2)  {Y=2; z1=1; z2=0;}
          break;
case 1: if (x3)        {Y=0; z1=0; z2=0;}
          break;
case 2: if (x3)        {Y=0; z1=0; z2=0;}
          break;}.
```

Эта модель, как будет показано ниже, похожа на обобщенную программную модель автоматов Мили и отличается от последней порядком расположения переменной состояния и выходных переменных, сохраняющим изоморфизм с соответствующими графами переходов.

Из изложенного следует, что для одного и того же ГП автомата Мура может быть выбрана одна из двух структурных моделей и одна из четырех программных моделей, использующих конструкции `switch`.

5.1.3. Использование конструкции switch при реализации автоматов Мили

Пусть автомат Мили задан ГП на рис. 4.113.

Используем структурную модель автомата Мили первого рода (рис. 3.8). Реализуем первоначально эту модель, применяя две конструкции `switch`, первая из которых соответствует комбинационной схеме КС2, а вторая — комбинационной схеме КС1, охваченной обратной связью:

```
        switch (P)  {
case 0: if(x̄1 & x̄2)   S=0;
        if(x1 & x2)    S=0;
        if(x1 ⊕ x2)    S=1; break;
case 1: if(x̄1 & x̄2)   S=1;
        if(x1 & x2)    S=1;
        if(x1 ⊕ x2)    S=0; break;}
        switch (P)  {
case 0: if(x̄1 & x̄2)   P=0;
        if(x1 ⊕ x2)    P=0;
        if(x1 & x2)    P=1; break;
case 1: if(x1 & x2)    P=1;
        if(x1 ⊕ x2)    P=1;
        if(x̄1 & x̄2)   P=0; break;}.
```

Если в результате перехода состояние автомата не изменяется, то соответствующая строка программы может быть исключена. При этом если текст первой конструкции `switch` не изменяется, то вторая конструкция приобретает следующий вид:

```
        switch (P)  {
case 0: if(x1 & x2)    P=1; break;
case 1: if(x̄1 & x̄2)   P=0; break;}.
```

Как и для автоматов Мура, имеется возможность объединить обе конструкции `switch` в одну, образовав обобщенную программную модель автомата Мили, аналог которой отсутствует в теории автоматов:

```
        switch (P) {
case 0: if(x̄1 & x̄2)   {S=0;/*P=0;*/}
        if(x1 ⊕ x2)    {S=1;/*P=0;*/}
        if(x1 & x2)    {S=0;  P=1;}  break;
case 1: if(x1 & x2)    {S=1;/*P=1;*/}
        if(x1 ⊕ x2)    {S=0;/*P=1;*/}
        if(x̄1 & x̄2)   {S=1;  P=0;}  break;}.
```

Эта модель изоморфна ГП, по которому она строилась, и поэтому ее наиболее целесообразно применять для реализации ГП автоматов Мили. Неизменяющиеся при переходе номера (коды) вершин (состояний) закомментированы.

Последняя модель внешне похожа на обобщенную программную модель автомата Мура второго рода, в которой, однако, нет необходимости отражать в явном виде переходы, соответствующие петлям ГП, а следующее состояние и значения выходных переменных располагаются в другом порядке.

Необходимо отметить, что при применении двух конструкций `switch` использование структурной и программной моделей для автомата Мили второго рода невозможно.

Таким образом, для одного и того же ГП автомата Мили может быть применена одна структурная модель и одна из двух программных моделей, использующих конструкции `switch`.

Покажем на примере рассмотренной обобщенной программной модели автомата Мили, как можно выполнить совместную реализацию булевых формул на дугах ГП за счет введения в модель входного преобразователя. Построим этот преобразователь, используя линейные арифметические полиномы и маскирование [92, 276]:

$$X = 8 - 3x_1 - 3x_2; \quad X1 = X \,\&\, 8; \quad X2 = X \,\&\, 4; \quad X3 = X \,\&\, 2;$$

```
        switch (P){
case 0: if(X1)     S=0;
        if(X2)     S=1;
        if(X3)    {S=0; P=1;} break;
case 1: if(X3)     S=1;
        if(X2)     S=0;
        if(X1)    {S=1; P=0;} break;}.
```

В заключение раздела отметим, что для всех рассмотренных программных моделей их среднее быстродействие может быть повышено за счет использования дополнительных операторов `break`. При этом конструкция `switch` в последнем примере приобретает следующий вид:

```
        switch (P){
case 0: if(X1)    {S=0; break;}
        if(X2)    {S=1; break;}
        if(X3)    {S=0; P=1; break;}
case 1: if(X3)    {S=1; break;}
        if(X2)    {S=0; break;}
        if(X1)    {S=1; P=0; break;}}.
```

5.2. Функциональные элементы задержки

В гл. 2 было показано, что управляющий автомат в общем случае состоит из автомата и ФЭЗ. Завершив рассмотрение вопросов, связанных с реализацией автоматов, перейдем к изложению аналогичных вопросов применительно к ФЭЗ.

5.2.1. Переходные процессы в одноконтурных схемах

Рассмотрим простейшие схемы с контурами — одноконтурные схемы.

Наличие контура в схеме не является достаточным для существования памяти. Так, в [138] приведен пример одноконтурной схемы из двухвходовых элементов И и ИЛИ, являющейся комбинационной.

Последовательностное поведение контура связано с наличием инверторов в нем. В [107] показано, что если контур содержит нечетное число инверторов, то схема переходит в генераторный режим. На рис. 5.1 приведена схема, которая при x - 1 преобразуется в «нечетный» контур их трех инверторов, функционирующий в соответствии с ГП (рис. 5.2), содержащим две неустойчивые вершины.

Если контур содержит четное число инверторов («четный» контур), то он обладает памятью. На рис. 5.3 приведена схема, которая при $x = 1$ преобразуется в «четный» контур из четырех инверторов, функционирующий в соответствии с ГП на рис. 5.4. Обе вершины этого ГП устойчивы.

Более внимательное рассмотрение функционирования этой схемы при переходе из одного состояния в другое показывает, что так как реальные элементы ЗАПРЕТ инерционны, то переходы в ГП осуществляются не непосредственно, а через промежуточные состояния, что требует самостоятельного исследования.

Ниже излагается метод построения ФЭЗ, длительность запаздывания которых может превышать сумму запаздываний элементов, их составляющих. Метод разработан В. В. Киселевым (НПО «Аврора») при участии автора [108, 109].

Теоретический интерес этого метода состоит в том, что он относится к группе методов, которые позволяют обеспечить для композиции элементов более высокие показатели качества по сравнению с ее составляющими. К таким методам относится, например, метод построения надежных схем из ненадежных элементов [46].

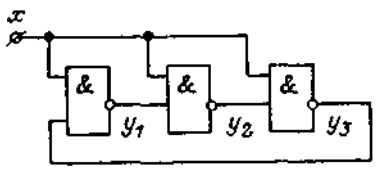

Рис. 5.1

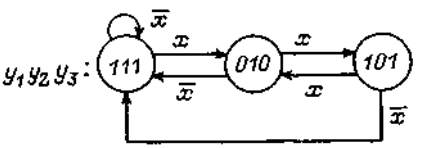

Рис. 5.2

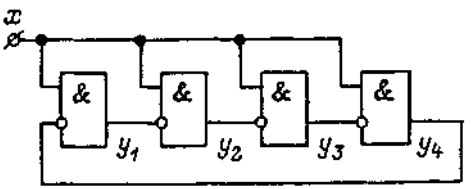

Рис. 5.3

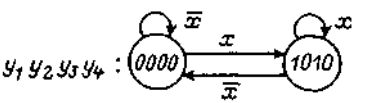

Рис. 5.4

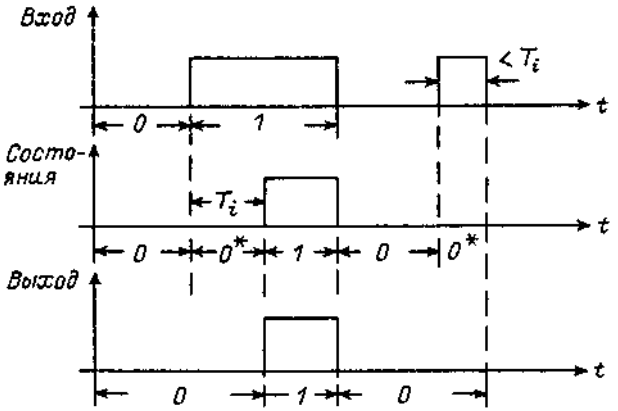

Рис. 5.5

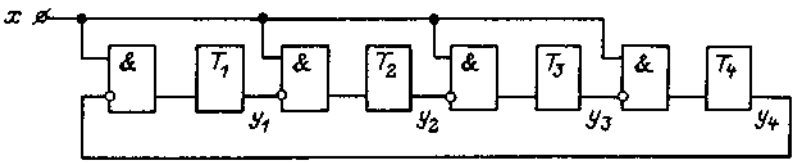

Рис. 5.6

Для практического использования «четных» контуров и исследования переключательных процессов в них рассмотрим схемы, содержащие наряду с элементами ЗАПРЕТ также и элементы задержки (ЭЗ). Так как время срабатывания элементов ЗАПРЕТ существенно меньше этого показателя для ЭЗ, будем считать, что комбинационные элементы безынерционны.

При этом будем предполагать, что каждый элемент задержки в этом случае функционирует следующим образом:

— при отсутствии входного потенциала сигнал на выходе отсутствует (обозначим состояние и выходной сигнал в этом случае символом «0»);

— при поступлении входного потенциала элемент переходит в «возбужденное» состояние (начинает «набирать» время), при этом на его выходе сигнал продолжает отсутствовать (обозначим возбужденное состояние символом «0*», а выходной сигнал символом «0»);

— если входной сигнал продолжает присутствовать, то при наступлении момента времени T_i- элемент срабатывает и на его выходе появляется потенциальный сигнал (обозначим состояние и выходной сигнал в этом случае символом «1»);

— если входной сигнал становится равным нулю, то выходной сигнал, если он был, мгновенно исчезает (становится равным нулю) независимо от того, в каком состоянии «0*» или «1» элемент находился.

Элемент, работающий указанным образом, называется «задержкой на срабатывание» (рис. 5.5).

Предположим, что задан «четный» контур, содержащий n элементов ЗАПРЕТ и n элементов задержки. Для n - 4 схема приведена на рис. 5.6.

При x - 0 схема находится в первом устойчивом состоянии, в котором все элементы задержки находятся в состоянии «0». Если входной сигнал становится равным единице, то все эти элементы возбуждаются и начинают «набирать» время и в схеме начинается переключательный процесс, время завершения которого (момент перехода схемы во второе устойчивое состояние) зависит от соотношения времен запаздывания элементов задержки.

По истечении времени, равного величине минимального запаздывания элементов задержки, за один такт происходит следующее: соответствующий ЭЗ срабатывает и его выходной сигнал переводит соседний ЭЗ в нулевое состояние. Под соседним будем понимать элемент задержки, расположенный справа, учитывая также и крайние элементы контура.

Срабатывание следующего элемента задержки переведет его «соседа» в нулевое состояние, который в свою очередь переведет уже своего «соседа» в возбужденное состояние «0*». В дальнейшем в ходе переключательного процесса число возбужденных элементов в схеме начинает изменяться.

При этом может возникнуть ситуация, в которой возбуждение двух соседних элементов заканчивается их одновременным переходом в состояние «1», т. е. наступает состязание этих элементов.

Определим параметры времен запаздываний элементов задержки, при которых это состязание возникает.

Рассмотрим контуры, соотношения между временами запаздываний в которых удовлетворяют одному из неравенств:

$$\ldots > T_{k-1} > T_k > T_{k+1} > \ldots; \tag{5.1}$$

$$\ldots < T_{k-1} < T_k < T_{k+1} < \ldots, \tag{5.2}$$

где T_k — время запаздывания элемента задержки с номером k.

При выполнении соотношения (5.1) переключательный процесс завершится устойчивым состоянием, в котором все ЭЗ с нечетными номерами будут находиться в состоянии «О», а все ЭЗ с четными номерами — в состоянии «1». Для этого случая характерно, что первый элемент перейдет из состояния «0*» в состояние «0», минуя состояние «1». В рассматриваемом случае невозможно совпадение расчетных моментов перехода в состояние «1» двух каких-либо соседних элементов задержки, т. е. их состязание отсутствует. При этом длительность переключательного процесса равна времени запаздывания первого элемента схемы.

Пример 5.1. Пусть в схеме (рис. 5.6) $T_1 = 4$, $T_2 = 3$, $T_3 = 2$, $T_4 = 1$ Построим таблицу возбуждения элементов задержки для этого случая (табл. 5.1).

При выполнении соотношения (5.2) переключательный процесс завершится устойчивым состоянием, в котором все ЭЗ с нечетными номерами будут находиться в состоянии «1», а все ЭЗ с четными номерами — в состоянии «О». При этом длительность переключательного процесса равна времени запаздывания предпоследнего элемента схемы.

Пример 5.2. Пусть в схеме (рис. 5.6) $T_1 = 1$, $T_2 = 2$, $T_3 = 3$, $T_4 = 4$ Построим таблицу возбуждения элементов задержки для этого случая (табл. 5.2).

Таблица 5.1

№	$T_1 = 4$	$T_2 = 3$	$T_3 = 2$	$T_4 = 1$
0	0	0	0	0
1	0*	0*	0*	0*
2	0	0*	0*	1
3	0*	0*	1	0
4	0*	1	0	0*
5	0	1	0	1

Таблица 5.2

№	$T_1 = 1$	$T_2 = 2$	$T_3 = 3$	$T_4 = 4$
0	0	0	0	0
1	0*	0*	0*	0*
2	1	0	0*	0*
3	1	0	0*	0*
4	1	0	1	0

Из рассмотрения табл. 5.1 и 5.2 следует, что при выполнении соотношений (5.1) и (5.2) в контуре по крайней мере один элемент не переходит в единичное состояние ни одного раза. Определим соотношения между временами запаздываний элементов схемы таким образом, чтобы каждый элемент задержки в ней неоднократно переходил бы в это состояние. Для этого выберем времена запаздывания элементов задержки схемы таким образом, чтобы для части этих элементов выполнялось соотношение (5.2), а для остальных — соотношение (5.1). Осуществим этот выбор следующим образом:

$$T_i < T_2 > T_3 > T_4 > \ldots > T_n . \qquad (5.3)$$

Соотношения для определения величин T_i, выберем так, чтобы расчетные моменты срабатывания для первого и второго элементов задержки наступали одновременно, т. е. имело бы место состязание этих элементов. Результатом состязания может быть новое устойчивое состояние схемы либо продолжение переключательного процесса. Первая ситуация наступает, если состязание выигрывает первый элемент, а вторая — если второй. В случае наступления устойчивого состояния нечетные элементы будут находиться в единичном состоянии, а четные — в нулевом.

Необходимо отметить, что при выполнении неравенства (5.3), во-первых, первый элемент впервые перейдет из состояния «0*» в состояние «0» через единичное состояние, что отсутствовало при выполнении неравенства (5.1), а во-вторых, с момента первого перехода этого элемента в единичное состояние для переключательного процесса характерна периодичность появления «передних фронтов» единичных импульсов, формируемых первым и вторым элементами задержки. На основе [108] можно показать, что эти события наступят одновременно в момент времени

$$t = T_{\text{н}} \frac{T_{\text{ч}} - T_2}{T_{\text{ч}} - T_{\text{н}}}, \tag{5.4}$$

где $T_{\text{ч}}$ — сумма времен запаздываний четных элементов; $T_{\text{н}}$ — сумма времен запаздываний нечетных элементов.

Если предположить, что все времена запаздываний целые числа, то из соотношения (5.4) следует, что величина t максимальна при

$$T_{\text{ч}} - T_{\text{н}} = 1. \tag{5.5}$$

Учитывая неравенство (5.3), соотношение (5.5) выполняется [108] при

$$T_1 = T_2 - \frac{n}{2}; \tag{5.6}$$

$$T_i = T_2 - i + 1, \tag{5.7}$$

где $i = 3, 4, \ldots, n$.

Покажем, что при указанном выборе времен запаздываний соотношение (5.5) действительно выполняется. Учитывая соотношения (5.6) и (5.7), получим соотношения для определения величин $T_{\text{н}}$ и $T_{\text{ч}}$, предварительно отметив, что

$$1 + 3 + 5 + \ldots + (n-1) = \left(\frac{n}{2}\right)^2; \quad 2 + 4 + 6 + \ldots + n = \frac{n}{2}\left(\frac{n}{2} + 1\right).$$

При этом

$$T_{\text{н}} = T_1 + T_3 + T_5 + \ldots + T_{n-1} = \left(T_2 - \frac{n}{2}\right) + (T_2 - 3 + 1) +$$

$$+ (T_2 - 5 + 1) + \ldots + (T_2 - (n-1) + 1) = \frac{n}{2} T_2 - \frac{n}{2} - \left(\left(\frac{n}{2}\right)^2 - 1\right) +$$

$$+ \frac{n}{2} - 1 = \frac{n}{2} T_2 - \left(\frac{n}{2}\right)^2 = \frac{n}{2}\left(T_2 - \frac{n}{2}\right);$$

$$T_{\text{ч}} = T_2 + T_4 + T_6 + \ldots + T_n = T_2 + (T_2 - 4 + 1) +$$

$$+ (T_2 - 6 + 1) + \ldots + (T_2 - n + 1) = \frac{n}{2} T_2 - \left(\frac{n}{2}\left(\frac{n}{2} + 1\right) - 2\right) +$$

$$+ \frac{n}{2} - 1 = \frac{n}{2} T_2 - \left(\frac{n}{2}\right)^2 + 1 = \frac{n}{2}\left(T_2 - \frac{n}{2}\right) + 1.$$

Таким образом,

$$T = T_н + T_ч = n\left(T_2 - \frac{n}{2}\right) + 1. \quad (5.8)$$

Полученные соотношения для $T_н$, $T_ч$ и соотношение (5.5) позволяют на основе формулы (5.4) определить t как функцию только от переменных n и T_2. Эта функция имеет вид

$$t = \frac{n}{2}\left(T_2 - \frac{n}{2}\right)\left(\frac{n}{2}\left(T_2 - \frac{n}{2}\right) + 1 - T_2\right). \quad (5.9)$$

Наиболее важным для рассматриваемого класса схем является то свойство, что время переключательного процесса (величина t) может превышать сумму времен запаздываний всех элементов задержки в контуре.

Необходимо отметить, что минимальная величина n, при которой обеспечивается указанное свойство, равна четырем, так как при $n = 2$ соотношение (5.3) не может быть выполнено.

При n - 4 определим, используя соотношения (5.3)—(5.7), минимальное значение T_2, при котором t превышает сумму времен запаздываний всех элементов задержки контура, равную T.

При $T_2 = 4$ из указанных соотношений следует, что $T_1 = 2$, $T_3 = 2$, $T_4 = 1$ и $(t = 4) < (T = 9)$.

При $T_2 = 5$ из этих соотношений следует, что $T_1 = 3$, $T_3 = 3$, $T_4 = 2$ и $(t = 12) < (T = 13)$.

При $T_2 = 6$ имеем $T_1 = 4$, $T_3 = 4$, $T_4 = 3$ и $(t = 24) > (T = 17)$.

Таким образом, искомое значение T_2 равно шести.

Пример 5.3. Построить таблицу возбуждения элементов задержки для схемы (рис. 5.6) при $T_1 = 4$, T_2 - 6, $T_3 = 4$, $T_4 = 3$ (табл. 5.3).

Обратим внимание на строку 24, в которой происходит состязание элементов $ЭЗ_1$, и $Э3_2$ (оба элемента одновременно «набрали» необходимое время). Строка 25 заполнена в предположении, что состязание выиграл первый элемент. При этом схема перешла в устойчивое состояние: переключательный процесс завершился.

Для того чтобы этот процесс завершился устойчивым состоянием без состязания элементов, можно увеличить значение $T_ч$ на величину $At = 1/(T_н - T_2)$ [109]. Для последнего примера эта величина равна 0.5.

Пример 5.4. Определить величину t в «четном» контуре при $n = 4$ и $T_2 = 1$.

Из соотношений (5.8) и (5.9) следует, что T - 21 и $t = 40$.

Пример 5.5. Определить величину t в «четном» контуре при л = 6 и $T_2 = 7$.

Из соотношений (5.8) и (5.9) следует, что $T = 25$ и $t = 72$.

В случае, когда переключательный процесс завершается устойчивым состоянием без состязания, его длительность определяется соотношением

$$T_{ПП} = T_н \left\lceil \frac{T_ч - T_2}{T_ч - T_н} \right\rceil. \quad (5.10)$$

Таблица 5.3

№	$T_1 = 4$	$T_2 = 6$	$T_3 = 4$	$T_4 = 3$	№	$T_1 = 4$	$T_2 = 6$	$T_3 = 4$	$T_4 = 3$
0	0	0	0	0	13	0*	0*	1	0
1	0*	0*	0*	0*	14	0*	0*	1	0
2	0*	0*	0*	0*	15	0*	0*	1	0
3	0*	0*	0*	0*	16	0*	1	0	0*
4	0	0*	0*	1	17	1	0	0*	0*
5	0*	0*	1	0	18	1	0	0*	0*
6	0*	0*	1	0	19	0	0*	0*	1
7	0*	1	0	0*	20	0	0*	0*	1
8	0*	1	0	0*	21	0*	0*		0
9	1	0	0*	0*	22	0*	0*		0
10	0	0*	0*	1	23	0*	0*		0
11	0	0*	0*	1	24	0*	0*		0
12	0	0*	0*	1	25	1	0		0

В соотношениях (5.4) и (5.10) величины $L_1 = \frac{T_ч - T_2}{T_ч - T_н}$ и $L_2 = \left]\frac{T_н - T_2}{T_ч - T_н}\right[$ равны числу единичных импульсов, формируемых первым элементом задержки контура, причем в первом случае в момент формирования последнего импульса происходит состязание, а во втором — переключательный процесс завершается без состязаний.

Это замечание необходимо учитывать при решении обратной задачи: пусть заданы значения величин n и t и требуется определить значения времен запаздываний элементов «четного» контура. При этом предполагается, что указанные значения могут настраиваться в диапазоне от λ_1 до λ_2 с кратностью Δ. Если значение L_1 — целое, то параметры контура обеспечивают получение $T_{пп} = t$. Если L_1 — дробное, то параметры контура должны быть выбраны так, чтобы значение $T_{пп}$ превосходило величину t, но было к нему наиболее близким.

Решая уравнение (5.9) относительно T_2, получим:

$$T_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \qquad (5.11)$$

где

$$a = \frac{n}{2}\left(\frac{n}{2} - 1\right); \quad b = -\frac{n}{2}\left(\frac{n^2}{2} - \frac{n}{2} - 1\right); \quad c = \frac{n^4}{16} - \frac{n^2}{4} - t.$$

Пример 5.6. Пусть $n = 4$, $t = 24$, $\lambda_1 = 0$, $\lambda_2 = 10$, $\Delta = 0.1$. Определить T_i, где $i = 1..4$.

Из соотношения (5.11) следует, что так как $a = 2$, $b = -10$, $c = -12$, то $T_2 = 6$. При этом из соотношений (5.6) и (5.7) следует, что $T_1 = 4$, $T_3 = 4$, $T_4 = 3$.

Пример 5.7. Пусть $n = 4$, t - 20, $\lambda_1 = 0$, $\lambda_2 = 10$, $\Delta = 0.1$. Определить Г,, где $i = 1..4$.

Из соотношения (5.11) следует, что так как $a = 2$, $b = -10$, $c = -8$, то $T_2 = 5.7$. При этом из соотношений (5.6) и (5.7) следует, что $T_1 = 3.7$, T_3 - 3.7, $T_4 = 2.7$. Используя формулу (5.10), получим $T_{пп} = 22.2$.

Уменьшим величину T_2. Пусть $T_2 = 5.4$. При этом из соотношений (5.6) и (5.7) следует, что $T_1 = 3.4$, $T_3 = 3.4$, $T_{пп} = 2.4$. Используя формулу (5.10), получим $T_{пп} = 20.4$.

При $T_2 = 5.3$ получим $T_{пп} = 19.8$. Таким образом, величина $T_{пп}$, превышающая двадцать и наиболее близкая к этому значению, равна 20.4. При этом Г, = 3.4, $T_2 = 5.4$, $T_3 = 3.4$ и $T_4 = 2.4$.

Построение таблицы возбуждения с шагом 0.1 показало, что в устойчивое состояние (1010) схема переходит в строке с номером 20.5. Так как возбуждение схемы начинается в строке 0.1, то запаздывание равно 20.4.

Пример 5.8. Пусть $n = 4$, t - 46, $\lambda_1 = 0$, $\lambda_2 = 10$, $\Delta = 0.1$. Определить Г,-, где $i = 1 \div 4$

Из соотношения (5.11) следует, что так как a - 2, $b = -10$, $c = -34$, то $T_2 = 7.3$. При этом из соотношений (5.6) и (5.7) следует, что $T_1 = 5.3$, $T_2 = 5.3$, $T_4 = 4.3$. Используя формулу (5.10), получим $T_{пп} = 53$.

Уменьшим величину T_2. Пусть $T_2 = 7.1$. При этом из соотношений (5.6) и (5.7) следует, что Г, = 5.1, $T_3 = 5.1$, $T_4 = 4.1$. Применяя формулу (5.10), получим $T_{пп} = 51$. При $T_2 = 7$ $T_{пп} = 40$. Таким образом, величина $T_{пп}$, превышающая 46 и наиболее близкая к этому значению, равна 51. При этом $T_1 = 5.1$, $T_2 = 7.1$, $T_3 = 5.1$, $T_4 = 4.1$.

Задача, поставленная в последнем примере, может быть решена точно при эвристическом выборе параметров элементов задержки: $T_1 = 8$, $T_2 = 12$, $T_3 = 7$, T_4 - 5. При этом переключательный процесс завершается без состязания, а из формулы (5.10) следует, что $T_{пп} = 45$ (табл. 5.4).

5.2.2. Модели функциональных элементов задержки

В предыдущем разделе изложен новый подход к построению ФЭЗ. Однако на практике при программной реализации распространение получили другие модели ФЭЗ.

Рассмотрим двухвходовой ФЭЗ (рис. 5.7). Представим этот элемент в виде автомата и простейшего элемента задержки (ПЭЗ) (рис. 5.8).

ПЭЗ работает следующим образом: при появлении сигнала «засечки времени» (z_i - 1) фиксируется время наступления этого события (t_3);

Таблица 5.4

№	$T_1 = 8$	$T_2 = 12$	$T_3 = 7$	$T_4 = 5$	№	$T_1 = 8$	$T_2 = 12$	$T_3 = 7$	$T_4 = 5$
0	0	0	0	0	23	0*	0*	1	0
1	0*	0*	0*	0*	24	0*	0*	1	0
2	0*	0*	0*	0*	25	0*	0*	1	0
3	0*	0*	0*	0*	26	0*	0*	1	0
4	0*	0*	0*	0*	27	0*	0*	1	0
5	0*	0*	0*	0*	28	0*	0*	1	0
6	0	0*	0*	1	29	0*	0*	1	0
7	0	0*	0*	1	30	0*	1	0	0*
8	0*	0*	1	0	31	1	0	0*	0*
9	0*	0*	1	0	32	1	0	0*	0*
10	0*	0*	1	0	33	1	0	0*	0*
11	0*	0*	1	0	34	1	0	0*	0*
12	0*	0*	1	0	35	0	0*	0*	1
13	0*	1	0	0*	36	0	0*	0*	1
14	0*	1	0	0*	37	0	0*	0*	1
15	0*	1	0	0*	38	0*	0*	1	0
16	1	0	0*	0*	39	0*	0*	1	0
17	1	0	0*	0*	40	0*	0*	1	0
18	0	0*	0*	1	41	0*	0*	1	0
19	0	0*	0*	1	42	0*	0*	1	0
20	0	0*	0*	1	43	0*	0*	1	0
21	0	0*	0*	1	44	0*	0*	1	0
22	0	0*	0*	1	45	0*	0*	1	0
					46	1	0	1	0

текущее время ($t_т$) сравнивается со временем «засечки» и определяется величина $\Delta = t_т - t_з$; если величина Δ меньше времени задержки (t), то выход ПЭЗ равен нулю ($T = 0$), в противном случае $T = 1$

Функционирование автомата, входящего в состав этого ФЭЗ, описывается ГП (рис. 5.9). При этом предполагается, что сигналы запуска и сброса ФЭЗ ортогональны ($z_1 z_2 = 0$), так как при построении ГП автомата, внешнего по отношению к ФЭЗ, в каждом состоянии обращение осущес-

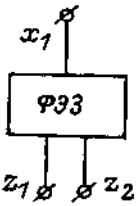

Рис. 5.7

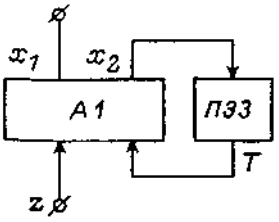

Рис. 5.8

твляется только к одной из двух подпрограмм, реализующих функции запуска и сброса. Это позволяет при программной реализации рассматривать этот элемент как одновходовой.

В нормальном режиме работы этот элемент выдает импульс, передний фронт которого задержан на время уставки относительно появления переднего фронта сигнала запуска, а задний фронт импульса формируется в момент появления переднего фронта сигнала сброса (t_c) (рис. 5.10).

Рассмотренный ФЭЗ обладает тем свойством, что он может быть сброшен при $\Delta < t$. Если необходимо использовать задержку, для которой обязательно должно выполняться соотношение $\Delta \geqslant t$, то эта задержка может быть одновходовой как при аппаратной, так и при программной реализациях.

Рассмотрим теперь одновходовой элемент задержки ФЭ31 (рис. 5.11). Представим этот элемент в виде автомата А1 и ПЭЗ (рис. 5.12). Выберем в качестве модели поведения автомата А1 граф переходов, приведенный на рис. 5.13. В нормальном режиме работы этот элемент выдает короткий сигнал после истечения времени задержки, отсчитываемого от момента обращения к ней (рис. 5.14).

При использовании элементов задержки в контуре управления целесообразно иметь также одновходовой элемент ФЭ32 (рис. 5.15, 5.16), формирующий импульс, длина которого равна величине задержки t (рис. 5.17). ГП автомата А2, входящего в состав этого элемента, приведен на рис. 5.18.

Рассмотренные ФЭЗ обладают также интересным свойством, состоящим в том, что если снять сигнал запуска ($z_1 = 0$ или $z = 0$) в момент, когда автомат находится в состоянии «2» («ФЭЗ набирает время»), то автомат перейдет в состояние «3» («ожидание нового запроса»). При этом ПЭЗ, являющийся относительно этого автомата внешней средой, продол-

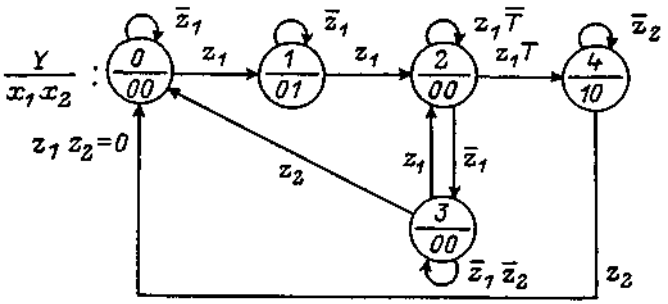

Рис. 5.9

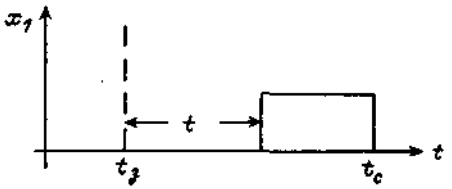

Рис. 5.10

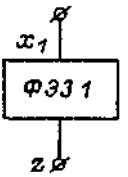

Рис. 5.11

жает набирать время. При повторном запросе на запуск (z_1 = 1 или z - 1) автомат вновь переходит в состояние «2». Если за время пребывания в третьем состоянии ПЭЗ еще не сработал (T = 0), то он продолжает набирать время и в состоянии «2», в противном же случае (T = 1) ФЭЗ срабатывает для первых двух типов элементов задержки единицей (x_1 = 1) в состоянии «4», а для третьего — нулем (x_2 = 0) в состоянии «0».

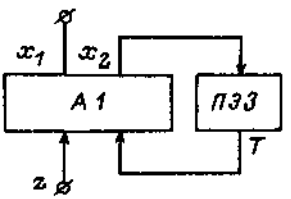

Рис. 5.12

Кроме рассмотренных, естественно, могут быть построены и другие ФЭЗ, и в частности такие, в которых снятие сигнала запуска (в состоянии «пока элемент еще не набрал время») может использоваться не для перехода в состояние ожидания, а для возвращения ФЭЗ в исходное состояние. Могут быть

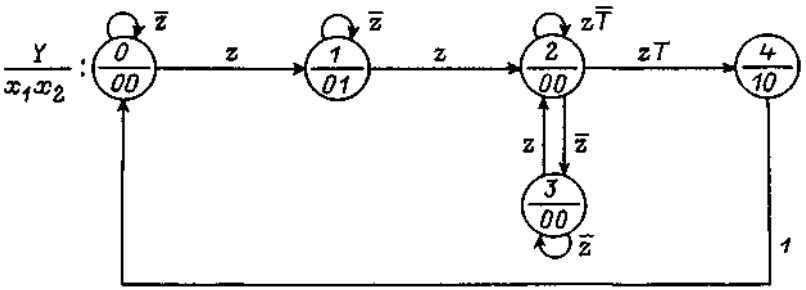

Рис. 5.13

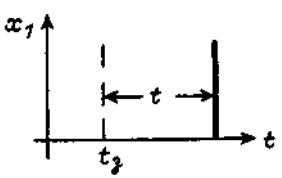

Рис. 5.14

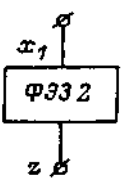

Рис. 5.15

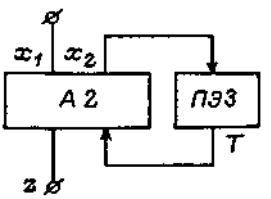

Рис. 5.16

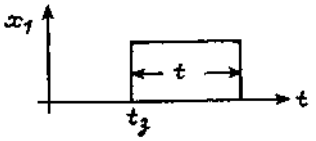

Рис. 5.17

построены также и элементы, в которых нет необходимости сохранять сигнал запуска в течение всего времени задержки, — импульсные по входу ФЭЗ.

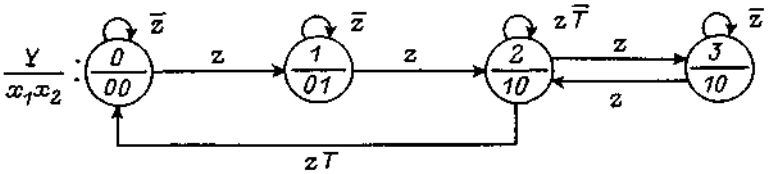

Рис. 5.18

5.2.3. Программная реализация функциональных элементов задержки

В Приложении 4 в качестве примера приведены две подпрограммы `_time (i, j)`, `reset_time (i)`, которые реализуют функциональный элемент задержки. Значения «единица» и «ноль», вырабатываемые этими подпрограммами, присваиваются переменной t [i]. При этом используются следующие обозначения: i — порядковый номер элемента задержки; l' — величина запаздывания в секундах.

Для элементов этого типа может быть использована также и процедура `dly(+(-)i,...,+(-)l, end)`, где `i,...,l` — порядковые номера элементов задержки, запускаемых (+) и сбрасываемых (-) этой процедурой. Величины запаздывания этих элементов в секундах задаются в специальном массиве. Значения, вырабатываемые этой процедурой для i-го ФЭЗ, присваиваются переменной d[i]

Еще один вариант построения элементов этого типа рассмотрен в разд. 14.3.2 для случая, когда в контроллере весьма сложно выполняются процедуры. При этом i-й ФЭЗ реализуется в виде i-го счетчика, в который раз в секунду прибавляется единица, что на языке СИ может быть промоделировано, например, процедурой `time (&t`$_i$`)`. Сбросу счетчика соответствует микрооперация $t_i = 0$. ФЭЗ срабатывает, если $t_i \geqslant T_i$

Элементы задержки могут быть реализованы также и с помощью таймеров (разд. 14.3.1).

5.3. Программная реализация моделей объектов управления

Автономная проверка управляющих автоматов или их совокупности осуществляется при подаче с помощью клавиатуры ПЭВМ входных переменных, имитирующих работу органов управления и сигнализаторов объектов управления.

Для комплексной проверки управляющих автоматов или их совокупности в программу необходимо ввести модели объектов управления, которые, получая информацию с выходов указанных автоматов, автоматически формируют часть входных воздействий, имитирующих работу сигнализаторов объектов. При этом с помощью клавиатуры приходится имитировать только работу органов управления, что является естественным.

Предположим, что в качестве объекта управления используется клапан с памятью, имеющий два входа (z_1, — закрытие, z_2 — открытие) и два выхода (x_3 — закрыт, x_4 — открыт). Предположим также, что его открытие и закрытие выполняются в течение трех секунд (пример 40 Приложения 5).

Выберем в качестве модели клапана управляющий автомат, состоящий из автомата Мура второго рода и ФЭЗ. При этом автомат Мура описывается ГП с четырьмя вершинами и программно реализуется, например, следующим образом:

```
        switch (Y)  {
case 0: if(z2)        Y = 1; break;
case 1: if(t[1])      Y = 2; break;
case 2: if(z1)        Y = 3; break;
case 3: if(t[1])      Y = 0; break;
                    }
        switch (Y)  {
case 0: x3=1; x4 = 0; reset _time(1);   break;
case 1: x3=0; x4 = 0;       _time1(1,3);break;
case 2: x3=0; x4 = 1; reset _time(1);   break;
case 3: x3=0; x4 = 0;       _time1(1,3);break;
                    }.
```

5.4. Примеры программной реализации алгоритмов логического управления

5.4.1. Примеры построения графов переходов

Рассмотрим несколько примеров построения ГП.

Пример 5.9. Пусть требуется построить счетный триггер, функционирование которого задается временными диаграммами x/t и z/t (рис. 5.19).

Из рассмотрения диаграмм следует, что в данном случае имеет место периодичность наступления событий, определяемых передними фронтами соседних входных сигналов. Для фиксации этих событий построим новую диаграмму Y/t (рис. 5.19).

Указанные временные диаграммы могут быть заменены замкнутой компактной формой представления заданного алгоритма — графом переходов (рис. 5.20). Этот ГП соответствует автомату Мура. Указанный алгоритм может быть реализован также универсальным автоматом без выходного преобразователя с принудительно-свободным кодированием. При этом младший разряд кодов состояний является выходом автомата (рис. 5.21).

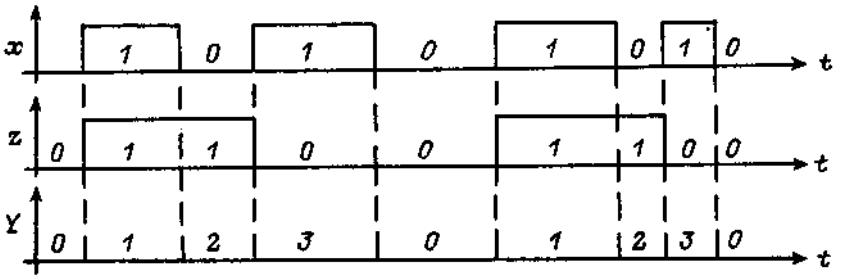

Рис. 5.19

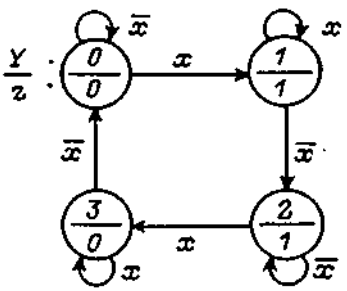

Рис. 5.20

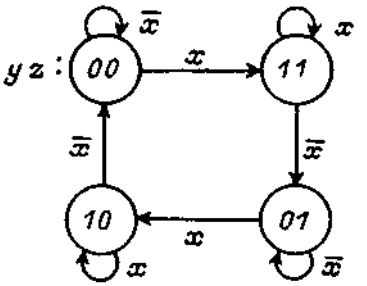

Рис. 5.21

Если говорить о программной реализации, то рассмотрение вопроса о построении спецификации можно считать завершенным, однако если иметь в виду также и аппаратную реализацию, то формализация должна быть продолжена.

Действительно, при x — 1 могут иметь место критические состязания, в результате которых автомат из состояния «00» может перейти не в состояние «11», как это определяется ГП, а в состояние «10», которое при указанном значении входного сигнала устойчиво.

При использовании противогоночного кодирования (в данном случае кода Грея: 0 —> 00; 1 —> 01; 2 —> 11; 3 —» 10) критические состязания устраняются (рис. 5.22). Этому ГП соответствует СБФ вида

$$y' = \bar{x}\, z \vee xy; \quad z = \bar{x}\, z \vee x\, \bar{y}.$$

Однако, выполнив реализацию триггера по этой системе формул, возникает проблема риска, связанная с тем, что в рассмотренные формулы одна и та же переменная входит одновременно в прямой и инверсной формах. При этом если $y = z = 1$, то СБФ приобретает вид:

$$y' = \bar{x} \vee x; \quad z = \bar{x}.$$

При $x = 0$ автомат находится в состоянии «11». При появлении $x = 1$ может возникнуть ситуация, в которой вместо состояния «10» автомат

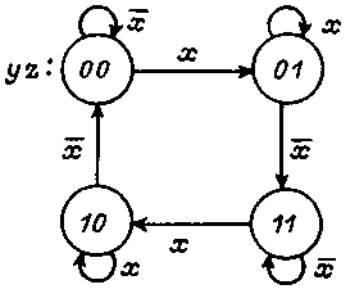

Рис. 5.22

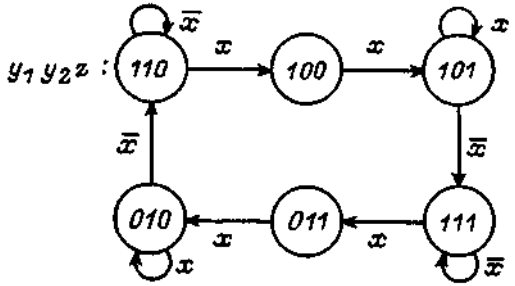

Рис. 5.23

перейдет в состояние «00», из которого при $x = 1$ он перейдет в состояние «10», являющееся при указанном значении входного сигнала устойчивым и ошибочным.

Для устранения указаного недостатка увеличим число состояний автомата и построим остов нового ГП автомата без выходного преобразователя. Используем и в этом случае противогоночный код, при котором в смежных состояниях комбинации отличаются только в одном разряде, а выходом является младший разряд кода (рис. 5.23).

Построенный ГП [13] является неполным как по переходам (из состояний «100» и «011»), так и по состояниям («000» и «001»). Используя карты Карно и доопределяя функции разрядов кода y_1, y_2, z с целью построения монотонных формул, получим:

$$y_1' = \bar{x} \vee \bar{y}_2 \vee y_1\bar{z}; \quad y_2' = \bar{x} \vee \bar{y}_1 y_2 z; \quad z = y_1 z \vee \bar{y}_2.$$

ГП, построенный по этой СБФ, приведен на рис. 5.24.

Пример 5.10. Пусть необходимо построить автомат, осуществляющий дистанционное управление трехпозиционным клапаном с памятью от трех кнопок без памяти: x_1, — открытие; x_2 — закрытие; x_3 — сброс команды.

Выходы автомата обозначим переменными z_1 и z_2 (рис. 5.25).

Автомат должен реализовать следующий алгоритм: если клапан был открыт, то нажатием кнопки x_2 он закрывается; после его закрытия кнопкой x_3 осуществляется сброс команды; если клапан был закрыт, то нажатием кнопки x_1 он открывается; после его открытия кнопкой x_3 осуществляется сброс команды.

В качестве структурной модели выберем автомат Мура. Построим по словесному заданию исходный ГП. При этом нулевая вершина соответствует закрытому и открытому состояниям клапана, первая — состоянию клапана «открывается», вторая — состоянию клапана «закрывается». В нулевой вершине должны формироваться выходные сигналы $z_1 = 0$, $z_2 = 0$; в первой — $z_1 = 0$, $z_2 = 1$; во второй — $z_1 = 1$, $z_2 = 0$ (рис. 5.26). Так как в данном случае все векторы значений выходных сигналов различны, то имеется возможность использовать принудительное кодирование состояний автомата (0 —> 00, 1 —> 01,2 —> 10) и перейти от автомата Мура к автомату без выходного преобразователя. Полученный ГП приведен на рис. 5.27.

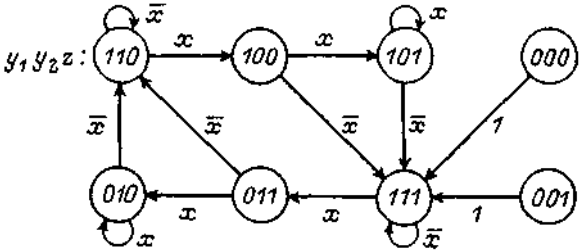

Рис. 5.24

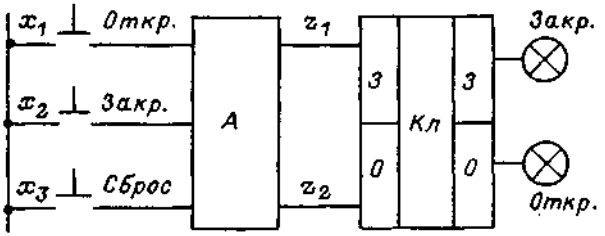

Рис. 5.25

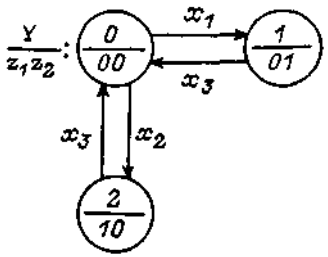

Рис. 5.26

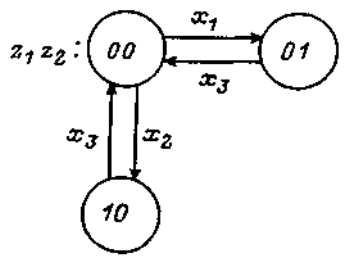

Рис. 5.27

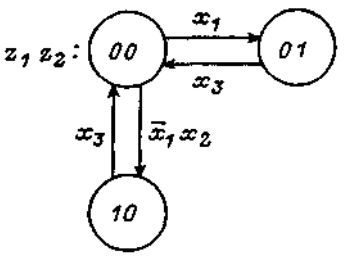

Рис. 5.28

Устраним противоречивость для вершины «00». Противоречие возникает, ввиду того что пометки исходящих из этой вершины дуг имеют одну и ту же составляющую x_1x_2, так как

$$x_1 = x_1\bar{x}_2 \vee x_1 x_2 \quad \text{и} \quad x_2 = \bar{x}_1 x_2 \vee x_1 x_2.$$

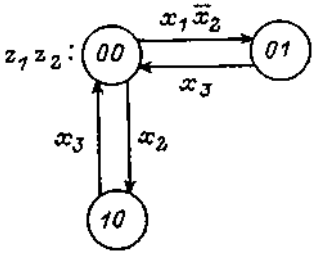

Рис. 5.29

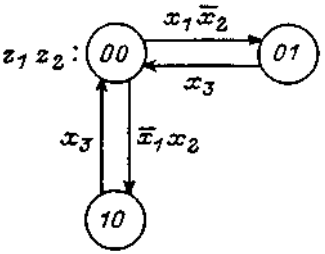

Рис. 5.30

Возможны различные варианты устранения противоречивости, связанные с исключением общей конъюнкции x_1x_2: в x_2 (приоритет открытия — рис. 5.28), в x_1 (приоритет закрытия — рис. 5.29) и в x_1, и x_2 одновременно (рис. 5.30).

После устранения противоречивости перейдем к устранению формальной неполноты для каждой вершины. При этом доопределение всех вершин будем проводить на основе одного общего правила: все конъюнкции, не вошедшие в пометки исходящих из рассматриваемой вершины дуг, отмечают петлю, обеспечивающую сохранение состояния, соответствующего этой вершине. Используя теорему 1 (разд. 2.3), обеспечим в качестве примера полноту ГП, представленного на рис. 5.30. Полный и непротиворечивый ГП приведен на рис. 5.31.

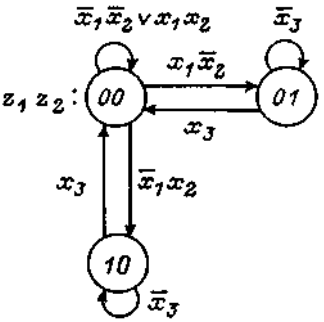

Рис. 5.31

Таким образом, противоречивость и неполнота словесного задания приводят к многовариантности построения спецификации.

Приведенный пример чрезвычайно важен для практики, так как показывает, что даже при реализации простейших условий работы возможно получение различных формальных спецификаций. Выбор и согласование одной из них должны осуществляться с участием Заказчика (Технолога). Согласование на ранней стадии ситуаций, не определенных техническим заданием, резко повышает достоверность постановки решаемой задачи и позволяет Разработчику и Заказчику лучше понять ее. После этого Заказчик может изменить или дополнить (не формально, а содержательно) исходное задание.

Пример 5.11. Предположим, что Заказчик согласовал устранение противоречивости без приоритетов (рис. 5.30). Однако при обеспечении полноты нулевой вершины он решил воспользоваться (в отличие от остальных вершин) не теоремой 1 (разд. 2.3), а зафиксировать ситуацию $x_1 = x_2 = 1$, не определенную в задании, на информационном табло с последующим его сбросом нажатием кнопки x_3,.

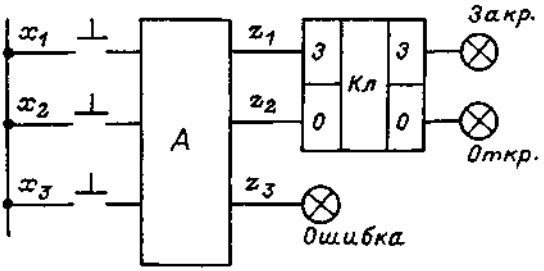

Рис. 5.32

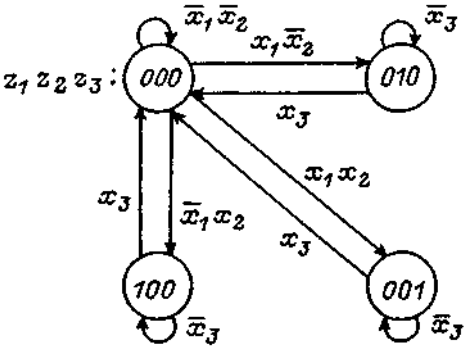

Рис. 5.33

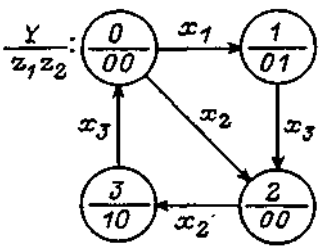

Рис. 5.34

Это приводит к увеличению числа выходов автомата до трех (рис. 5.32), а числа его состояний до четырех. При этом могут быть использованы автомат без выходного преобразователя и принудительное кодирование его состояний (рис. 5.33).

Многовариантность построения формальной спецификации на этом не исчерпывается.

Пример 5.12. Приведенное в примере *5.10* техническое задание позволяет в качестве исходного выбрать также ГП, соответствующий автомату Мура, который содержит четыре вершины. Такой ГП является более естественным, так как позволяет сопоставить каждому состоянию клапана состояние управляющего им автомата и обеспечивает возможность определения предполагаемого состояния объекта управления по состоянию этого автомата. При этом нулевая вершина графа переходов соответствует состоянию «клапан закрыт»; первая — состоянию «клапан открывается», вторая — «клапан открыт», а третья — состоянию «клапан закрывается». При этом в нулевой и второй вершинах графа должны формироваться

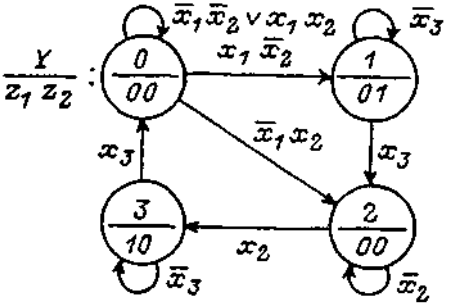

Рис. 5.35

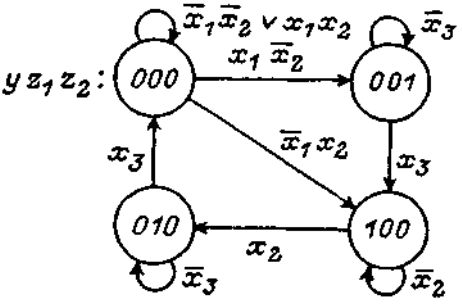

Рис. 5.36

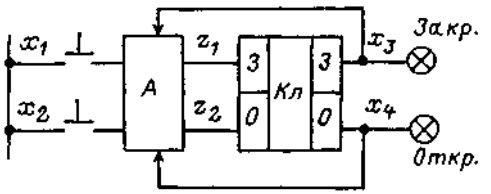

Рис. 5.37

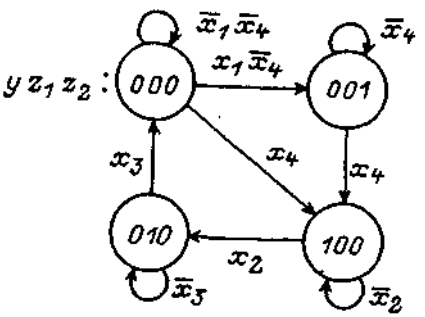

Рис. 5.38

выходные сигналы $z_1 = 0$, $z_2 = 0$; в первой — $z_1 = 0$, z_2 — 1, а в третьей — сигналы $z_1 = 1$, $z_2 = 0$ (рис. 5.34). В этом случае противоречивость и неполнота устраняются так же, как в примере *5.10* (рис. 5.35). Если отказаться от применения автомата Мура, то, в силу того что в построенном ГП в двух вершинах должны формироваться одинаковые выходные сигналы, автомат без выходного преобразователя с принудительным кодированием состояний использовать невозможно, а требуется применение универсального автомата этого класса с принудительно-свободным кодированием. ГП с выбранным в качестве примера вариантом кодирования приведен на рис. 5.36.

Пример 5.13. Построить автомат, аналогичный рассмотренному в предыдущем примере, для случая, когда сброс команд выполняется автоматически (рис. 5.37).

Спецификация для этого примера приведена на рис. 5.38.

Пример 5.14. Построить автомат, аналогичный рассмотренному в предыдущем примере, который дополнительно обеспечивает сигнализацию того, что по истечении интервала времени D открытие (закрытие) клапана не состоялось. Сброс сигнализации осуществляется нажатием кнопки x_5.

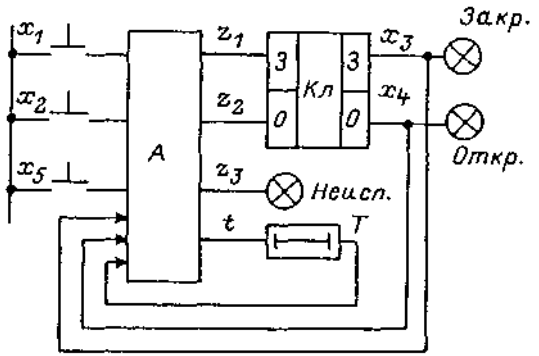

Рис. 5.39

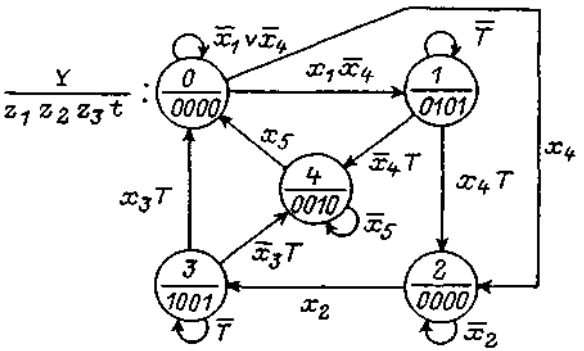

Рис. 5.40

Автомат в данном случае имеет четыре выхода и шесть входов (рис. 5.39). ГП с пятью вершинами реализует заданный алгоритм (рис. 5.40).

Пример 5.15. Построить автомат Мура, формирующий импульсный выходной сигнал по переднему фронту потенциального входного сигнала.

Автомат в данном случае имеет три состояния. При этом единичное значение выходного сигнала формируется во втором из них, являющемся неустойчивым состоянием. Этот автомат реализуется следующей программой, по которой однозначно и изоморфно может быть восстановлен ГП:

```
        switch (Y)  {
case 0: z = 0;
        if (x)     Y = 1;
        break;
case 1: z = 1;     Y = 2;
        break;
```

```
case 2: z = 0;
        if (x̄)     Y = 0;
        break;
          }.
```

Приведенная реализация соответствует автомату Мура первого рода.

Рассмотренные примеры связаны с реализацией заданного алгоритма одним автоматом и не отражают всего многообразия вопросов, которые возникают при программной реализации систем логического управления. К этим вопросам относятся, например, вопросы реализации совокупности автоматов, моделирования объектов управления, «параллельной» работы ФЭЗ и т. д. Эти вопросы рассмотрены в следующем разделе. Другие особенности построения алгоритмов логического управления изложены в гл. 8, 9 и разд. 12.4.

5.4.2. Сравнение вариантов программной реализации алгоритмов логического управления

В предыдущем разделе был рассмотрен вопрос о разработке алгоритма управления трехпозиционным клапаном (клапан с памятью).

Однако выбор столь простого объекта управления не позволяет продемонстрировать различные особенности разработки управляющих алгоритмов. Поэтому рассмотрим задачу о реализации различных алгоритмов логического управления двумя клапанами с памятью с помощью двух кнопок без памяти (Приложение 5).

Предположим, что алгоритм управления задан в словесной форме (алгоритм 1.1). Будем первоначально реализовывать его одним автоматом. Схема связей «ИИ—А—ИМ» приведена на рис. 5.41. В этой и последующих схемах связей лампы сигнализации положений клапанов для упрощения схем не приводятся.

Специфицировать этот алгоритм будем с помощью графов переходов. В данном случае могут быть построены две спецификации (два ГП), отличающиеся между собой числом состояний (вершин). В первой из них число состояний равно пяти (рис. 5.42), а во второй — шести (рис. 5.43).

В первом ГП нулевое состояние соответствует открытому и закрытому состояниям обоих клапанов; во втором ГП в нулевом состоянии клапаны закрыты, а в третьем состоянии графа переходов они открыты.

Вторая спецификация является более естественной: траектория (0, 1, 2, 3) — открытие клапанов; траектория (3, 4, 5, 0) — их закрытие.

Реализация этих спецификаций связана с выбором:

— структурной модели, включая ее род (модификацию);

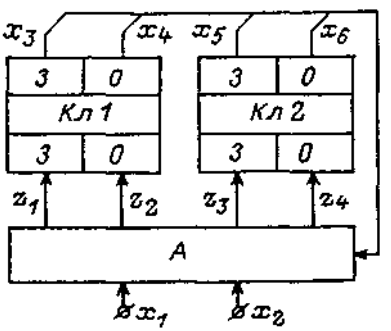

Рис. 5.41

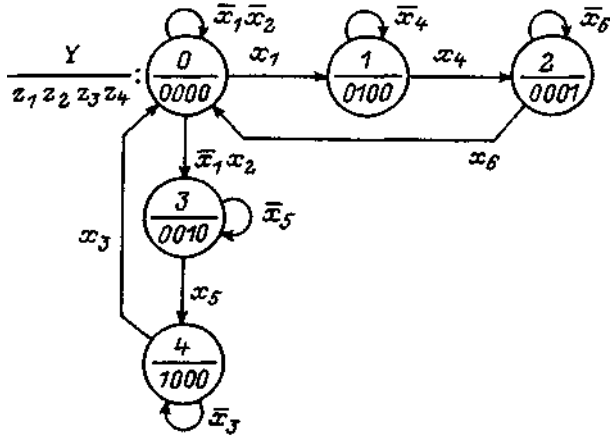

Рис. 5.42

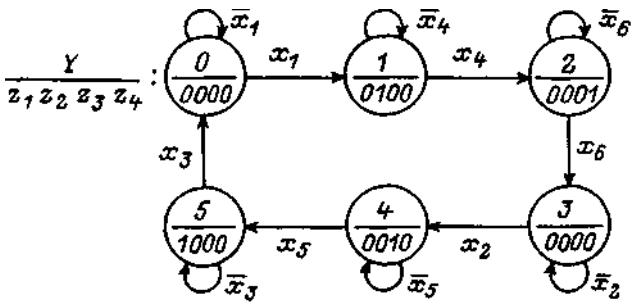

Рис. 5.43

— вида кодирования состояний;
— алгоритмической модели;
— программной модели.

В Приложении 5 для случая реализации алгоритма 1.1 одним автоматом приведено 28 программ, каждая из которых отражает принятые решения по перечисленным выше вопросам. Перечень принятых решений для каждой программы приводится в тексте, изложенном в начале этого приложения. В табл. 5.5 приведены коды состояний, а в ряде случаев и выходов для каждой из программ.

Особый интерес представляет программа *11,* в которой в качестве алгоритмической модели выбрана ГСА.

Эта ГСА была построена по ГП (рис. 5.42) таким образом, чтобы эти модели были изоморфны. Основное требование при построении ГСА — отсутствие возвратов назад в любом месте граф-схемы, за исключением возврата из конца в начало. Эта ГСА весьма сложна, так как содержит шесть условных вершин типах,-, четыре такие же вершины типа Y_j-, девять операторных вершин типа $Y_r = c_j$ и пять таких же вершин типа $z_l = c_h$.

Таблица 5.5

Номер варианта	Номер вершины					
	0	1	2	3	4	5
1	0000	0100	0001	0010	1000	—
2	0	4	1	2	8	—
3	00000	01000	00010	00001	00100	10000
4—7	$\frac{000}{0000}$	$\frac{001}{0100}$	$\frac{010}{0001}$	$\frac{011}{0010}$	$\frac{100}{1000}$	—
8—11	$\frac{Y_0}{0000}$	$\frac{Y_1}{0100}$	$\frac{Y_2}{0001}$	$\frac{Y_3}{0010}$	$\frac{Y_4}{1000}$	—
12—15	$\frac{0}{0000}$	$\frac{1}{0100}$	$\frac{2}{0001}$	$\frac{3}{0010}$	$\frac{4}{1000}$	—
16	$\frac{0}{0000}$	$\frac{1}{-1--}$	$\frac{2}{--0-1}$	$\frac{3}{---1-}$	$\frac{4}{1-0-}$	—
17—20	$\frac{0}{0000}$	$\frac{1}{0100}$	$\frac{2}{0001}$	$\frac{3}{0010}$	$\frac{4}{1000}$	—
21, 22	$\frac{0}{0000}$	$\frac{1}{0100}$	$\frac{2}{0001}$	$\frac{3}{0000}$	$\frac{4}{0010}$	$\frac{5}{1000}$
23, 24	000	001	010	011	100	—
25, 26	Y_0	Y_1	Y_2	Y_3	Y_4	—
27	0	1	2	3	4	—
28	0	1	2	3	4	5

Этот пример демонстрирует недостатки ГСА при их использовании не только в качестве спецификации (как это отмечалось в разд. 2.3), но и в качестве алгоритмической модели, даже в случае когда применяется дешифратор состояний, что связано с его реализацией на условных переходах, а не в виде единой конструкции `switch`.

Для обоснования принципов взаимодействия автоматов рассмотрим реализацию алгоритма 1.1 двумя автоматами. Предположим первоначально, что автоматы связаны между собой лишь по входным переменным (рис. 5.44). На рис. 5.45 приведены ГП, описывающие функционирование каждого автомата. При этом их взаимосвязь обеспечивается указанием одинаковых пометок на дугах различных ГП.

При построении ГП учитывается та особенность программной реализации, что в каждом цикле последовательно для каждого графа реализуется не более одного перехода.

Задать алгоритм управления двумя ГП в некотором смысле проще, чем одним графом. Во многих практически важных случаях описать алгоритм

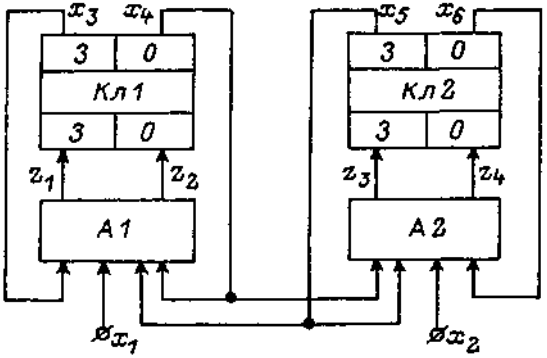

Рис. 5.44

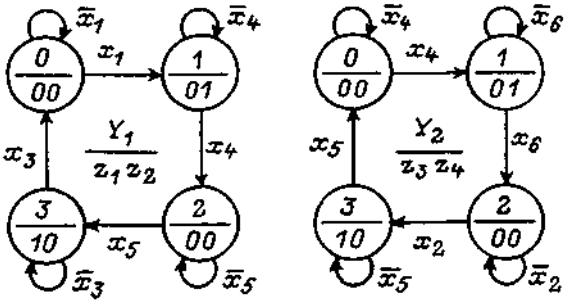

Рис. 5.45

в виде единого ГП оказывается весьма сложно. Однако, когда наступает «момент истины» — проверка правильности построения спецификации (совокупности ГП) или программы, написанной по этой спецификации, совокупность автоматов приходится рассматривать как единое целое.

В этой связи автором предлагается строить граф переходов проверки, в вершинах которого указываются значения состояний и выходов каждого автомата совокупности, которые желательно получить при подаче входных переменных, помечающих соответствующие дуги этого графа. Графы проверки могут быть двух типов, первый из которых отражает каждое изменение состояний каждого автомата, а второй — только их устойчивые состояния. Граф первого типа назовем пошаговым, а второго типа — циклическим. В первом случае программа должна быть построена так, чтобы после «прохождения» каждого ГП его состояние было наблюдаемо, а во втором — оно было наблюдаемо лишь после «прохождения» всех ГП совокупности.

Программа *29* содержит указанные «контрольные точки» для каждого ГП. Используя в этом случае в качестве теста пошаговый граф проверки (рис. 5.46), построим пошаговый граф переходов функционирования (ГПФ) совокупности автоматов. ГПФ, как и графы переходов

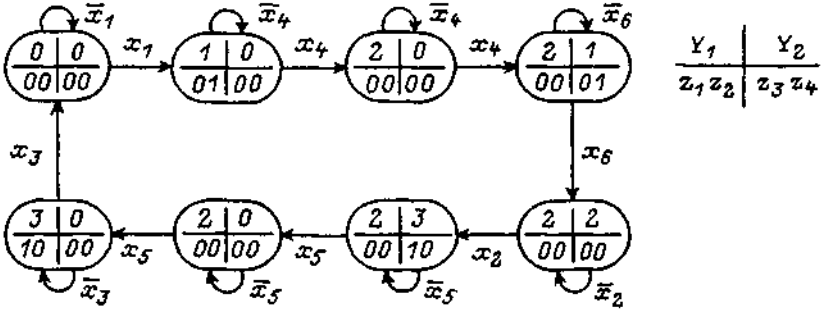

Рис. 5.46

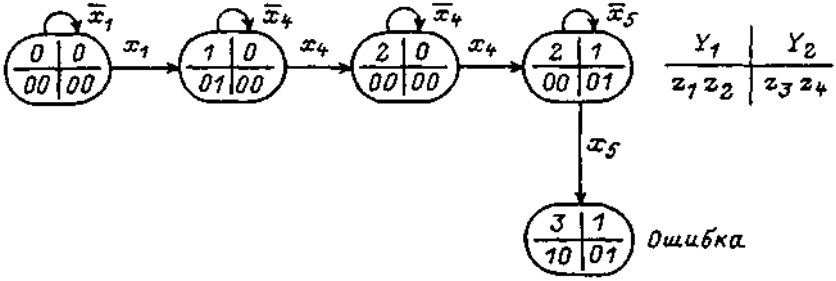

Рис. 5.47

проверки, также могут быть двух типов: с отражением каждого состояния и с указанием только устойчивых состояний. Фрагмент ГП функционирования первого типа приведен на рис. 5.47. Этот фрагмент демонстрирует, что построенная спецификация содержит ошибку, возникшую в связи с тем, что в исходном состоянии $x_5 = 1$, и поэтому при $x_4 = 1$ вместо перехода из состояния «10» (через состояние «20») в состояние «21» совокупность автоматов проходит эти состояния и переходит в состояние «31», которое отсутствует в графе переходов проверки.

Правильное функционирование двух автоматов при использовании только входных переменных обеспечивается при введении в первый автомат дополнительной переменной x_6 и пятого состояния с нулевыми выходами. При этом в ГП для автомата А1 (рис. 5.45) разрывается дуга, помеченная переменной x_5, и в точку разрыва вводится пятая вершина, для которой входная дуга помечается символом x_6, а выходная — x_5. Такое решение нельзя признать приемлемым, так как получающиеся ГП обладают различным числом состояний, в то время как по физической природе оба клапана равноценны.

Для устранения этого недостатка вместо введения в автомат А1 переменной x_6 обеспечим обмен между автоматами по значениям внут-

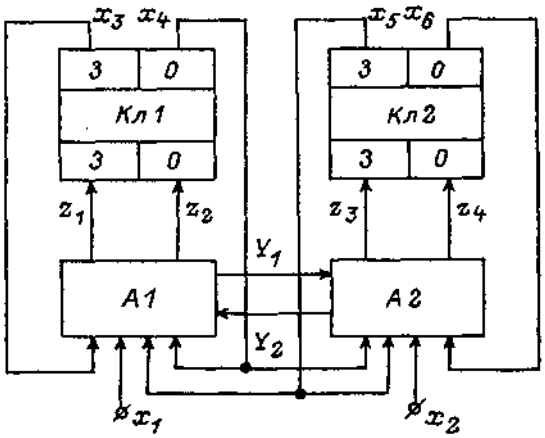

Рис. 5.48

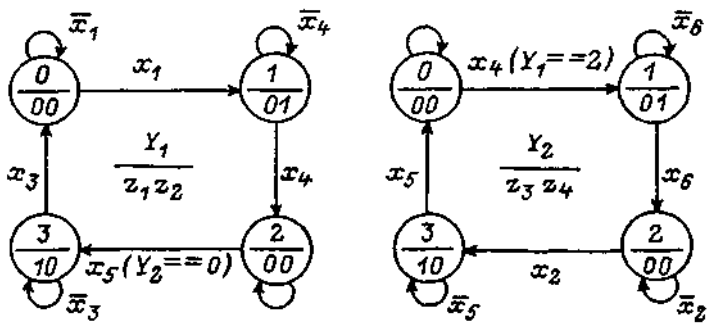

Рис. 5.49

ренних переменных (рис. 5.48). При написании программы (программа *30)* по ГП (рис. 5.49) совокупность автоматов начинает правильно функционировать (рис. 5.50). Действительно, если в этом случае в качестве циклического графа проверки внешнего поведения использовать ГП (рис. 5.43) одного автомата (рис. 5.41), то формируемый при этом циклический граф функционирования (рис. 5.50) рассматриваемой совокупности из двух автоматов (рис. 5.48) эквивалентен по внешнему поведению графу проверки.

Необходимо отметить, что нормальное функционирование будет обеспечено также и в том случае, если связь автоматов по переменной У, будет исключена.

Однако более естественной является ситуация, в которой каждый автомат получает информацию только от «своих» источников, а их взаимосвязь осуществляется только по состояниям (рис. 5.51, 5.52). Пошаговый ГП функционирования для программы *31* приведен на рис. 5.53.

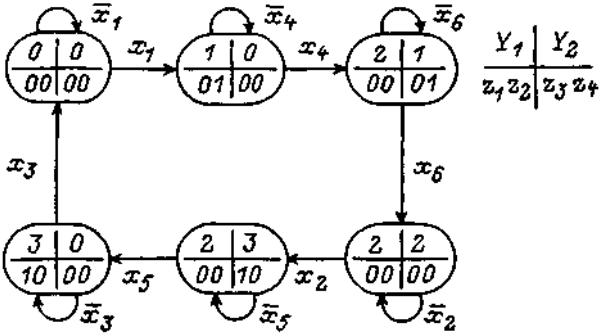

Рис. 5.50

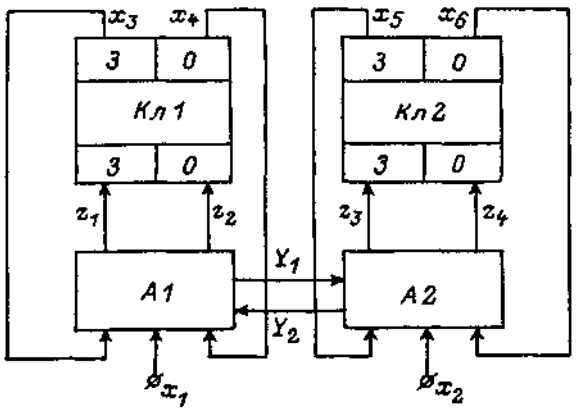

Рис. 5.51

Отметим, что переход от одного автомата к двум связан в этом случае с увеличением суммарного числа состояний в ГП с шести до восьми. Увеличивается также и число состояний в пошаговом ГП функционирования.

Рассмотренная система автоматов (рис. 5.52) обладает тем недостатком, что в ней выходной вектор формируется последовательно по частям: сначала компоненты выходного вектора, реализуемые автоматом А1, а затем — автоматом А2. В общем случае для быстродействующих объектов управления это может быть весьма опасно. Указанный недостаток устраняется при применении двух автоматов без выхода, объединенных выходным преобразователем (комбинационной схемой), формирующим выходной вектор в целом (однотактно) (рис. 5.54). Программа *32* реализует предложенную структуру.

Взаимосвязь автоматов между собой не является единственно возможным принципом их взаимодействия. Более рациональным бывает исполь-

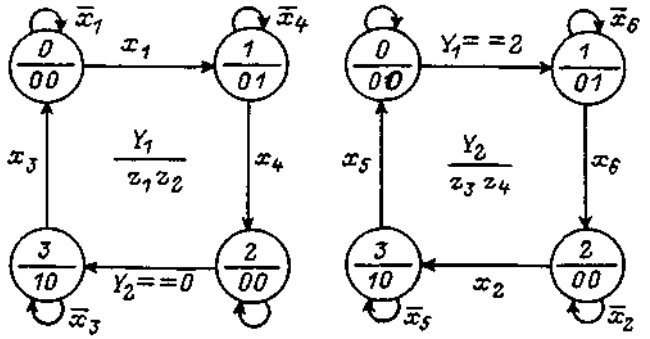

Рис. 5.52

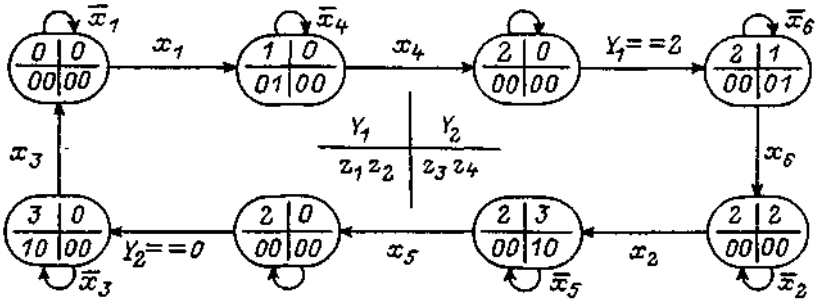

Рис. 5.53

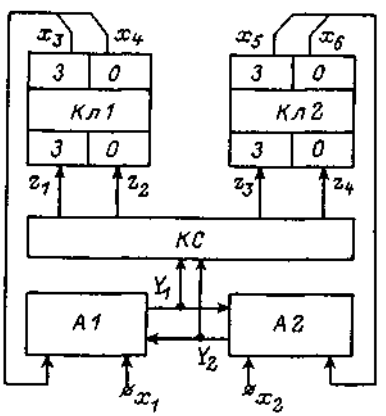

Рис. 5.54

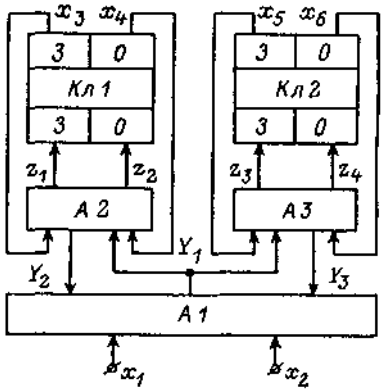

Рис. 5.55

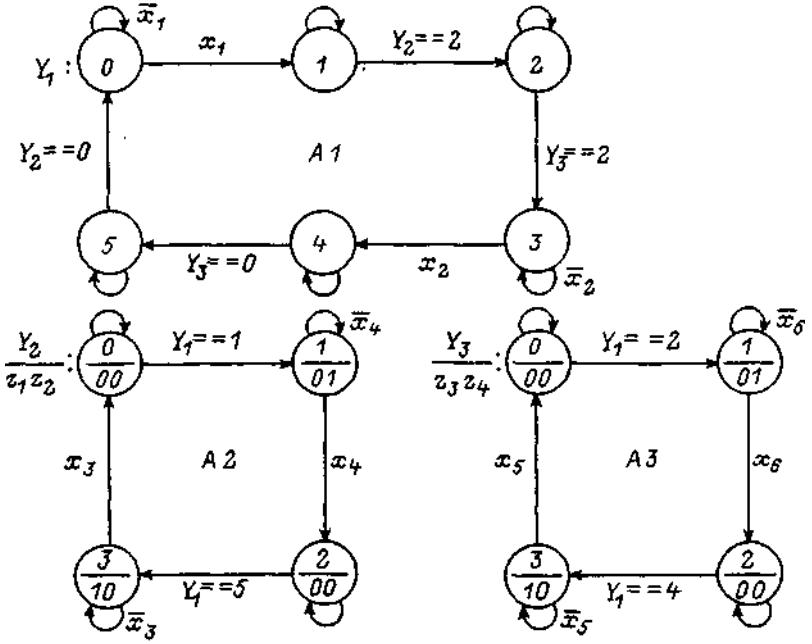

Рис. 5.56

зование структуры, в которой автоматы между собой непосредственно не связаны, а их взаимодействие осуществляется с помощью координирующего автомата (рис. 5.55). (В общем случае возможна многоуровневая иерархия взаимодействующих автоматов). Это позволяет в случае необходимости вводить изменения только в этот автомат, без корректировки структуры остальных автоматов, модифицируя в последних лишь пометки дуг, что далеко не всегда возможно при их непосредственном взаимодействии без координирующего автомата.

Достижение нового качества при реализации алгоритма 1.1 связано с дальнейшим увеличением числа состояний: суммарное число состояний в трех автоматах равно 14 (рис. 5.56). Программа *33* построена по этой спецификации. Используя и в этом случае в качестве циклического графа проверки внешнего поведения ГП (рис. 5.43), получим эквивалентный по внешнему поведению циклический граф функционирования (рис. 5.57).

В последнем примере количество вершин в головном графе для автомата А1 совпадает с их числом в ГП при реализации алгоритма одним автоматом (рис 5.43). При этом можно считать, что каждая из вершин головного графа соответствует определенному состоянию клапанов: «О» — оба закрыты; «1» — открытие первого; «2» — открытие второго; «3» — оба открыты; «4» — закрытие второго; «5» — закрытие первого.

В рассмотренном примере (рис. 5.55, 5.56) осуществляется двусторонний обмен номерами состояний между автоматами. Для рассматриваемой задачи обмен можно сделать односторонним (исключаются связи Y_2 и Y_3),

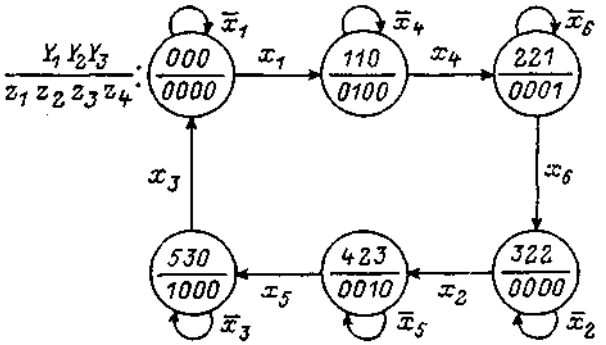

Рис. 5.57

если ввести в автомат А1 информацию от всех сигнализаторов положения $x_1,\ldots, x_6$. При этом в ГП для автомата А1 выполняются следующие замены: Y_2==2 на x_4; Y_3==2 на x_6; Y_3==0 на x_5; Y_2==0 на x_3.

При этом, правда, реализуется алгоритм, несколько отличный от заданного, так как, вместо того чтобы в режиме закрытия сначала снять сигнал на закрытие второго клапана, а потом подать сигнал на закрытие первого клапана, осуществляется формирование этих сигналов в обратном порядке. Указанное изменение для данной задачи с технической точки зрения несущественно.

Из изложенного следует, что исключение двустороннего обмена между автоматами возможно, но не всегда.

В последнем примере декомпозиция алгоритма на три взаимосвязанных автомата выполнена пообъектно. Три взаимосвязанных автомата можно получить также и при декомпозиции автомата по режимам (рис. 8.18). Сравнивая графы, построенные по режимам (рис. 5.43 и 8.18), с графами, спроектированными по объктному принципу (рис. 5.52 и 5.56), можно утверждать, что первые имеют более простое описание и соответственно более простую реализацию, а также легче читаются, а вторые — более унифицированы (допускают использование в разных задачах) и в них проще вносить изменения.

Действительно, если в рассмотренном примере кроме режимного управления необходимо обеспечить также и индивидуальное управление каждым клапаном, то в первом случае требуется перестроить ГП, а во втором — только изменить пометки дуг.

Все рассмотренные программы написаны в предположении, что моделируемая система автоматов является разомкнутой: значения всех входных переменных $x_1 \div x_6$ задаются с помощью клавиатуры ПЭВМ.

Следующим шагом в направлении обеспечения полноты моделирования является переход от проверки функционирования собственно управляющих автоматов к моделированию замкнутых систем «ИИ—А—ФЭЗ—СПИ—ИМ» или «ИИ—СВА—ФЭЗ—СПИ—ИМ». Для этой цели в программы дополнительно вводятся модели объектов управления (клапаны) и средств представления информации (табло).

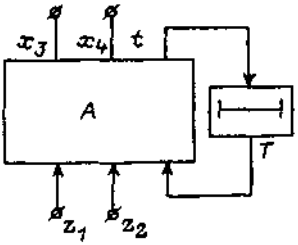

Рис. 5.58

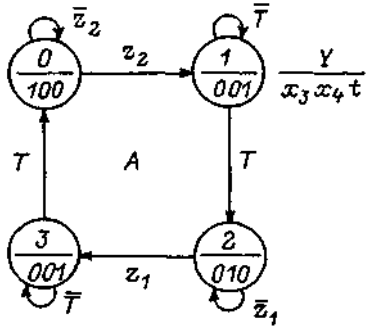

Рис. 5.59

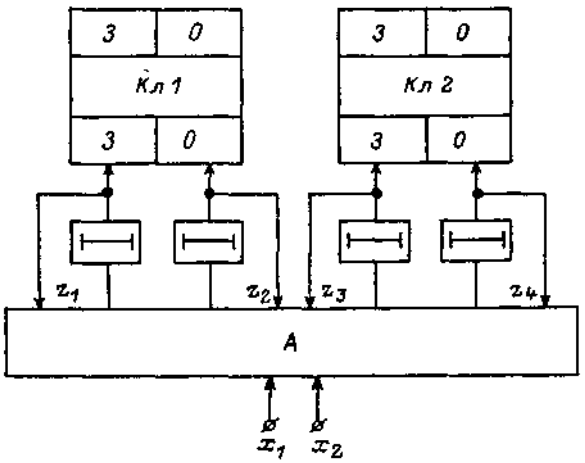

Рис. 5.60

В настоящей работе выбраны простейшие модели клапанов (разд. 5.3). Для этих моделей входными переменными являются переменные z_1, z_2 (для первого клапана) и z_3, z_4 (для второго клапана), а выходными — переменные x_3, x_4 и x_5, x_6 — соответственно. Каждая модель, реализуемая автоматом Мура, имеет 4 состояния: «закрыт» («О»), «открывается» («1»), «открыт» («2»), «закрывается» («3»). Закрытое состояние каждого клапана представляется на соответствующем табло ровным голубым цветом, открытое — ровным зеленым, а переходные (открывается и закрывается) — мигающим желтым цветом.

В моделях времена открытия и закрытия клапанов выбраны одинаковыми и равными 3 с (алгоритм 1.2). Введение в модели времени потребовало разработки механизма реализации ФЭЗ и их взаимодействия с автоматом (разд. 5.2). При этом ФЭЗ для автомата, являющегося основой модели клапана, рассматривается в качестве внешнего устройства (рис. 5.58). ФЭЗ запускается двоичной переменной *t,* формируемой автоматом, и выдает в него двоичную переменную *T* по истечении времени задержки. ГП, реализующий модель клапана, приведен на рис. 5.59.

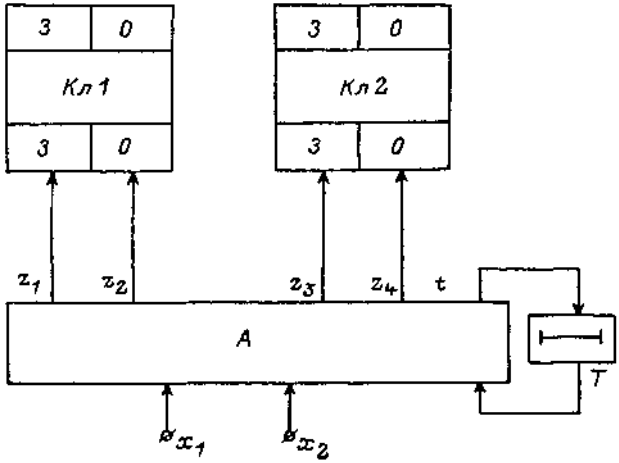

Рис. 5.61

Программа *34* реализует изложенный подход для случая реализации алгоритма 1.2 одним автоматом, а программа *35* — тремя автоматами без выхода (один координирующий — ведущий и два управляющих — ведомые) и выходным преобразователем, однотактно формирующим выходной вектор по сигналам состояния ведомых автоматов.

Программы *36* и *37* демонстрируют еще один подход к построению разомкнутых систем, при котором автомат получает информацию не от сигнализаторов положения по факту их срабатывания, а от ФЭЗ, уставка которых выбрана таким образом, чтобы за это время объект управления заведомо сработал (алгоритм 1.3). Такой подход целесообразен в случае, если сигнализаторы положения недостаточно надежны.

Эти программы интересны также и тем, что в них в отличие от рассмотренных выше программ ФЭЗ входят в состав управляющих автоматов, а не моделей объектов управления. Отметим, что в этих программах по сравнению с предыдущими используется другой тип ФЭЗ, в которых до срабатывания выходной сигнал выдается, а после срабатывания — исчезает. В программе *36* применяются четыре ФЭЗ (рис. 5.60), а в программе *37* — один (рис. 5.61).

Введение ФЭЗ в управляющие автоматы в последних двух программах во многом связано с моделированием работы объектов управления. При наличии сигнализаторов положения, ФЭЗ в составе указанных автоматов для этой цели могут не использоваться.

Больший интерес представляют реализации алгоритмов, которые вне зависимости от свойств объектов управления принципиально являются временными, т. е. без применения ФЭЗ не могут быть выполнены (алгоритм 2).

На рис. 5.62 представлена схема связей моделей клапанов УА1 и УА2 с моделью управляющего устройства УА3. Каждая из этих моделей реализована управляющим автоматом, состоящим из автомата и ФЭЗ. Граф переходов автомата А3, входящего в состав управляющего устрой-

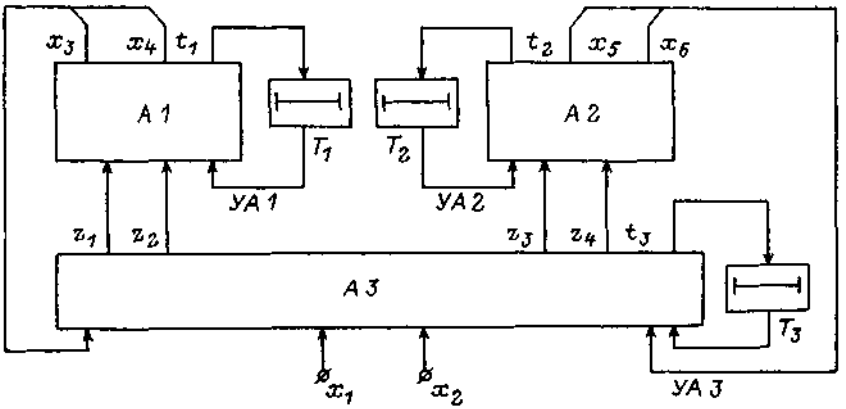

Рис. 5.62

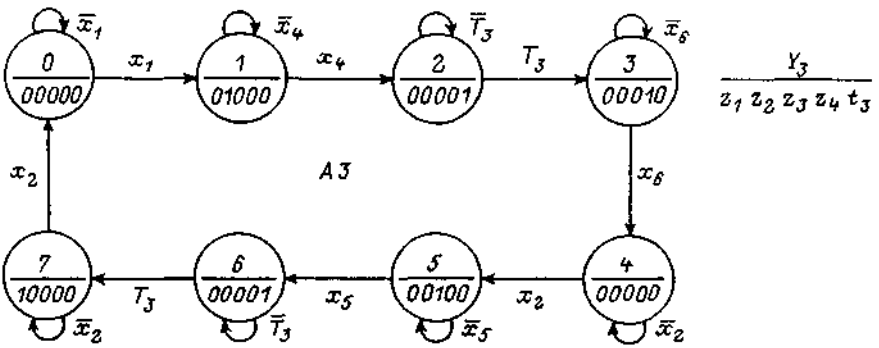

Рис. 5.63

ства, приведен на рис. 5.63. Из рассмотрения этого ГП следует, что усложнение реализуемого алгоритма привело к увеличению числа состояний соответствующего автомата до восьми. Программа *38* обеспечивает моделирование рассматриваемой системы.

Наибольший интерес представляют алгоритмы управления, в которых требуется параллельная работа ФЭЗ (алгоритм 3). Этот тип алгоритмов является одним из наиболее сложных для задач логического управления при однопроцессорной реализации.

На рис. 5.64 представлена схема связи моделей клапанов УА1 и УА2 с моделью управляющего устройства УАЗ для этого случая. Граф переходов автомата А3, входящего в состав управляющего устройства, приведен на рис. 5.65. Усложнение алгоритма привело к увеличению числа состояний реализующего его автомата до двенадцати. Программа *39* обеспечивает моделирование рассмотренной системы.

Наиболее сложной из всех рассмотренных является программа *40,* реализующая композицию из пяти управляющих автоматов (рис. 5.66), два из которых соответствуют моделям клапанов (УА1, УА2), а три —

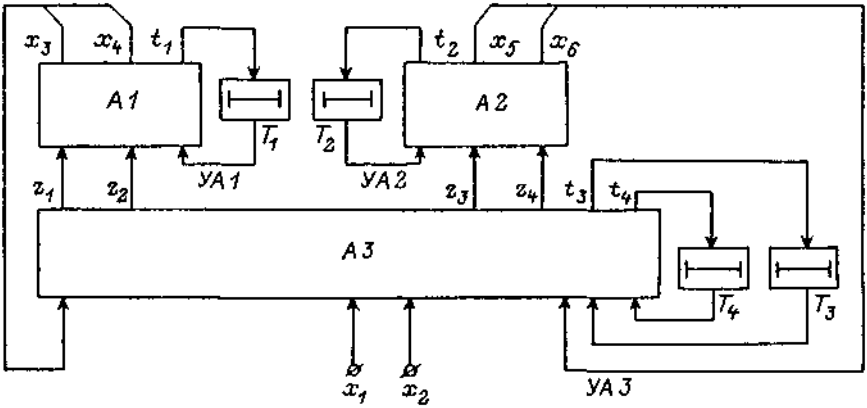

Рис. 5.64

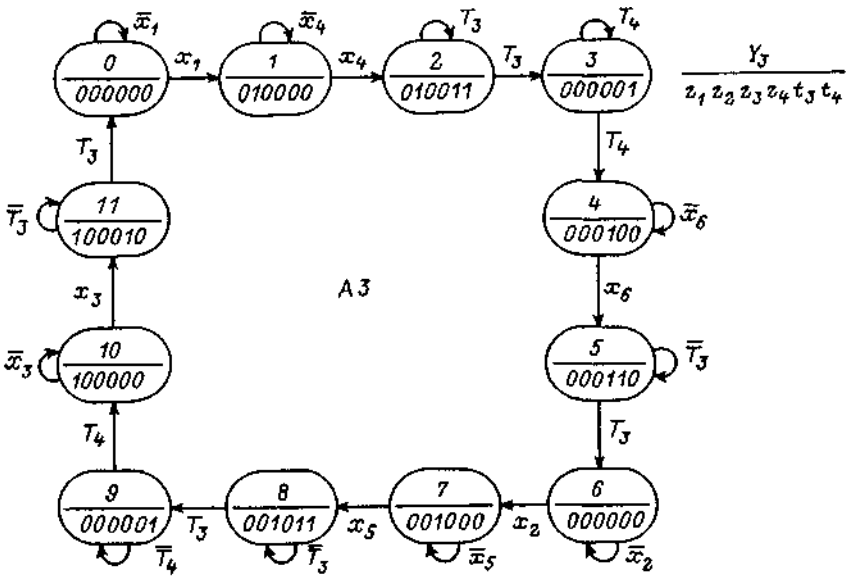

Рис. 5.65

составляющим управляющего устройства (УА3, УА4, УА5). Каждый из управляющих автоматов состоит из автомата и ФЭЗ. ГП этих автоматов приведены на рис. 5.67. При этом необходимо отметить, что суммарное число состояний в этих автоматах, реализующих управляющее устройство, возросло до 18. Табл. 5.6 иллюстрирует пошаговый процесс моделирования работы системы. Рассмотренный пример построения двухуровнего управляющего «устройства» позволяет аналогично реализовывать такие «устройства» с произвольным числом уровней.

Отметим, что в схеме на рис. 5.66 сигналы x_4 и x_5 из автомата А3 могут быть исключены, если ввести связь по переменным состояний автоматов А4 и А5 с указанным автоматом. При этом в ГП для автомата

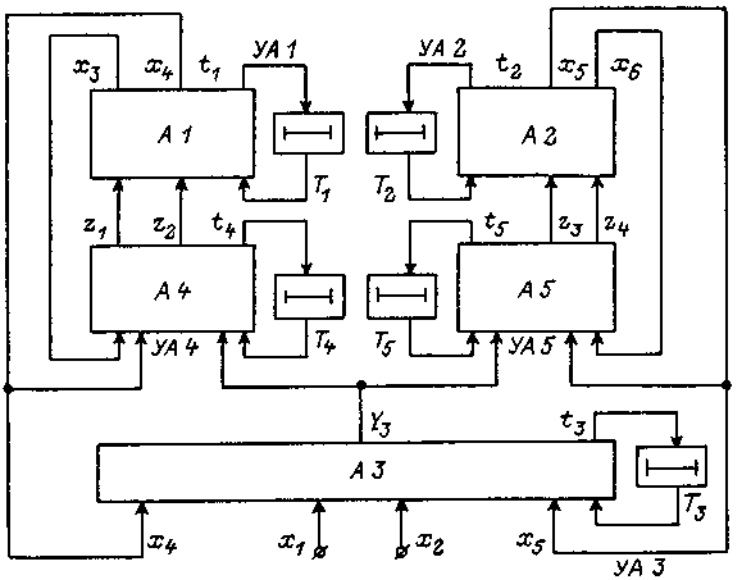

Рис. 5.66

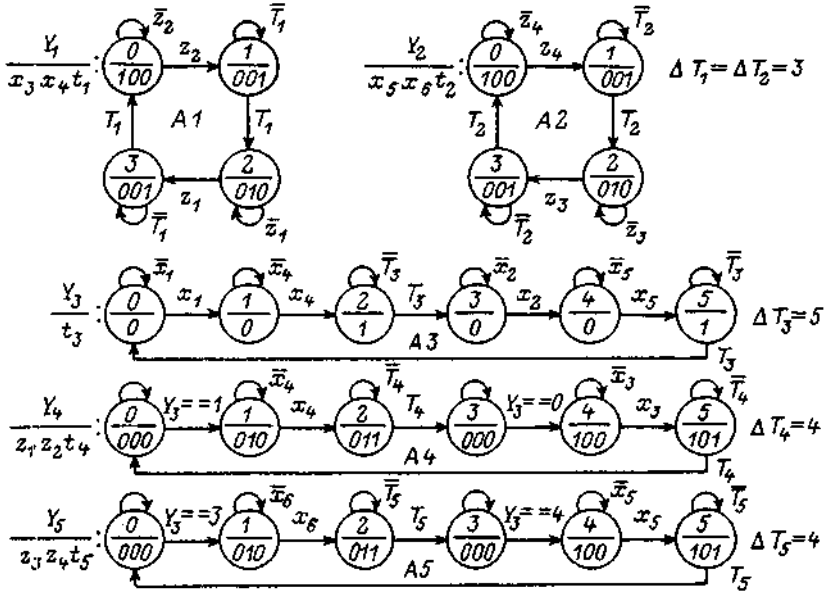

Рис. 5.67

Таблица 5.6

Вход	Номер шага	t_1	t_2	t_3	t_4	t_5	A_1	A_2	A_3	A_4	A_5	z_1	z_2	z_3	z_4
$x_1 = 1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0
	2	0*	0	0	0	0	1	0	1	1	0	0	1	0	0
	3	1*	0	0	0	0	1	0	1	1	0	0	1	0	0
	4	2*	0	0	0	0	1	0	1	1	0	0	1	0	0
	5	0	0	0*	0*	0	2	0	2	2	0	0	1	0	0
	6	0	0	1*	1*	0	2	0	2	2	0	0	1	0	0
	7	0	0	2*	2*	0	2	0	2	2	0	0	1	0	0
	8	0	0	3*	3*	0	2	0	2	2	0	0	1	0	0
	9	0	0	4*	0	0	2	0	2	3	0	0	0	0	0
	10	0	0	0	0	0	2	0	3	3	1	0	0	0	1
	11	0	0*	0	0	0	2	1	3	3	1	0	0	0	1
	12	0	1*	0	0	0	2	1	3	3	1	0	0	0	1
	13	0	2*	0	0	0	2	1	3	3	1	0	0	0	1
	14	0	0	0	0	0*	2	2	3	3	2	0	0	0	1
	15	0	0	0	0	1*	2	2	3	3	2	0	0	0	1
	16	0	0	0	0	2*	2	2	3	3	2	0	0	0	1
	17	0	0	0	0	3*	2	2	3	3	2	0	0	0	1
	18	0	0	0	0	0	2	2	3	3	3	0	0	0	0
$x_2 = 1$	19	0	0	0	0	0	2	2	4	3	4	0	0	1	0
	20	0	0*	0	0	0	2	3	4	3	4	0	0	1	0
	21	0	1*	0	0	0	2	3	4	3	4	0	0	1	0
	22	0	2*	0	0	0	2	3	4	3	4	0	0	1	0
	23	0	0	0*	0	0*	2	0	5	3	5	0	0	1	0
	24	0	0	1*	0	1*	2	0	5	3	5	0	0	1	0
	25	0	0	2*	0	2*	2	0	5	3	5	0	0	1	0
	26	0	0	3*	0	3*	2	0	5	3	5	0	0	1	0
	27	0	0	4*	0	0	2	0	5	3	0	0	0	0	0
	28	0	0	0	0	0	2	0	0	4	0	1	0	0	0
	29	0*	0	0	0	0	3	0	0	4	0	1	0	0	0
	30	1*	0	0	0	0	3	0	0	4	0	1	0	0	0
	31	2*	0	0	0	0	3	0	0	4	0	1	0	0	0

Таблица 5.6 *(продолжение)*

Вход	Номер шага	t_1	t_2	t_3	t_4	t_5	A_1	A_2	A_3	A_4	A_5	z_1	z_2	z_3	z_4
$x_2 = 1$	32	0	0	0	0*	0	0	0	0	5	0	1	0	0	0
	33	0	0	0	1*	0	0	0	0	5	0	1	0	0	0
	34	0	0	0	2*	0	0	0	0	5	0	1	0	0	0
	35	0	0	0	3*	0	0	0	0	5	0	i	0	0	0
	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0

А3 на рис. 5.67 переменная x_4 должна быть заменена выражением Y_4==2, а переменная x_5 — выражением Y_5 ==5.

Отметим, что в первых двадцати восьми программах в Приложении 5 мы сталкиваемся с редкой ситуацией, когда все они корректно реализуют один и тот же алгоритм. Это позволяет проводить их достаточно объективное сравнение по разным критериям, так как обычно такое многообразие программ для одного и того же алгоритма не рассматривается. Поэтому традиционный довод о том, что программа нормально работает, ничего доказать не может, так как в данном случае все 28 программ работают правильно. Наилучшей из рассмотренных программ, по-видимому, является программа *22,* так как, по мнению автора, важнейшими критериями качества программ для систем логического управления являются такие неформализуемые понятия, как наглядность и читаемость, а все остальные критерии (объем памяти, быстродействие и т. д.) должны использоваться только в тех случаях, когда соответствующие ограничения не удается выполнить.

Сокращение сложности программы и повышение ее быстродействия обеспечиваются при реализации рассматриваемого алгоритма с помощью обобщенной программной модели автомата Мура второго рода, состоящей из одной конструкции `switch`.

В рассмотренных примерах исследовалась однопроцессорная реализация, однако предложенный подход может быть использован и для случая, когда каждый управляющий автомат, входящий в состав управляющего устройства, реализуется своим процессором.

Из изложенного следует, что основное достоинство предлагаемого подхода состоит в том, что проектирование сложной программы производится не в программистских терминах, а в терминах взаимодействия автоматов, что, по мнению автора, более приемлемо для Разработчиков и Заказчиков систем логического управления.

В заключение раздела отметим, что под руководством автора разработана «оболочка» для отладки программ, которая позволяет выбрать число входных, временных и выходных переменных, а также число автоматов, используемых в программе.

На экран дисплея с помощью клавиатуры можно ввести значения входных переменных и вывести вычисленные значения выходных переменных. На экране также могут быть сформированы табло для представ-

ления состояний клапанов и окна для указания значений выдержек времени.

Главная особенность предлагаемой «оболочки» состоит в том, что на экран автоматически выводятся также значения состояний каждого автомата, используемого в программе, применяя для этого только одну внутреннюю многозначную переменную. Это обеспечивает наблюдаемость процесса выполнения программы, превращая ее в «белый ящик» [250] с минимальным числом внутренних переменных. При применении конструкции switch программа является управляемой, так как структура каждого ее функционального фрагмента изоморфна соответствующему ГП, который весьма просто корректируется.

«Оболочка» наряду с циклическим и пошаговым (принудительным) режимами выполнения программы обеспечивает также возможность реализации автоматического режима.

В циклическом режиме по сигналу запуска осуществляется один проход программы в целом. При пошаговом режиме по тому же сигналу реализуется фрагмент программы, соответствующий одному переходу одного автомата. В автоматическом режиме пусковой сигнал инициирует реализацию переходного процесса (режима), завершающегося устойчивым состоянием, для выхода из которого в данный момент времени отсутствуют соответствующие инициирующие значения входных переменных. В каждом из режимов изменения значений входных переменных осуществляются до подачи сигнала запуска.

Кроме указанной «оболочки», построенной, в частности, для визуализации программ *1—40* (Приложение 5), под руководством автора разработана еще одна «оболочка» (прикладной генератор), позволяющая на экране дисплея строить графы переходов автоматов Мура и автоматически транслировать их в программы на языке СИ. После трансляции появляется возможность моделирования поведения системы в форме изменения активности вершин каждого графа при смене значений входных переменных.

Программы, приведенные в Приложении 5, не содержат головной программы, которая, в частности, формирует массив значений входных переменных, получаемый в результате циклического опроса клавиш клавиатуры. Обработка этих значений каждой из программ выполняется после нажатия клавиши «Enter».

В Приложении 6 приведены две программы, реализующие ГП автомата Мура с приоритетами, который эквивалентен ГП автомата без выходного преобразователя (рис. 5.38). В первой из них значения входных переменных формируются в результате циклического опроса клавиш, а во второй — по прерываниям, возникающим при нажатии клавиш.

Особенность реализуемого ГП состоит в том, что его дуги помечены лишь одиночными входными переменными. Это позволяет отказаться в приведенных программах от применения клавиши «Enter» для перехода к обработке сформированных значений входных переменных. При пометках дуг более сложными логическими выражениями и отказе от применения клавиши «Enter» программа должна содержать существенно более сложный «обработчик» прерываний клавиатуры.

Все программы, приведенные в Приложении 5, реализованы на языке СИ. В этих программах для упрощения их понимания и проверки, даже для реализации однотипных фрагментов алгоритмов, функции (подпрограммы, процедуры) не применялись.

Так, например, два однотипных, но различных графа переходов автоматов Мура (рис. 5.52), описывающих алгоритм управления двумя клапанами, реализованы программой *31,* содержащей четыре конструкции `switch`, в каждой паре которых первая конструкция реализует функцию переходов автомата, а вторая — функции его выходов. Отметим, что каждый из этих автоматов может быть реализован также и одной конструкцией `switch`, совместно реализующей функцию переходов автомата и его функции выходов.

Анализ программ, построенных с помощью конструкции `switch` без использования функций, целесообразно начинать с проверки изоморфизма между каждым графом переходов и соответствующим фрагментом текста программы (двух или одной конструкции `switch`). Такую проверку можно назвать проверкой программы в статике. Если при этом в программе ошибки не обнаружены или устранены, то можно переходить к проверке ее в динамике. При этом в качестве теста может применяться пошаговый граф проверки (рис. 5.53) с целью построения эквивалентного ему графа функционирования.

Использование механизма функций позволяет для рассматриваемого примера уменьшить число конструкций `switch` до одной (Приложение 7). При этом длина функциональной части программы сокращается, а «похожесть» графов переходов и текста программы снижается. В результате статическая проверка программы усложняется, особенно учитывая тот факт, что для сокращения текста программы применяется передача параметров по ссылкам. Отсутствие изоморфизма в этом случае может привести к появлению семантических ошибок. Обнаружение этих ошибок и проверка программы проводятся тестированием, как изложено выше.

В Приложении 8 приведены четыре объектно-ориентированные программы, написанные на языке СИ++. Первая и вторая программы реализуют алгоритм, заданный с помощью двух однотипных графов переходов (рис. 5.52), а третья и четвертая программы реализуют тот же алгоритм с помощью трех графов переходов, из которых два однотипны (рис. 5.56).

В первой программе каждому автомату сопоставлен свой класс. Во второй программе вводится один класс, который содержит два экземпляра «объектов» (автоматов). В третьей программе используются два класса, первый из которых содержит один экземпляр «объектов», а второй — два экземпляра «объектов». За счет усложнения в четвертой программе применяется один класс, содержащий три экземпляра «объектов», один из которых существенно отличается от двух других (имеет большее число состояний), в то время как они между собой отличаются менее значительно (имеют только различную пометку дуг).

Сопоставление этих программ с программами, написанными без использования объектно-ориентированного подхода (рассмотренные выше), для одних и тех же примеров оставляет открытым вопрос о целесообраз-

ности применения объектно-ориентированного программирования для рассматриваемого класса задач реального времени.

В Приложении 9 приведены две программы, написанные на языке программирования «Форт», базирующемся на использовании обратной польской записи [122]. Первая из программ реализует один ГП автомата Мура, который эквивалентен ГП, приведенному на рис. 5.38, а вторая — систему из двух взаимосвязанных ГП (рис. 5.52). При построении этих программ применяется конструкция, аналогичная конструкции `switch` языка СИ.

Программы, приведенные в Приложениях 6—9, написаны С. Б. Терентьевым (НПО «Аврора») совместно с автором.

5.4.3. Реализация алгоритма управления системой воздуха среднего давления

Описание объекта управления. Система воздуха среднего давления состоит из клапана (Кл), компрессора (КОМ) и сигнализатора давления воздуха в трубопроводе (Д), обозначаемого переменной x_4. Исполнительные механизмы клапана z_1, и компрессора z_2 являются одновходовыми и памятью не обладают. Сигнализатор положения x_5 срабатывает, когда клапан открыт, а блок-контакт x_6 — в случае, если подан сигнал на включение компрессора.

Органы управления и средства представления информации. На пульте управления должны быть размещены три кнопки без памяти: ПУСК (П) — x_1, СТОП (С) — x_2, РАЗБЛОКИРОВКА (Р) — x_3, и два табло: НОРМАЛЬНАЯ РАБОТА (НР) — z_3, НЕИСПРАВНОСТЬ (НС) — z_4.

Схема связей «ИИ—У А—СПИ—ИМ» приведена на рис. 5.68.

Словесное описание алгоритма управления. При нажатии кнопки ПУСК открывается клапан. После его открытия включается компрессор. Через 5 с после пуска компрессора клапан закрывается.

Если через 20 с после пуска компрессора давление в трубопроводе достигнет или превысит 10 кг /см^2, то засвечивается табло НОРМАЛЬНАЯ РАБОТА. В противном случае открывается клапан, а затем засвечивается табло НЕИСПРАВНОСТЬ.

Через 10 с после открытия клапана компрессор останавливается, а клапан закрывается.

Последующий запуск системы возможен после снятия сигнала НЕИСПРАВНОСТЬ путем нажатия кнопки РАЗБЛОКИРОВКА.

При нажатии кнопки СТОП, когда компрессор работает в установившемся режиме, открывается клапан. Через 10 с после открытия клапана компрессор останавливается, а клапан закрывается.

Система готова к дальнейшей работе [207].

Схема связей «ИИ—А—ФЭЗ—СПИ—ИМ» приведена на рис. 5.69.

Построение графа переходов автомата. По словесному заданию строится ГП (рис. 5.70). Построенный ГП содержит 13 вершин, в которых формируются только 10 различных комбинаций значений выходных переменных. В восьми вершинах формируются уникальные комбинации, а соответственно в двух и трех вершинах — одинаковые. Различие вершин с одинаковыми комбинациями выходных воздействий осуществляется не

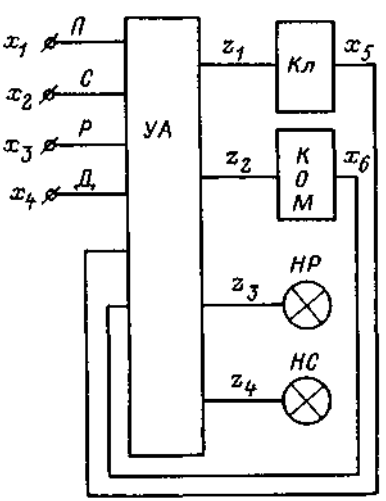

Рис. 5.68

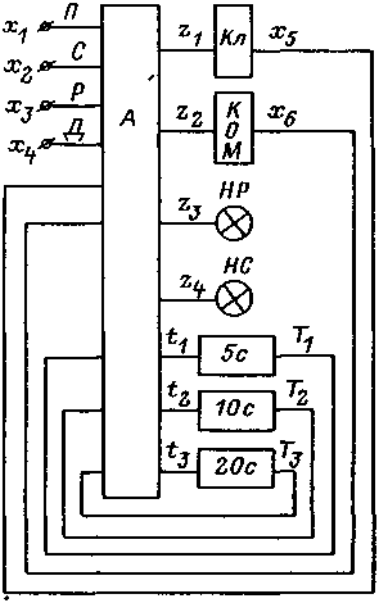

Рис. 5.69

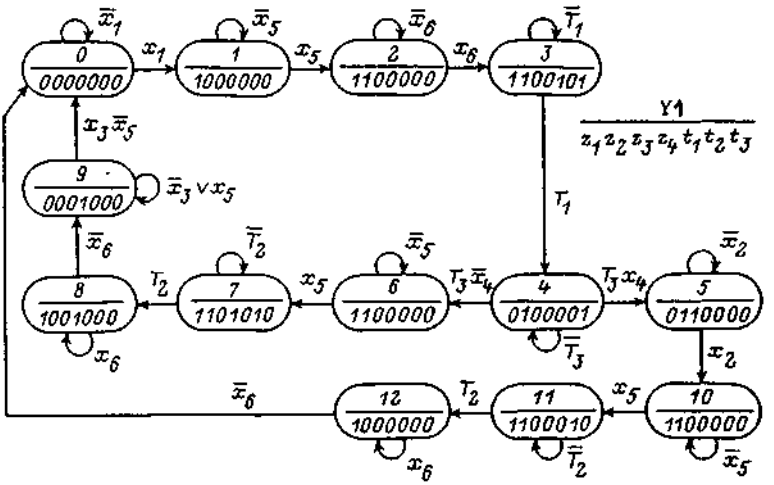

Рис. 5.70

за счет введения в явном виде дополнительных (промежуточных) переменных, а косвенно — за счет различной нумерации вершин: 1, 12 и 2, 6, 10 соответственно. При программировании номера вершин в десятичном коде записываются в метках `case` конструкции `switch`.

Реализация графа переходов. Построенный ГП «вкладывается» в одну конструкцию `switch`, включающую в свой состав обращения к процедуре, реализующей функциональные элементы задержки (разд. 5.2.3). Получающаяся программа структурирована, легко читаема, наблюдаема и управляема:

```
        switch (Y1)  {
case  0: if(x1)      {Y1=1;  z1=1; z2=0; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case  1: if(x5)      {Y1=2;  z1=1; z2=1; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case  2: if(x6)      {Y1=3;  z1=1; z2=1; z3=0; z4=0;
                      dly( 1, -2,  3, end); } break;
case  3: if(d[1])    {Y1=4;  z1=0; z2=1; z3=0; z4=0;
                      dly(-1, -2,  3, end); } break;
case  4: if(x4&d[3]) {Y1=5;  z1=0; z2=1; z3=1; z4=0;
                      dly(-1, -2, -3, end); }
         if(x̄4&d[3]) {Y1=6;  z1=1; z2=1; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case  5: if(x2)      {Y1=10; z1=1; z2=1; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case  6: if(x5)      {Y1=7;  z1=1; z2=1; z3=0; z4=1;
                      dly(-1,  2, -3, end); } break;
case  7: if(d[2])    {Y1=8;  z1=1; z2=0; z3=0; z4=1;
                      dly(-1, -2, -3, end); } break;
case  8: if(x̄6)      {Y1=9;  z1=0; z2=0; z3=0; z4=1;
                      dly(-1, -2, -3, end); } break;
case  9: if(x3&x̄5)   {Y1=0;  z1=0; z2=0; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case 10: if(x5)      {Y1=11; z1=1; z2=1; z3=0; z4=0;
                      dly(-1,  2, -3, end); } break;
case 11: if(d[2])    {Y1=12; z1=1; z2=0; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;
case 12: if(x̄6)      {Y1=0;  z1=0; z2=0; z3=0; z4=0;
                      dly(-1, -2, -3, end); } break;}.
```

5.4.4. Реализация логико-вычислительных алгоритмов

Подход, разработанный в настоящей работе для реализации алгоритмов логического управления, может быть использован также и для реализации логико-вычислительных алгоритмов, таких, например, как алгоритм синхронизации судового генератора с шинами главного распределительного щита (Приложение 10). Эта программа разработана А. В. Никитиным и В. В. Галаном совместно с автором.

При ее создании применялась классическая идея о том, что любое вычислительное устройство может быть построено из операционного устройства и устройства управления [7]. Такой же подход может быть

использован и при написании программ, выделяя отдельно устройство управления и вычислительные блоки, рассматривая последние в качестве операционного устройства.

Вычислительные блоки вырабатывают двоичные сигналы, поступающие в управляющее устройство, которое в свою очередь, получая сигналы также и от других источников информации, выдает сигналы для обращения к указанным блокам. Это позволяет весьма автономно разрабатывать вычислительные блоки и устройство управления, применяя подход, излагаемый в настоящей работе.

Другой подход к разработке алгоритмов рассматриваемого класса состоит в построении ГП, в которых в качестве выходных переменных используются вычислительные процессы. Процессы, соответствующие переменным, размещенным в вершинах графа, реализуются в течение всего времени пребывания автомата в вершине, в общем случае — многократно (разд. 12.4).

5.5. Сертификация программ, реализующих одиночные автоматы

Как следует из изложенного в предыдущих разделах, для автоматов с памятью соотношения «вход—выход» не позволяют однозначно описать их поведение. Так, например, в табл. 5.7, соответствующей Л-триггеру, в первой строке имеет место неоднозначность выходных реакций на одно и то же входное воздействие.

Таблица 5.7

R	S	Y
0	0	0, 1
0	1	1
1	0	0
1	1	0

Этого недостатка лишена кодированная таблица переходов (табл. 4.1), которая однозначно описывает соотношения «выход (состояние), вход—следующий выход (следующее состояние)». По этой таблице однозначно строится ГП (рис. 4.15), который, во-первых, является более компактной формой представления кодированной таблицы переходов, а во-вторых, отражает поведение R-триггера.

Графу переходов рассмотренного автомата без выходного преобразователя однозначно соответствует интервальная таблица (табл. 5.8). Число строк в этой таблице равно числу конъюнкций в булевых формулах (представленных в дизъюнктивной нормальной форме), помечающих все ортогонализованные дуги (включая петли) ГП.

Таблица 5.8

Y	R	S	Y'
0	1	—	0
	—	0	0
	0	1	1
1	1	—	0
	0	—	1

Эта таблица содержит всего пять вместо восьми строк в кодированной таблице переходов и может использоваться для сертификации программ, которые построены по ГП формально и изоморфно.

Действительно, если по ГП (рис. 4.15) формально построить изоморфную программу:

```
        switch (Y) {
case 0: if (R̄&S)   Y=1;
        break;
case 1: if (R)     Y=0;
        break; },
```

то, поэлементно сопоставив ее с ГП, можно быть уверенным в правильности преобразования. При этом табл. 5.8 может применяться для сертификации программы, т. е. демонстрации того, что поведение программы соответствует графу переходов. Поэтому таблицы указанного типа, построенные по ГП, могут быть названы сертификационными.

Сертификация отличается от тестирования и верификации. При сертификации демонстрируется соответствие программы спецификации (графу переходов), а не проверяется или доказывается правильность программы. При этом если ГП не корректен, то некорректной (даже при формальном и изоморфном переходе) является и программа, что при сертификации может и не обнаруживаться. Поэтому, по мнению автора, основные усилия для построения «правильной» программы должны быть направлены на построение «правильного» ГП, который в свою очередь должен преобразовываться в программу формально и изоморфно.

Для обеспечения «правильности» ГП должен быть построен так, чтобы он был понятен максимальному числу заинтересованных Специалистов, которые его согласуют.

С увеличением размерности эффективность применения интервальных таблиц для сертификации может возрастать. Так, табл. 5.9, построенная по ГП автомата Мура (рис. 7.4), содержит лишь 7 строк, в то время как соответствующая кодированная таблица переходов состоит из 48 строк.

Упрощение приведенной таблицы по сравнению с КТП связано с тем, что в ГП каждый переход существенно зависит не от всех входных

Таблица 5.9

Y	z_1	z_2	x_1	x_2	x_3	x_4	Y'
0	0	0	0	0	—	—	0
			1	0	—	—	1
			—	1	—	—	2
1	0	1	—	—	—	1	0
			—	—	—	0	1
2	1	0	—	—	1	—	0
			—	—	0	—	2

переменных, а только от некоторых из них, так как от всех остальных переменных переход зависит несущественно (не зависит).

Если каждый переход зависит от всех входных переменных существенно, как. например это имеет место для счетного триггера (рис. 4.124), то сертификационная таблица не является интервальной и совпадает с КТП. По сертификационной таблице автомата Мура сначала в статике демонстрируется соответствие каждого кортежа значений выходов соответствующему состоянию, которое должно быть наблюдаемым, а затем в динамике проверяется правильность выполнения переходов. При этом предполагается, что при подаче каждого сертификационного входного набора реализуется только один переход ГП.

Для автоматов Мили сертификационная таблица имеет другую структуру. Табл. 5.10 построена по ГП автомата Мили (рис. 4.127).

Таблица 5.10

Y	x_1	x_2	x_3	z	Y'
0	1	1	—	1	0
	1	—	0	1	0
	0	1	—	0	0
	0	—	0	0	0
	—	0	1	0	1
1	0	1	—	0	0
	1	1	—	1	0
	—	0	—	1	1

Данная таблица содержит в два раза меньше строк по сравнению с соответствующей кодированной таблицей переходов. Однако сертификационная таблица для автоматов Мили, так же как было показано для автоматов Мура, не всегда содержит меньшее число строк по сравнению с КТП. Так, например, для последовательного сумматора в ГП автомата Мили (рис. 4.113) каждый переход зависит от всех входных переменных, что не позволяет сократить сертификационную таблицу по сравнению с КТП.

Для С-автоматов сертификационные таблицы имеют более сложную структуру. Табл. 5.11 построена по ГП смешанного автомата с приорите-

Таблица 5.11

Y	Выходы Мура				Входы								Выходы Мили		Y'
	z_1	z_2	t_1	t_2	x_1	x_2	x_3	x_4	x_5	x_6	T_1	T_2	t_1	t_2	
0	0	0	0	0	0	0	0	0	—	—	—	—	—	—	0
					1	0	—	0	—	—	—	—	—	—	1
					—	0	1	0	—	—	—	—	—	—	1
					—	1	—	—	—	—	—	—	—	—	2
					—	—	—	1	—	—	—	—	—	—	2
1	0	1	1	1	—	—	—	—	1	—	1	—	—	—	0
					—	—	—	—	—	0	0	—	—	—	1
					—	—	—	—	0	0	—	—	—	—	1
					—	—	—	—	—	1	0	—	0	—	2
					—	—	—	—	0	1	—	—	0	—	2
2	1	—	1	*	—	—	—	—	1	—	—	1	—	—	0
					—	—	1	—	—	—	1	0	0	0	1
					—	—	1	—	0	—	1	—	0	0	1
					—	—	—	—	—	—	0	0	—	—	2
					—	—	—	—	0	—	0	—	—	—	2
					—	—	0	—	—	—	—	0	—	—	2
					—	—	0	—	0	—	—	—	—	—	2

тами (рис. 14.23). При этом рассматриваемый в качестве примера ГП предварительно преобразуется следующим образом:

— в соответствии с приоритетами выполняется ортогонализация всех дуг, исходящих из каждой вершины;

— вводится пометка петель;

— булевы формулы в скобочных формах, помечающие дуги, преобразуются в ДНФ.

В отличие от сертификационных таблиц, рассмотренных выше, в данном случае для «выходов Мура» используется символ «звездочка», который означает сохранение приобретенного ранее значения, а для «выходов Мили» указываются только те значения, которые формируются на соответствующем переходе.

В предлагаемой технологии сертификационные таблицы могут применяться в качестве основы методики проверки функционирования.

5.6. Исследование функциональных возможностей автоматов и систем автоматов

В разд. 5.4.4 для случая реализации заданного алгоритма системой взаимосвязанных автоматов строятся:

— спецификация в виде ГП или СВГП;

— программа, формально и изоморфно реализующая ГП или СВГП;

— граф переходов проверки, отражающий желаемое поведение автомата или системы взаимосвязанных автоматов;

— граф переходов функционирования как результат применения графа переходов проверки в качестве сертификационного теста для построенной программы.

При этом считалось, что если граф переходов функционирования может быть сведен (исключением некоторых вершин, а возможно, и переобозначением номеров оставшихся вершин) к графу переходов проверки, то программа корректно реализует заданный алгоритм. В противном случае спецификация и (или) программа должны корректироваться.

Такой подход не позволяет, во-первых, определить все функциональные возможности программы (так как не исключено, что она может делать еще что-то кроме того, что проверено с помощью графа переходов проверки), а во-вторых, отделить ошибки спецификации от ошибок программирования.

Эти недостатки могут быть устранены, если после разработки спецификации построить по ней граф достижимых маркировок (ГДМ), вершины которого помечены по крайней мере кортежами, образованными номерами вершин, в которых одновременно могут находиться все ГП системы.

Вершины ГДМ могут быть помечены в том числе и одинаковыми кортежами этого типа, что может привести в дальнейшем к необходимости перекодирования номеров этих вершин. Число вершин в ГДМ определяет число состояний в СВГП.

ГДМ должен строиться даже для одного ГП, если он содержит флаги и умолчания значений выходных переменных, с целью описания в явном

виде всех состояний, в которых может находиться автомат, поведение которого задает этот граф переходов.

Вершины и (или) дуги ГДМ могут быть помечены также кортежами, образованными значениями всех выходных и временных переменных системы.

Из изложенного следует, что ГДМ определяет все функциональные возможности СВГП и является (или может быть сведен к) ГП единого автомата, эквивалентного этой системе.

В случае, если исследование ГДМ показывает, что построенная спецификация корректна, возможны два варианта:

— по спецификации выполняется программирование, а ГДМ после «сведения» используется в качестве сертификационного теста для проверки правильности выполнения программирования;

— ГДМ после «сведения» применяется в качестве «понятной» спецификации, по которой проводится программирование, и в качестве теста для сертификации построенной по нему программы.

При этом отметим, что в рамках предлагаемой технологии программирование должно проводиться по спецификации, формально и желательно изоморфно, что резко уменьшает число ошибок при программировании.

Если с помощью ГДМ в спецификации обнаружены ошибки, то она должна корректироваться и по новой спецификации должен строиться новый ГДМ.

Однако так как формальный метод корректировки спецификации с целью получения «желаемого» ГДМ не известен, возможен другой подход для построения корректной спецификации. Он состоит в преобразовании полученного ГДМ в желаемый ГДМ, который и используется (после замены пометок его вершин, определяющих состояния компонент исходной спецификации, значениями одной многозначной переменной) в качестве корректной спецификации в форме ГП автомата Мура.

Обратим внимание на то, что при построении исходной спецификации предлагается использовать некоторые ограничения, выполнение которых обеспечивается при ее реализации программными средствами. Эти ограничения, влияющие также и на построение ГДМ, состоят в следующем:

— значение каждой входной переменной за время однократного прохода СВГП не изменяется;

— в течение одного прохода СВГП в каждом графе выполняется не более одного перехода;

— «состязания» внутренних переменных в каждом ГП устраняются или переобозначением этих переменных (при различных видах двоичного кодирования), или использованием одной многозначной внутренней переменной, которой не с чем состязаться в одном графе переходов.

Первое ограничение для одного ГП устраняет проблему «риска» при вычислении повторных булевых формул, помечающих его дуги.

Для СВГП это ограничение устраняет также «состязания» внутренних переменных, принадлежащих разным ГП системы, т. е. переходы в ГДМ по разным траекториям из одной его вершины в другую в зависимости от того, при выполнении (перед выполнением) какого ГП осуществляется изменение значения хотя бы одной входной переменной.

Если это ограничение не выполняется, то можно полагать (по аналогии со срабатыванием элементов асинхронной схемы), что если в некотором состоянии системы одновременно выполняются условия переходов в нескольких ГП системы (по одному в каждом из них), то переход выполняется только в одном графе системы. Это позволяет изображать в вершине ГДМ, соответствующей этому состоянию системы, лишь N дуг вместо 2^N, требующихся в общем случае. Здесь N — число одновременно выполняемых условий в СВГП.

Пусть ГДМ для СВГП (рис. 5.45) находится в состоянии «10», тогда если первое ограничение не выполняется, то в случае, когда переменная x_4 примет единичное значение при выполнении первого ГП, в графе достижимых маркировок появится траектория «10—20—21», а в случае, когда это произойдет при выполнении второго ГП, появится траектория «10—11—21».

При выполнении указанного ограничения имеет место только первая траектория.

Выполнение второго ограничения, во-первых, позволяет установить дисциплину реализации всех графов системы, а во-вторых, делает значения каждой выходной и внутренней переменных каждого ГП доступными всем другим ГП системы. Если множества указанных переменных для различных ГП системы не пересекаются, то указанные значения доступны и для вывода во «внешний мир».

Это ограничение делает возможным для одного ГП, во-первых, чтобы для одной вершины пометки входящих и исходящих дуг совпадали, а во-вторых, чтобы значения переменных, помечающих эти дуги, изменялись при «прохождении» рассматриваемой вершины.

Выполнение указанного ограничения не позволяет:

— при $x_4 \cdot x_5 = 1$ перейти за один проход в первом ГП (рис. 5.45) из первой вершины в третью;

— не пропустить первую вершину при $z = 1$ в ГП (рис. 5.9).

Однако это ограничение обеспечивает возможность применения значения внутренней переменной одного ГП в пометках дуг другого ГП (рис. 5.52).

Как отмечалось выше, изложенный подход может использоваться не только для СВГП, но и для одного ГП, когда он содержит флаги и (или) умолчания значений выходных переменных.

Если в ГП флаги и умолчания не применяются, то, задавая поведение автомата, он также и описывает поведение в явном виде, так как в этом случае ГП и ГДМ совпадают. В противном случае ГП задает, но не описывает в явном виде поведение автомата. При этом ГП и соответствующий ему ГДМ различны.

Пусть в качестве примера построен ГП с тремя вершинами и умолчанием значений выходной переменной в одной из них (рис. 5.71). Он задает поведение автомата, но в отличие от его ГДМ с четырьмя вершинами, в которых нет умолчаний (рис. 5.72), не описывает поведение автомата в явном виде.

Если в построенном ГДМ в одной из вершин пометку «2» заменить на пометку «3», то он преобразуется в хорошо «читаемый» и реализуемый ГП, так как он не содержит умолчаний значений выходной переменной и все его вершины имеют различные коды.

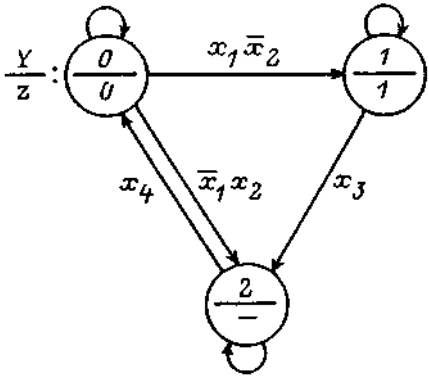

Рис. 5.71

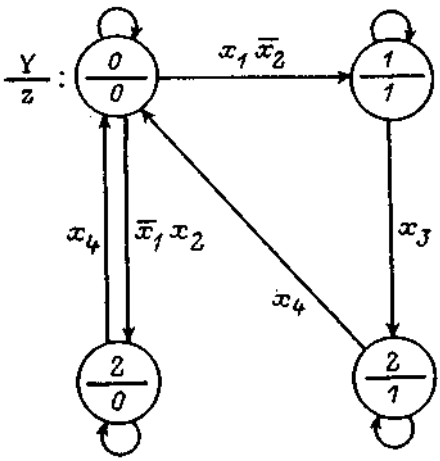

Рис. 5.72

Сравнивая полученный и исходный ГП, можно считать, что им соответствуют различные автоматы, отличающиеся числом состояний и принадлежащие разным классам: первый является автоматом Мура, а второй — автоматом Мура с сохранением значений выходных переменных. Однако в этом случае (по аналогии с минимизацией числа состояний автоматов, проводимой, правда, внутри одного класса автоматов (разд. 4.4.5)) можно считать также, что имеется один автомат, в котором в результате эквивалентных преобразований увеличено число его состояний.

Пусть задан ГП автомата с флагом, содержащий семь вершин (рис. 9.9). Его поведение в явном виде описывает ГДМ с десятью вершинами, который получается из ГП на рис. 9.7 при следующих заменах номеров вершин: 4 на 6, 6 на 2, 7 на 4, 8 на 1, 9 на 2.

ГП автомата с двумя флагами на рис. 9.10 содержит пять вершин, в то время как соответствующий ему ГДМ содержит 10 вершин.

Обратим внимание на тот факт, что если сложность ГДМ незначительно превышает сложность соответствующего ему ГП и находится в рамках допустимых в каждом конкретном случае ограничений, то при использовании исходного или преобразованного ГДМ в качестве спецификации его усложнение компенсируется за счет значительного улучшения его «читаемости».

Построение ГДМ ввиду трудоемкости может проводиться не целиком, а до обнаружения с его помощью первой ошибки в спецификации. На рис. 5.47 приведен начальный фрагмент ГДМ для СВГП (рис. 5.45), в пятой вершине которого формируется кортеж значений выходных переменных, не соответствующий реализуемому алгоритму. Этот фрагмент содержит лишь пять вершин, в то время как ГДМ в целом — одиннадцать.

Спецификация после обнаружения с помощью ГДМ ошибки в ней должна корректироваться, например, как показано на рис. 5.52. Этой

спецификации соответствует простой и правильно функционирующий ГДМ (рис. 5.53), содержащий восемь вершин.

После этого можно выполнять программирование по СВГП (рис. 5.52) или, исключив в ГДМ две вершины и переобозначив остальные, получить один ГП (рис. 5.43) с шестью вершинами, по которому также может быть написана программа.

При этом исходный или преобразованный ГДМ может применяться в качестве сертификационного теста для проверки программы. В этом случае в отличие от подхода, описанного в начале раздела, ошибки, обнаруживаемые этим тестом, принадлежат только программе, а не спецификации.

Несомненно, что и в этом случае спецификация может содержать ошибки, но они таковы, что все заинтересованные в разработке Специалисты, участвующие в согласовании спецификации, на этой стадии их не «видят». При таком подходе и тестирование формально и изоморфно построенной программы вряд ли поможет обнаружить ошибки спецификации, за исключением случаев, когда вдруг кто-то из Специалистов внезапно «прозреет» или на испытаниях программы появится Специалист, который не участвовал в согласовании спецификации. Ошибки в согласованной спецификации могут быть обнаружены на каждой из следующих стадий создания системы логического управления:

— при моделировании, т. е. совместном функционировании разработанной управляющей программы с программой, реализующей модель объекта управления;

— при испытаниях системы совместно с физической моделью объекта управления, если последняя имеется;

— при испытаниях системы на реальном объекте управления и даже в ходе ее многолетней эксплуатации.

Изложенный подход, ориентированный на построение одного ГДМ для СВГП, может оказаться весьма трудоемким из-за чрезвычайно большого числа вершин в нем, особенно при наличии параллельных процессов в реализуемом алгоритме. В этом случае может использоваться подход, описанный в начале раздела.

Если СВГП можно разбить на независимые подсистемы, то вместо построения единого ГДМ для системы в целом должен строиться ГДМ для каждой из подсистем в отдельности.