УДК 519.714

© 1996 г. А. А. ШАЛЫТО, канд. техн. наук
(НПО "Аврора", Санкт-Петербург)

ИСПОЛЬЗОВАНИЕ ГРАФ-СХЕМ И ГРАФОВ ПЕРЕХОДОВ ПРИ ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ. II

Рассматриваются методы построения "читаемых" графов переходов (ГП) по граф-схемам алгоритмов и обратная задача – построение "читаемых" граф-схем по графам переходов. Излагаются методы программирования, обеспечивающие для языков различных уровней читаемость программ. Выполнено сравнение предлагаемого подхода с основным методом структурного программирования – методом Ашкрофта и Манны. Показаны преимущества и определены требования, при выполнении которых целесообразно использовать ГП в качестве языка общения и спецификации для задач рассматриваемого класса.

1. Введение

В [1] рассмотрены особенности граф-схем алгоритмов (ГСА) и программ, затрудняющие их понимание. Предложена классификация ГСА и показано, что переход к графам переходов (ГП), удовлетворяющим определенным условиям, позволяет решить проблему чтения и понимания спецификаций.

Настоящая работа посвящена описанию методов построения понимаемых ГП по ГСА и решению обратной задачи – построению понимаемых ГСА по ГП. Излагаются методы программирования, обеспечивающие для языков различных уровней понимаемость программ, реализующих функциональные задачи логического управления. Выполнено сравнение предлагаемого подхода с основным методом структурного программирования – методом Ашкрофта и Манны.

2. Метод построения читаемого графа переходов по граф-схеме алгоритма с обратными связями

Изложение метода выполним на примере. Пусть задана граф-схема (рис. 1) алгоритма с внутренними обратными связями (ВОС) (ГСА1 в терминологии [1]) и требуется понять эту граф-схему (ГС).

Если закодировать числами (числа в кружках) точку начала и конца (если она имеется), а также точки, следующие за операторными вершинами (ОВ), и определить пути в ГСА между смежными точками [2], то можно построить ГП автомата Мили (АМИ) с неоднозначными значениями выходных переменных (рис. 2), число вершин в котором равно количеству точек, введенных в ГСА. Этот ГП может быть эффективно запрограммирован, например, на языке СИ, но понять (из-за умолчаний некоторых значений выходных переменных), как он функционирует и соответственно как функционирует ГСА1, весьма сложно.

Поэтому построим по ГСА1 граф переходов автомата Мура (ГП АМ), который по принципам построения должен пониматься лучше [1]. Для этого закодируем числами (без кружков) каждую ОВ (рис. 1) и определим все пути между смежными вершинами [2]. Используя эту информацию, построим ГП АМ с неоднозначными

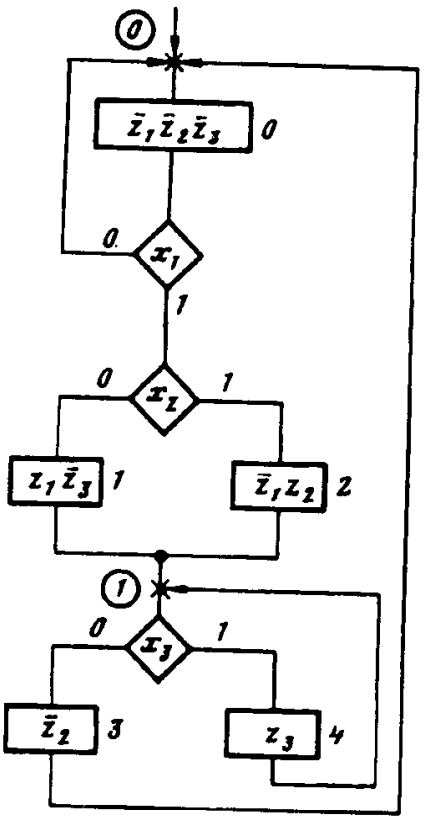

Рис. 1

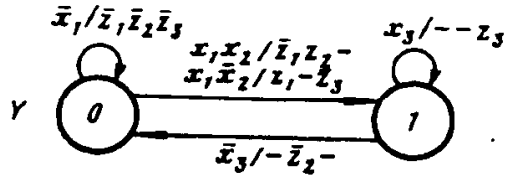

Рис. 2

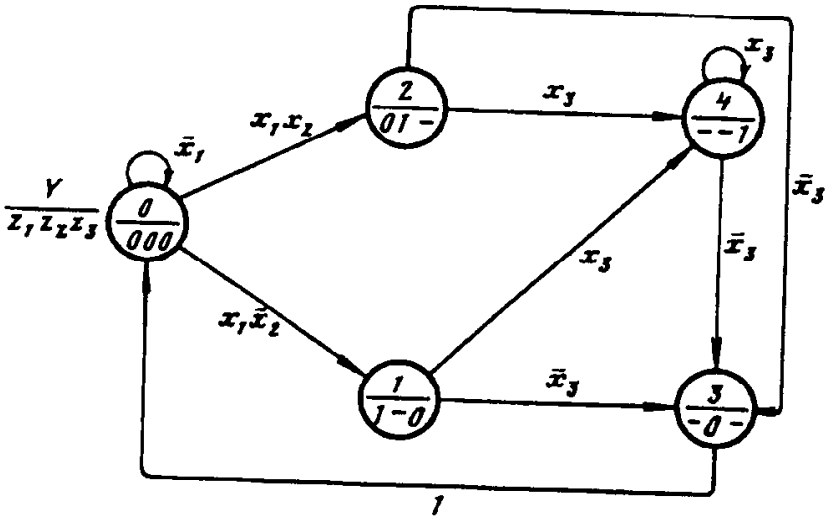

Рис. 3

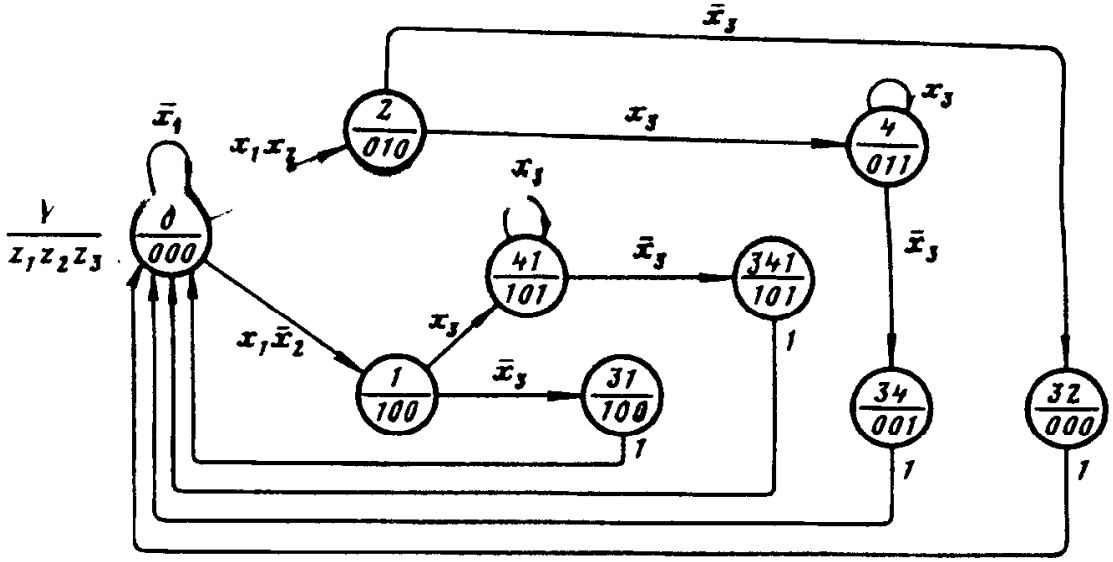

Рис. 4

значениями выходных переменных, содержащий пять вершин, закодированных многозначными (десятичными) числами (рис. 3). Этот граф понимается лучше, чем предыдущий, но также весьма сложно, так как и этот ГП содержит умолчания.

Анализируя значения, формируемые при прохождении различных путей в этом ГП, построим по результатам анализа ГП АМ без умолчаний с девятью вершинами (рис. 4). Так как в этом ГП в каждой паре вершин (41, 341) и (1, 31) формируются одинаковые значения выходных переменных, то вторые вершины в этих парах вместе с дугами, помеченными единицами (безусловные переходы), могут быть исключены. При этом первые вершины этих пар соединяются с нулевой вершиной.

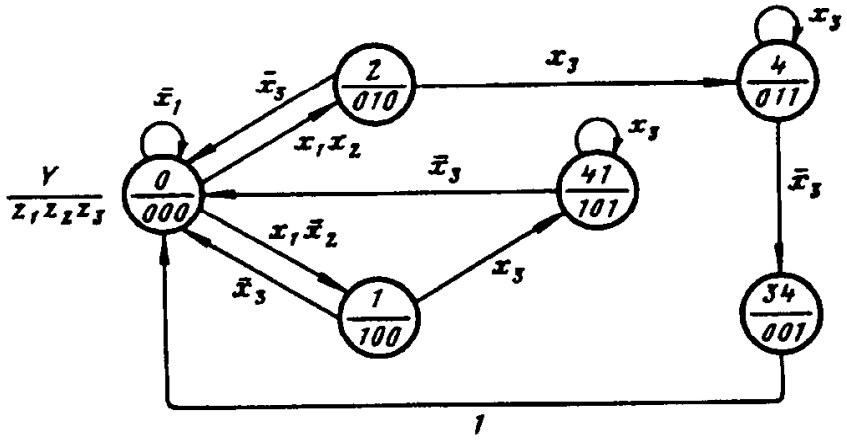

Рис. 5

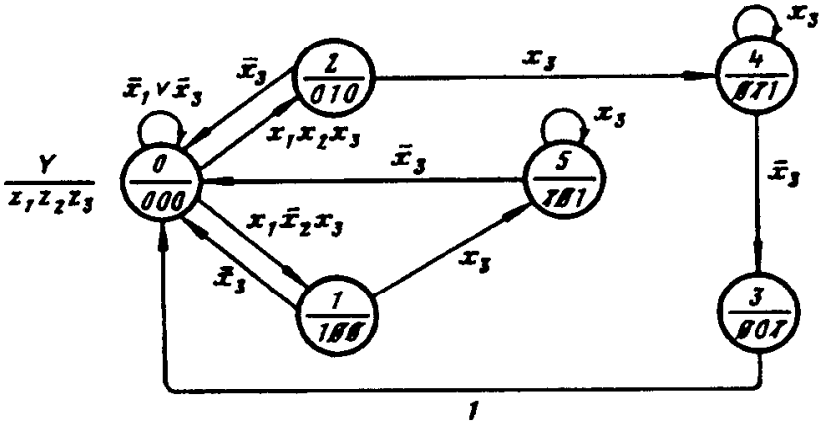

Рис. 6

С этой же вершиной соединяется вершина 2, так как вершина 32, в которой формируются те же значения выходных переменных, что и в нулевой вершине, может быть исключена. Получающийся при этом ГП АМ содержит шесть вершин (рис. 5). Этот ГП "абсолютно" понятен, так как он компактен и при его чтении не требуется помнить предысторию. При этом необходимо отметить, что его структура, в отличие от ранее рассмотренных графов переходов (рис. 2, 3), существенно отличается от исходной ГСА.

Полученный ГП полон и непротиворечив, однако содержит, как и ГСА1, два генерирующих контура $0 = 1$ и $0 = 2$, устраняя которые, например введением переменной x_3 в пометку дуг $0 - 1$ и $0 - 2$, получим корректный ГП (рис. 6). Обратим при этом внимание на тот факт, что в граф переходов указанные изменения были внесены весьма просто без корректировки его структуры, в то время как для графа схемы алгоритма это потребовало бы существенных изменений схемы, что может приводить к ошибкам.

Из изложенного следует, что именно ГП (рис. 6), а не ГСА1 и должен использоваться в качестве "предмета обсуждения" с заказчиком и спецификации на программирование при отсутствии жестких ограничений на объем памяти программ (П). Приведенный ГП, в отличие от ГСА1, описывает поведение автомата (А) в явной и понятной форме и содержит достаточно информации, чтобы быть формально реализованным с помощью различных алгоритмических моделей (системой булевых формул (СБФ), функциональной схемой (ФС), ГСА4 [1] и т.д.) [3, 4].

Каждая из этих алгоритмических моделей, в свою очередь, может быть изоморфно отражена программной моделью, однако степень изоморфизма последней с ГП различна. Действительно, если тексты П, записанные, например, на языке СИ, формально построенные по СБФ или ФС, функционально эквивалентны ГП, то внешне

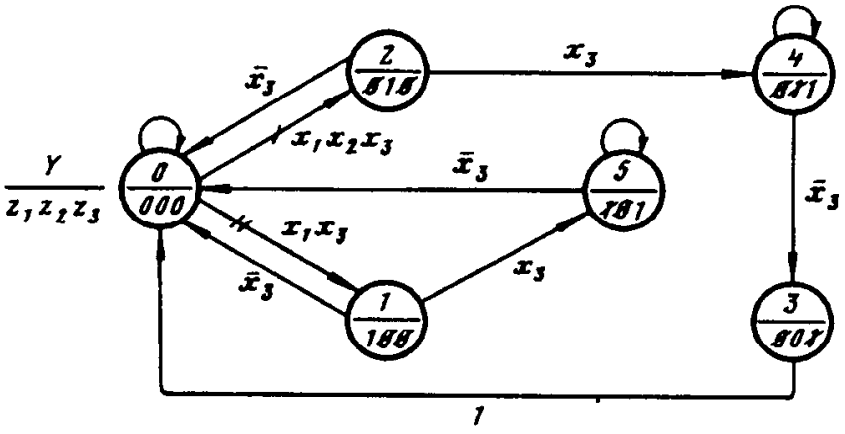

Рис. 7

эти тексты на ГП непохожи, особенно для СБФ, описывающей ФС с триггерами. При этом отметим, что эта СБФ отличается от СБФ, построенной непосредственно по ГП. С другой стороны, в состав указанного языка входит такая управляющая конструкция (УК), как переключатель (switch), при использовании которой удается обеспечить изоморфизм между текстами П и ГП, что открывает возможность простого чтения не только спецификаций, но и текстов программ.

В заключение раздела отметим, что если для построения некоторых алгоритмических и программных моделей, например СБФ или ФС, вся информация, приведенная на рис. 6, необходима, то при использовании УК switch в ГП пометки петель могут умалчиваться (предполагая, что в каждой вершине обеспечивается логическая полнота пометок дуг, исходящих из нее), а вместо ортогонализации этих пометок могут расставляться приоритеты, указываемые на дугах штрихами, число которых тем меньше, чем выше номер приоритета (рис. 7). Это в общем случае позволяет резко упростить ГП и уменьшить объем П, практически без ухудшения их понимаемости.

Перейдем к рассмотрению вопроса о построении читаемой граф-схемы алгоритма без внутренних обратных связей по графу переходов без умолчаний. В [1] было показано, что "плохо организованные" ГСА без ВОС (ГСА2 в терминологии [1]) весьма трудно читаются. Покажем, какой структурой должны обладать ГСА без ВОС, чтобы устранить указанный недостаток.

3. Реализация автоматов без выходного преобразователя с принудительным кодированием состояний

Пусть задан ГП автомата без выходного преобразователя (АБВП) с принудительным кодированием состояний (рис. 8), в котором при импульсных переменных x_1 и x_2 генерирующие контуры отсутствуют. Кодирование названо принудительным, так как коды, определяющие состояния, совпадают со значениями выходных переменных, формируемых в соответствующих состояниях (вершинах). Этот вид кодирования использован ввиду того, что в рассматриваемом ГП комбинации значений выходных переменных во всех вершинах различны. Этот ГП предлагается реализовать ГСА4 [1] (рис. 9), тело которой состоит из трех слоев. Первый из них содержит условные вершины (УВ), помеченные переменными z_1 и z_2, и является дешифратором состояний. Второй слой образован УВ, помеченными входными переменными, и реализует функции переходов автомата. Третий слой содержит ОВ, в которых указаны значения выходов, совпадающие в данном случае с кодами следующих состояний.

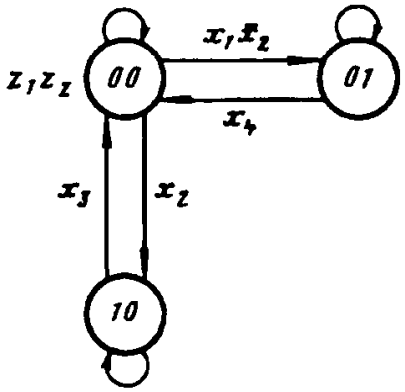

Рис. 8

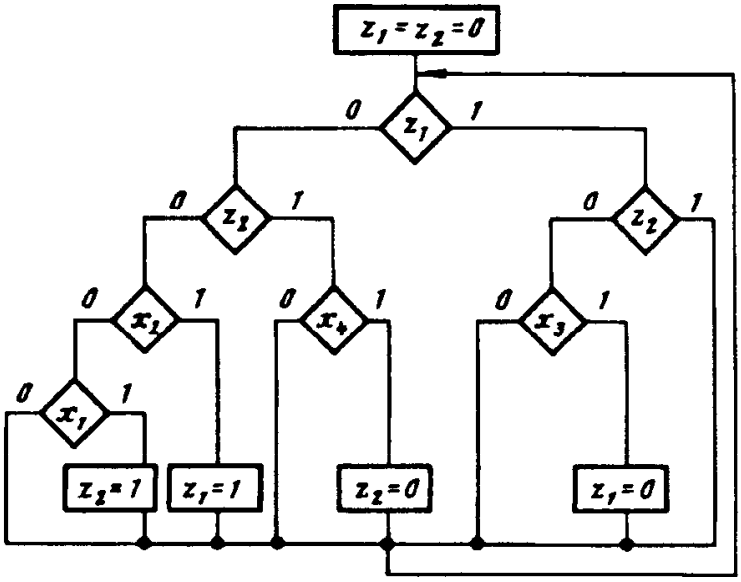

Рис. 9

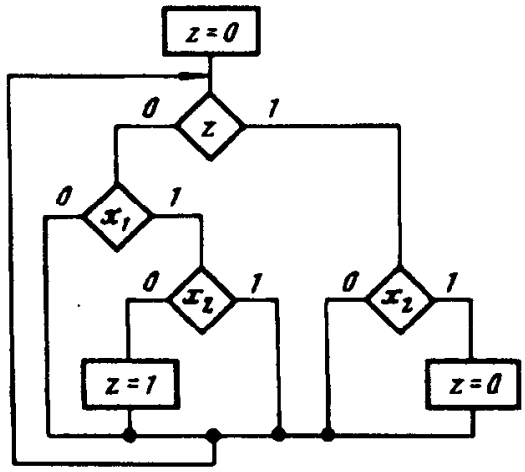

Рис. 10

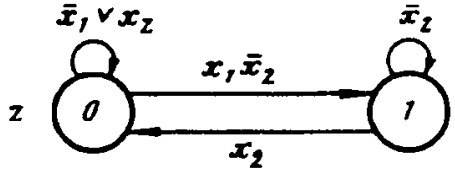

Рис. 11

Несмотря на то, что в ОВ этой ГС используются умолчания, она достаточно хорошо понимается, так как в этом случае, в отличие от ГСА2, сохраняемые значения нет необходимости помнить, а их можно определить, "поднимаясь" вверх по соответствующему пути в ГС.

Понимаемость этой ГС придает также и то, что она обладает, в отличие, например, от ГСА2 (рис. 2 [1]), глубиной, определяемой только одним переходом в ГП, что является чрезвычайно важным и полезным свойством этого класса граф-схем. Кроме того, в этой ГС за один проход, в отличие от ГС (рис. 3 [1]), не приходится неоднократно пересчитывать значения ни одной выходной переменной, а в отличие от ГС (рис. 5 [1]) – многократно проверять значения одной и той же входной переменной. Эта ГС не только структурирована в традиционном понимании, но и хорошо организована в том смысле, что УВ с пометками Z и X не перемешаны между собой и с ОВ.

Такая организация ГСА (настоящее состояние – условие перехода – следующее состояние) соответствует нормальному человеческому поведению, при котором, например, просыпаясь утром, человек сначала определяет свое внутреннее состояние (жив – мертв, здоров – болен) и лишь потом "опрашивает" значения входных переменных (например, тепло или холодно на улице), а затем в зависимости от значений этих переменных переходит в следующее состояние (например, одевается соответ-

ствующим образом). Ясно, что в этом случае поведение без учета внутреннего состояния только по значениям входных переменных будет выглядеть странным.

Однако при алгоритмизации с помощью ГСА без ВОС понятие "состояние" обычно в явном виде не используется и ее строят, начиная с опроса входных переменных, и в зависимости от их значений формируют те или иные значения выходных, а возможно, и дополнительно введенных внутренних переменных, которые определяют состояния косвенным (компонентным) способом.

Например, несмотря на то, что ГСА2 (рис. 3 [1]), реализующая R-триггер, идеально соответствует принципам структурного программирования, она обладает двумя недостатками, затрудняющими ее чтение: пересчет значения z при $x_1 = x_2 = 1$ и отсутствие в модели указания в явном виде значения z при $x_1 = x_2 = 0$. Этих недостатков лишена ГСА с дешифратором состояний (рис. 10), которая, несмотря на большую сложность по сравнению с ГСА (рис. 3 [1]), понимается лучше. Простоту чтения и компактность изображения совмещает в себе ГП (рис. 11), который при программировании с помощью конструкции switch может быть еще более упрощен (умолчание пометок петель).

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

Пусть требуется реализовать счетный триггер, поведение которого описывается ГП (рис. 12). Для того чтобы различать вершины ГП, используем принудительно-свободное кодирование состояний A, введя отсутствующую в алгоритме управления (АУ) внутреннюю переменную y (рис. 13). Название этого вида кодирования определяется тем, что значения части разрядов кода принудительно задаются значениями выходных переменных z, а значения остальных разрядов, обозначаемых переменными y, могут быть выбраны при программной реализации (ПР) свободно (произвольно). На рис. 14 приведена ГСА, эквивалентная этому ГП.

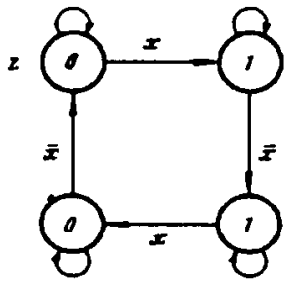

Рис. 12

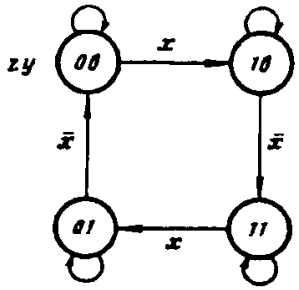

Рис. 13

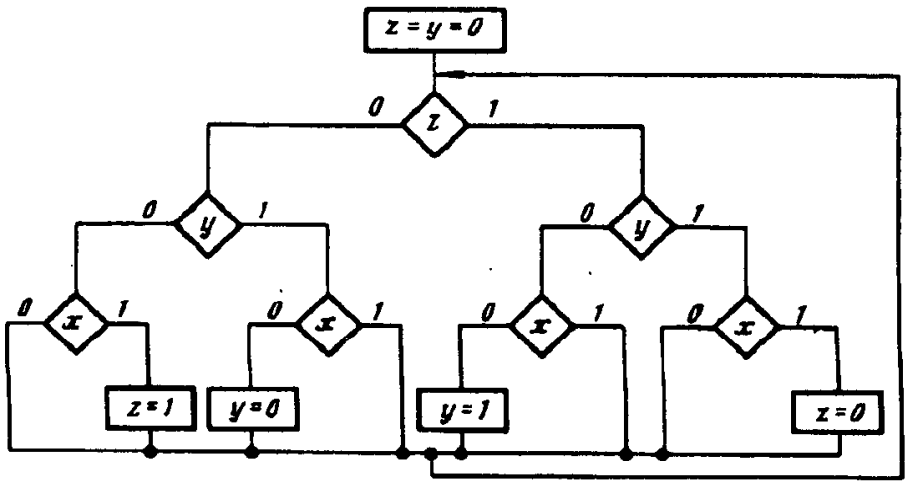

Рис. 14

Зависимость состояний A от значений выхода z в рассмотренных ГСА (рис. 9. 10 и 14) может создать трудности при условии, что эти значения могут изменяться в других компонентах АУ. Поэтому сделаем независимыми состояния A от его выходных значений, перейдя к модели АМ.

5. Реализация автоматов Мура с двоичным логарифмическим кодированием состояний

На рис. 15 приведен ГП АМ, реализующий счетный триггер, для рассматриваемого варианта кодирования, а на рис. 16 – соответствующая ему ГСА. Тело этой ГСА является четырехслойным (дешифратор состояний, формирование значений выходов, реализация функций переходов, формирование следующего состояния). Недостатки этой ГСА состоят в значительной глубине пирамидального дешифратора и большом числе вновь вводимых переменных y_i, где $0 < i <]\log_2 s[-1$, где s – число состояний A. Первый из этих недостатков устраняется в разделе 6, а оба недостатка – в разделе 7.

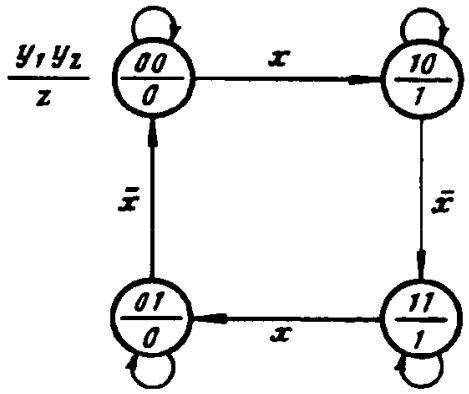

Рис. 15

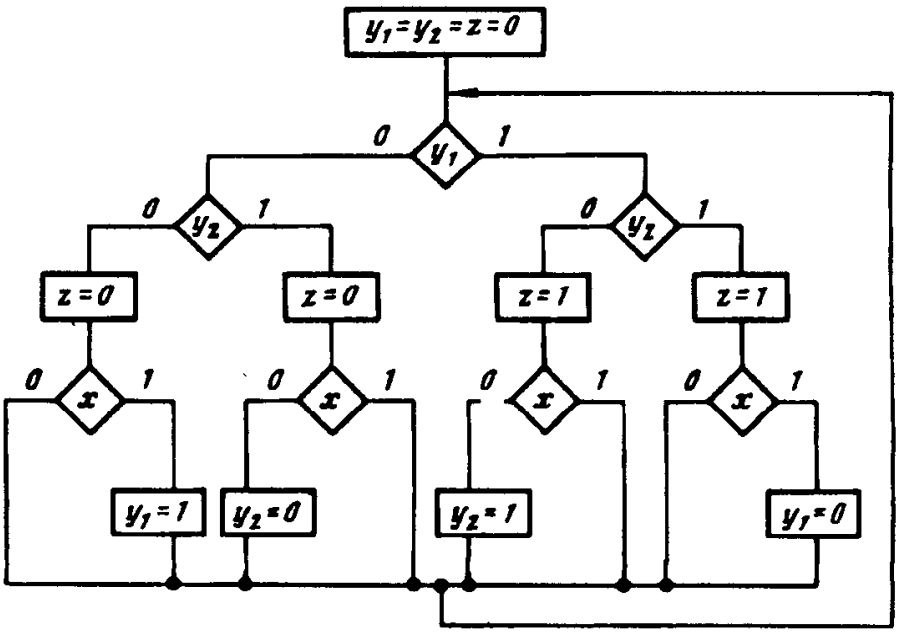

Рис. 16

6. Реализация автоматов Мура с двоичным (единичным) кодированием состояний

Такой вариант кодирования вершин ГП АМ (рис. 17), названный в [2] единичным (в j-й вершине ГП только битовая переменная y_j принимает значение, равное единице, а в остальных вершинах значение этой переменной равно нулю), позволяет использовать в ГСА (рис. 18) линейный дешифратор состояний вместо пирамидального, который применялся в ГСА АМ при других видах кодирования. Недостаток ГСА в этом случае состоит в еще большем (по сравнению с предыдущим случаем) числе вновь вводимых битовых переменных y_j $(0 < j < s-1)$, каждую из которых приходится не только устанавливать, но и принудительно сбрасывать.

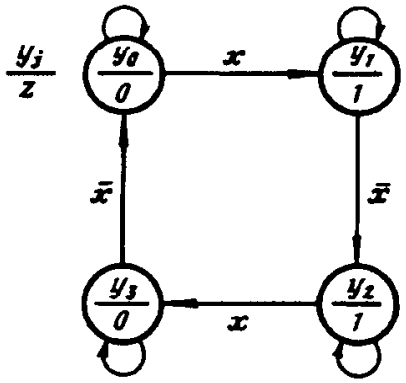

Рис. 17

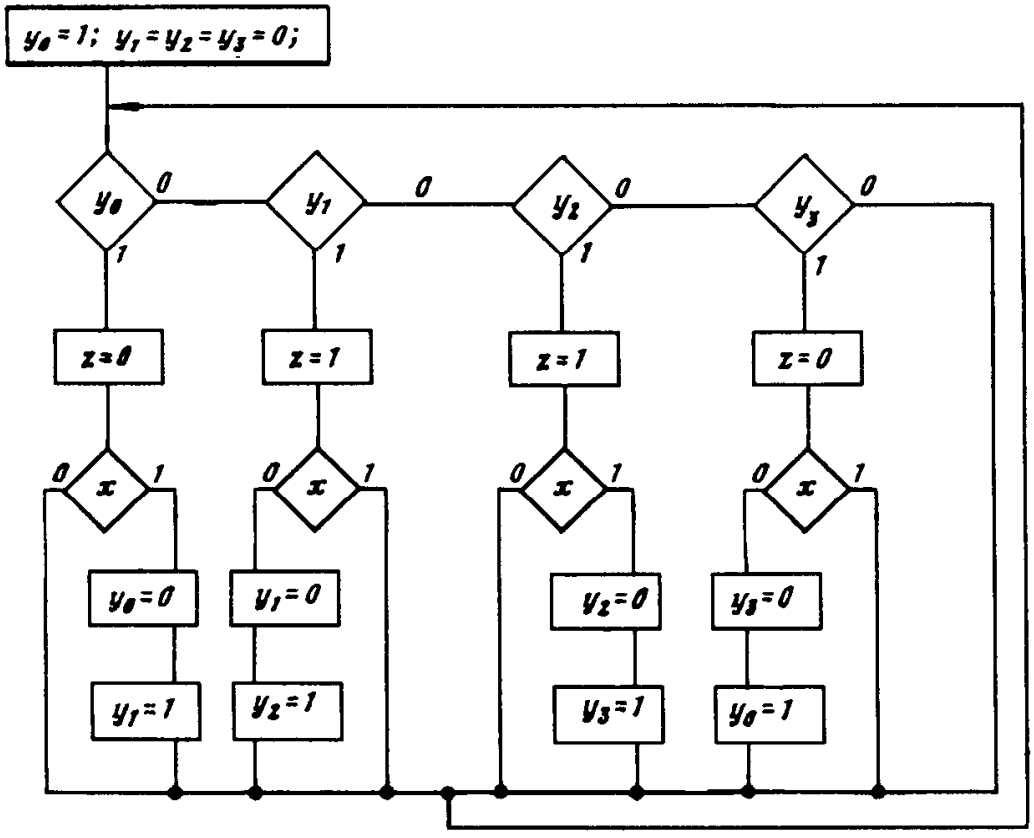

Рис. 18

7. Реализация автоматов Мура с многозначным кодированием состояний

Все недостатки предыдущих вариантов кодирования устраняются при переходе к многозначному кодированию состояний автоматов [5, 6]. При этом для АМ, АМИ и смешанных автоматов (СА), реализуемых в виде одной компоненты, для кодирования состояний достаточно иметь ОДНУ переменную Y значности s, которую не требуется принудительно сбрасывать, так как она автоматически сбрасывается при переходе от одного значения к другому. При этом для реализации N-компонентного АУ (реализуемого N графами) с помощью указанных классов A требуется N многозначных переменных Y_j, где $j = 0 \ldots N-1$. Этот вариант кодирования, практически нереализуемый при аппаратной реализации A, по мнению автора, при отсутствии жестких ограничений на объем памяти П и возможности работы с многозначными переменными должен стать основным при программной реализации A.

На рис. 19 в качестве примера приведена ГСА для рассматриваемого варианта кодирования, построенная по ГП АМ счетного триггера (рис. 7 [1]). В этой граф-схеме за счет ухудшения понимаемости УВ с пометкой $Y = 3$ и вторая ОВ с пометкой $z = 1$ могут быть исключены.

ГСА аналогичной структуры может быть построена для любого ГП АМ. При этом, в отличие от ГП, такая ГСА всегда может быть планарной. Применение всего лишь одной промежуточной переменной при "хорошей" организации структуры делает использование ГСА этого типа весьма привлекательным. Единственный недостаток, кроме громоздкости, такого типа структур (если умалчивание значений выходных переменных исключено) состоит в том, что так как в них связь между фрагментами осуществляется не непосредственно, как в ГП, а по данным (значениям переменной Y), то визуальное обнаружение генерирующих контуров становится весьма затруднительным.

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

При использовании многозначного кодирования открывается весьма эффективная в изобразительном отношении дополнительная возможность обеспечения связи между компонентами АУ (в том числе и при реализации параллельных процессов)

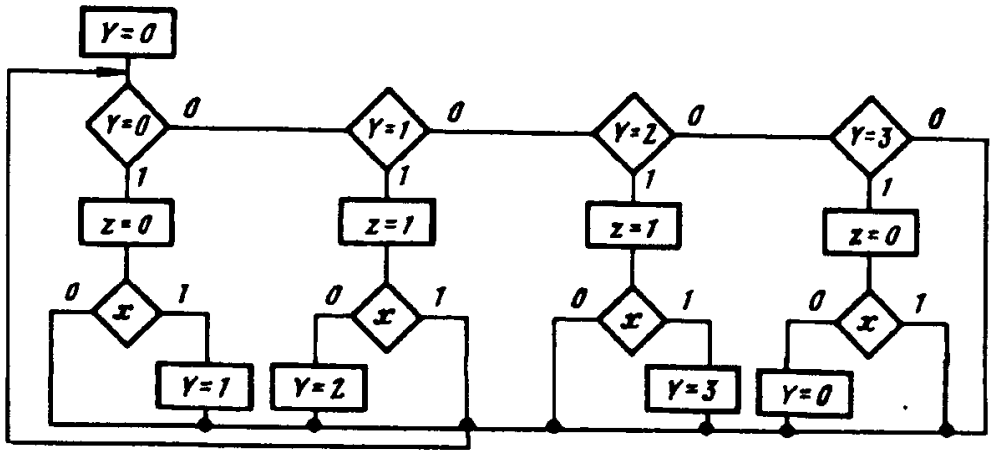

Рис. 19

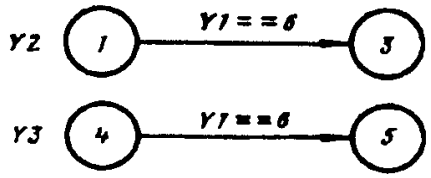

Рис. 20

за счет обмена десятичными значениями используемых при взаимодействии номеров состояний компонент (возможность взаимодействия по входным, выходным и дополнительно вводимым внутренним переменным сохраняется). Пусть, например, требуется, чтобы вторая компонента перешла из первого состояния в третье, а третья компонента – из четвертого состояния в пятое при условии, что первая компонента находится в шестом состоянии. Тогда без введения дополнительных переменных этот параллельный процесс весьма наглядно отражается фрагментом системы взаимосвязанных ГП (СВ ГП), представленным на рис. 20. В [7] показано, что СВ ГП обладают широкими изобразительными возможностями и, в частности, в явном виде отражают такие важные свойства сложных систем, как параллелизм и иерархичность.

При этом отметим, что и один ГП, являясь последовательным по состояниям, обычно является параллельным по входам и выходам (например, ГП рис. 7 и 8).

8. Реализация автоматов Мили с многозначным кодированием состояний

В автоматах Мили, так же как в автоматах Мура, могут использоваться различные виды кодирования состояний, однако, исходя из изложенного, рассмотрим только многозначное кодирование. На рис. 8 [1] приведен ГП АМИ, реализующий

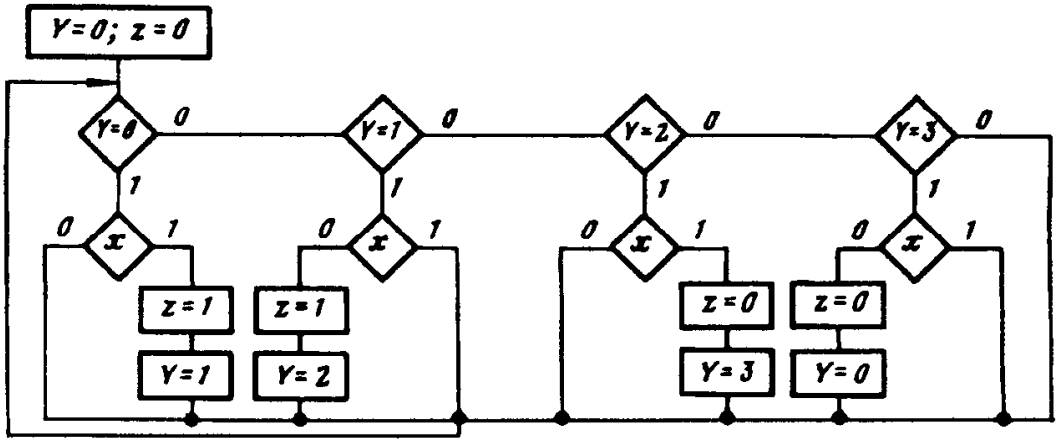

Рис. 21

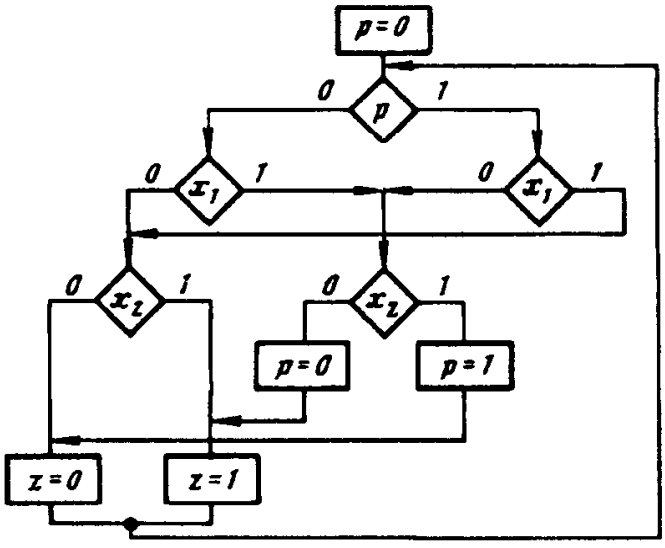

Рис. 22

счетный триггер при принятом варианте кодирования, а на рис. 21 – соответствующая ГСА. В этой ГС могут быть исключены УВ с пометкой $Y = 3$ и вторые ОВ с пометками $z = 1$ и $z = 0$. Отличие этой ГС от ГС АМ состоит в размещении слоя ОВ, помеченных значениями выходной переменной, не до дешифраторов значений входных переменных, а после них.

Второй пример реализации этого класса A приведен на рис. 22 для последовательностного одноразрядного сумматора (рис. 10 [1]). В этой ГС для минимизации числа вершин слой значений выходной переменной z расположен после слоя со значениями следующего состояния p. Тело этой ГС, состоящее всего лишь из девяти вершин, построено с помощью модификации метода Блоха, изложенной в [8].

9. Реализация смешанных автоматов с многозначным кодированием состояний

На рис. 23 изображено тело ГСА, реализующей C-автомат с флагом, представленный на рис. 11 [1]. Эта ГС является пятислойной: дешифратор состояний, формирование значений выходных переменных, реализация функций переходов, формирование значений флага, формирование следующего состояния. В приведенной ГС переменная x_1 определяет содержимое счетного триггера (другой компоненты АУ), который также управляется и от другого источника информации – кнопки. В этой ГСА, в отличие от ГП, противоречивость устранена ортогонализацией и используется булев признак T вместо неравенства $t \geqslant D$.

При этом необходимо отметить, что если при использовании СВ ГП каждая компонента замкнута, то для ГС, реализуемых в программируемых логических контроллерах (ПЛК) [9], замыкание выполняется для АУ в целом.

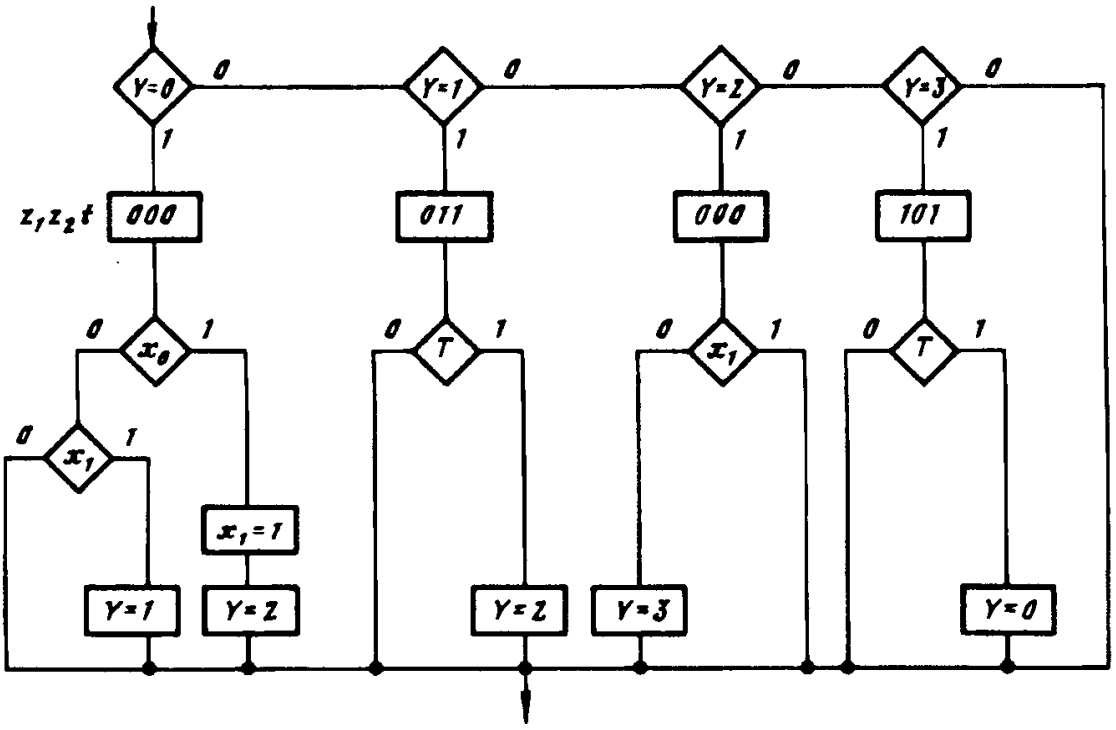

Рис. 23

10. Программирование графов переходов и граф-схем алгоритмов с многозначным кодированием состояний в базисе языков высокого уровня

Использование многозначного кодирования позволяет для ГСА4, так же как и для ГП, изоморфно переходить к тексту П, минуя построение промежуточных ГС [1]. При этом несмотря на то, что ГП содержат петли и контура (негенерирующие), их не приходится принудительно расцикливать и структурировать, а эти проблемы решаются при программировании выбором соответствующей УК.

Наиболее просто это достигается при использовании такой УК, как, например, оператор switch языка СИ [5, 6, 10]. Приведем в качестве примера П, использующую этот оператор для реализации ГП АМ (рис. 7 [1]) и ГСА (рис. 19):

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

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

В приведенной П в состоянии 2 (case 2) значение выходной переменной, не изменяющееся при переходе из предыдущего состояния, закомментировано. Это, с одной стороны, уменьшает объем памяти, а с другой – сохраняет хорошую читаемость П. В этой программе каждый оператор break осуществляет передачу управления за закрывающуюся фигурную скобку, что при невыполнении условия в операторе if сохраняет предыдущее состояние, а при его выполнении не позволяет выполнить более одного перехода в ГП. При этом для СВ ГП за один программный цикл в каждом ГП реализуется не более одного перехода. Это делает доступным номер любого состояния рассматриваемого A для других компонент АУ вне зависимости от значений входных переменных, в отличие, например, от ГСА2 (рис. 3 [1]), в которой при $x_1 = x_2 = 1$ значение $z = 1$ для других компонент недоступно.

Так, например, если первая компонента АУ реализована указанным образом и значение переменной $Y1$, равное шести, доступно другим компонентам, то фрагменту СВ ГП (рис. 20) соответствует следующий фрагмент программы, использующий значение шестого состояния первой компоненты в качестве условия перехода в двух других компонентах:

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

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

Приведем теперь пример П, построенной по ГП АМИ (рис. 8 [1]) и ГСА4 (рис. 21), предполагая, что в начале $Y = 0$; $z = 0$:

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

С помощью УК switch также эффективно реализуются и C-автоматы.

Из изложенного следует, что ГП, во-первых, может одновременно использоваться в качестве спецификации как АУ, так и П, а во-вторых, с помощью УК switch он изоморфно отражается в текст структурированной П, которая может быть наблюдаемой не только по двоичным выходам, но и, что самое главное, по десятичному номеру состояния. При этом для проверки П на экран дисплея для каждого ГП АМ, АМИ или СА достаточно вывести только ОДНУ десятичную внутреннюю переменную (сигнатуру). Это позволяет при исследовании поведения A в динамике следить только за значениями этой переменной (предварительно определив, например, для АМ соответствие между номером состояния и значениями выходных переменных в этом состоянии), а не за многими двоичными внутренними переменными, как это делается при традиционном подходе.

Это позволяет ввести в программирование понятие "наблюдаемость" по аналогии с понятием "измеримость" (по Л. Заде), используемым в автоматическом управлении [11].

Получаемые таким образом программы обладают высокой модификационной способностью (в некотором смысле легко управляемы) и хорошо читаются при условии, что значения выходных переменных задаются однозначно, так как в этом случае они не зависят от предыстории. ГП автоматов с флагами также изоморфно отражаются в текст П с помощью УК switch. Однако зависимость следующего состояния от предыстории резко уменьшает их модификационную способность.

Отметим также, что основная причина простоты внесения изменений в П, построенные указанным образом по ГП, состоит в том, что в последних вершины связаны между собой непосредственно, в отличие от ГС, в которых ОВ в общем случае

соединены через УВ и поэтому изменение в одном переходе оказывает существенное влияние на другие переходы.

При использовании предлагаемого подхода верификация П может производиться сверкой текста П с ГП. Для однокомпонентных АУ граф переходов может использоваться в качестве теста для проверки П, а для многокомпонентных АУ (по аналогии с сетями Петри) по СВ ГП должен строиться граф достижимых маркировок, позволяющий исследовать все функциональные возможности системы и представить ее поведение одним ГП. Эта задача практически разрешима при отсутствии параллелизма по состояниям в АУ или для задач малой размерности. Для независимых параллельных процессов строить единый ГП нет необходимости, так как они могут проверяться по отдельности.

11. Программирование граф-схем алгоритмов с внутренними обратными связями в базисе языков высокого уровня

Подход, излагаемый в настоящей работе, позволяет предложить метод [redacted] программирования граф-схем алгоритмов с ВОС, которые в многозадачном режиме использования применяемого вычислительного устройства (ВУ) не реализуется традиционным путем без предварительного расцикливания [1].

Суть предлагаемого метода состоит в том, что по заданной ГСА строится эквивалентный ГП, который затем изоморфно отражается в текст П с помощью УК switch.

Пусть, например, задана ГСА с ВОС (рис. 1). По этой ГСА может быть построен ГП АМИ (рис. 2) или ГП АМ (рис. 3), каждый из которых может быть однозначно реализован конструкцией switch. Несмотря на то, что получающиеся при этом П весьма компактны, эти ГП содержат генерирующие контуры и их понимание весьма затруднительно из-за неоднозначности значений выходных переменных. Эти недостатки отсутствуют в П, построенной по преобразованному ГП АМ (рис. 7):

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

Число строк в этой П равно $s + d + 1$, где d – число дуг (включая петли) в ГП. В операторе case 0 оператор if, соответствующий дуге с наибольшим приоритетом, исходящей из вершины 0, размещается ниже оператора if, соответствующего дуге с меньшим приоритетом. При такой ПР проблемы расцикливания и структурирования исходной ГСА решаются автоматически в ходе построения П. При этом

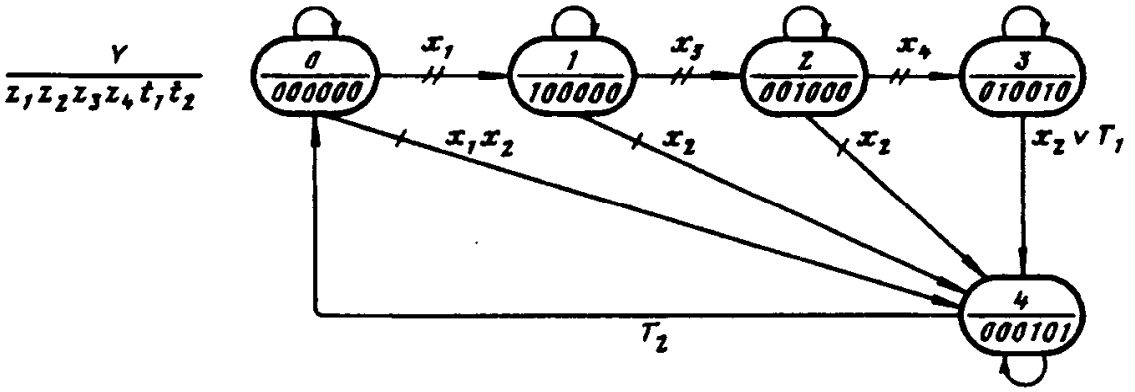

Рис. 24

отметим, что построение ГП по ГСА с ВОС не является необходимым. Для написания П в этом случае достаточно выполнить многозначное кодирование (раздел 2) ОВ заданной ГСА (для построения П, соответствующей АМ) или точек, следующих за ОВ (для построения П, соответствующей АМИ).

Быстродействие П может быть повышено при усложнении ее структуры:

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

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

В качестве второго примера рассмотрим ГСА с ВОС (рис. 2 [1]) и построим эквивалентный ей ГП АМ (рис. 24). В этом ГП, входящем в состав управляющего автомата (УА), каждой единице на позициях t_1 и t_2 соответствует обращение к процедуре time (i, D), где D – время задержки i-го функционального элемента задержки (ФЭЗ). Этот ГП, так же как и предыдущий, реализуется с помощью конструкции switch.

12. Программирование граф-схем алгоритмов и графов переходов с многозначным кодированием состояний в базисе языков низкого уровня

Как отмечалось в [1], для учета свойств УК используемого языка программирования целесообразно переходить от исходной ГСА к тексту П не непосредственно, а используя промежуточную форму ГСА (ГСА3 в терминологии [1]).

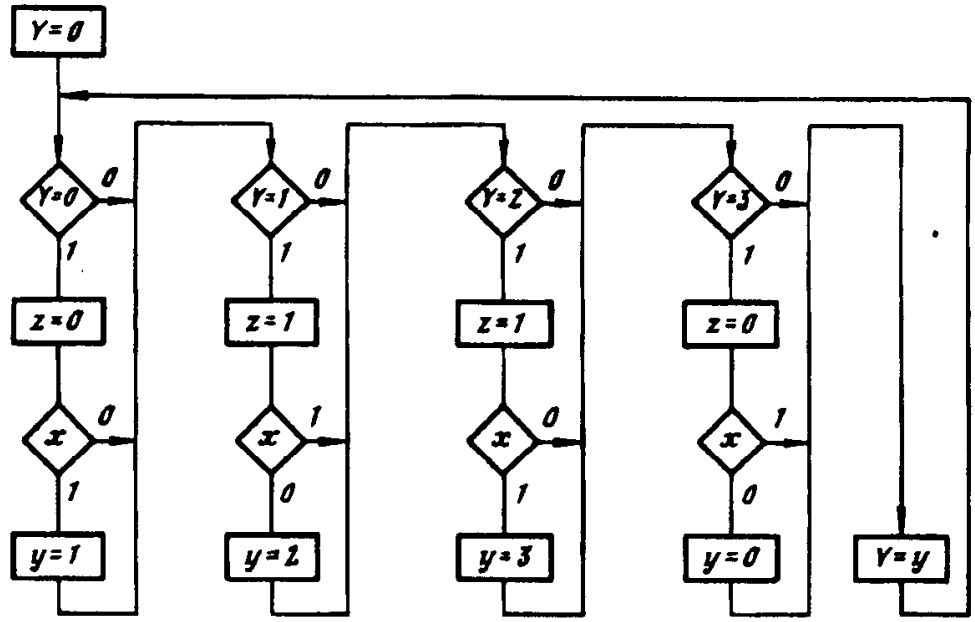

Рис. 25

На рис. 25 приведена линеаризованная и структурированная специальным образом (все УВ в каждом блоке при не выполнении условия передают управление в одну точку) ГСА, эквивалентная граф-схеме алгоритма (рис. 19). Полученная ГСА, в свою очередь, может быть практически изоморфно преобразована в ГСП, учитывающую семантику применяемых команд и архитектурные особенности используемого ВУ, например ПЛК [12], по которой также изоморфно может быть построен текст П на мнемокоде этого контроллера:

```
STR   R   C  0; Ввод константы 0 в регистровый сумматор (PC:=0)
EQU   R   M  Y; if (Y = 0) битовый сумматор (БС)=1
IF        T   ; if (БС) перейти к следующей команде, иначе на
              ; метку CONT
EQ       RO  z; if (БС) z = 0
IF         I  x; if (x) перейти к следующей команде, иначе на
              ; метку CONT
STR   R   C  1; PC:= 1
EQ    R  SM  y; if (БС) y = 1
CONT           ; переход к следующей команде

STR   R   C  1
EQU   R   M  Y
IF        T
EQ       SO  z
IF         I  x
STR   R   C  2
EQ    R  SM  y
...

STR   R   M  y; PC := y
EQ    R   M  Y; Y := PC
STOP           ; передача управления в начало П
```

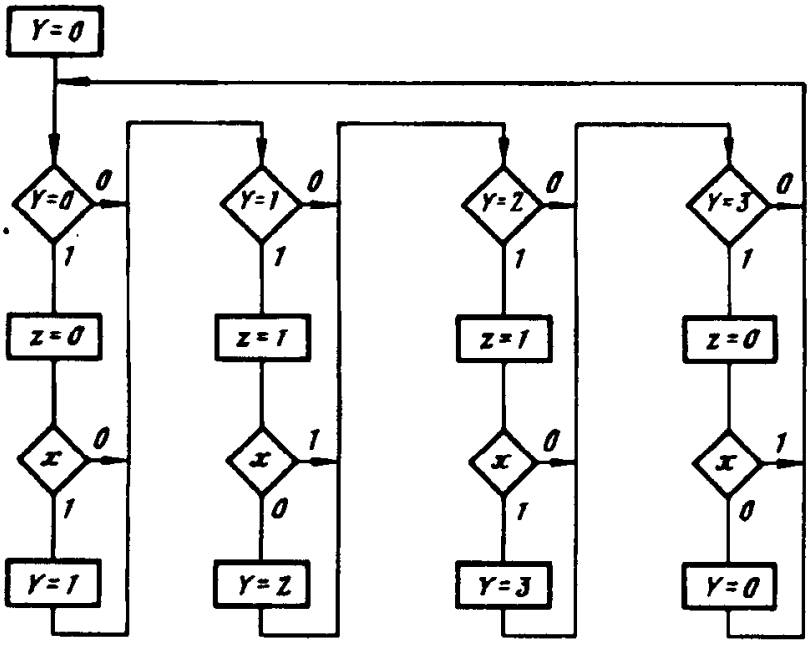

Рис. 26

Если в качестве языка спецификации в этом случае использовать не ГСА, а ГП (рис. 7 [1]), то последний может быть практически изоморфно отражен в следующий текст П:

```
STR   R   C  0;  Ввод константы 0 в регистровый сумматор (РС:=0)
EQU   R   M  Y;  if (Y = 0) битовый сумматор (БС)=1
EQ        RO z;  if (БС) z = 0
AND       I  x;  БС:=БС&x
STR   R   C  1;  РС:=1
EQ    R  SM  y;  if (БС) y = 1

STR   R   C  1;  РС:=1
EQU   R   M  Y;  if (Y = 1) БС=1
EQ        SO z;  if (БС) z = 1
AND       NI x;  БС:=БС&x̄
STR   R   C  2;  РС:=2
EQ    R  SM  y;  if (БС) y = 2
...

STR   R   M  y;  РС := y
EQ    R   M  Y;  Y := РС
STOP             ; передача управления в начало П
```

Эта программа содержит меньшее число команд по сравнению с предыдущей. Она не включает условных переходов и выполняется в режиме последовательного выполнения всех команд и поэтому может быть еще более упрощена за счет исключения третьей команды, если заменить девятую команду на команду EQ 0 z ($z :=$ БС) при условии, что в начале программы $z = 0$. Это объясняется тем, что в этом случае, пока A находится в первой вершине, в ячейку z записывается единица и ноль, когда A "уходит" из этой вершины. Дальнейшее упрощение П связано с еще большим уменьшением степени ее изоморфизма с ГП. При этом каждая пара команд 5–6 и 11–12 может быть заменена командой INC R M y ($y := y + 1$).

Рассмотренные П обеспечивают реализацию за один программный цикл не более одного блока в ГСА (рис. 19) или одного перехода в ГП (рис. 7 [1]) за счет использования второй многозначной внутренней переменной y, в которой присваивается значение Y на время одного программного цикла.

При этом необходимо отметить, что одной дополнительной переменной y достаточно для реализации любой СВ ГП. Если отказаться от требования, чтобы за один проход выполнялось не более одного перехода в ГП, то ГСА (рис. 25) может быть преобразована в граф-схему алгоритма (рис. 26). Это позволяет еще более упростить приведенные выше П. Построенная ГСА корректна, так как в ПЛК [12] в течение одного прохода программы значения введенных входных переменных не изменяются.

Если в качестве языка спецификации использовать ГП и снять ограничение на число переходов, реализуемых за один программный цикл, то появляется возможность применения при написании П такой нетрадиционной УК, как шаговый регистр (ШР), существующей в системах команд многих ПЛК [9, 12]. Однако в этих работах ШР предлагается использовать в режиме формирования последовательных шагов и не упоминается возможность применения его для реализации произвольных A. Поэтому настоящую работу можно считать в качестве обоснования такой возможности.

Приведем в качестве примера П, реализующую ГП (рис. 7 [1]), в которой используется нулевой (из 32 имеющихся) ШР, находящийся в начальный момент времени на нулевом (из 256 допустимых) шаге (в нулевом состоянии):

```
READ    S   0; Выбор в качестве рабочего ШР с номером 0

STR     S   0; if (S=0) БС=1
EQ     RO   z; if (БС) z = 0
AND     I   x; БС := БС&x
STEP    S   1; if (БС) S=1

STR     S   1; if (S=1) БС=1
EQ     SO   z; if (БС) z = 1
AND    NI   x; БС := БС&!x
STEP    S   2; if (БС) S=2
...
STOP
```

Так как и эта П не содержит условных переходов, то и в ней третья команда может быть исключена при замене седьмой команды на команду EQ 0 z при условии, что в начальный момент времени $z = 0$.

Подходы, изложенные в настоящем разделе, могут быть использованы и для АМИ, C-автоматов и автоматов с флагами. Так, например, ГП АМИ (рис. 8 [1]), описываемый на языке СИ следующей П:

```
     Y = 0;
M :  if((Y == 0) & x)    {z = 0;   Y = 1;}
     if((Y == 1) & !x)   {z = 1;   Y = 2;}
     if((Y == 2)& x)     {z = 1;   Y = 3;}
     if((Y == 3)& !x)    {z = 0;   Y = 0;}
     goto M;
```

реализуется следующей П, использующей ШР:

```
READ    S  0   STR     S  1
                AND    NI  x
STR     S  0   EQ     S0  z
AND     I  x   STEP    S  2
EQ     RO  z   ...
STEP    S  1   STOP
```

Из изложенного следует, что использование ШР делает программирование A на мнемокоде весьма высокоуровневым. Эта же тенденция сохраняется и при реализации УА, тем более, что в ряде случаев ФЭЗ могут быть реализованы командой NEXT Si D, которая, если ШР находится на шаге с номером i в течение D секунд, обеспечивает переход на шаг с номером $i+1$. При применении этой команды может возникнуть необходимость в преобразовании исходного ГП (за счет увеличения числа вершин в нем) для обеспечения перехода только в соседнюю вершину по истечении выдержки времени. При этом необходимо отметить, в данном случае управляющий A не декомпозируется на A и ФЭЗ, а программируется как единое целое. Реализуем в качестве примера ГП АМ с флагом (рис. 11 [1]), предварительно увеличив число вершин в нем до пяти, и при условии, что в начале программы $z_1 = z_2 = 0$:

READ	S	0	STR	S	2
IF	S	0	AND	NM	x_1
STR	M	x_1	STEP	S	3
STEP	S	1			
STR	I	x_0	STR	S	3
EQ	SM	x_1	EQ	0	z_1
STEP	S	2	NEXT	S3	D
CONT					
			STR	S	4
STR	S	1	STEP	S	0
EQ	0	z_2			
NEXT	S1	D	STOP		

В этой П изоморфизм с ГП обеспечивается по вершинам последнего. При этом фрагмент П, соответствующий дуге с большим приоритетом, исходящей из нулевой вершины, расположен ниже фрагмента, соответствующего дуге с меньшим приоритетом, исходящей из этой вершины. Так как в этой П команды выдачи [redacted] значений выходных переменных входят только в "линейный" участок (не "защищены" командой if), то команды формирования их нулевых значений не применяются, а получение единичных значений выполняется командами EQ 0 z_i.

Если изоморфизм обеспечивать не по вершинам ГП, а по его дугам, то фрагмент программы, соответствующий дуге с большим приоритетом, размещается выше фрагмента, соответствующего дуге с меньшим приоритетом:

READ	S	0	STR	S	2
			AND	NM	x_1
STR	S	0	STEP	S	3
AND	I	x_0			
EQ	SM	x_1	STR	S	3
STEP	S	2	EQ	0	z_1
STR	S	0	NEXT	S3	D
AND	M	x_1			
STEP	S	1	STR	S	4
			STEP	S	0
STR	S	1			
EQ	0	z_2	STOP		
NEXT	S1	D			

13. Сравнение предлагаемого подхода с методом построения структурированных граф-схем, предложенным Ашкрофтом и Манной

Универсальный метод построения структурированных ГСА (СГСА) был предложен Ашкрофтом и Манной [13, 14], который в терминологии [1] по неструктурированной ГСА1 или ГСА2 строит структурированную ГСА4 или ГСА5 с многозначным кодированием состояний.

Однако, по мнению автора, этот метод обладает рядом недостатков, не позволяющих получать хорошо понимаемые ГСА:

в явном виде не используется понятие "состояние";

не делается различие по типам A и типам используемых переменных;

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

из-за связи фрагментов СГСА по данным визуально весьма сложно обнаружить генерирующие "контуры";

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

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

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

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

предложенный метод ориентирован на использование языков высокого уровня.

Так как указанный метод предназначен для построения СГСА, то из его содержания следует, что если исходная ГСА уже структурирована, то к ней этот метод применяться не должен. По этой причине, если, например, в качестве исходной задана СГСА2 (рис. 3 [1]), то исходя из изложенного она не должна подвергаться дальнейшим преобразованиям. Однако, как показано в [1], эта ГСА "плохо" понимается и вместо нее для целей общения целесообразно использовать граф-схему алгоритма (рис. 10) или ГП (рис. 11).

Таким образом, в целом идея, предложенная Ашкрофтом и Манной, порочна, так как незачем сначала строить неструктурированную ГСА, чтобы потом ее структурировать. Следует либо сразу строить СГСА с дешифратором состояний для принятого варианта кодирования, либо, что более целесообразно, производить исходное описание в виде ГП для выбранного типа A, в котором, по-возможности, должны отсутствовать умалчиваемые значения переменных и который изоморфно реализуется, например, с помощью УК switch языка СИ, производящей одновременно расцикливание и структурирование и обеспечивающей доступ к любому значению многозначной переменной, соответствующей состояниям A. Подход применим и для языков низкого уровня, например мнемокодов ПЛК.

Введение понятия "состояние" делает каждую компоненту П "наблюдаемой" изнутри (с помощью одной многозначной переменной), а не только по входам и выходам. В получаемые П легко вносятся изменения. Они достаточно просто проверяются (в качестве теста используется ГП) и ввиду изоморфизма с первичным описанием легко верифицируются.

ГП по форме изображения являются в общем случае "существенно плоскостными", что позволяет отражать АУ в более естественной форме, чем в виде граф-схемы или диаграммы Графсет [9], которые по стандартам обычно изображаются в форме, приближающейся к "линейной", в направлении сверху вниз. Такая форма представления соответствует последовательному книжному изображению и чтению текстов, но не соответствует плоскостному (параллельному) изображению картин, более удобных для восприятия человеком. При этом естественно, что ГП должны быть в максимальной степени планарными. ГП являются более ранней "сущностью" теории алгоритмизации (теории А (ТА)) по сравнению с другими, рассмотренными в настоящей работе, и поэтому в соответствии с принципом Оккама [4], по которому "не следует размножать сущности без необходимости", можно утверждать, что для многих задач логического управления такая необходимость не наступает.

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

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

Проблемы, возникающие для задач большой размерности при использовании ТП как основной модели, для которой разработаны алгоритмы ТА при двоичной аппаратной, и в особенности асинхронной реализации, видимо, явились тем сдерживающим фактором, который по традиции распространился и на диаграммы переходов (основное название ГП в ТА), являющиеся в ТА вспомогательной (иллюстрационной) моделью, и не позволил до настоящего времени широко применять ГП в качестве языка общения и спецификации при ПР задач логического управления, при которой традиционные задачи ТА либо отсутствуют, либо не являются определяющими.

По мнению автора, при современном уровне развития элементной базы при ПР практических задач логического управления основным критерием построения П должна являться их хорошая читаемость и, как следствие, безошибочность, а остальные критерии должны быть вспомогательными (хотя в ПЛК их ограниченные ресурсы могут определять выбор моделей в процессе алгоритмизации). Поэтому автор надеется, что настоящая работа позволит ГП занять подобающее им место при ПР рассматриваемого класса задач.

Предлагаемый подход может быть использован не только для логико-временных АУ, но и для логико-вычислительных алгоритмов. На рис. 27 в качестве примера приведена ГСА1, реализующая алгоритм определения наибольшего общего делителя двух целых положительных чисел M и N (алгоритм Евклида). Эта ГСА является не автоматной, а логико-вычислительной. Она может программироваться с помощью подхода предлагаемого в настоящей работе по ГП, описывающему логико-вычислительный процесс и являющемуся АМИ с флагами (рис. 28). Необходимо отметить, что этот ГП существенно компактнее, чем соответствующая ГСА.

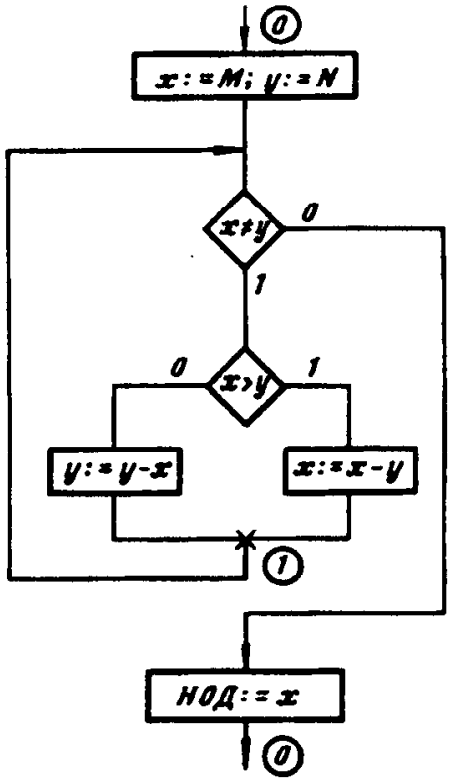

Рис. 27

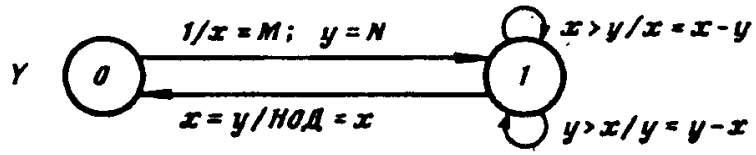

Рис. 28

Предлагаемый подход, по мнению автора, может представлять интерес для специалистов по объектно-ориентированному программированию в качестве математического аппарата для работы с "состояниями", введенными для описания "объектов".

14. Заключение

Изложенный подход, названный SWITCH-технологией алгоритмизации и программирования задач логического управления, является одним из направлений широко развиваемой в настоящее время CASE-технологии [15] (Computer Aided Software Engineering) (интересное совпадение с меткой case в операторе switch). Этой технологии "практически нет альтернативы, так как без специальной методики и инструментария составить техническое задание на систему, адекватно описывающее все нюансы ее поведения, практически невозможно" [16]. Это особенно важно, если учитывать один из законов Мэрфи, по которому, "если что-то кажется простым, то обычно является сложным, а если кажется сложным, то может оказаться и вовсе невыполнимым" [4].

Подход обеспечивает возможность всем участникам разработки не только контролировать поведение "черного ящика", содержащего П, как это делается обычно, но и позволяет "влезать" в него – наблюдать внутренние состояния каждой компоненты, следя только за одной ее переменной, и выполнять, при необходимости, проверку текстов П, реализующих функциональные задачи. Для обеспечения этой возможности под руководством автора разработана программная оболочка для отладки систем УА большой размерности, заданных СВ ГП и описанных на языке

СИ, а Кузнецовым Б. П. (НПО "Аврора") совместно с автором реализован транслятор "СИ – язык инструкций", учитывающий изложенные выше особенности ПР автоматов в ПЛК, описанных в [12].

Предложенная оболочка позволяет моделировать не только АУ, но и комплекс "АУ – объект управления (ОУ)", в котором компоненты модели ОУ (например, клапаны) также описываются с помощью ГП. При этом оболочка обеспечивает возможность изменения значений входных переменных и наблюдения получающихся при этом значений всех выходных и временных переменных, а также значений номеров состояний каждой компоненты комплекса как в пошаговом (один программный цикл), так и автоматическом (от одного устойчивого состояния комплекса до другого) режимах.

Подход позволяет разделить работу, а самое главное, ответственность между заказчиком (технологом), разработчиком и программистом в случае, когда они представляют разные организации, а тем более страны, так как в противном случае возникают существенные языковые, а в конечном счете, и экономические проблемы. Подход позволяет снять с программиста необходимость знания особенностей технологического процесса, а с разработчика – особенностей программирования. При этом появляется возможность общения между заказчиком, разработчиком, программистом и пользователем не традиционным путем в терминах технологического процесса (например, не "идет" режим экстренного пуска), а на промежуточном полностью формализованном языке (своего рода техническом эсперанто). Например, "в третьем ГП в пятой вершине на четвертой позиции изменить значение 0 на значение 1", что не вызывает разночтений, характерных даже для одного естественного языка, как, например, это имеет место в такой фразе, как "косой пошел с косой" [17], и не требует привлечения специалистов, знающих технологический процесс, для корректного внесения изменений. Более того, если для разработчика программирование является "открытым", что, правда, не всегда имеет место в особенности при работе с некоторыми зарубежными фирмами, то у него появляется возможность и вовсе отказаться от услуг функционального программиста и перейти к автопрограммированию.

При этом отметим, что при применении ГСА в большинстве случаев переход от алгоритмизации к программированию для задач логического управления сложными технологическими процессами представляет большую проблему. Это объясняется тем, что обычно процесс алгоритмизации почти никогда не завершается тем, чем положено, – созданием алгоритма в математическом смысле, который, по определению, должен однозначно выполняться любым вычислителем, а оканчивается некоторой "картинкой", называемой алгоритмом, которую в той или иной степени приходится додумывать при программировании. В этой ситуации либо разработчик должен сам программировать, либо программист должен знать все особенности технологического процесса, либо они вместе должны устранять неминуемые ошибки традиционного проектирования программ при испытаниях.

При использовании подхода, предлагаемого в настоящей работе, алгоритмизация должна происходить и завершаться на другом уровне – в процессе взаимодействия заказчика и разработчика. При этом выдача технического задания превращается из однократного события с последующими дополнениями в процесс, завершающийся созданием СВ ГП, в котором учтены все детали с точностью до каждого состояния, перехода и бита. При этом программист для функциональных задач ничего не должен додумывать, а только однозначно реализовать эту систему ГП, что позволяет резко снизить требования к его квалификации.

Подход в настоящее время успешно апробирован при создании НПО "Аврора" совместно с фирмой NORCONTROL (Норвегия) системы логического управления судовым дизель-генератором [18] и при создании системы управления дизель-генератором того же типа, построенной на основе аппаратуры "Selma-2" фирмы ABB

STROMBERG (Финляндия) [19]. При этом в первом случае программирование выполнялось на языке высокого уровня PL/M, а во втором – на специализированном языке функциональных блоков.

В последнем случае автором совместно с Кондратьевым В. Н. (НПО "Аврора") было предложено использовать только такие функциональные блоки библиотеки (цифровые и логические мультиплексоры), которые обеспечивают изоморфизм между согласованными с заказчиком графами переходов и получаемой функциональной схемой. Этот подход принципиально отличается от традиционного, при котором добиваются только функциональной эквивалентности ФС и спецификации (если последняя имеется), но не обеспечивается их изобразительная эквивалентность, позволяющая проводить верификацию сверкой изображений. Функциональная схема при традиционном подходе реализует заданное поведение, но описывает его в форме, не отображающей динамику переходов из состояния в состояние и динамику изменения значений выходных переменных A.

Если спецификация программно реализуемого АУ выполняется с помощью ФС, построенной на триггерах и логических элементах, охваченных обратной связью (ОС), то несмотря на то, что ФС, в отличие от исходного задания, полностью определена (неполнота, если она имеется, снимается в ходе построения схемы) и не содержит умолчаний (за исключением, быть может, не указания типов применяемых триггеров) и поэтому, так же как и ГП, может формально и независимо программироваться, ее весьма трудно понимать. Это объясняется тем, что, во-первых, она, в отличие от ГП, не обладает свойством локальности описания (если схема имеет несколько входных переменных, то без глубокого ее анализа не ясно, какие из этих переменных влияют на переход из рассматриваемого состояния в следующее), во-вторых, при взаимосвязанной реализации не обладает свойством локальности по внесению изменений и, что самое главное, не содержит предварительно определенных значений номеров состояний и выходных переменных. Поэтому ее чтение сводится к логическим вычислениям по схеме, включая запоминание "в голове" значений промежуточных переменных в триггерах. Анализ ФС (проверка непротиворечивости, отсутствие генерации, последовательность появления значений выходных переменных и т.д.) обычно осуществляется тестированием – подачей входных воздействий и вычислением выходных реакций. При этом необходимо отметить, что определение тестов в этом случае является большой проблемой, так как наличие триггеров и ОС резко затрудняет выбор тестов для определения всех функциональных возможностей схемы. Поэтому вместо тестирования более целесообразно выполнить верификацию – по ФС формально записать СБФ, а по ней также формально построить ГП, поведение которого и следует анализировать.

Если поведение ГП отличается от желаемого, то изменяется не исходная ФС, а строится корректированный ГП, который, в свою очередь, может быть реализован с помощью различных алгоритмических моделей, в том числе и отличных от ФС. При использовании последней по новому ГП формально строится новая СБФ, которая формально реализуется ФС.

При этом необходимо отметить, что если при аппаратной (в особенности асинхронной) реализации поведение реальной схемы может отличаться от поведения, задаваемого моделью, то при ПР это ограничение устраняется весьма просто, например при использовании СБФ эта проблема решается переобозначением внутренних переменных.

Аналогичный подход может быть использован при применении в качестве языка программирования релейно-контактных (лестничных) схем.

Предложенный подход позволяет обеспечить программирование с единых позиций в базисе различных языков, входящих в состав программного обеспечения современных ПЛК, таких, как, например, ПЛК [20], для которых программирование может выполняться, во-первых, с помощью языков, рекомендуемых международным

стандартом IEC 1131: последовательных функциональных диаграмм (Sequential Function Chart – SFC), функциональных блоков (Function Blok Diagram – FBD), лестничных схем (Ladder Diagram – LD), инструкций (Instruction List – IL), а во-вторых, с помощью таких языков как Ассемблер и СИ.

Изложенный подход идеологически близок к использованному при разработке в Институте проблем управления (Москва) под руководством Кузнецова О. П. языка "Ярус" [21] и его модификаций [22] и может рассматриваться как его осмысление и дальнейшее развитие [23]. Подход соответствует основным тенденциям, развиваемым в настоящее время для построения автоматизированных систем управления сложными энергетическими системами [24].

СПИСОК ЛИТЕРАТУРЫ

1. *Шалыто А. А.* Использование граф-схем алгоритмов и графов переходов при программной реализации задач логического управления. I // АиТ. 1996. № 6. С. 148–158.
2. *Баранов С. И.* Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.
3. *Шалыто А. А.* Реализация алгоритмов судовых управляющих логических систем при использовании микропроцессорной техники. Л.: Ин-т повышения квалификации руководящих работников и специалистов судостроительной промышленности, 1988.
4. *Шалыто А. А.* Программная реализация алгоритмов логического управления судовыми системами. Л.: Ин-т повышения квалификации руководящих работников и специалистов судостроительной промышленности, 1989.
5. *Шалыто А. А.* Программная реализация управляющих автоматов // Судостроительная промышленность. Сер. Автоматика и телемеханика. 1991. Вып. 13. С. 41–42.
6. *Шалыто А. А.* Технология программной реализации алгоритмов логического управления как средство повышения живучести // Тез. докл. научно-технической конференции "Проблемы обеспечения живучести кораблей и судов". С.-Петербург: Судостроение, 1992. С. 87–89.
7. *Руднев В. В.* Системы взаимосвязанных графов и алгоритмическое программирование дискретных управляющих устройств // АиТ. 1979. № 7. С. 110–121.
8. *Рубинов В. И., Шалыто А. А.* Построение граф-схем бинарных программ для систем булевых функций, заданных таблицами истинности // Автоматика и вычислительная техника. 1988. № 1. С. 87–92.
9. *Мишель Ж.* Программируемые контроллеры. Архитектура и применение. М.: Машиностроение, 1992.
10. *Джамп Д.* AUTOCAD. Программирование. М.: Радио и связь, 1992.
11. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний. М.: Наука, 1970.
12. Autolog 32. Руководство пользователя. FF–Elektroniikka Fredriksson Ky, 1990.
13. *Иодан Э.* Структурное проектирование и конструирование программ. М.: Мир, 1979.
14. *Лингер Р., Миллс Х., Уитт С.* Теория и практика структурного программирования. М.: Мир, 1982.
15. *Schmalhofer F., Thoben J.* A model-based construction of a CASE-oriented expert system // The European Journal on Artificial Intelligence. 1992. V. 5. № 1. P. 38–45.
16. CASE-технология в России: настоящее или будущее? // Computer world Moscow. 1992. № 40–41. С. 8–9.
17. *Криницкий Н. А.* Алгоритмы вокруг нас. М.: Наука, 1984.
18. Functional Description. Warm-up & prelubrication logic. Generator Control Unit. Severnaya hull no 431. NORCONTROL, 1993.
19. SELMA 2. Диспетчерская система. Инструкция по программированию и эксплуатации. RU 5351265-8C. ABB ASEA BROWN BOVERI. STROMBERG.

20. *Гельфанд А. М., Шумилов В. Н., Аблин И. Д. и др.* Многофункциональный комплекс программно-аппаратных средств для построения распределенных систем управления – МФК "Техноконт" // Приборы и системы управления. 1994. № 1. С. 2–9.
21. *Кузнецов О. П., Макаревский А. Я., Марковский А. В. и др.* ЯРУС – язык описания работы сложных автоматов // АиТ. 1972. № 6. С. 80–89, № 7. С. 72–82.
22. *Кузнецов О. П., Шипилина Л. Б., Марковский А. В. и др.* Проблемы разработки языков логического программирования и их реализация на микро-ЭВМ (на примере языка ЯРУС-2) // АиТ. 1985. № 6. С. 128–138.
23. *Shalyto A. A.* Cognitive Properties of Hierarchical Representations of Complex Logical Structures // Proceedings of the 1995 ISIC (International Symposium on Intelligent Control). Workshop. Monterey, California, 1995.
24. *Прангишвили И. В., Амбарцумян А. А.* Научные основы построения АСУ ТП сложных энергетических систем. М.: Наука, 1992.

Поступила в редакцию 10.03.95

Российская академия наук

Автоматика и телемеханика

Журнал основан в 1936 году
Выходит 12 раз в год

7
июль
·Наука· Москва
1996