С автором можно связаться по адресу: aurora@peterlink.ru, "для Шалыто".

Статья опубликована журнале "Автоматика и телемеханика", 1996. N6, С.148-158; N7, С.144-169.

Имеется перевод: Shalyto A.A. Algorithmic graf schemes and transition graphs: their application in software realization of logical control algorithms. //Automation and Remote Control. 1996. N6, P.890-897; N7, P.1027-1045.

УДК 519.714

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

© 1996 А.А.Шалыто, д-р техн. наук

Федеральный научно-производственный центр
Государственное унитарное предприятие "НПО "Аврора""

Санкт-Петербургский институт точной механики и оптики
(технический университет)

В работе выполнен анализ недостатков, затрудняющих понимание граф-схем алгоритмов и программ, построенных на их основе. Приведена классификация граф-схем алгоритмов. Показано, что основные трудности их понимания возникают из-за умолчаний и не использования в них такого понятия как "состояние". Введение этого понятия позволяет переходить от граф-схем алгоритмов к графам переходов. Сформулированы требования к графам переходов, выполнение которых обеспечивает их "понятность".

1.ВВЕДЕНИЕ

В настоящее время при программной реализации алгоритмов логического управления технологическими процессами наряду с программируемыми логическими контроллерами все чаще используются компьютеры, предназначенные для работы в производственных условиях.

Однако вне зависимости от типа управляющего вычислительного устройства в силу большой важности задач этого класса (например, при управлении ядерными реакторами) необходимо, чтобы Заказчик, Разработчик, Программист, Оператор и Контролер однозначно и полностью понимали друг друга.

Назовем язык общения указанных сторон языком спецификаций [1]. Несмотря на наличие для задач логического управления большого числа таких языков [2-26] на практике, видимо, в связи с традициями в области вычислительной техники [5], наиболее широкое использование в этом качестве получили блок-схемы, называемые также граф-схемами или просто схемами [27, 28], что закреплено соответствующими стандартами [29, 30].

Однако в этих стандартах определяются лишь правила изображения граф-схем и отсутствуют требования (кроме

изобразительных) к их построению для обеспечения возможности легкого их понимания.

Необходимо отметить, что и в литературе недостаточно рассматривались свойства граф-схемам алгоритмов, упрощающие их применение в качестве языка общения для указанного класса задач. Так в [31, 32] были предложены методы построения граф-схем, структурная организация которых улучшает их понимание. При этом авторы этих работ считали, что если граф-схемы построены только из вложенных базовых управляющих конструкций без использования операторов go to, то это решает проблему их легкого понимания и не учитывали ряда других факторов, затрудняющих чтение, и, в частности, умолчания значений переменных.

Однако в последнее время специалисты по проектированию программ [33] обратили внимание на то, что только структурное проектирование указанной проблемы не решает и предложили объектно-ориентированный подход к проектированию программ, в рамках которого было введено понятия "объекта" и его "состояния" и рекомендовано использовать графы переходов для описания динамики реализуемых процессов. Однако, кроме одного примера использования графов переходов, эта работа не содержит теоретического обоснования требований к их построению, обеспечивающих, в частности, простоту понимания программ.

В настоящей работе разрабатываются требования к построению легко понимаемых граф-схем и графов переходов, а во второй ее части предлагаются методы построения понимаемых графов переходов по граф-схемам алгоритмов и понимаемых граф-схем по графам переходов, а также методы их программирования в базисе языков различных уровней.

2.ГРАФ-СХЕМЫ. ОСНОВНЫЕ ПРОБЛЕМЫ

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

При этом будем различать граф-схемы алгоритмов (ГСА) по следующим основным признакам:

- наличию внутренних обратных связей;
- используемым типам переменных;
- наличию дешифратора состояний;
- наличию умолчаний и неоднозначно задаваемых переменных;
- наличию переменных, которые могут неоднократно изменяться за один проход граф-схемы;
- месту выдачи значений выходных переменных.

Используя некоторые из этих признаков и не претендуя на полноту классификации, выделим пять подклассов граф-схем алгоритмов:

- граф-схемы алгоритмов с внутренними обратными связями и выдачей значений выходных переменных в любой операторной вершине, обозначаемые ГСА1;
- граф-схемы алгоритмов без внутренних обратных связей и дешифратора состояний, в которых выдача значений выходных переменных выполняется в конце "тела" граф-схемы, обозначаемые ГСА2;

- граф-схемы алгоритмов без внутренних обратных связей, учитывающие особенности используемых управляющих конструкций языка программирования, обозначаемые ГСА3;
- граф-схемы алгоритмов без внутренних обратных связей, но с дешифратором состояний и выдачей значений выходных переменных в конце "тела" граф-схемы, которые не содержат выходных переменных, неоднократно изменяющихся за один проход граф-схемы, обозначаемые ГСА4;
- граф-схемы алгоритмов без внутренних обратных связей, но с дешифратором состояний и выдачей значений выходных переменных в конце "тела" граф-схемы, содержащие выходные переменные, которые могут неоднократно изменяться за один проход граф-схемы, обозначаемые ГСА5.

При этом отметим, что внешняя обратная связь в управляющих алгоритмах и программах существует всегда (режим сканирования в программируемых логических контроллерах). Отметим также, что если в вычислительном устройстве реализуется алгоритм управления в виде одной компоненты (задачи), то ГСА1, используя ГСА3, могут при наличии соответствующих управляющих конструкций изоморфно отражаться в граф-схемы программ.

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

На рис.1 в качестве примера приведена схема связи управляющего автомата с объектом управления, состоящим из трех клапанов (Кл1, Кл2, Кл3) и двигателя (Д). При этом управляющий автомат декомпозирован на автомат (А) и функциональные элементы задержки (ФЭЗ).

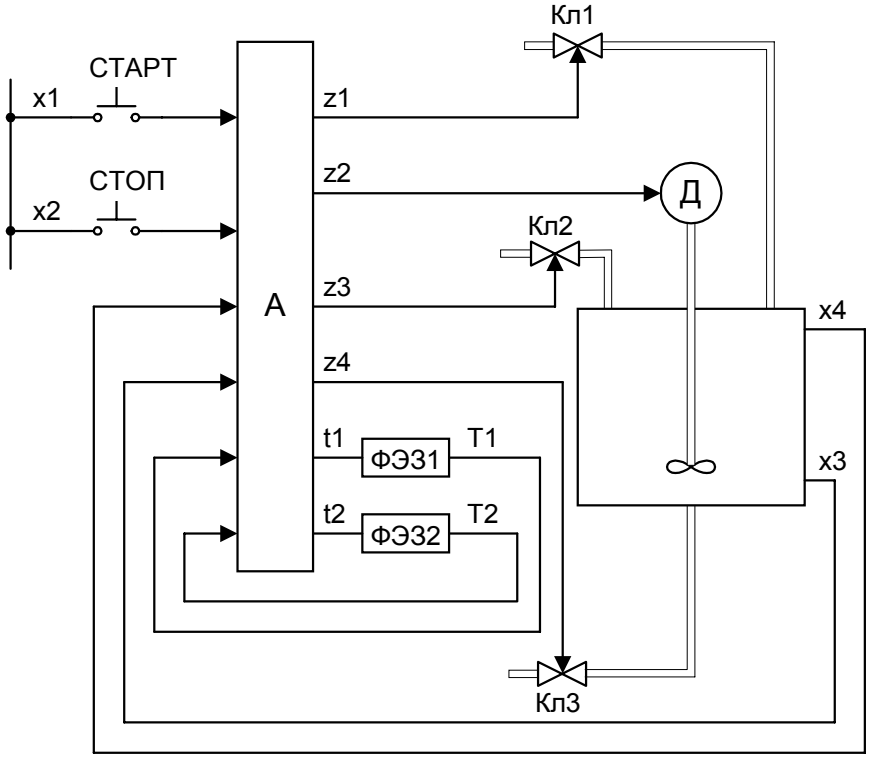

Рис. 1

На рис.2 для рассматриваемого примера приведена ГСА1 с пятью внутренними обратными связями [34].

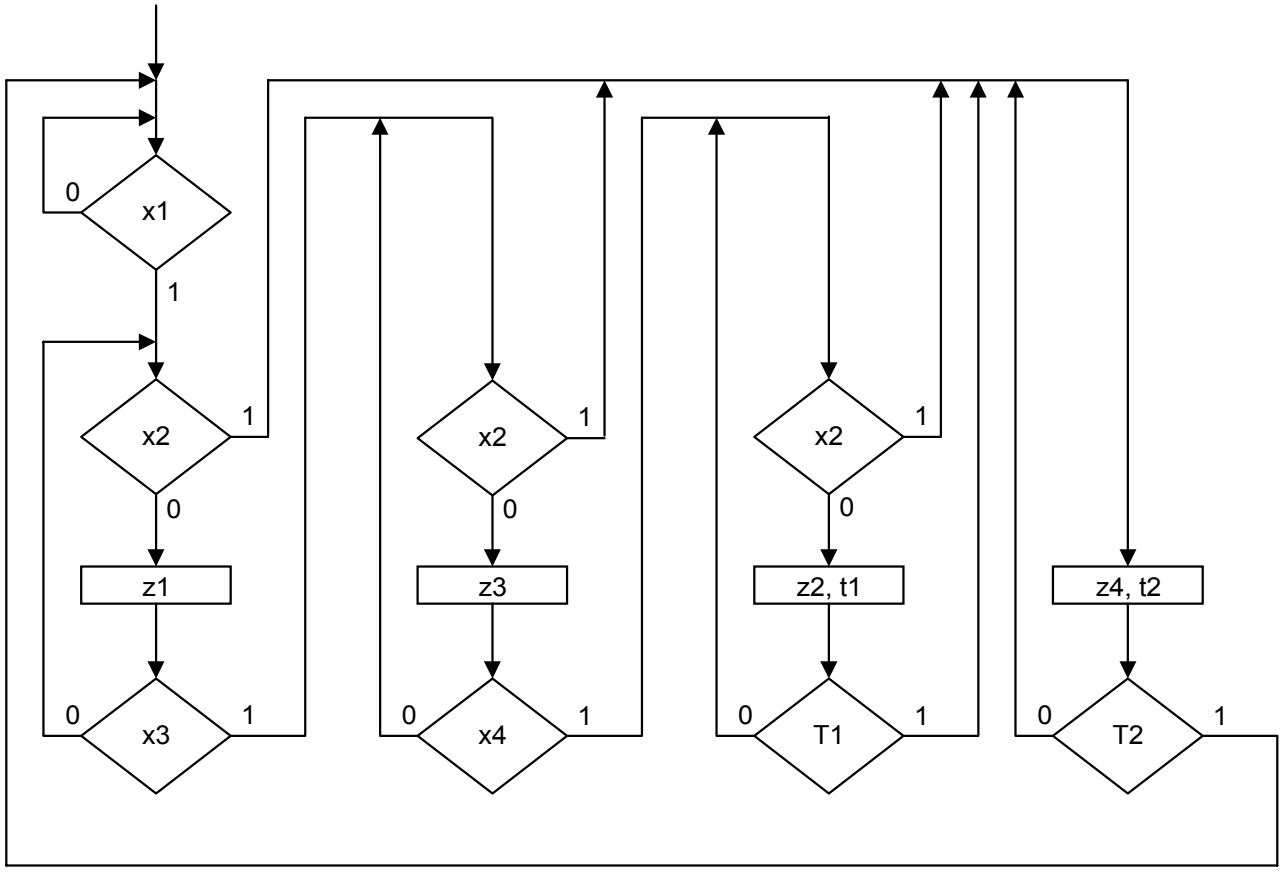

Рис. 2

Эта граф-схема реализует либо управляющий автомат в целом (ФЭЗ1 и ФЭЗ2 вычисляются в третьей и четвертой операторных вершинах), либо только автомат (временные переменные t1 и t2 рассматриваются как обращения к процедуре, реализующей функциональные элементы задержки, а двоичные переменные T1 и T2 сигнализируют о срабатывании элементов задержки).

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

Однако предположение о том, что если в операторных вершинах, в которых некоторая выходная или временная переменная отсутствует, то ее значение равно нулю, справедливо далеко не всегда, так как более часто при использовании граф-схем считают, что умалчиваемые переменные сохраняют предыдущее значение. Такие граф-схемы также могут быть использованы для вычислений, так как вычислительное устройство помнит предыдущие значения всех переменных, сохраняющиеся во внешней по отношению к граф-схемам памяти, но они весьма трудно понимаются человеком, которому сложно помнить предысторию, особенно по нескольким переменным одновременно. При этом отметим, что граф-схемы предназначены в основном для отображения связей по управлению и в существенно меньшей степени -

по данным [28, 35]. Таким образом, граф-схемы с умалчиваемыми значениями переменных нецелесообразно применять в качестве языка общения.

Поэтому качественной можно считать только такую спецификацию, в которой кроме связей по управлению в максимальной степени отражены также и данные [35]. Отсутствие в явном виде некоторых данных в граф-схемах не позволяет использовать ее также и в качестве теста для проверки программы, реализующей эту граф-схему. Более того, если программа строится по граф-схеме эвристически, то даже при отсутствии умолчаний с помощью граф-схемы можно проверить лишь то, что программа реализует граф-схему, но невозможно установить, что программа не делает ничего дополнительно.

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

ГСА1 могут быть построены так, что в условных вершинах применяются только входные переменные типов X и T, а в операторных вершинах — выходные переменные типов Z и t, где T — переменные, фиксирующие факт срабатывания функциональные элементы задержки. Наличие в ГСА1 только тех переменных, которые упоминаются в алгоритме управления, является важным достоинством этого подкласса граф-схем алгоритмов.

В ГСА2 крайне редко используются только те переменные, которые указаны в алгоритме управления и не применяются "лишние" переменные. В качестве такого примера на рис.3 приведена ГСА2, реализующая R-триггер, в которой используются только переменные, определенные алгоритмом управления: x1 — переменная установки триггера; x2 — переменная сброса триггера; z — выходная переменная.

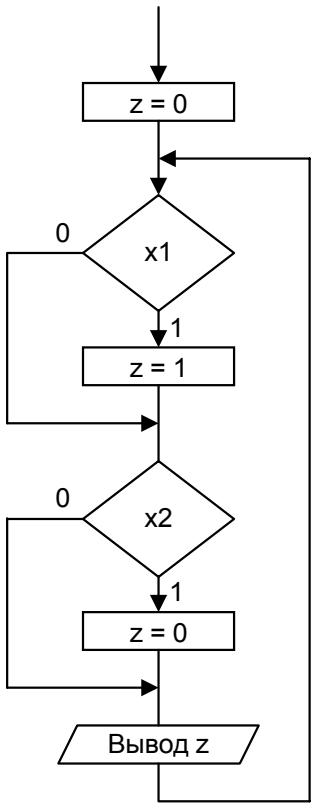

Рис. 3

Даже эта граф-схема алгоритма (без дополнительных переменных) весьма трудно понимается, так как выдача результатов в ней происходит в конце тела программы и поэтому при x1 = x2 = 1 значения z вычисляются дважды (пересчитываются), а кроме того в ней при x1 = x2 = 0 имеется "пустой" от операторных вершин путь, при прохождении которого сохраняется предыдущее значение z, что осуществляется за счет запоминания во внешней относительно граф-схемы алгоритма ячейке памяти значений z, указанных в операторных вершинах.

Из рассмотренного примера следует, что если в ГСА2 существует хотя бы один путь, при прохождении которого не устанавливается ни нулевое, ни единичное значение хотя бы одной выходной переменной, то такая граф-схема реализует последовательностный автомат и однотактный - в противном случае.

Как отмечалось выше, ГСА2 без "лишних" переменных встречаются крайне редко. В общем случае в этом подклассе граф-схем алгоритмов в условных вершинах наряду с входными переменными типов X и T, проверяются также и значения выходных переменных Z и отсутствующие в алгоритме управления промежуточные (внутренние) переменные Y, которые устанавливаются и сбрасываются наряду с выходными переменными в операторными переменными. В силу того, что обычно

используются битовые переменные Y, то ГСА2 в этом случае весьма громоздки. Кроме того они обычно неупорядочены по структуре, так как порядок расположения вершин и их пометка в граф-схемах алгоритмов стандартами не оговаривается. ГСА2 весьма трудно понимаются и по причине применения умолчаний сохраняющихся значений переменных Y и Z.

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

Необходимо отметить, что, если для Программиста проверки переменных Y и Z вполне естественны, а для Разработчика объяснимы, то для Заказчика, Оператора и Контролера их наличие неприемлемо, так как, например, Заказчик в техническом задании обычно не просит устанавливать какую-либо внутреннюю переменную в единицу. Особые сложности могут возникнуть при чтении в тех случаях, когда проверяемые переменные Y и Z одной ГСА2 могут изменяться из других компонент алгоритма управления, реализуемых в том же вычислительном устройстве.

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

Проблемы с построением ГСА1 и ГСА2 на изложенном не заканчиваются, так как для безошибочного перехода к граф-схеме программы и возможности проведения экспертизы (для ответственных объектов управления) желательно по исходной граф-схеме алгоритма построить ГСА3, учитывающую основные свойства управляющих конструкций используемого языка программирования. Например, в большинстве программируемых логических контроллеров для команд IF допустимы переходы только вперед и запрещены возвраты назад. Другая принципиальная особенность команд IF многих логических контроллеров состоит в том, что, во-первых, они одноадресные, а, во-вторых, обладают тем свойством, что если некоторая команда IF при невыполнении условия передает управление на команду (метку) CONT, то тогда и все размещаемые между этими командами другие команды IF при невыполнении условий также должны передавать управление на использованную метку CONT [34]. При этом необходимо отметить, что команда безусловного перехода STOP позволяет реализовать только внешнюю обратную связь.

В этих условиях ГСА3 должна быть линеаризованной и структурированной только с помощью управляющих конструкций "последовательное соединение" и "неполный выбор". В тех случаях, когда ГСА2 в исходном виде обладает такой структурой (например, рис.3), необходимость в построении ГСА3 отпадает. Однако, если, например, ГСА2, реализующая однотактный автомат, описываемый булевой формулой z = !x1x2 v x1!x2, имеет "плоскостную" (рис.4), а не "линейную" структуру, то для перехода к граф-схеме программы и тексту программы по ней целесообразно предварительно построить ГСА3 (рис.5).

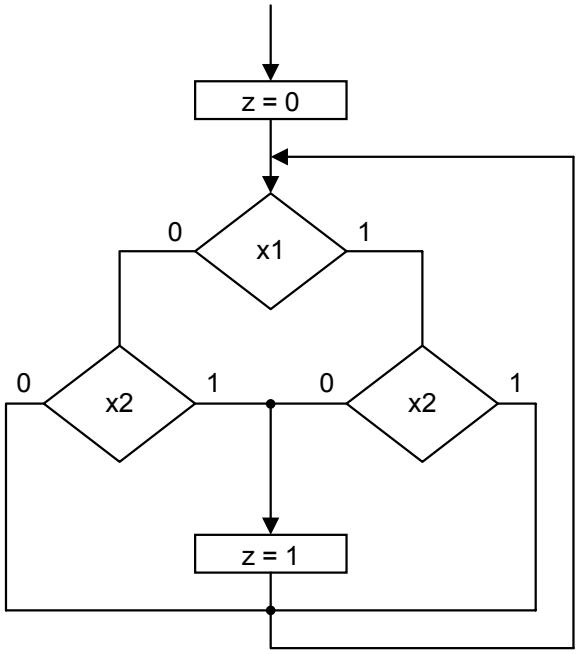

Рис. 4

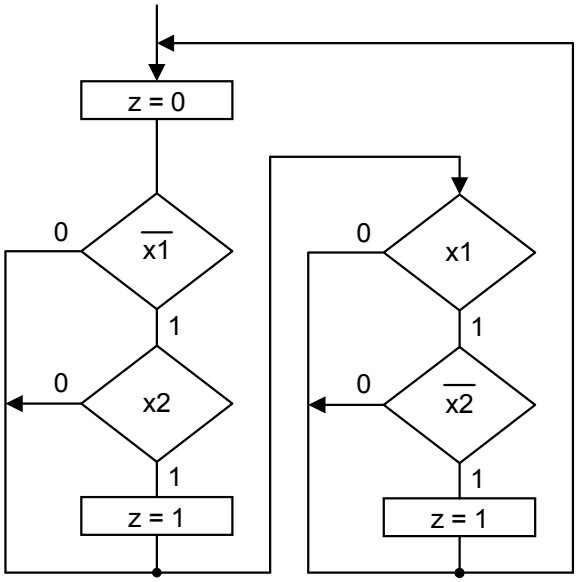

Рис. 5

Из рассмотрения последней граф-схемы следует, что сложность ее понимания по сравнению с ГСА2 (рис.4) увеличилась (даже несмотря на то, что в этих граф-схемах не используются проверки переменных Y и Z), так как при прохождении любого пути в ГСА3 входные переменные приходится проверять неоднократно. При этом отметим, что если программирование булевых формул проводить не в бинарной, а в операторной форме (непосредственно по формуле), то ГСА2 всегда будет иметь линейную структуру, что, правда, в общем случае приводит к снижению быстродействия и требует применения промежуточных переменных.

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

Таким образом, если в ГСА1 отражается только семантика реализуемого алгоритма управления, в ГСА2 кроме семантики учитывается возможность реализации алгоритма управления, состоящего из нескольких компонент, в одном вычислительном устройстве, то в ГСА3 отражается также специфика используемых управляющих конструкций применяемого языка программирования. Однако ГСА3 еще значительно отличаются от граф-схем программ, так как в последних должна отражаться также и семантика всех используемых команд или операторов применяемого языка программирования. При этом в граф-схемах программ появляются упоминания о таких архитектурных особенностях вычислительных устройств, которые в исходной граф-схеме алгоритма не используются. Например, на рис.6 приведена граф-схема программы,

реализующая ГСА2 (рис.3), в которой символом "БС" обозначен битовый сумматор программируемого логического контроллера.

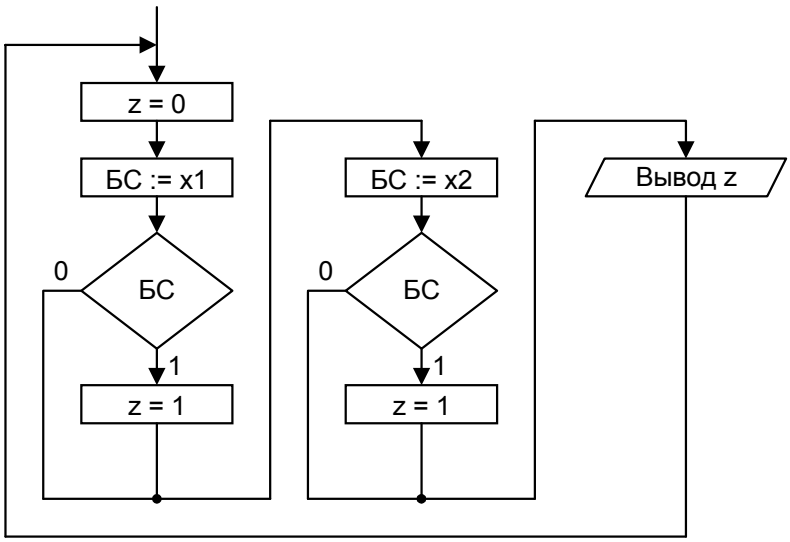

Рис. 6

Из изложенного следует, что граф-схемы программ, а тем более тексты программ не должны применяться в качестве языка общения.

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

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

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

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

3.ГРАФЫ ПЕРЕХОДОВ. КЛАССИФИКАЦИЯ

Определение 1. Граф переходов, в котором в каждой вершине явно указаны значения каждой выходной переменной, назовем графом переходов автомата Мура с явным заданием значений всех выходных переменных.

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

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

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

Графы переходов этого типа читаются несколько хуже, чем графы переходов автоматов Мура первого типа, однако их целесообразно использовать для сокращения объема памяти программ, изоморфно отражающих графы переходов автоматов Мура. Графы переходов этого класса могут быть изображены таким образом, что в нем в каждой вершине в явном виде указаны значения каждой выходной переменной, а неизменяющиеся относительно "смежных" вершин значения этих переменных отмечаются перечеркиванием.

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

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

Это преимущество с точки зрения компактности описания и сокращения длины программы порождает одновременно и главный недостаток графов переходов этого типа — трудность их чтения и понимания. Именно по этой причине такие графы переходов не рекомендуется использовать в качестве языка общения.

Спецификацию задач рассматриваемого класса целесообразно производить с помощью двух первых моделей графов переходов автоматов Мура, а в случае использования для спецификации ГСА1 последнюю целесообразно преобразовать к одной из этих моделей графов переходов.

Это, естественно, не исключает применение графов переходов для других классов автоматов. Аналогичная классификация может быть предложена для графов переходов автоматов Мили, в которых значения выходных переменных формируются не в вершинах графов переходов, а

на дугах [36]. Во многих задачах логического управления переход от модели автомата Мура к модели автомата Мили не уменьшает числа состояний (вершин графа переходов). На рис.7 в качестве примера приведен граф переходов автомата Мура, реализующий счетный триггер, а на рис.8 эта же задача описана с помощью графа переходов автомата Мили.

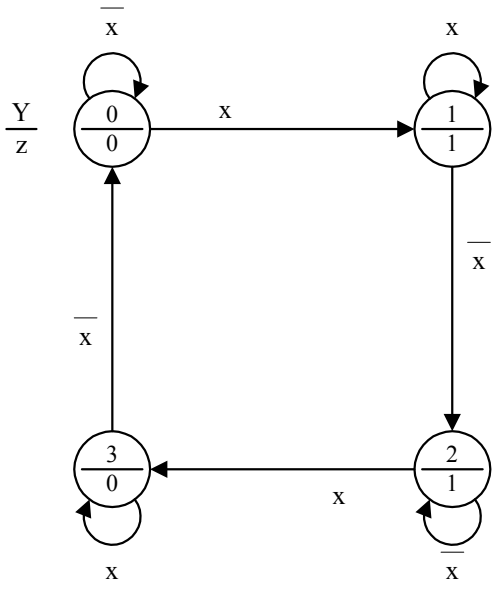

Рис. 7

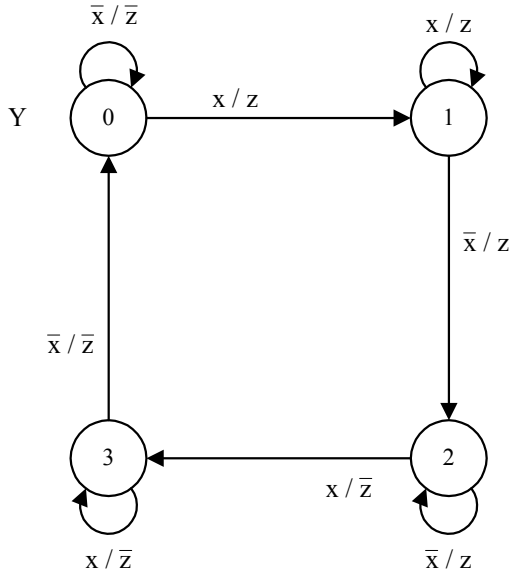

Рис. 8

В других случаях переход от графа переходов автомата Мура или графа переходов автомата без выходного преобразователя к графу переходов автомата Мили позволяет сократить число вершин. На рис.9 приведен граф переходов автомата без выходного преобразователя с четырьмя вершинами, реализующий последовательностный одноразрядный сумматор, а на рис.10 этот алгоритм описан графом переходов автомата Мили с двумя вершинами.

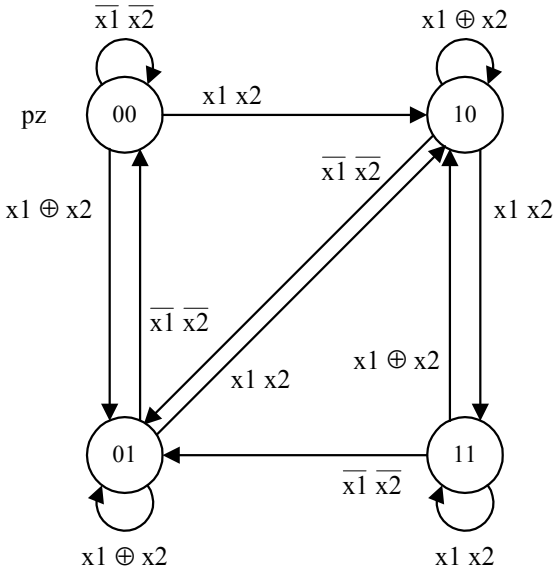

Рис. 9

При этом необходимо отметить, что если для автоматов Мура и автоматов без выходного преобразователя с однозначным заданием значений выходов число состояний равно числу комбинаций этих значений (среди которых учитываются повторяющиеся), а для автоматов, в которых все комбинации различны, равно числу различных комбинаций, то уже для автоматов этих классов с неоднозначным их заданием понятие "состояние" становится менее связанным со значениями выходов и поэтому становится более абстрактным и менее понятным.

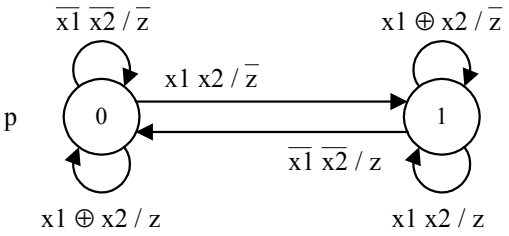

Рис. 10

Это положение усугубляется для автоматов Мили, а для автоматов этого класса с неоднозначными значениями выходов в [37] вместо понятия "состояние" было введено понятие "ситуация", а вместо термина "граф переходов" — термин "граф переключений". Однако несмотря на то, что граф переключений может содержать меньшее число вершин, чем эквивалентный граф переходов, применение графов переключений в качестве языка общения нецелесообразно ввиду сложности их понимания.

Аналогичная ситуация с числом состояний из-за умолчаний значений переменных может иметь место и для такого типа автоматов как смешанные автоматы — "автоматы без выходного преобразователя-Мили (СА1)" и "автоматы Мура-Мили (СА2)", а также для автоматов всех перечисленных классов с флагами, в которых одна и та же переменная может использоваться в одном графе переходов как в качестве входной, так и выходной переменной.

В качестве флага в ряде случаев могут использоваться переменные, если они находятся в ячейках памяти, значения которых проверяются автоматом и могут быть изменены не только внешним относительно рассматриваемого автомата источником информации, например кнопкой, но и собственно автоматом. В графе переходов СА2 (рис.11) в качестве флага используется переменная x1, значения которой, сохраняемые во внешней относительно автомата битовой ячейке памяти, могут быть изменены в устойчивых состояниях автомата кнопкой или собственно автоматом на переходе 0-2, инициируемом входной переменной x0. Значения переменной x1 не только формируются автоматом, но и проверяются им на переходах 0-1 и 2-3.

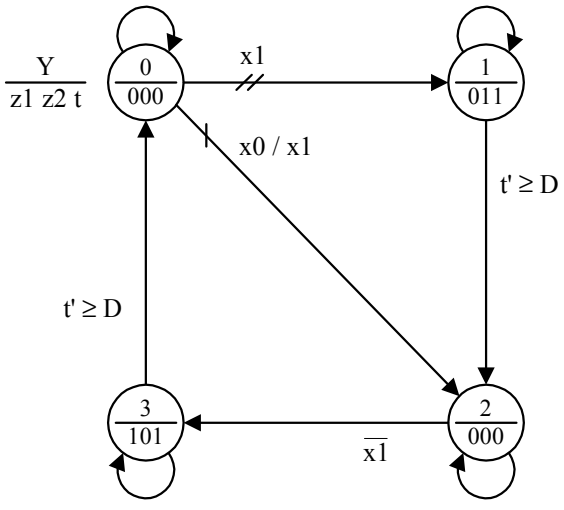

Рис. 11

Более типично использование флагов как дополнительно введенных переменных, которые воздействуют на ячейки памяти,

содержимое которых не может быть изменено от других источников информации. В случае, когда флаги вводятся в автомат без выходного преобразователя, то эти ячейки могут рассматриваться как принадлежность автомата наряду с ячейками, сохраняющими значения переменных, различающих состояния.

Для автоматов Мура, Мили и смешанных автоматов флагами являются дополнительно вводимые переменные F, значения которых запоминаются в ячейках памяти, внешних относительно рассматриваемого автомата. Если при использовании многозначного кодирования состояний автомата любой сложности, принадлежащих этим классам, достаточно иметь в автомате только **одну** промежуточную переменную, то при введении флагов во внешней относительно автомата среде появляются дополнительные ячейки памяти, в том числе и многозначные. При этом номер следующего состояния определяется не только номером рассматриваемого состояния и входным воздействием, как это имеет место в автоматах без флагов, но зависит от предыстории.

Таким образом, для таких автоматов не только значения выходов, но и значения состояний могут зависеть и изменяться от предыстории, что, естественно, резко усложняет их чтение.

На рис.12 приведен не содержащий ни одной устойчивой вершины граф переходов автомата Мура с десятью вершинами. Число вершин в графе переходов автомата Мура удается сократить до семи за счет введения двоичного флага F1, формируемого и проверяемого автоматом, и неоднозначности значений флага в некоторых вершинах графа переходов (рис.13). Дальнейшее сокращение числа вершин в графе переходов автомата Мура обеспечивается введением дополнительного многозначного флага F2 и использованием умолчаний его значений (рис.14).

Методы построения понимаемых графов переходов по граф-схемам алгоритмов и понимаемых граф-схем алгоритмов по графам переходов, а также методы программирования рассматриваемых моделей в базисе языков различных уровней рассматриваются во второй части работы.

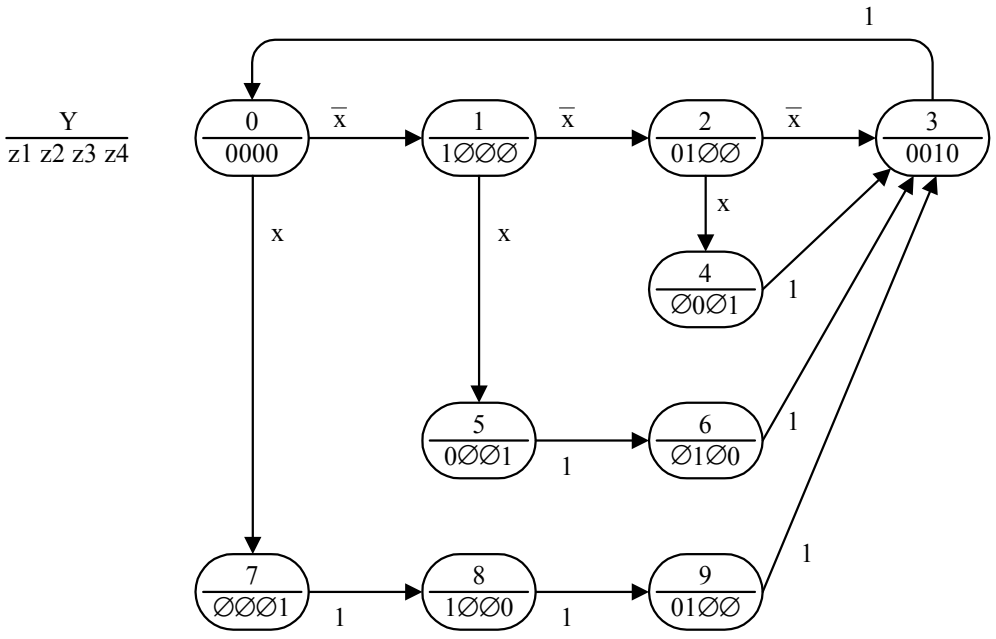

Рис. 12

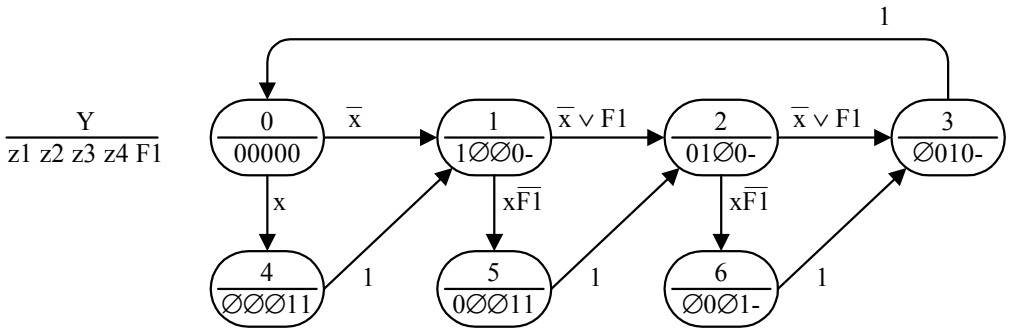

Рис. 13

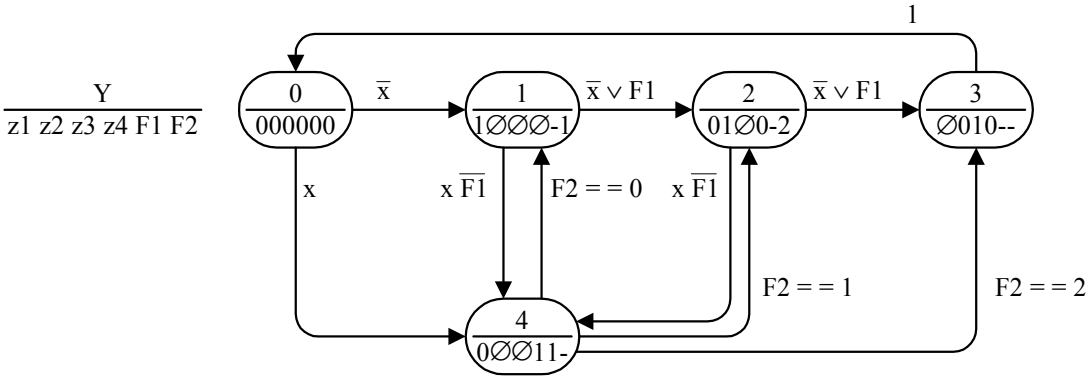

Рис. 14.

СПИСОК ЛИТЕРАТУРЫ

1. Требования к спецификации программ /Под ред. Агафонова В.Н. М.: Мир, 1984.
2. Клини С. Представление событий в нервных сетях и конечных автоматах //Автоматы. М.: Изд-во иностр. лит, 1956. С.17-27.
3. Ляпунов А.А. О логических схемах программ //Проблемы кибернетики. М.: Физматгиз, 1958. Вып.1. С.5-28.
4. Янов Ю.И. О логических схемах алгоритмов //Проблемы кибернетики. М.: Физматгиз, 1958. Вып.1. С.29-53.
5. Калужнин Л.А. Об алгоритмизации математических задач //Проблемы кибернетики. М.: Физматгиз, 1958. Вып.1. С.58-63.
6. Таль А.А. Анкетный язык и абстрактный синтез минимальных последовательностных машин //Автоматика и телемеханика (А и Т). 1964. N 6. С.38-49.
7. Гаврилов М.А., Девятков В.В., Чичковский А.Б. Язык операторных схем параллельных алгоритмов с памятью (язык ОСПАП) //Абстрактная и структурная теория релейных устройств. М.: Наука, 1975. С.18-30.
8. Глушков В.М., Капитонова Ю.В., Летичевский А.А. О применении метода формализованных технических заданий к проектированию программ обработки структур данных //Программирование. 1978. N6. С.5-12.
9. Кузнецов О.П. Теория алгоритмических конечноавтоматных языков //А и Т. 1981. N 3. С.122-133. N4. С.127-135.
10. 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) //А и Т. 1985. N 6. С.128-138.

11. Девятков В.В., Чичковский А.Б. Условие-82 - язык программно-логического управления //Автоматизация проектирования. М.: Машиностроение, 1990. Вып.2. С.58-67.
12. Амбарцумян А.А., Искра С.А., Кривандина Н.Ю. и др. Проблемно-ориентированный язык описания поведения систем логического управления ФОРУМ-М //Проектирование устройств логического управления. М.: Наука, 1984. С.35-47.
13. Юдицкий С.А., Покалев С.С. Логическое управление гибким интегрированным производством. Препринт. М.: Ин-т проблем управления, 1989.
14. Закревский А.Д. Языки логического управления. Препринт. Минск: Ин-т технической кибернетики, 1988.
15. Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов. М.: Энергоатомиздат, 1989.
16. Баранов С.И. Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.
17. Флорин Ж. Таблицы этапов или сети Петри? //Теория дискретных управляющих устройств. М.: Наука, 1982. С.35-42.
18. Захаров В.Н. Секвенциальное описание управляющих автоматов //Изв. АН СССР. Техн.кибернетика. 1972. N2. С.58-65.
19. Фрайтаг Г.,Годе В., Якоби Х. и др. Введение в технику работы с таблицами решений. М.: Энергия, 1979.
20. Базиль Д. Реализация конечного автомата на языке Форт //Форт в исследованиях и разработках. Л.: Ленинградский гос. ун-т, 1991. Т.1. N1. С.5-8.
21. Сосье Г. Управляющие автоматы: моделирование, декомпозиция и реализация. //Теория дискретных управляющих устройств. М.: Наука. 1982. С.49-58.
22. Бардзинь Я.М., Калниньш А.А., Стродс Ю.Ф. и др. Язык спецификации SDL и методика его использования. Рига: Латвийский гос. ун-т. 1986.
23. Бергер Г. Программирование управляющих устройств на языке STEP 5. SIEMENS, 1982.
24. Мишель Ж. Программируемые контроллеры. Архитектура и применение. М.: Машиностроение, 1992.
25. Шалыто А.А. Программная реализация управляющих автоматов //Судостроительная промышленность. Сер. Автоматика и телемеханика. 1991. Вып.13. С.41-42.
26. Шалыто А.А. Технология программной реализации алгоритмов логического управления как средство повышения живучести //Тез. докл. научно-технической конф."Проблемы обеспечения живучести кораблей и судов". СПб.: Судостроение, 1992. С.87-89.
27. Котов В.Е., Сабельфельд В.К. Теория схем программ. М.: Наука, 1991.
28. Ершов А.П. Введение в теоретическое программирование. М.: Наука, 1977.
29. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. ГОСТ 19.701-90 (ИСО 5807-85).
30. Р - схемы алгоритмов и программ. ГОСТ 19.005 - 85.
31. Иодан Э. Структурное проектирование и конструирование программ. М.: Мир, 1979.
32. Лингер Р., Миллс Х., Уитт С. Теория и практика структурного программирования. М.: Мир, 1982.
33. Буч Г. Объектно - ориентированное проектирование с примерами применения. Киев: Диалектика, 1992.
34. Al 2000. Technical Description. FF-ELEKTRONIIKKA FREDRIKSSON KY, 1992.
35. Вирт Н. Программы = алгоритмы + данные. М.: Мир, 1988.
36. Брауэр В. Введение в теорию конечных автоматов. М.: Радио и связь, 1987.
37. Кузнецов О.П. Графы логических автоматов и их преобразования. //А и Т. 1975. N9. С.118-129.

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

© 1996 А.А.Шалыто, д-р техн. наук

Федеральный научно-производственный центр
Государственное унитарное предприятие "НПО "Аврора""

Санкт-Петербургский институт точной механики и оптики
(технический университет)

Рассматриваются методы построения "читаемых" графов переходов 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 графы переходов 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 граф-схем алгоритмов и показано, что переход к графам переходов, удовлетворяющим определенным условиям, позволяет решить проблему чтения и понимания спецификаций.

Настоящая работа посвящена описанию методов построения понимаемых графов переходов по граф-схемам алгоритмов и решению обратной задачи — построению понимаемых граф-схем алгоритмов по графам переходов. 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 алгоритма с внутренними обратными связями (ГСА1 в терминологии [1]), представленная на рис.1, и требуется понять эту граф-схему.

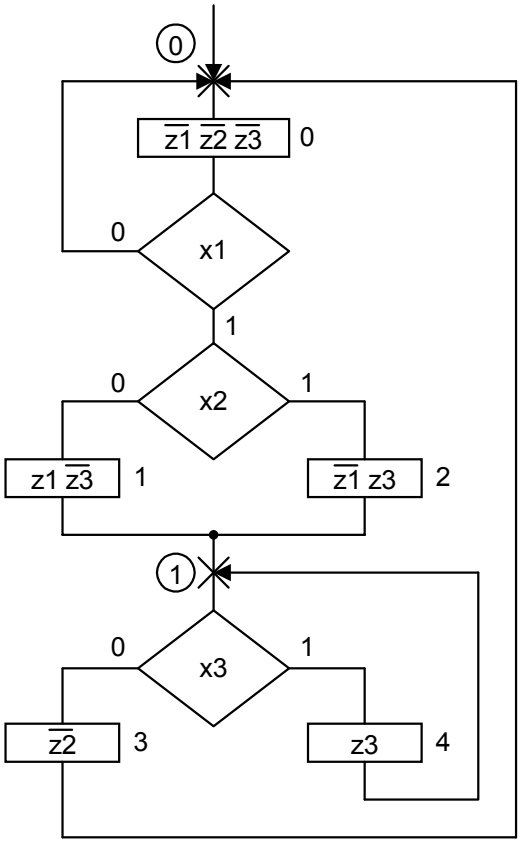

Рис. 1

Если закодировать числами (числа в кружках) точку начала и конца (если она имеется), а также точки, следующие за операторными вершинами, и определить пути в граф-схеме алгоритма между смежными точками [2], то можно построить граф переходов автомата Мили с неоднозначными значениями выходных переменных (рис.2), число вершин в котором равно количеству точек, введенных в граф-схему.

Этот граф переходов может быть эффективно запрограммирован, например, на языке Си, но понять (из-за умолчаний некоторых значений выходных переменных) как он функционирует и соответственно как функционирует ГСА1 весьма сложно.

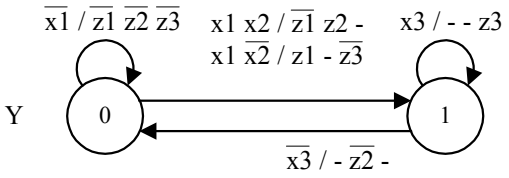

Рис. 2

Поэтому построим по ГСА1 граф переходов автомата Мура, который по принципам построения должен пониматься лучше [1]. Для этого закодируем числами (без кружков) каждую операторную вершину (рис.1) и определим все пути между смежными вершинами [2]. Используя эту информацию, построим граф переходов автомата Мура с неоднозначными значениями выходных переменных, содержащий пять вершин, закодированных многозначными (десятичными) числами (рис.3).

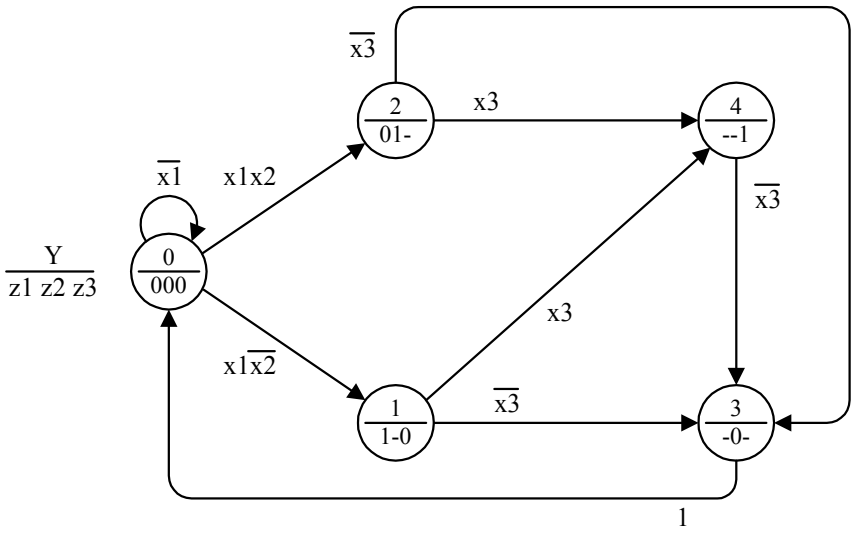

Рис. 3

Этот граф понимается лучше, чем предыдущий, но также весьма сложно, так как и этот граф переходов содержит умолчания.

Анализируя значения, формируемые при прохождении различных путей в этом графе переходов, построим по результатам анализа граф переходов автомата Мура без умолчаний с девятью вершинами (рис.4).

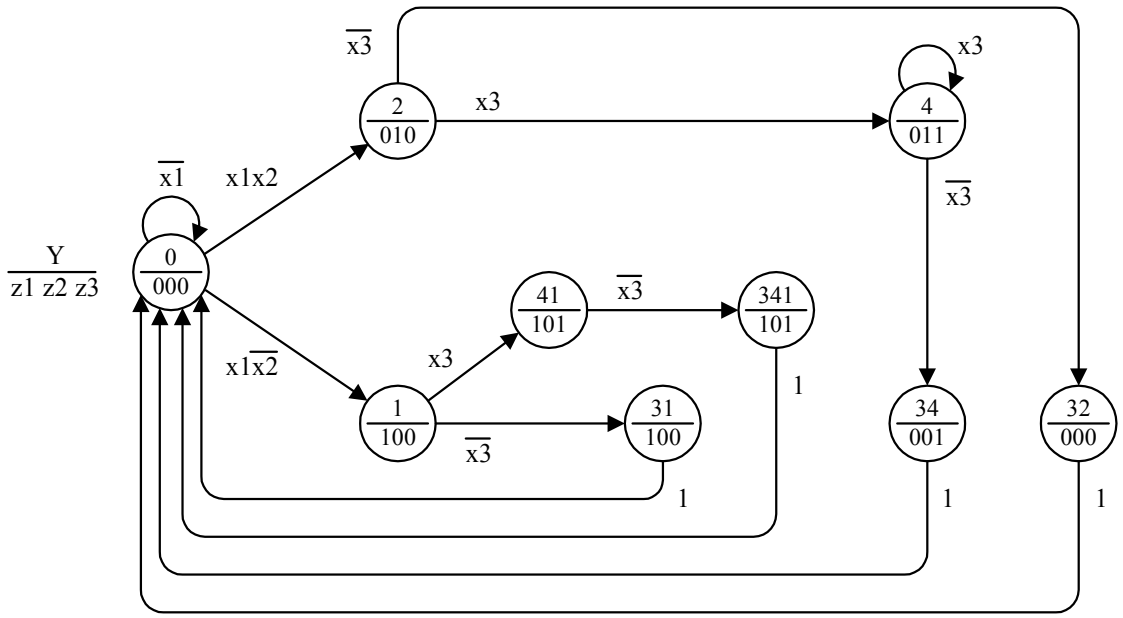

Рис. 4

Так как в этом графе переходов в каждой паре вершин (41, 341) и (1, 31) формируются одинаковые значения выходных переменных, то вторые вершины в этих парах вместе с дугами, помеченными единицами (безусловные переходы), могут быть исключены. При этом первые вершины этих пар соединяются с нулевой вершиной. С этой же вершиной соединяется вершина 2, так как вершина 32, в которой формируются те же значения выходных переменных, что и в нулевой вершине, может быть исключена. Получающийся при этом граф переходов автомата Мура содержит шесть вершин (рис.5).

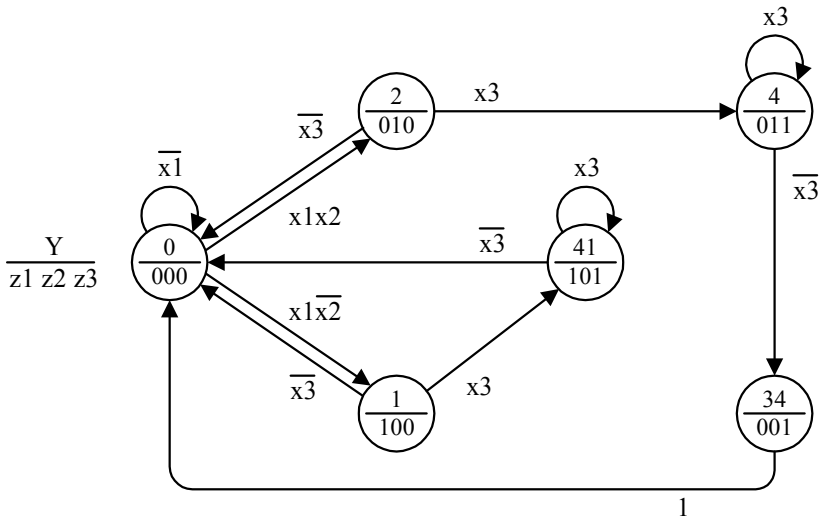

Рис. 5

Этот граф переходов "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 граф-схемы.

Полученный граф переходов полон и непротиворечив, однако содержит, как и ГСА1, два генерирующих контура 0-1 и 0-2, устраняя которые, например введением переменной x3 в пометку дуг 0-1 и 0-2, получим корректный граф переходов (рис.6). В этом графе не изменяющиеся значения выходных переменных перечеркнуты.

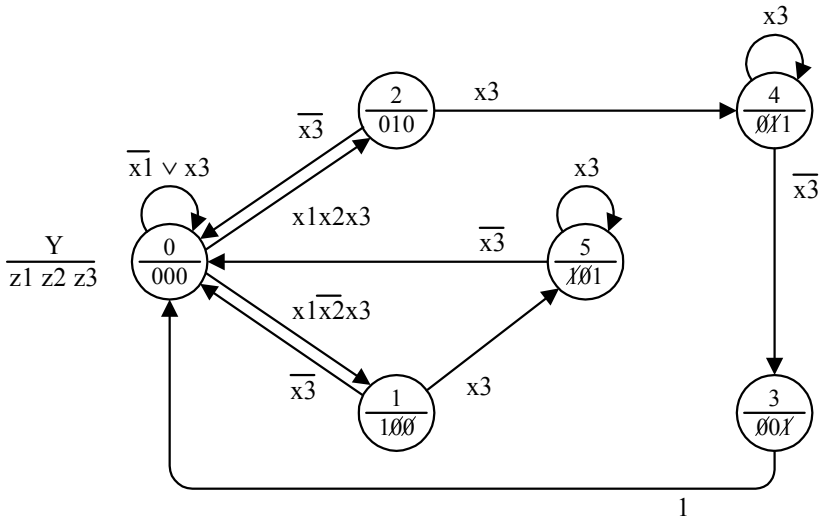

Рис. 6

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 граф-схемы алгоритма это потребовало бы существенных изменений схемы, что может приводить к ошибкам.

Из изложенного следует, что именно граф переходов (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. Приведенный граф переходов в отличие от ГСА1 описывает поведение автомата в явной и понятной форме и содержит достаточно информации, чтобы быть формально реализованным с помощью различных алгоритмических моделей (системой булевых формул, функциональной схемой, 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 графом переходов различна. Действительно, если тексты программ, записанные, например на языке Си, формально построенные по системе булевых формул или функциональной схеме, функционально эквивалентны графу переходов, то внешне эти тексты на граф переходов непохожи, особенно для системы булевых формул, описывающей функциональную схему с триггерами. При этом отметим, что эта система булевых формул отличается от системы таких формул, построенной непосредственно по графу переходов. С другой стороны, в состав указанного языка входит такая управляющая конструкция как переключатель (switch), при использовании которой удается обеспечить изоморфизм между текстом программы и графом переходов, 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 системы булевых формул или функциональной схемы, вся информация, приведенная на рис.6 необходима, то при использовании управляющей конструкции switch в графе переходов пометки петель могут умалчиваться (предполагая, что в каждой вершине обеспечивается логическая полнота пометок дуг, исходящих из нее), а вместо ортогонализации этих пометок могут расставляться приоритеты, указываемые на дугах штрихами, число которых тем меньше, чем выше номер приоритета (рис.7).

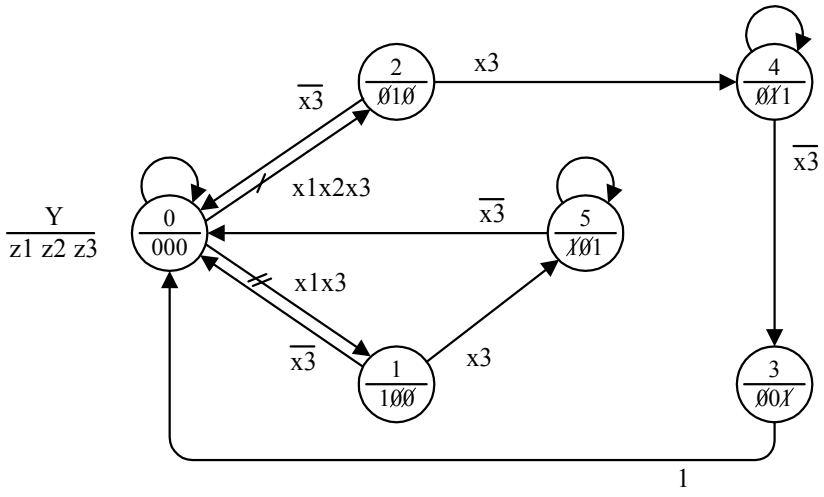

Рис. 7

Это в общем случае позволяет резко упростить граф переходов и уменьшить объем программы, 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" граф-схемы алгоритмов без внутренних обратных связей (ГСА2 в

терминологии [1]) весьма трудно читаются. Покажем какой структурой должны обладать граф-схемы этого класса, чтобы устранить указанный недостаток.

3. РЕАЛИЗАЦИЯ АВТОМАТОВ БЕЗ ВЫХОДНОГО ПРЕОБРАЗОВАТЕЛЯ С ПРИНУДИТЕЛЬНЫМ КОДИРОВАНИЕМ СОСТОЯНИЙ

Пусть задан граф переходов автомата без выходного преобразователя с принудительным кодированием состояний (рис.8), в котором при импульсных переменных x1 и x2 генерирующие контура отсутствуют.

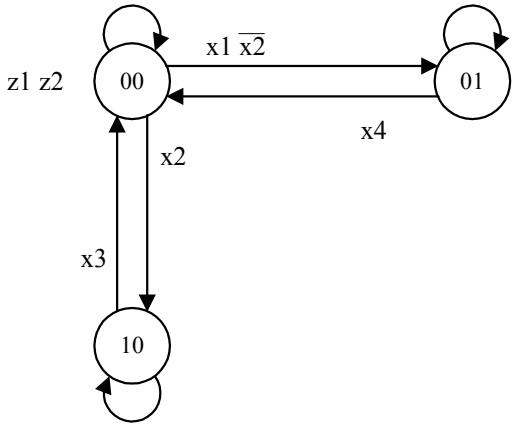

Рис. 8

Кодирование названо принудительным, так как коды, определяющие состояния, совпадают со значениями выходных переменных, формируемых в соответствующих состояниях (вершинах). Этот вид кодирования использован ввиду того, что в рассматриваемом графе переходов комбинации значений выходных переменных во всех вершинах различны. Этот граф переходов предлагается реализовать ГСА4 [1] (рис.9), тело которой состоит из трех слоев.

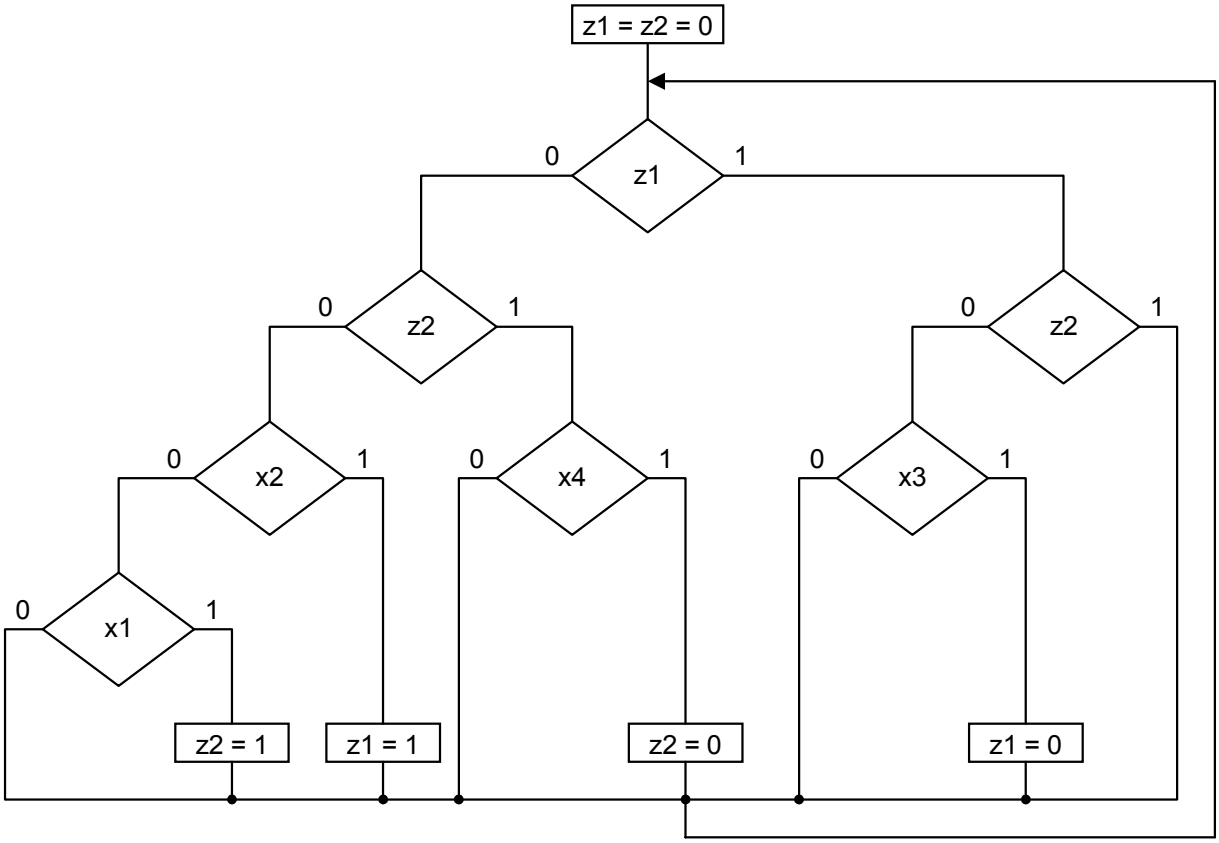

Рис. 9

Первый из них содержит условные вершины, помеченные переменными z1 и z2, и является дешифратором состояний. Второй слой образован условными вершинами, помеченными входными переменными, и реализует функции переходов автомата. Третий слой содержит операторные вершины, в которых указаны значения выходов, совпадающие в данном случае с кодами следующих состояний.

Несмотря на то, что в операторных вершины этой граф-схемы 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 граф-схеме.

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

Такая организация граф-схем алгоритмов (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 соответствующим образом). Ясно, что в этом случае поведение без учета внутреннего состояния только по значениям входных переменных будет выглядеть странным.

Однако при алгоритмизации с помощью граф-схем алгоритмов без внутренних обратных связей 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 x1 = x2 = 1 и отсутствие в модели указания в явном виде значения z при x1 = x2 = 0. Этих недостатков лишена граф-схема алгоритма с дешифратором состояний (рис.10), которая, несмотря на большую сложность по сравнению с граф-схемой (рис.3 [1]), понимается лучше.

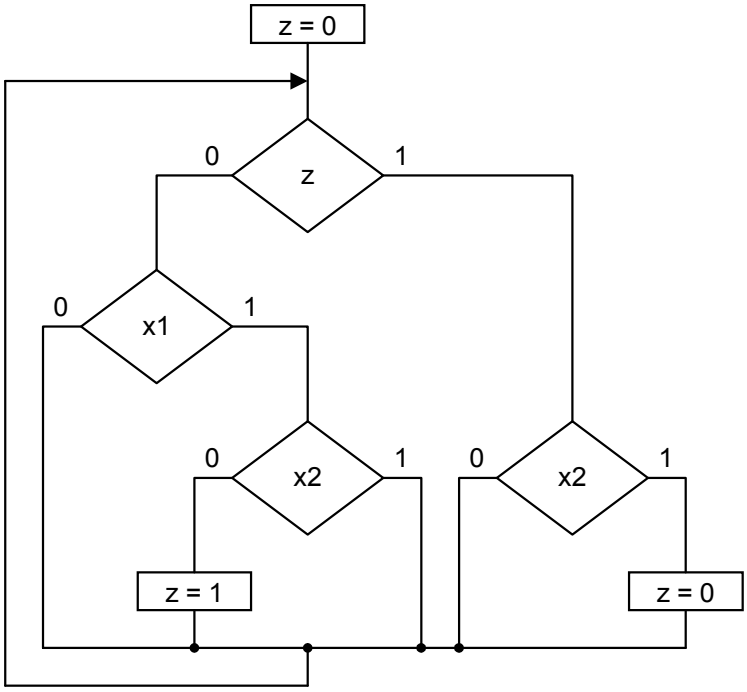

Рис. 10

Простоту чтения и компактность изображения совмещает в себе граф переходов (рис.11), который при программировании с помощью конструкции switch может быть еще более упрощен (умолчание пометок петель).

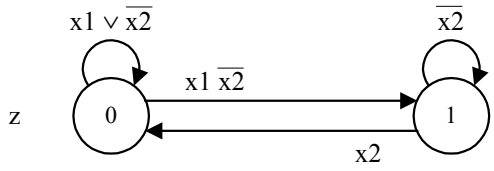

Рис. 11

4. РЕАЛИЗАЦИЯ АВТОМАТОВ БЕЗ ВЫХОДНОГО ПРЕОБРАЗОВАТЕЛЯ С ПРИНУДИТЕЛЬНО-СВОБОДНЫМ КОДИРОВАНИЕМ СОСТОЯНИЙ

Пусть требуется реализовать счетный триггер, поведение которого описывается графом переходов (рис.12).

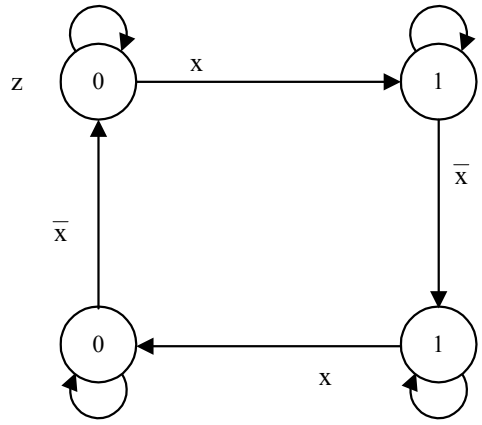

Рис. 12

Для того, чтобы различать вершины графа переходов, используем принудительно-свободное кодирование состояний автомата, введя отсутствующую в алгоритме управления внутреннюю переменную y (рис.13).

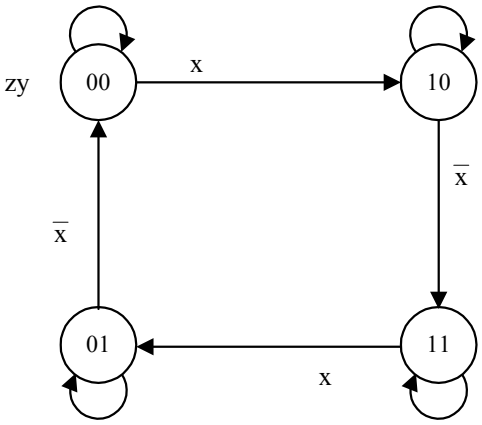

Рис. 13

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 свободно (произвольно). На рис.14 приведена граф-схема алгоритма, эквивалентная этому графу переходов.

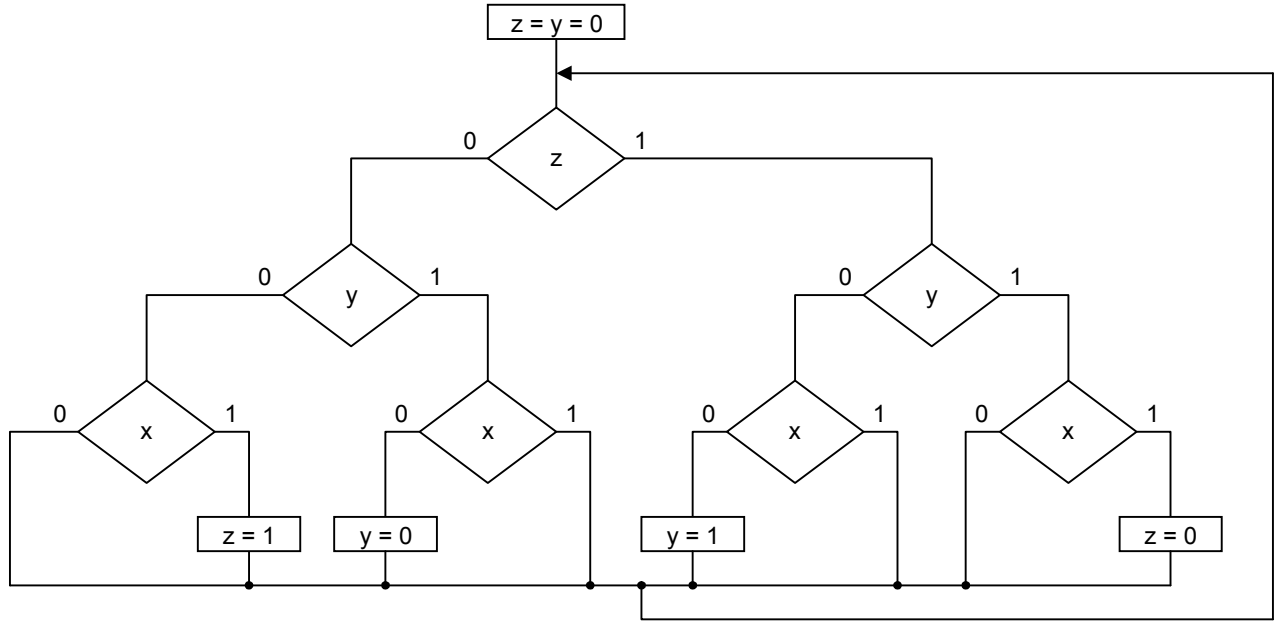

Рис. 14

Зависимость состояний автомата от значений выхода z в рассмотренных граф-схем алгоритмов (рис.9, 10 и 14) может создать трудности при условии, что эти значения могут изменяться в других компонентах алгоритма управления. Поэтому сделаем независимыми состояния автомата от его выходных значений, перейдя к модели автомата Мура.

5. РЕАЛИЗАЦИЯ АВТОМАТОВ МУРА С ДВОИЧНЫМ ЛОГАРИФМИЧЕСКИМ КОДИРОВАНИЕМ СОСТОЯНИЙ

На рис.15 приведен граф переходов автомата Мура, реализующий счетный триггер, для рассматриваемого варианта кодирования, а на рис.16 — соответствующая ему граф-схем алгоритма.

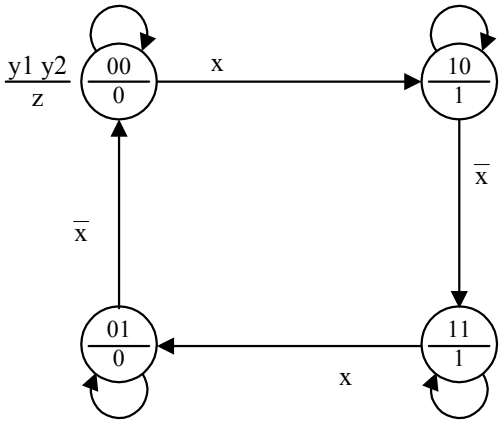

Рис. 15

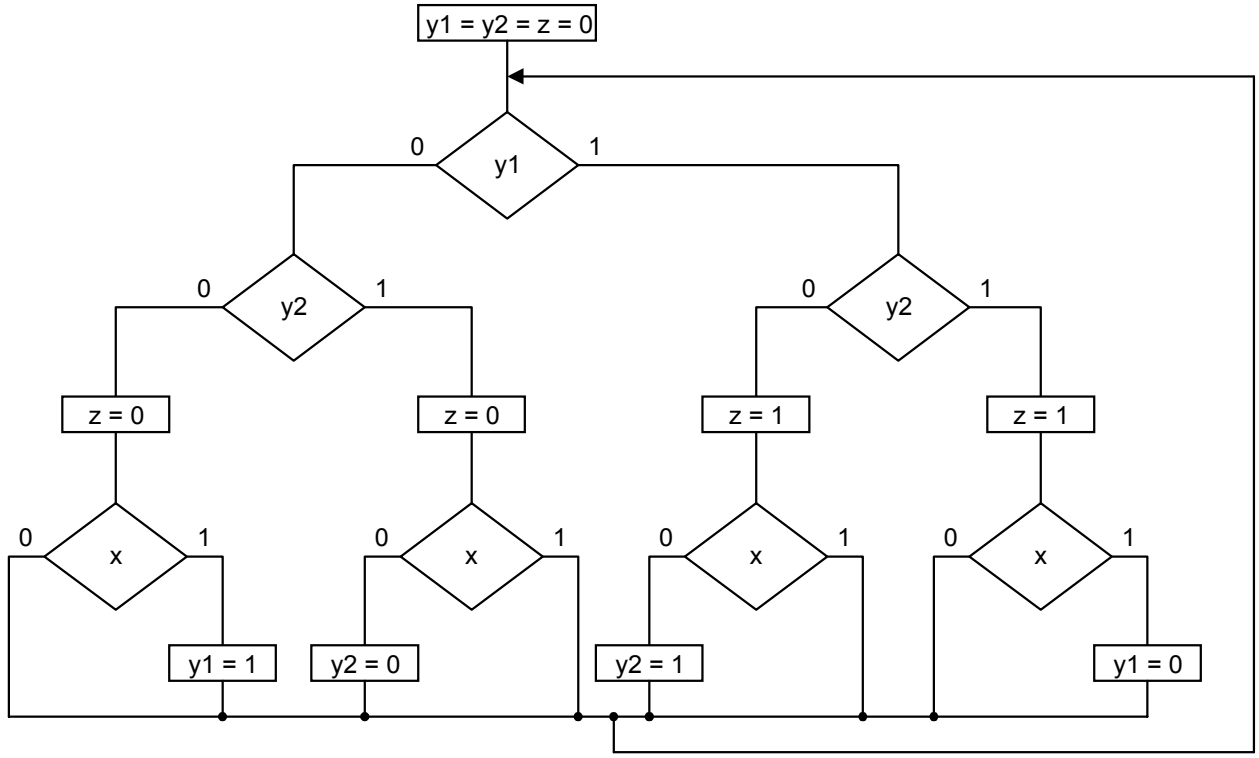

Рис. 16

Тело этой граф-схемы является четырехслойным (дешифратор состояний, формирование значений выходов, реализация функций переходов, формирование следующего состояния). Недостатки этой граф-схемы состоят в значительной глубине пирамидального дешифратора и большом числе вновь вводимых переменных yi, где $0 < i < \lceil \log s \rceil - 1$; s — число состояний автомата. Первый из этих недостатков устраняется в разд. 6, а оба недостатка — в разд. 7.

6. РЕАЛИЗАЦИЯ АВТОМАТОВ МУРА С ДВОИЧНЫМ (ЕДИНИЧНЫМ) КОДИРОВАНИЕМ СОСТОЯНИЙ

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

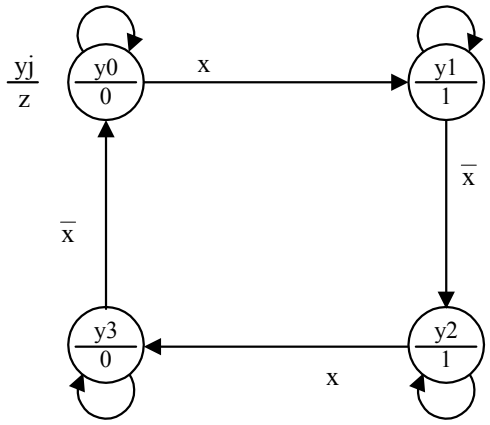

Рис. 17

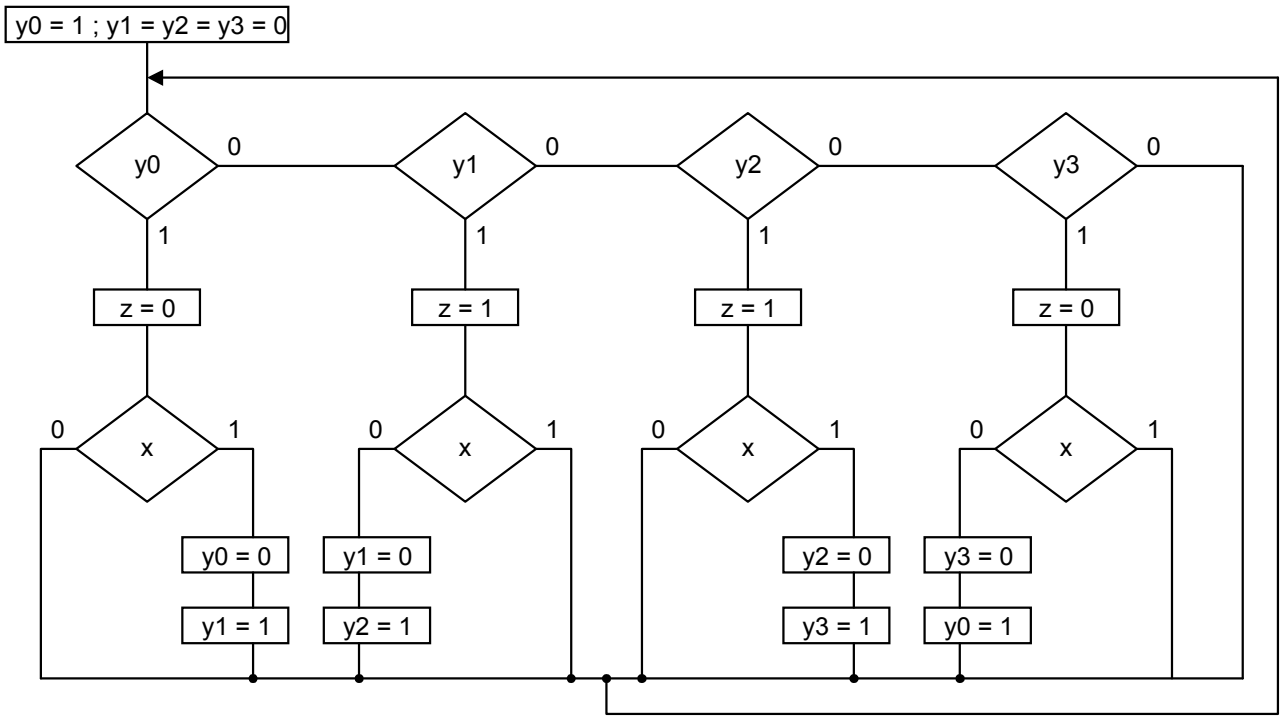

Рис. 18

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 автоматов Мура, Мили и смешанных автоматов, реализуемых в виде одной компоненты, для кодирования состояний достаточно иметь **одну** переменную Y значности s, которую не требуется принудительно сбрасывать, так как она автоматически сбрасывается при переходе от одного значения к другому. При этом

для реализации N-компонентного алгоритма управления (реализуемого N графами) с помощью указанных классов автоматов требуется N многозначных переменных Yj, где j=0,..., N-1. Этот вариант кодирования, практически нереализуемый при аппаратной реализации автоматов, по мнению автора, при отсутствии жестких ограничений на объем памяти программ и возможности работы с многозначными переменными должен стать основным при программной реализации автоматов.

На рис.19 в качестве примера приведена граф-схема алгоритма для рассматриваемого варианта кодирования, построенная по графу переходов автомата Мура счетного триггера (рис.7 [1]).

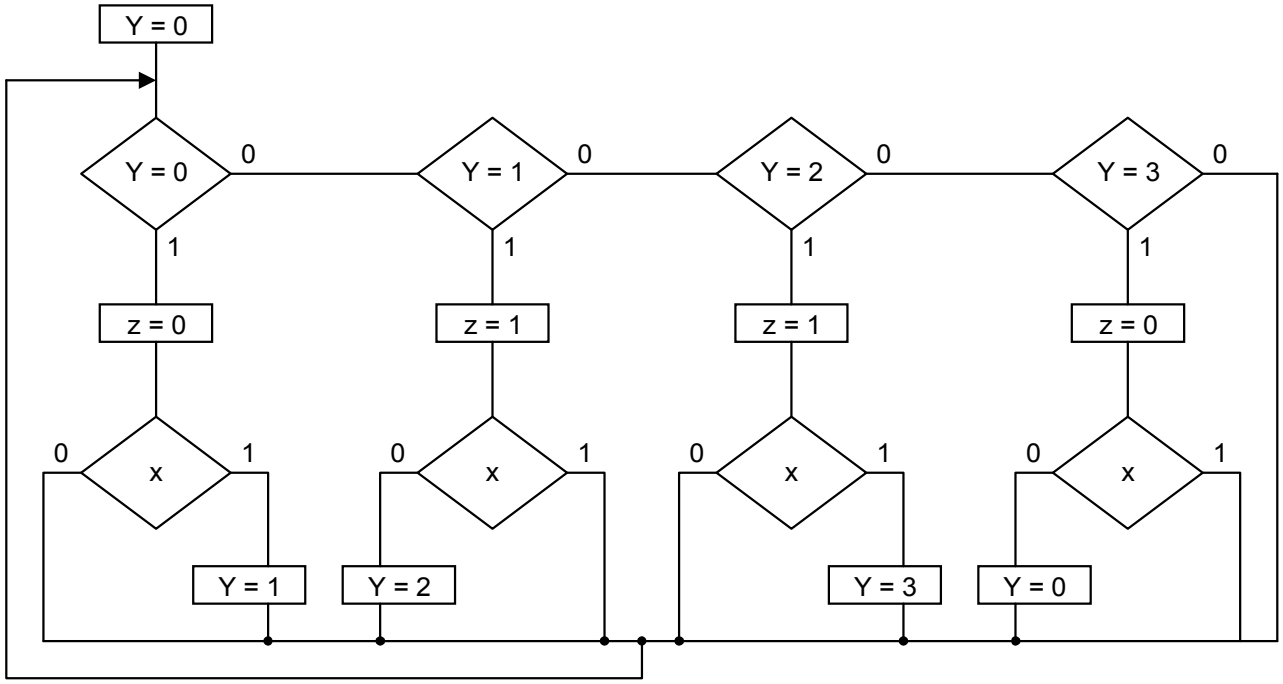

Рис. 19

В этой граф-схеме за счет ухудшения понимаемости условная вершина с пометкой Y = 3 и вторая операторная вершина с пометкой z = 1 могут быть исключены.

Граф-схема алгоритма аналогичной структуры может быть построена для любого графа переходов автомата Мура. При этом в отличие от графа переходов такая граф-схема алгоритма всегда может быть планарной. Применение всего лишь одной промежуточной переменной при "хорошей" организации структуры делает использование граф-схем 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 графах переходов, 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 граф-схема алгоритма без внутренних обратных связей, то в этом случае целесообразно применять граф-схема алгоритмов с дешифратором многозначно закодированных состояний.

При использовании многозначного кодирования открывается весьма эффективная в изобразительном отношении дополнительная

возможность обеспечения связи между компонентами алгоритма управления (в том числе, и при реализации параллельных процессов) 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 графов переходов, представленным на рис.20.

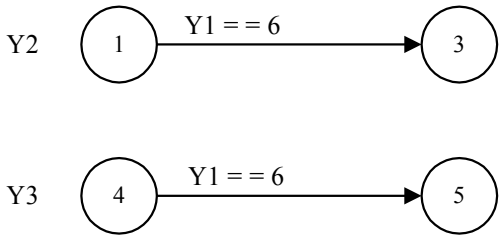

Рис. 20

В [7] показано, что системы взаимосвязанных графов переходов 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 граф переходов являясь последовательным по состояниям обычно является параллельным по входам и выходам (например, графы переходов на рис.7 и 8).

8. РЕАЛИЗАЦИЯ АВТОМАТОВ МИЛИ С МНОГОЗНАЧНЫМ КОДИРОВАНИЕМ СОСТОЯНИЙ

В автоматах Мили, также как в автоматах Мура, могут использоваться различные виды кодирования состояний, однако, исходя из изложенного, рассмотрим только многозначное кодирование. На рис.8 [1] приведен граф переходов автомата Мили, реализующий счетный триггер при принятом варианте кодирования, а на рис.21 — соответствующая граф-схема алгоритма.

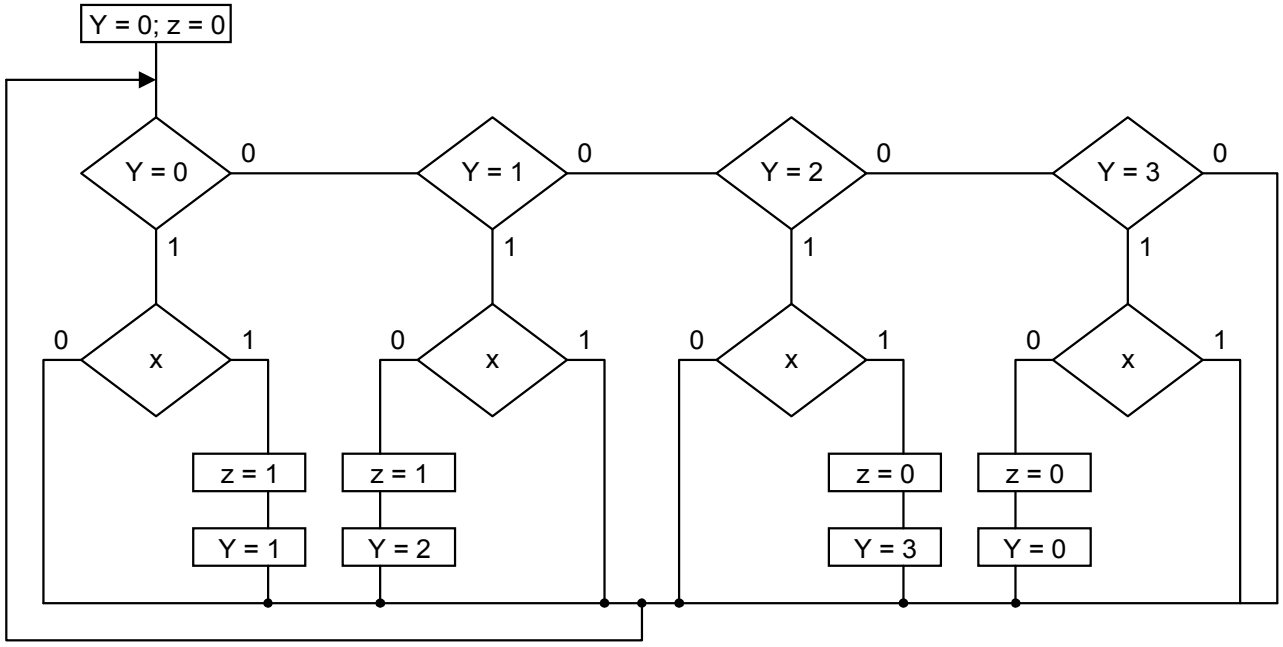

Рис. 21

В этой граф-схеме могут быть исключены условная вершина с пометкой Y = 3 и вторые операторные вершины с пометками z = 1 и z = 0. Отличие этой граф-схемы от граф-схемы автомата Мура состоит в размещении слоя операторных вершин, помеченных значениями выходной переменной, не до дешифраторов значений входных переменных, а после них.

Второй пример реализации этого класса автоматов приведен на рис.22 для последовательностного одноразрядного сумматора (рис.10 [1]).

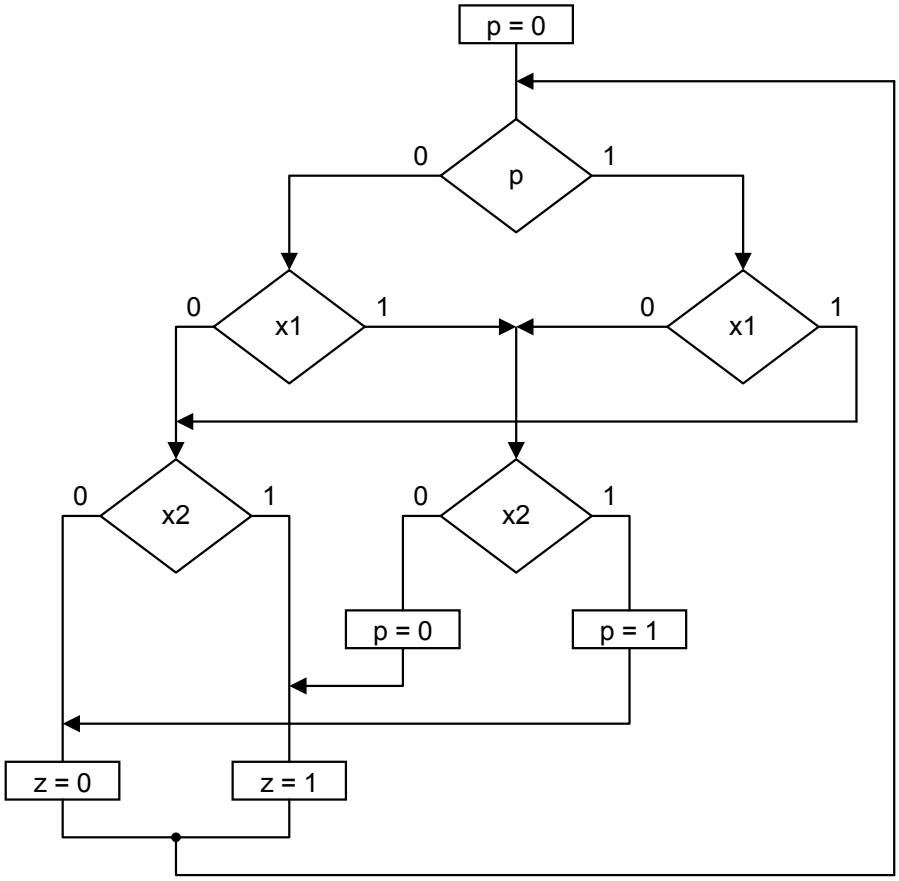

Рис. 22

В этой граф-схеме для минимизации числа вершин слой значений выходной переменной z расположен после слоя со значениями следующего состояния P. Тело этой граф-схемы, 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 граф-схемы алгоритма, реализующей смешанный автомат с флагом, представленный на рис.11 [1].

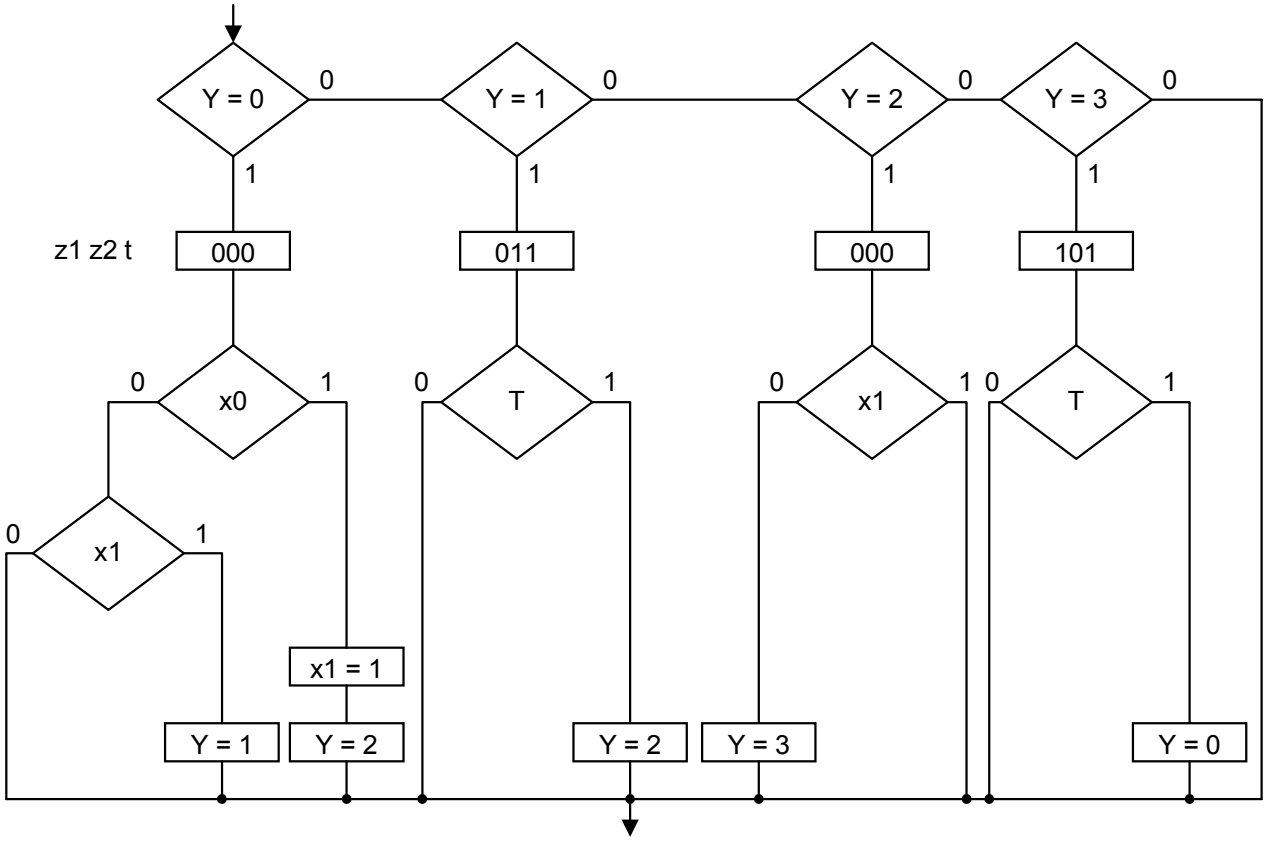

Рис. 23

Эта граф-схема является пятислойной: дешифратор состояний, формирование значений выходных переменных, реализация функций переходов, формирование значений флага, формирование следующего состояния. В приведенной граф-схеме переменная x1 определяет содержимое счетного триггера (еще одной компоненты алгоритма управления), который также управляется и от другого источника информации — кнопки. В этой граф-схеме в отличие от графа переходов противоречивость устранена ортогонализацией и используется булев признак T вместо неравенства t > D.

При этом необходимо отметить, что если при использовании системы взаимосвязанных графов переходов каждая компонента замкнута, то для граф-схем, реализуемых в программируемых логических контроллерах [9], замыкание выполняется для алгоритма управления в целом.

10.ПРОГРАММИРОВАНИЕ ГРАФОВ ПЕРЕХОДОВ И ГРАФ-СХЕМ АЛГОРИТМОВ С МНОГОЗНАЧНЫМ КОДИРОВАНИЕМ СОСТОЯНИЙ В БАЗИСЕ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

Использование многозначного кодирования позволяет для ГСА4, также как и для графов переходов, изоморфно переходить к текстам программ, минуя построение промежуточных граф-схем [1]. При этом несмотря на то, что графы переходов содержат петли и контура (негенерирующие), их не приходится принудительно расцикливать и структурировать, а эти проблемы решаются при программировании выбором соответствующей управляющей конструкции.

Наиболее просто это достигается при использовании такой управляющей конструкции, как например, оператор switch языка Си [5, 6, 10]. Приведем в качестве примера программу, использующую этот оператор для реализации графа переходов автомата Мура (рис.7 [1]) и граф-схемы алгоритма (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 программа обладает замечательным свойством — она с гарантией делает только то, что описывает граф переходов и не делает ничего лишнего, что легко проверяется сверкой текста программы с графом переходов, так как при безошибочном переходе от графа переходов к программе должен быть обеспечен их полный изоморфизм. Этим свойством обладают не все алгоритмические модели. Так, например, если автомат с s состояниями реализуется системой m булевых формул ($m =]\log_2 s[$), а $s \neq 2^m$, то система реализует другой автомат с числом состояний 2^m , в котором исходный автомат является лишь ядром построенного, а переходы из новых 2^m-s состояний в заданные зависят от принятого варианта доопределения.

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

рассматриваемого автомата для других компонент алгоритма управления вне зависимости от значений входных переменных, в отличие, например, от ГСА2 (рис.3 [1]), в которой при x1 = x2 = 1 значение z = 1 для других компонент недоступно.

Так, например, если первая компонента алгоритма управления реализована указанным образом и значение переменной Y1 равное шести доступно другим компонентам, то фрагменту системы взаимосвязанных графов переходов (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 программы, но и на чтение спецификации — графа переходов или системы взаимосвязанных графов переходов. Для обеспечения универсальности, в том числе и для графов переходов с неустойчивыми вершинами, целесообразно считать, что в каждом графе переходов за один проход выполняется не более одного перехода, что на программном уровне поддерживается указанным образом.

Приведем теперь пример программы, построенной по графу переходов автомата Мура (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 управляющей конструкции switch также эффективно реализуются и смешанные автоматы.

Из изложенного следует, что граф переходов, во-первых, может одновременно использоваться в качестве спецификации как алгоритма управления, так и программы, а, во-вторых, с помощью управляющей конструкции switch он изоморфно отражается в текст структурированной программы, которая может быть наблюдаемой не только по двоичным выходам, но и, что самое главное, по

десятичному номеру состояния. При этом для проверки программы на экран дисплея для каждого графа переходов автомата Мура, Мили или смешанного автомата достаточно вывести только **одну** десятичную внутреннюю переменную (сигнатуру). Это позволяет при исследовании поведения автомата в динамике следить только за значениями этой переменной (предварительно определив, например, для автомата Мура, 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. Графы переходов автоматов с флагами также изоморфно отражаются в текст программы с помощью управляющей конструкции switch. Однако зависимость следующего состояния от предыстории резко уменьшает их модификационную способность. Отметим также, что основная причина простоты внесения изменений в программы, построенные указанным образом по графу переходов, состоит в том, что в последних вершины связаны между собой непосредственно в отличие от граф схем, в которых операторные вершины в общем случае соединены через условные вершины и поэтому изменение в одном переходе оказывает существенное влияние на другие переходы. При использовании предлагаемого подхода верификация программы может производиться сверкой текста программы с графом переходов. Для однокомпонентных алгоритмов управления граф переходов может использоваться в качестве теста для проверки программы, а для многокомпонентных алгоритмов управления (по аналогии с сетями Петри) по системе взаимосвязанных графов переходов должен строиться граф достижимых маркировок, позволяющий исследовать все функциональные возможности системы и представить ее поведение одним графом переходов. Эта задача практически разрешима при отсутствии параллелизма по состояниям в алгоритме управления или для задач малой размерности. Для независимых параллельных процессов строить единый граф переходов 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 программирования граф-схем алгоритмов с внутренними обратными связями, которые в многозадачном режиме использования применяемого вычислительного устройства не реализуется традиционным путем без предварительного расцикливания [1].

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

Пусть, например, задана граф-схема алгоритма с внутренними обратными связями (рис.1). По этой граф-схеме может быть построен граф переходов автомата Мили (рис.2) или граф переходов автомата

Мура (рис.3), каждый из которых может быть однозначно реализован управляющей конструкцией switch. Несмотря на то, что получающиеся при этом программы весьма компактны, эти графы переходов содержат генерирующие контура и их понимание весьма затруднительно из-за неоднозначности значений выходных переменных. Эти недостатки отсутствуют в программе, построенной по преобразованному графу переходов автомата Мура (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 программе равно s + d + 1, где d — число дуг (включая петли) в графе переходов. В метке case 0 оператор if, соответствующий дуге с наибольшим приоритетом, исходящей из вершины 0, размещается ниже оператора if, соответствующего дуге с меньшим приоритетом. При такой программной реализации проблемы расцикливания и структурирования исходной граф-схемы решаются автоматически в ходе построения программы. При этом отметим, что построение графа переходов по граф-схеме алгоритма с внутренними обратными связями не является необходимым. Для написания программы в этом случае достаточно выполнить многозначное кодирование (раздел 2) операторных вершин заданной граф-схем алгоритма (для построения программы, соответствующей автомату Мура) или точек, следующих за операторными вершинами (для построения программы, соответствующей автомату Мили). Быстродействие программы может быть повышено при усложнении ее структуры:

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

  case 1 :
    z1=1;    /*z2=0;   z3=0;*/
    if(!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 программе фрагмент, соответствующий дуге с большим приоритетом, исходящей из нулевой вершины, располагается выше фрагмента, соответствующего дуге с меньшим приоритетом, исходящей из той же вершины.

В качестве второго примера рассмотрим граф-схему алгоритма с внутренними обратными связями (рис.2 [1]) и построим эквивалентный ей граф переходов автомата Мура (рис.24).

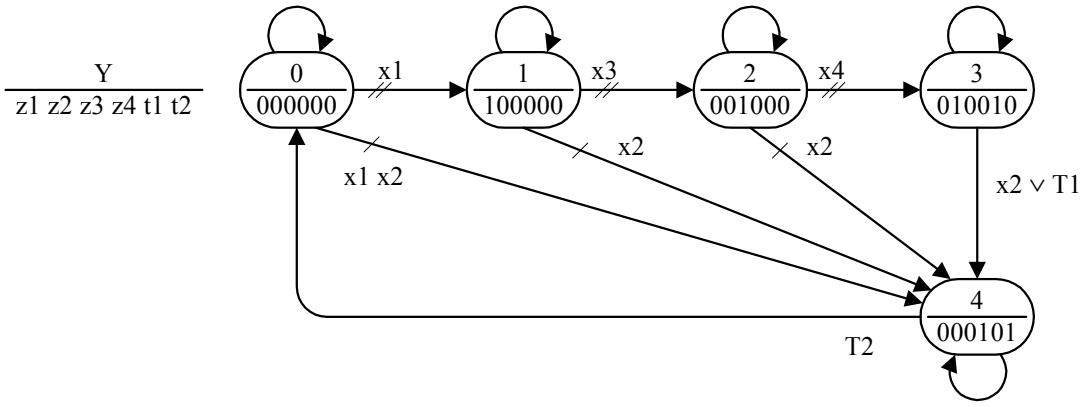

Рис. 24

В этом графе переходов, входящем в состав управляющего автомата, каждой единице на позициях t1 и t2 соответствует обращение к процедуре time(i,D), где D — время задержки i-го функционального элемента задержки. Этот граф переходов, также 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 управляющих конструкций используемого языка программирования целесообразно переходить от исходной граф-схем алгоритма к тексту программы не непосредственно, а используя промежуточную форму граф-схем алгоритма (ГСА3 в терминологии [1]).

На рис.25 приведена граф-схем алгоритма, линеаризованная и структурированная специальным образом (все условные вершины в каждом блоке при не выполнении условия передают управление в одну точку), эквивалентная граф-схеме алгоритма (рис.19).

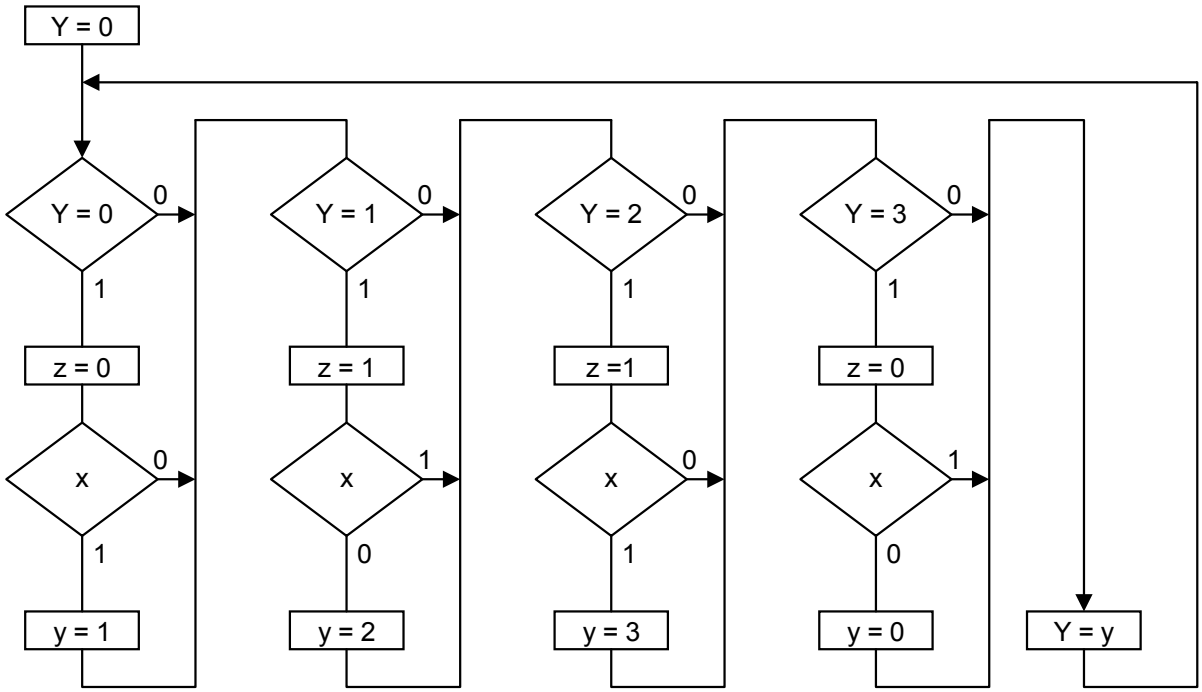

Рис. 25

Полученная граф-схема алгоритма, в свою очередь, может быть практически изоморфно преобразована в граф-схему программы, учитывающую семантику применяемых команд и архитектурные особенности используемого вычислительного устройства, например программируемого логического контроллера [12], по которой также изоморфно может быть построен текст программы на языке инструкций этого контроллера:

```
STR  R   C  0 ; Ввод константы 0 в регистровый сумматор (РС:=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 программы
```

Если в качестве языка спецификации в этом случае использовать не граф-схему алгоритма, а граф переходов (рис.7 [1]), то последний может быть практически изоморфно отражен в следующий текст программы:

```
STR  R   C  0 ; Ввод константы 0 в регистровый сумматор (PC:=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 ; PC:=1
EQ   R  SM  y ; if(БС)  y=1

STR  R   C  1 ; PC:=1
EQU  R   M  Y ; if(Y=1) БС=1
EQ      SO  z ; if(БС)  z=1
AND     NI  x ; БС:=БС&!x
STR  R   C  2 ; PC:=2
EQ   R  SM  y ; if(БС)  y=2
...
STR  R   M  y ; PC:=y
EQ   R   M  Y ; Y:=PC
STOP          ; передача управления в начало программы
```

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 O z (z:=БС) при условии, что в начале программы z = 0. Это объясняется тем, что в этом случае пока автомат находится в первой вершине в ячейку z записывается единица и ноль, когда он "уходит" из этой вершины. Дальнейшее упрощение программы связано с еще более значительным уменьшением степени ее изоморфизма с графом переходов. При этом каждая пара команд 5-6 и 11-12 может быть заменена командой INC R M y (y:=y+1).

Рассмотренные программы обеспечивают реализацию за один программный цикл не более одного блока в граф-схеме алгоритма (рис.19) или одного перехода в графе переходов (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 системы взаимосвязанных графов переходов. Если отказаться от требования, чтобы за один проход выполнялось не более одного перехода в графе переходов, то граф-схема (рис.25) может быть преобразована в граф-схему алгоритма (рис.26).

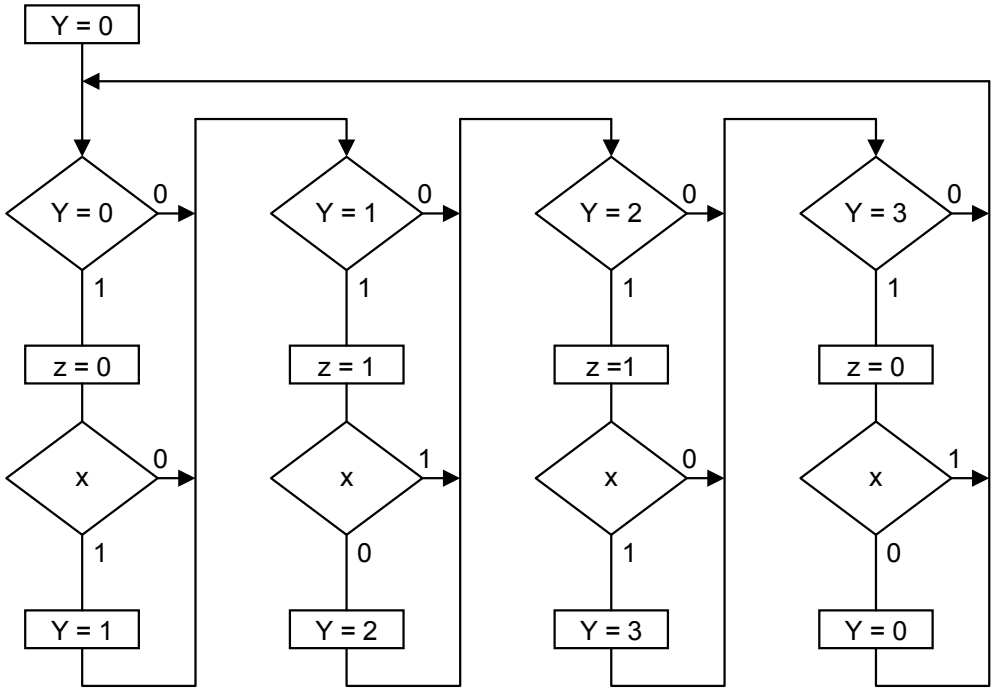

Рис. 26

Это позволяет еще более упростить приведенные выше программы. Построенная граф-схема алгоритма корректна, так как в программируемом логическом контроллере [12] в течение одного прохода программы значения введенных входных переменных не изменяются.

Если в качестве языка спецификации использовать граф переходов и снять ограничение на число переходов, реализуемых за один программный цикл, то появляется возможность применения при написании программы такой нетрадиционной управляющей конструкции как шаговый регистр, существующей в системах команд многих программируемых логических контроллеров [9,12]. Однако в этих работах шаговый регистр предлагается использовать в режиме формирования последовательных шагов и не упоминается возможность применения его для реализации произвольных автоматов. Поэтому настоящую работу можно считать в качестве обоснования такой возможности.

Приведем в качестве примера программу, реализующую граф переходов (рис.7 [1]), в которой используется нулевой (из 32 имеющихся) шаговый регистр, находящийся в начальный момент времени на нулевом (из 256 допустимых) шаге (в нулевом состоянии):

```
READ   S  0 ; Выбор в качестве рабочего шагового регистра 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 программа не содержит условных переходов, то и в ней третья команда может быть исключена при замене седьмой

команды на команду EQ O z при условии, что в начальный момент времени z = 0.

Подходы, изложенные в настоящем разделе, могут быть использованы и для автоматов Мили, смешанных автоматов и автоматов с флагами. Так, например, граф переходов автомата Мили (рис.8 [1]), описываемый на языке Си следующей программой:

```
    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 программой, использующей шаговый регистр:

```
READ   S  0

STR    S  0
AND    I  x
EQ    RO  z
STEP   S  1

STR    S  1
AND   NI  x
EQ    SO  z
STEP   S  2
...

STOP
```

Из изложенного следует, что использование шаговых регистров делает программирование автоматов на языке инструкций весьма высокоуровневым. Эта же тенденция сохраняется и при реализации управляющих автоматов, тем более, что в ряде случаев функциональные элементы задержки могут быть реализованы командой NEXT Si D, которая, если шаговый регистр 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 графа переходов (за счет увеличение числа вершин в нем) для обеспечения перехода только в соседнюю вершину по истечении выдержки времени. При этом необходимо отметить, в данном случае управляющий автомат не декомпозируется на автомат и функциональные элементы задержки, а программируется как единое целое. Реализуем в качестве примера граф переходов автомата Мура с флагом (рис.11 [1]), предварительно увеличив число вершин в нем до пяти, и при условии, что в начале программы z1 = z2 = 0 :

```
READ   S   0
IF     S   0
STR    M  x1
STEP   S   1
STR    I  x0
EQ    SM  x1
STEP   S   2
CONT

STR    S   1
EQ     O   z2
NEXT   S1  D

STR    S   2
AND   NM   x1
STEP   S   3

STR    S   3
EQ     O   z1
```

```
NEXT   S3  D

STR    S   4
STEP   S   0

STOP
```

В этой программе изоморфизм с графом переходов обеспечивается по вершинам последнего. При этом фрагмент программы, 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 программе команды выдачи 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 O zi.

Если изоморфизм обеспечивать не по вершинам графа переходов, 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  0
AND    I  x0
EQ    SM  x1
STEP   S  2

STR    S  0
AND    M  x1
STEP   S  1

STR    S  1
EQ     O  z2
NEXT   S1 D

STR    S  2
AND   NM  x1
STEP   S  3

STR    S  3
EQ     O  z1
NEXT   S3 D

STR    S  4
STEP   S  0

STOP
```

13. СРАВНЕНИЕ ПРЕДЛАГАЕМОГО ПОДХОДА С МЕТОДОМ ПОСТРОЕНИЯ СТРУКТУРИРОВАННЫХ ГРАФ-СХЕМ, ПРЕДЛОЖЕННЫМ АШКРОФТОМ И МАННОЙ

Универсальный метод построения структурированных граф-схем алгоритмов 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 граф-схем алгоритмов:

- в явном виде не используется понятие "состояние";
- не делается различие по типам автоматов и типам используемых переменных;

- используется только один тип кодирования фрагментов неструктурированной граф-схемы алгоритмов при объединении в каноническую структуру;
- из-за связи фрагментов структурированных граф-схем алгоритмов по данным визуально весьма сложно обнаружить генерирующие "контура";
- глубина структурированных граф-схем алгоритмов не ограничена одним переходом, что может не позволить использовать значения некоторых ее внутренних переменных в других компонентах алгоритмов управления;
- при применении в качестве исходной ГСА1 в ней не устраняется неоднозначность значений выходов;
- при использовании в качестве исходной ГСА2, в ней не только не устраняется неоднозначность значений выходов, но кроме того она может содержать большое число битовых промежуточных и флаговых переменных (в том числе с умалчиваемыми значениями), которые не исчезают при структурировании и к которым добавляется еще одна многозначная переменная, соответствующая номерам выделенных структурированных фрагментов;
- человек привыкает и понимает исходную форму представления алгоритма управления и обычно психологически не готов к работе с другими формами представления алгоритма;
- предложенный метод ориентирован на использование языков высокого уровня.

Так как указанный метод предназначен для построения структурированных граф-схем алгоритмов, то из его содержания следует, что если исходная граф-схема 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 граф-схема "плохо" понимается и вместо нее для целей общения целесообразно использовать граф-схему алгоритма (рис.10) или граф переходов (рис.11).

Таким образом, в целом идея, предложенная Ашкрофтом и Манной порочна, так как незачем сначала строить неструктурированную граф-схему алгоритма, чтобы потом ее структурировать. Следует либо сразу строить структурированную граф-схему с дешифратором состояний для принятого варианта кодирования, либо, что более целесообразно, производить исходное описание в виде графа переходов для выбранного типа автомата, в котором, по-возможности, должны отсутствовать умалчиваемые значения переменных, и который изоморфно реализуется, например, с помощью управляющей конструкции switch языка Си, производящей одновременно расцикливание и структурирование и обеспечивающей доступ к любому значению многозначной переменной, соответствующей состояниям автомата. Подход применим и для языков низкого уровня, например, языков инструкций программируемых логических контроллеров.

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

Графы переходов по форме изображения являются в общем случае "существенно плоскостными", что позволяет отражать алгоритмы управления в более естественной форме, чем в виде граф-схемы или диаграммы Графсет [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 графы переходов должны быть в максимальной степени планарными. Графы переходов являются более ранней "сущностью" теории алгоритмизации (теории автоматов) 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 графы переходов в отличие от таблиц переходов, оперирующих с минтермами и содержащих обычно большое число пустых клеток, существенно более обозримы и практически применимы для задач большой размерности. Основное достоинство графов переходов, позволяющее описывать задачи такой размерности, состоит в том, что если граф ортогонализован, то переход между двумя вершинами определяется не всеми входными переменными, упоминаемыми в пометках всех дуг графа, как это имеет место при использовании таблиц переходов, а только теми 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 графов переходов, так как для дуг с меньшими приоритетами, исходящих из некоторой вершины, не удается сразу определить (прочесть) перечень всех переменных, от которых зависит каждый переход по этим дугам. Для определения указанного перечня в этом случае приходится анализировать пометки всех дуг, исходящих из рассматриваемой вершины — локальность описания уменьшается.

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

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

Предлагаемый подход может быть использован не только для логико-временных алгоритмов управления, 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).

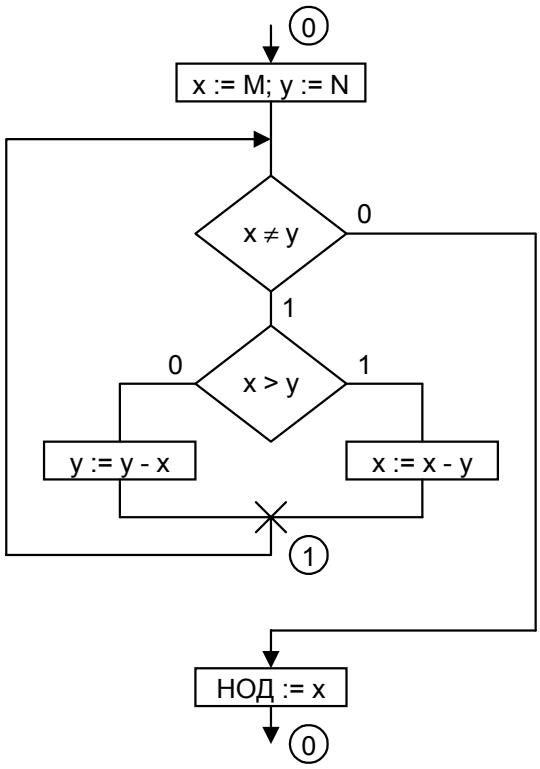

Рис. 27

Эта граф-схема алгоритма является не автоматной, а логико-вычислительной. Она может программироваться с помощью подхода предлагаемого в настоящей работе по графу переходов, описывающему логико-вычислительный процесс и являющемуся автоматом Мили с флагами (рис.28).

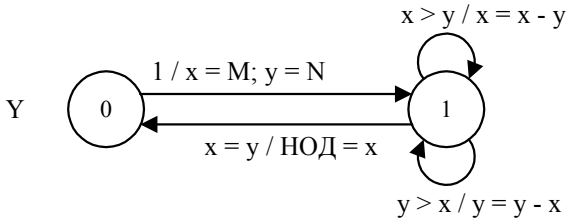

Рис. 28

Необходимо отметить, что этот граф переходов существенно компактнее, чем соответствующая граф-схема алгоритма.

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, предназначен для алгоритмизации и программирования задач логического управления и 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 программу, 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 программ, реализующих функциональные задачи. Для обеспечения этой возможности под руководством автора разработана программная оболочка для отладки систем управляющих автоматов большой размерности, заданных системами взаимосвязанных графов переходов и описанных на языке Си, а Б.П.Кузнецовым (НПО "Аврора") совместно с автором реализован транслятор "Си — язык инструкций", учитывающий изложенные выше особенности программной реализации автоматов в программируемых логических контроллерах, описанных в [12]. Предложенная оболочка позволяет моделировать не только алгоритмы управления, но и комплекс "алгоритм управления — объект управления", в котором компоненты модели объекта управления (например, клапана) также описываются с помощью графов переходов. 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 графе переходов 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 граф-схем алгоритмов в большинстве случаев переход от алгоритмизации к программированию для задач логического управления сложными технологическими процессами представляет большую проблему. Это объясняется тем, что обычно в 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 системы взаимосвязанных графов переходов , 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 графов переходов, 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 В.Н. Кондратьевым (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 функциональных схем 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 автомата.

Если спецификация программно реализуемого алгоритма управления выполняется с помощью функциональной схемы, построенной на триггерах и логических элементах, охваченных обратной связью, то несмотря на то, что функциональная схема в отличии 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, также как и граф переходов, может формально и независимо программироваться, ее весьма трудно понимать. Это объясняется тем, что, во-первых, она в отличие от графа переходов

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 функциональной схемы (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 обратных связей резко затрудняет выбор тестов для определения всех функциональных возможностей схемы. Поэтому вместо тестирования более целесообразно выполнить верификацию — по функциональной схеме формально записать систему булевых формул, а по ней также формально построить граф переходов, поведение которого и следует анализировать.

Если поведение графа переходов отличается от желаемого, то изменяется не исходная функциональная схема, а строится корректированный граф переходов, который, в свою очередь, может быть реализован в помощью различных алгоритмических моделей, в том числе и отличных от функциональной схемы. При использовании последней по новому графу переходов формально строится новая система булевых формул, которая формально реализуется функциональной схемой.

При этом необходимо отметить, что если при аппаратной (в особенности асинхронной) реализации поведение реальной схемы может отличаться от поведения, задаваемого моделью, то при программной реализации это ограничение устраняется весьма просто, например, при использовании системы булевых формул 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 программируемых логических контроллеров, таких как, например, программируемые логические контроллеры [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 д.т.н. О.П. Кузнецова 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 //Автоматика и телемеханика (А и Т). 1996. N6. 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 //А и Т. 1979. N7. 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. N1. 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. V5. N1. P.38-45.
16. CASE - технология в России: настоящее или будущее? //Computer world Moscow. 1992. N40-41. 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. N1. С.2-9.
21. Кузнецов О.П., Макаревский А.Я., Марковский А.В. и др. ЯРУС - язык описания работы сложных автоматов //А и Т. 1972. N6. С.80-89, N7. 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) //А и Т. 1985. N6. 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). Work-shop. Monterey, California, 1995.

24. Прангишвили И.В., Амбарцумян А.А. Научные основы построения АСУ ТП сложных энергетических систем. М.: Наука, 1992.