УДК 519.687

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

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

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 из-за неиспользования 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 (ПЛК) все чаще используются ЭВМ, предназначенные для работы в производственных условиях.

Однако вне зависимости от типа управляющего вычислительного устройства (ВУ) 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 (ГС) или просто схемами [27, 28], что закреплено соответствующими стандартами [29, 30].

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

Необходимо отметить, что и в литературе недостаточно рассматривались свойства ГСА, упрощающие их применение в качестве языка общения для указанного класса задач. Так, в [31, 32] были предложены методы построения ГС, структурная организация которых улучшает их понимание. При этом авторы этих работ считали, что если ГС построены только из вложенных базовых управляющих конструкций (УК) 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 были введены понятия "объекта" и его "состояния" и рекомендовано использовать графы переходов (ГП) для описания динамики реализуемых процессов. Однако, кроме одного примера использования ГП, 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 ГС и ГП, а во второй ее части предлагаются методы построения понимаемых ГП по ГСА и понимаемых ГСА по ГП, а также методы их программирования в базисе языков различных уровней.

2. Граф-схемы. Основные проблемы

Будем рассматривать в настоящей работе автоматные ГС, в которых в операторных вершинах (ОВ) формируются или сохраняются только единичные и нулевые значения переменных. Автоматные ГС разобьем на два класса: ГС алгоритмов (ГСА) и ГС программ (ГСП).

При этом будем различать 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 ГСА:

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

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

ГСА без ВОС, учитывающие особенности используемых УК языка программирования, обозначаемые ГСА3;

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

ГСА без ВОС, 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 ПЛК). Отметим также, что если в ВУ реализуется алгоритм управления (АУ) в виде одной компоненты (задачи), то [redacted] ГСА1, [redacted] используя ГСА3, могут при наличии соответствующих УК изоморфно отражаться в ГСП.

Если же АУ реализуется в ВУ в виде нескольких компонент, то для ГСА1 отсутствует возможность изоморфного отражения в ГСП. Это объясняется тем, что в задачах логического управления при использовании ГСА1 при определенных условиях на длительное время может наступать зацикливание по ВОС, что исключает возможность перехода к реализации других компонент АУ 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 (УА) с объектом управления (ОУ), состоящим из трех клапанов (Кл1, Кл2, Кл3) и двигателя (Д). При этом УА декомпозирован на автомат (А) и функциональные элементы задержки (ФЭЗ). На рис. 2 для рассматриваемого примера приведена ГСА1

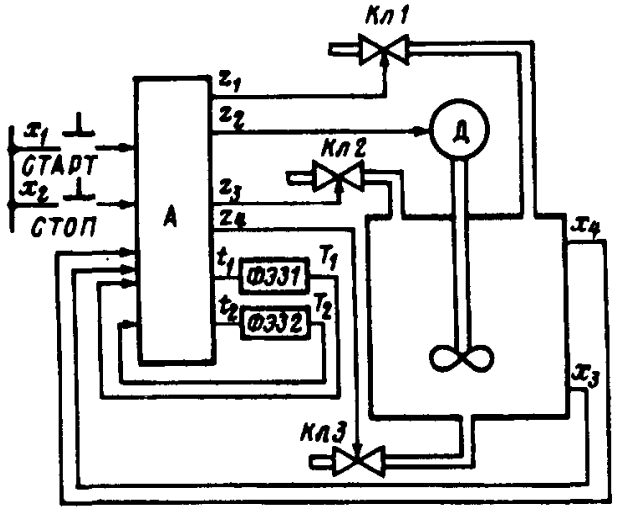

Рис. 1

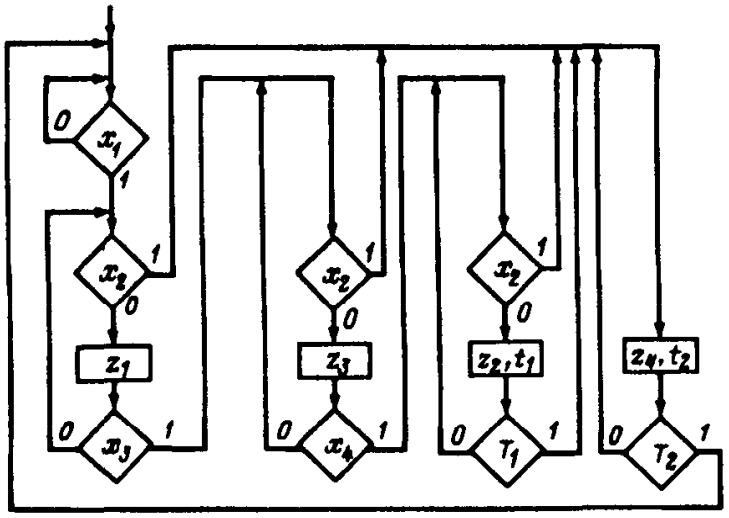

Рис. 2

с пятью ВОС [34]. Эта ГС реализует либо УА в целом (ФЭЗ1 и ФЭЗ2 вычисляются в третьей и четвертой операторных вершинах), либо только А (временные переменные t_1 и t_2 рассматриваются как обращения к процедуре, реализующей ФЭЗ, а двоичные переменные T_1 и T_2 сигнализируют о срабатывании элементов задержки).

Эта ГС содержательно неполна, так как в ней не указано, в каких ОВ сбрасываются выходные z_i $(i = 1 \ldots 4)$ и временные t_j $(j = 1, 2)$ переменные. Поэтому эта ГС не может использоваться в качестве формальной спецификации задачи и требует доопределения. Обращаясь к принципам работы одновходовых исполнительных механизмов, используемых в рассматриваемом ОУ, можно утверждать, что они не обладают памятью и поэтому в ГСА1 (рис. 2) по умолчанию предполагается, что в тех ОВ, в которых переменная не указана, ее значение равно нулю.

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

степени – по данным [28, 35]. Таким образом, ГС 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 ГС не позволяет использовать ее также и в качестве теста для проверки программы (П), реализующей эту граф-схему. Более того, если П строится по ГС эвристически, то даже при отсутствии умолчаний с помощью ГС можно проверить лишь то, что программа реализует ГС, но невозможно установить, что П не делает ничего дополнительно.

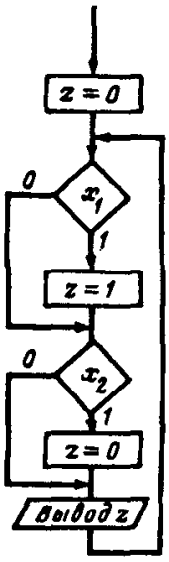

Рис. 3

Возвращаясь к описанию особенностей различных подклассов ГСА, отметим, что для ГСА1 характерно, что в них выдача значений выходных переменных осуществляется не в конце ГС, а в любых ОВ.

ГСА1 могут быть построены так, что в условных вершинах (УВ) применяются только входные переменные типов X и T, а в ОВ – выходные переменные типов Z и t, где T – переменные, фиксирующие факт срабатывания ФЭЗ. Наличие в ГСА1 только тех переменных, которые упоминаются в АУ, является важным достоинством этого подкласса ГСА.

В ГСА2 крайне редко используются только те переменные, которые указаны в АУ, и не применяются "лишние" переменные. В качестве такого примера на рис. 3 приведена ГСА2, реализующая R-триггер, в которой используются только переменные, определенные АУ : x_1 – переменная установки триггера; x_2 – переменная сброса триггера; z – выходная переменная. Даже эта ГСА (без дополнительных переменных) весьма трудно понимается, так как выдача результатов в ней происходит в конце тела П и поэтому при $x_1 = x_2 = 1$ значения z вычисляются дважды (пересчитываются), а кроме того, в ней при $x_1 = x_2 = 0$ имеется "пустой" от ОВ путь, при прохождении которого сохраняется предыдущее значение z, что осуществляется за счет запоминания во внешней относительно ГСА ячейке памяти значений z, указанных в ОВ.

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 ГС реализует последовательностный А и однотактный – в противном случае.

Как отмечалось выше, ГСА2 без "лишних" переменных встречаются крайне редко. В общем случае в этом подклассе ГСА в условных вершинах наряду с входными переменными типов X и T проверяются также и значения выходных переменных Z и отсутствующие в АУ промежуточные (внутренние) переменные Y, которые устанавливаются и сбрасываются наряду с выходными переменными в ОВ. В силу того, что обычно используются битовые переменные Y, то ГСА2 в этом случае весьма громоздки. Кроме того, они обычно не упорядочены по структуре, так как порядок расположения вершин и их пометка в ГСА 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 ОВ), 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 ОВ. В последнем случае возникают также большие проблемы с безошибочным внесением изменений в ГС.

Необходимо отметить, что если для программиста проверки переменных Y и Z

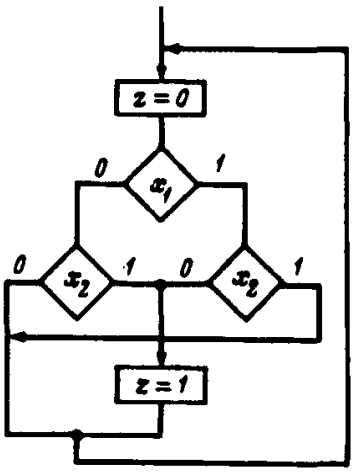

Рис. 4

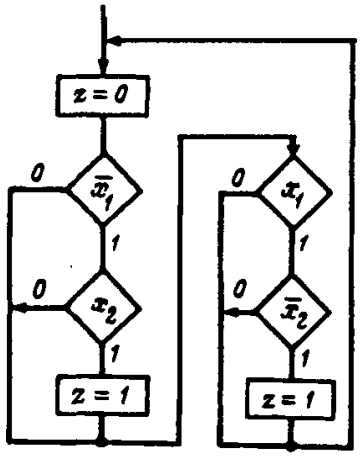

Рис. 5

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 АУ, реализуемых в том же ВУ.

Из изложенного следует, что если ГСА1 и ГСА2 без проверки переменных Y и Z в УВ отражают только семантику реализуемого АУ, то ГСА2 с такими проверками представляют собой алгоритм реализации заданного алгоритма управления в ВУ, что делает нецелесообразным использование таких ГС в качестве языка общения.

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

В этих условиях ГСА3 должна быть линеаризованной и структурированной только с помощью УК "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 А, описываемый одной из булевых формул $z = \overline{x}_1 \& x_2 \vee x_1 \& \overline{x}_2 = x_1 \oplus x_2$, имеет "плоскостную" (рис. 4), а не "линейную" структуру, то для перехода к ГСП и тексту П по ней целесообразно предварительно построить ГСА3 (рис. 5). Из рассмотрения последней ГС следует, что сложность ее понимания по сравнению с ГСА2 (рис. 4) увеличилась (даже несмотря на то, что в этих ГС 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 ВУ, в которых ввод входных переменных может осуществляться по мере необходимости (как это имеет место в некоторых типах ПЛК), эти переменные за

один проход ГСА могут не только неоднократно проверяться, но и изменять свои значения, что может приводить к нарушению функционального соответствия, задаваемого спецификацией (таблицей истинности), которое по аналогии с аппаратной реализацией может быть названо риском программной реализации. Для уменьшения степени риска для таких ПЛК должна использоваться буферизация или использоваться максимально бесповторная реализация.

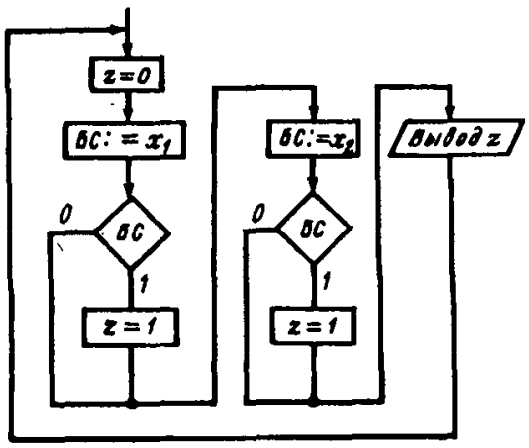

Рис. 6

Таким образом, если в ГСА1 отражается только семантика реализуемого АУ, в ГСА2, кроме семантики, учитывается возможность реализации АУ, состоящих из нескольких компонент, в одном ВУ, то в ГСА3 отражается также специфика используемых УК применяемого языка программирования. Однако ГСА3 еще значительно отличаются от ГСП, так как в последних должна отражаться также и семантика всех используемых команд или операторов применяемого языка программирования. При этом в ГСП появляются упоминания о таких архитектурных особенностях ВУ, которые в исходной ГСА не используются. Например, на рис. 6 приведена ГСП, реализующая ГСА2 (рис. 3), в которой символом БС обозначен битовый сумматор ПЛК. Из изложенного следует, что ГСП, а тем более тексты П не должны использоваться в качестве языка общения.

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

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 А, некорректна для последовательностных задач. Создание же методики, учитывающей также значения всех внутренних, а в ряде случаев и предшествующие значения выходных переменных, при неупорядоченной структуре ГСА проблематично из-за большой размерности.

По мнению автора, весь этот клубок проблем во многом связан с тем, что в ГСА, ГСП и П 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 ГСА к графам переходов (ГП) и обратно и использовать ГП в качестве языка общения и спецификации.

Определение 1. ГП, в котором в каждой вершине явно указаны значения каждой выходной переменной, назовем ГП автомата Мура (АМ) с явным заданием значений всех выходных переменных.

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

Определение 2. ГП, в котором в вершинах, кроме явно определенных значений одних переменных, используются также неявно, но однозначно определенные значения других переменных, назовем ГП АМ 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 прочерки.

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

Определение 3. ГП, в котором в вершинах, кроме явно определенных значений одних переменных, используются также неявно и неоднозначно заданные значения других переменных, назовем ГП АМ с неоднозначным заданием значений выходных переменных.

ГП этого класса могут содержать для одной и той же задачи меньшее число вершин, чем ГП АМ 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 П порождает одновременно и главный недостаток ГП этого типа – трудность их чтения и понимания. Именно по этой причине такие ГП не рекомендуется использовать в качестве языка общения.

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

Это, естественно, не исключает применение ГП для других классов А. Аналогичная классификация может быть предложена для ГП автоматов Мили (АМИ), в которых значения выходных переменных формируются не в вершинах ГП, а на дугах [36]. Во многих задачах логического управления переход от модели АМ к модели АМИ не уменьшает числа состояний (вершин ГП). На рис. 7 в качестве примера приведен ГП АМ, реализующий счетный триггер, а на рис. 8 эта же задача описана с помощью ГП АМИ. В других случаях переход от ГП АМ или ГП автомата без выходного преобразователя (АБВП) к ГП АМИ позволяет сократить число вершин. На рис. 9 приведен ГП АБВП с четырьмя вершинами, реализующий последовательностный одноразрядный сумматор, а на рис. 10 этот алгоритм описан АМИ с двумя вершинами.

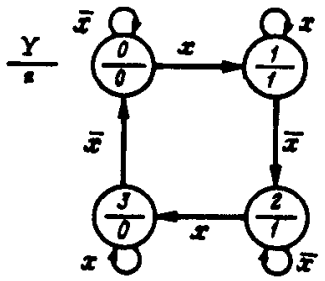

Рис. 7

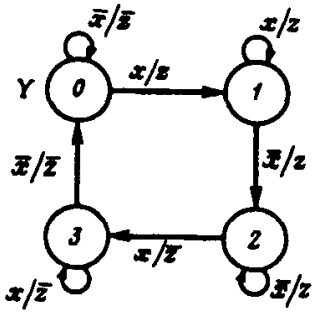

Рис. 8

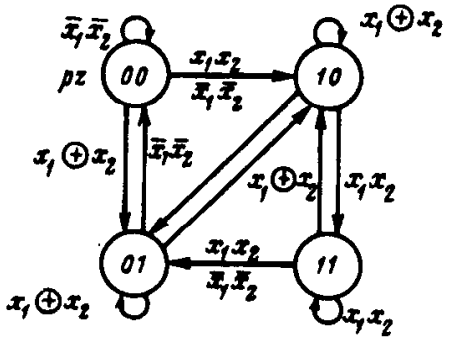

Рис. 9

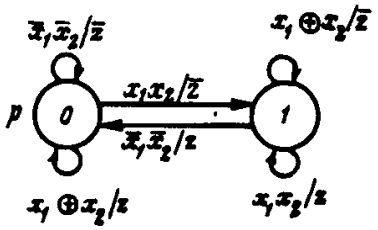

Рис. 10

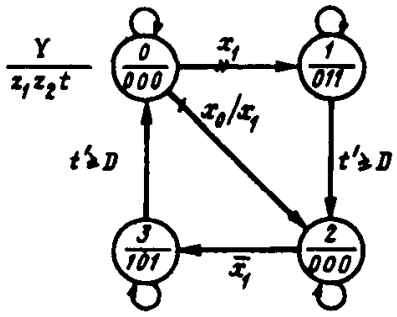

Рис. 11

При этом необходимо отметить, что если для АМ и АБВП с однозначным заданием значений выходов число состояний равно числу комбинаций этих значений (среди которых учитываются повторяющиеся), а для А, в которых все комбинации различны, равно числу различных комбинаций, то уже для А этих классов с неоднозначным их заданием понятие "состояние" становится менее связанным со значениями выходов и поэтому становится более абстрактным и менее понятным. Это положение усугубляется для АМИ, а для АМИ с неоднозначными значениями выходов в [37] вместо понятия "состояние" было введено понятие "ситуация", а вместо термина "граф переходов" – термин "граф переключений" (ГПП). Однако несмотря на то, что ГПП может содержать меньшее число вершин, чем эквивалентный ГП, применение ГПП 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 А, как смешанные автоматы (СА) – "автоматы без выходного преобразователя – Мили (СА1)" и "автоматы Мура – Мили (СА2)", а также для А всех перечисленных классов с флагами, в которых одна и та же переменная может использоваться в одном ГП 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 А и могут быть изменены не только внешним относительно рассматриваемого А источником информации, например кнопкой, но и собственно А. В ГП СА2 (рис. 11) в качестве флага используется переменная x_1, значения которой, сохраняемые во внешней относительно А битовой ячейке памяти, могут быть изменены в устойчивых состояниях А кнопкой или собственно автоматом на переходе 0 – 2, инициируемом входной переменной x_0. Значения переменной x_1 не только формируются А, но и проверяются им на переходах 0 – 1 и 2 – 3.

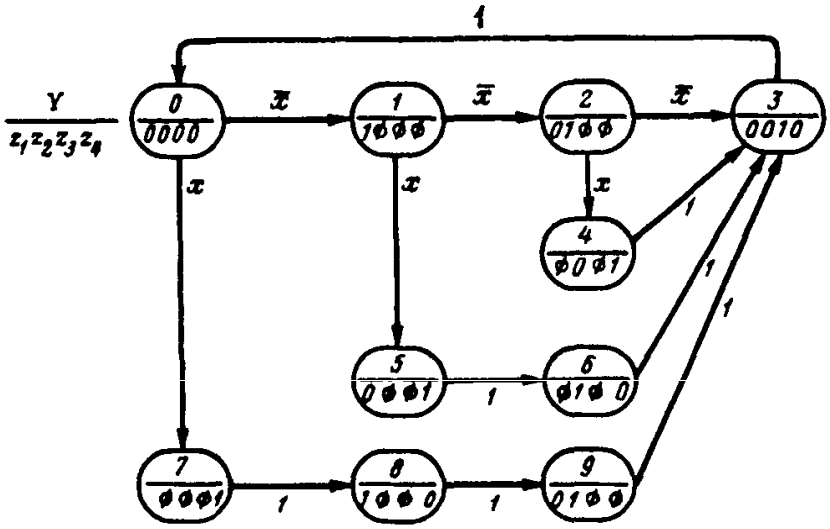

Рис. 12

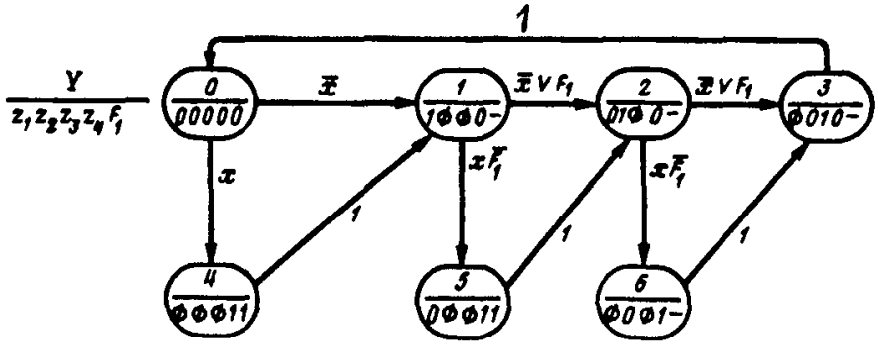

Рис. 13

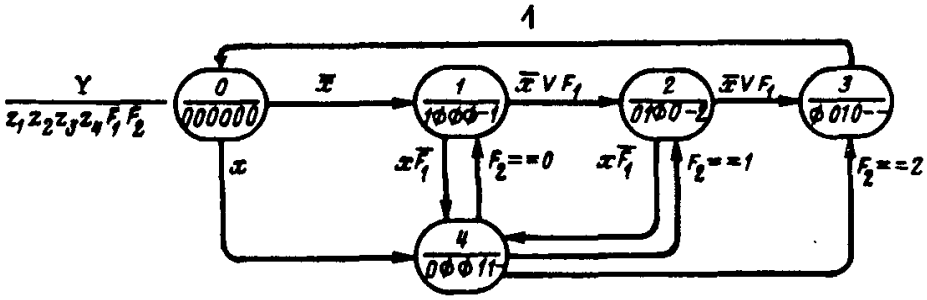

Рис. 14

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 АБВП, эти ячейки могут рассматриваться как принадлежность А наряду с ячейками, сохраняющими значения переменных, различающих состояния.

Для АМ, АМИ и СА флагами являются дополнительно вводимые переменные F, значения которых запоминаются в ячейках памяти, внешних относительно рассматриваемого А. Если при использовании многозначного кодирования состояний А любой сложности, принадлежащих этим классам, достаточно иметь в автомате только ОДНУ промежуточную переменную, то при введении флагов во внешней относительно А 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 А без флагов, но зависит от предыстории.

Таким образом, для таких A 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 ГП АМ с десятью вершинами. Число вершин в ГП АМ удается сократить до семи за счет введения двоичного флага F_1, формируемого ~~в этом случае только~~ A, и неоднозначности значений флага в некоторых вершинах ГП (рис. 13). Дальнейшее сокращение числа вершин в ГП АМ обеспечивается введением дополнительного многозначного флага F_2 и использованием умолчаний его значений (рис. 14).

Методы построения понимаемых ГП по ГСА и понимаемых ГСА по ГП, а также методы программирования рассматриваемых моделей в базисе языков различных уровней рассматриваются во второй части работы.

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 // АиТ. 1964. № 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. № 6. С. 5–12.
9. *Кузнецов О.П.* Теория алгоритмических конечноавтоматных языков // АиТ. 1981. № 3. С. 122–133. № 4. 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) // АиТ. 1985. № 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. № 2. 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. № 1. 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". С.-Петербург: 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. AI 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 // АиТ. 1975. № 9. С. 118–129.

Поступила в редакцию 10.03.95

Российская академия наук

Автоматика и телемеханика

Журнал основан в 1936 году

Выходит 12 раз в год

6

июнь

·Наука· Москва 1996