**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

А. А. Шалыто

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ ЗАДАЧ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ

**Санкт-Петербург
1998**

УДК 681.3.06:62-507
Шалыто А. А. Алгоритмизация и программирование задач логического управления. СПбГУ ИТМО. 1998.

Подход, предлагаемый в работе, позволяет помочь участникам разработки систем управления: Заказчику, Проектанту (Технологу), Разработчику, Программисту, Пользователю (Оператору) и Контролеру однозначно понимать, ***что должно быть сделано, что делается и что сделано*** *в программно реализуемом проекте. Это позволяет разделять работу и ответственность между Специалистами разных областей знаний, а также между организациями, что особенно важно при работе с инофирмами, в частности, из-за наличия языкового барьера и неоднозначности понимания естественных языков.*

Разработанный подход существенно дополняет международный стандарт IEC1131-3, распространяющийся на описания языков программирования программируемых логических контроллеров, позволяя при разных языках программирования иметь один язык спецификаций.

Излагаемая технология названа ***SWITCH-технология****, а соответствующая область программирования –* ***автоматное программирование****.*

Автор*: доктор технических наук, профессор* ***А.А. Шалыто****. Санкт-Петербургский государственный университет информационных технологий, механики и оптики (E-mail: shalyto@mail.ifmo.ru).*

ОГЛАВЛЕНИЕ

Введение 5
1. Языки описания алгоритмов логического управления 7
1.1. Классические языки логического управления 8
1.2. Традиционные языки описания алгоритмов логического управления 11
2. Графы переходов как язык алгоритмизации 16
2.1.Стратегии синтеза алгоритмов логического управления 16
2.2. Графы переходов: сущность, термины и определения 18
3. Методика построения графа переходов управляющего автомата, реализуемого программно 29
Заключение 51
Литература 54

Введение

При создании ответственных технологических объектов одной из важнейших составляющих процесса их проектирования является комплексная автоматизация технических средств.

Различная природа физических процессов, протекающих в объектах, сложный характер взаимодействия между ними и управляющими системами обуславливает трудности алгоритмизации и программирования задач управления.

Вопросы унификации аппаратного обеспечения в настоящее время решаются достаточно успешно на основе микропроцессорных и микроконтроллерных комплектов, программируемых логических контроллеров и промышленных компьютеров.

Однако при формировании общих подходов к созданию алгоритмического и программного обеспечения возникают трудности, связанные с необходимостью достижения их наглядности, структурированности, наблюдаемости и управляемости. Эти трудности усугубляются тем, что при создании различных систем управления обычно используются и различные технологии алгоритмизации и программирования. Спектр таких технологий широк: от технологий на основе алгоритмических языков высокого уровня для промышленных компьютеров до технологий на основе специализированных языков для программируемых логических контроллеров (например, функциональных и лестничных схем).

Очевидно, что алгоритмизация и программирование систем управления техническими средствами, должны основываться на единой методологии, позволяющей строить, читать, проверять, верифицировать алгоритмы и программы.

До настоящего времени алгоритмы логического (основанные на истинности и ложности) управления задаются Проектантом (Технологом) в словесной форме, и поэтому они могут быть противоречивыми и неполными. Это вносит дополнительные трудности в процесс алгоритмизации и программирования управляющих устройств, так как каждый участник разработки **(**Заказчик, Проектант (Технолог), Разработчик, Программист, Пользователь (Оператор) и Контролер**)** понимает в силу неоднозначности словесного описания алгоритмы логического управления по-своему.

По этой причине даже самый "точный" словесный алгоритм может удовлетворять каждого из участников разработки в отдельности, но, в конечном счете, являться некорректным. Ситуация усугубляется при создании систем управления, ядро которых представляет собой систему взаимосвязанных управляющих устройств. При этом, с одной стороны, совокупность таких устройств (вследствие большой размерности) становится плохо обозримой, наблюдаемой и управляемой, а с другой – плохо понимаемой Заказчиком и Проектантом (Технологом). Это во многом определяется тем, что обычно автоматизация каждого комплекса технических средств выполняется самостоятельным коллективом Разработчиков и Программистов, которые используют специфическую технологию алгоритмизации и программирования.

В настоящее время в практике проектирования систем управления, наиболее часто употребляемыми языками спецификаций алгоритмов логического управления являются язык функциональных схем и язык блок-схем алгоритмов, называемых обычно *схемами алгоритмов*. Указанные языки удобны, как правило, только для одних участников разработки, но крайне неудобны для других участников. Так, например, наиболее часто при согласовании алгоритмов управления судовыми техническими средствами применяется язык функциональных схем, который достаточно хорошо понятен только Разработчику, так как такой язык для последовательностных алгоритмов, реализуемых схемами с обратными связями, описывает их структуру (статику), но не отражает в явном виде их поведение (динамику).

В работе предлагается:

- ***при синтезе алгоритмического и программного обеспечения использовать автоматный подход;***
- ***применять графы переходов и их системы в качестве языка спецификаций;***
- ***ввести понятия "состояние" и "наблюдаемость" в программирование;***
- ***наглядно представлять поведение управляющих автоматов и композиций из них;***
- ***отображать только желаемые состояния;***
- ***отражать динамику переходов автомата из состояния в состояние;***
- ***контролировать непротиворечивость, полноту, реализуемость и отсутствие генерации в алгоритмах;***
- ***отказаться от применения умолчаний и флагов (флаг – переменная, которая в графе переходов одновременно является входной и выходной переменной);***
- ***компактно описывать алгоритмы за счет символьных пометок дуг графов переходов булевыми формулами только тех переменных, которые обеспечивают соответствующие переходы;***
- ***использовать многозначное кодирование состояний, позволяющее минимизировать число внутренних переменных в алгоритмах и программах;***
- ***достаточно просто представлять взаимодействие графов переходов за счет указания значений внутренних многозначных переменных на дугах связанных между собой графов;***
- ***применять конструкцию switch (или ее аналоги), позволяющую структурировать и модифицировать программы и обеспечивать их изоморфизм (изобразительную эквивалентность) со спецификацией (графом переходов).***

Указанные особенности графов переходов обеспечили возможность на их основе разработать методологию алгоритмизации и построения программ логического управления, которая предполагает:

- ***системное (архитектурное) проектирование;***
- ***разработку формальных спецификаций автоматов;***
- ***построение алгоритмических моделей автоматов;***
- ***программирование.***

На **первом этапе** описываются объекты управления. Выделяются источники информации и исполнительные механизмы. В дальнейшем на этой стадии определяется состав дополнительных источников информации (органов управления) и средств представления информации, например, ламп и табло. Формируется система взаимосвязанных управляющих автоматов. Каждый управляющий автомат декомпозируется на автомат и функциональные элементы задержки. При этом выделяется система взаимосвязанных автоматов. Системное проектирование завершается построением **схемы связей** “источники информации – система взаимосвязанных автоматов – функциональные элементы задержки – средства представления информации – исполнительные механизмы”.

На **этапе разработки формальных спецификаций** осуществляется выбор структурной модели для каждого автомата. Такими моделями являются: комбинационный автомат, автомат без выходного преобразователя, автомат Мура, автомат Мили, смешанный автомат, автомат с флагами и т.д. [1]. Выполняется кодирование состояний каждого автомата с памятью, и строится граф переходов, однозначно соответствующий выбранной структурной модели. При этом учитываются взаимосвязи с другими автоматами. Для каждого автомата обеспечиваются непротиворечивость, полнота, реализуемость и отсутствие генерации в контурах графа переходов, отличных от его петель. Строятся формальные спецификации функциональных элементов задержки и моделей объектов управления.

На **третьем этапе** выполняется выбор, построение и оптимизация алгоритмических моделей, реализующих формальные спецификации, с учетом рода (первого или второго) [34] принятой структурной модели каждого автомата с памятью. Отметим, что граф переходов также является алгоритмической моделью.

Этап программирования включает выбор языка программирования и собственно формализованное (на основе предлагаемых методов) и желательно изоморфное программирование по построенным алгоритмическим моделям.

На основе предлагаемого подхода для перехода от потенциальной наблюдаемости алгоритмов и программ к реальной наблюдаемости, разработана программная оболочка, позволяющая проводить моделирование в терминах автоматов.

Таким образом, использование предложенной технологии, которая получила название *SWITCH-технология* [1,2], позволяет с единых позиций проводить алгоритмизацию и программирование при создании систем логического управления и снизить число ошибок в алгоритмах и программах [3,4]. Эта технология может быть названа также *STATE-технология* или более точно *AUTOMATON-технология*, а соответствующая область программирования – *автоматное программирование.*

В работе обосновывается целесообразность применения графов переходов в качестве языка алгоритмизации и приводится методика их построения и программной реализации.

1. Языки описания алгоритмов логического управления

Актуальность разработки указанной технологии, как отмечалось выше, определяется, во-первых, необходимостью того, чтобы Заказчик, Проектант (Технолог), Разработчик, Программист, Пользователь (Оператор) и Контролер однозначно и полностью понимали друг друга, а, во-вторых, целесообразностью создания для различных типов управляющих вычислительных устройств и языков программирования единого подхода к формальному и желательно изоморфному построению хорошо "читаемых” и “понимаемых" алгоритмов и программ для задач рассматриваемого класса.

Эта проблема является актуальной и для других классов задач. Так, **Э. Дейкстра** во введении к работе [5] пишет: "*Программы могут очаровывать глубиной своего логического изящества, но мне постоянно приходилось убеждаться, что большинство из них появляются в виде, рассчитанном на механическое выполнение, и что они совершенно непригодны для человеческого восприятия. Меня не удовлетворяло также и то, что программы часто приводятся в форме готовых изделий, почти без упоминания тех рассуждений, которые проводились в процессе разработки и служили обоснованием для окончательного вида завершенной программы"*.

Продвижение в направлении решения этой проблемы для задач логического управления имеет особую важность в связи с большой ответственностью их решения для многих объектов управления, например, для ядерных или химических реакторов, а предпосылки для такого продвижения определяются наличием развитого математического аппарата теории автоматов.

В рамках разработанной технологии предлагается использовать два уровня языков – языки алгоритмизации или спецификации (языки общения) и языки программирования (языки реализации). Языки этих классов могут, как совпадать (при наличии транслятора с языка алгоритмизации), так и различаться между собой. Так, например, при аппаратной реализации систем в базисе релейно-контактных схем в качестве языка алгоритмизации применялись функциональные схемы, а в качестве языка реализации – собственно релейно-контактные

схемы. Однако плохая "читаемость", как функциональных схем, так и релейно-контактных схем, привела к необходимости использования промежуточного языка – функционально-принципиальных схем, которые в релейно-контактном виде отражают лишь алгоритм управления и не содержат другой информации, характерной для принципиальных схем (например, обозначений разъемов, гасящих сопротивлений, устройств контроля и т.д.). Однако и эти схемы, весьма удобные для представления комбинационных схем (автоматов без памяти), трудно читаются для автоматов с памятью, так как они обычно реализуют, но не отображают в своей структуре динамику переходов и смену состояний синтезируемого автомата.

Отметим, что, например, при программной реализации на базе аппаратуры "Selma-2" фирмы “ABB Stromberg” (Финляндия) в качестве, как языка алгоритмизации, так и языка программирования, применяются функциональные схемы. Для программируемых логических контроллеров "Autolog" фирмы "FF – Automation" (Финляндия) в качестве языка программирования используется язык программирования *ALPro*, в то время как язык алгоритмизации не определен. Последний не определен также и для многих других типов программируемых логических контроллеров, таких как , например, контроллеры "Melsec" фирмы "Mitsubishi Electric" (Япония), языками программирования которых являются язык инструкций и язык лестничных схем. Последний из языков является языком релейно-контактных схем, который дополнен большим числом вычислительных операций.

В настоящее время в качестве языков алгоритмизации в системах логического управления наиболее часто используются функциональные схемы и схемы алгоритмов. При этом в качестве языков программирования в зависимости от типа управляющего вычислительного устройства применяются три разновидности языков: алгоритмические языки высокого уровня (например, *СИ*), алгоритмические языки низкого уровня (языки инструкций, ассемблеры) и специализированные языки (функциональные и лестничные схемы).

Ниже обосновывается целесообразность применения в качестве языка алгоритмизации для описания алгоритмов с памятью управляющих графов – графов переходов, и предлагается единый методологический подход к их реализации в базисе языков программирования различных типов. Это позволяет для различных языков программирования иметь одно и то же алгоритмическое описание, не зависящее от типа используемого управляющего вычислительного устройства.

1.1. Классические языки логического управления

Булевы функции, таблицы истинности и таблицы решений. В системах логического управления для описания алгоритмов логического управления используются булевы функции и системы булевых функций, задаваемые в форме таблиц истинности для полностью определенных функций и таблиц решений для не полностью определенных функций. При этом таблицы истинности, описывающие автоматы с памятью, носят название кодированных таблиц переходов или кодированных таблиц переходов и выходов.

Применение таблиц истинности ограничивается задачами небольшой размерности, а использование таблиц решений – в основном автоматами без памяти – комбинационными схемами. Табличное задание автоматов с памятью ненаглядно.

Формулы и другие аналитические формы представления алгоритмов логического управления. Аналитической формой представления булевых функций являются булевы формулы и системы булевых формул, которые позволяют описывать как автоматы с памятью, так и автоматы без памяти большой размерности. Системы булевых формул могут быть изоморфно реализованы лестничными или функциональными схемами.

Иногда используются также и другие аналитические формы представления булевых функций, например пороговые [6], спектральные [7] или арифметические [8]. Основным ограничением на применение систем булевых формул для автоматов с памятью является их низкая наглядность.

Функциональные схемы. К достоинствам функциональных схем при их использовании в качестве языка алгоритмизации относятся традиционность и однозначность описания, в том числе и параллельных процессов, а к недостаткам:

- применение в большинстве случаев двоичных внутренних переменных, запоминаемых в триггерах, реализуемых виртуально, которые позволяют работать с многозначными переменными;
- отсутствие указания значений выходных и внутренних переменных в схеме;
- трудоемкость их чтения (понимания) с целью получения исчерпывающего представления о реализованном в них последовательностном процессе;
- проблема выбора тестов для их полной проверки;
- сложность гарантированного внесения изменений.

Необходимо отметить, что чтение функциональных схем представляет собой вычисления по их отдельным цепям с целью определения значений выходных переменных при различных наборах входных переменных. В этой ситуации при наличии даже сравнительно небольшого числа входов по функциональной схеме весьма трудно определить какие воздействия влияют на тот или иной переход в ней. В схемах этого класса достаточно сложно составить целостное представление о поведении даже сравнительно небольшого фрагмента схемы, содержащего триггера и обратные связи. Так, например, при трех взаимосвязанных триггерах в схеме, непосредственно по ней (без вычислений) весьма трудно определить, какое число состояний эта схема реализует, так как с помощью указанного числа триггеров может быть закодировано от трех до восьми состояний.

При этом необходимо отметить, что использование в качестве тестов соотношений “вход-выход”, обеспечивающих полноту проверки для схем без памяти, не решает проблему определения всех функциональных возможностей для схем с памятью, так как в этом случае необходимо проверять также и правильность порядка изменений выходных переменных. Однако, именно соотношения «вход-выход» и применяются при создании методик проверки на функционирование большинства систем логического управления. Это, как отмечалось выше, не обеспечивает качественной их проверки, так как такие соотношения не позволяют анализировать все имеющиеся в схеме переходы между состояниями. Более того, число состояний и переходов в схемах обычно не известно, так как они часто строятся эвристически без использования понятия “состояние”.

Функциональные схемы при их применении в качестве языка программирования обладают всеми достоинствами декларативных языков функционального программирования [9], ″основным из которых является функциональность (прозрачность по ссылкам). При этом каждое выражение определяет единственную величину, а все ссылки на нее эквивалентны самой этой величине, и тот факт, что на выражение можно ссылаться из другой части программы, никак не влияет на величину рассматриваемого выражения. Это свойство определяет различие между математическими функциями и функциями, которые можно написать на процедурных языках программирования таких, например, как Паскаль, позволяющих функциям ссылаться на глобальные данные и применять "разрушающее" присваивание. Такое присваивание может привести к побочным эффектам, например к изменению значения функции при повторном ее вызове даже без изменения значений аргументов. Это приводит к тому, что функцию трудно использовать, так как для того, чтобы определить, какая величина получится при ее вычислении, необходимо рассмотреть текущую величину глобальных данных, что, в свою очередь, требует изучения **предыстории** вычислений для определения того, что порождает эту величину в каждый момент времени”.

При определенных условиях (переобозначениях выходных переменных) в системах булевых формул, по которым функциональные схемы могут строиться, даже для автоматов с памятью удается обеспечить и другое достоинство декларативных языков – независимость результатов от порядка вычисления формул.

Временные диаграммы и циклограммы. Достоинство таких форм представления алгоритмов состоит в изображении динамики процессов, а их недостаток – в практической невозможности отражения всех допустимых значений выходных (а тем, более внутренних) переменных при всех возможных изменениях значений входных переменных даже для задач сравнительно небольшой размерности. Поэтому на практике такие диаграммы строятся в основном для описания "основного" режима, а алгоритм в целом отражается лишь в программе, которая по указанной причине строится по таким диаграммам во многом неформально.

Схемы алгоритмов. К достоинствам схем алгоритмов при их использовании в качестве языка алгоритмизации относится возможность отражения в явном виде последовательностей изменений значений входных переменных и реакций на эти изменения, представляемых в виде значений выходных переменных, в том числе и вычисляемых параллельно. При наличии двоичных значений переменных, записываемых в явном виде в операторных вершинах схем алгоритмов, резко упрощается понимание последних по сравнению с функциональными схемами.

К недостаткам схем алгоритмов относятся:

- отсутствие требований к тому, что должна отражать схема: алгоритм управления; алгоритм реализации алгоритма управления; алгоритм, учитывающий свойства управляющих конструкций используемого языка программирования; алгоритм выполнения программы;
- отсутствие требований к их организации (за исключением структурирования), обеспечивающих простоту "чтения";
- необходимость в общем случае их многократных преобразований для учета свойств управляющих конструкций языка программирования (например, линеаризация и структурирование);
- наличие внутренних (промежуточных) переменных, отсутствующих в «словесном алгоритме» логического управления, которые резко затрудняют чтение схем алгоритмов другими, отличными от Разработчика, Специалистами и, в особенности, Заказчиком;
- наличие в общем случае большого числа внутренних обычно битовых переменных, каждую из которых приходится не только устанавливать, но и принудительно сбрасывать, и которые характеризуют лишь отдельные компоненты состояний автомата, а его состояния в целом обычно не известны;
- использование битовых внутренних переменных является естественным при аппаратной реализации алгоритмов, но при их реализации с помощью языков программирования, позволяющих обрабатывать многозначные переменные, применение битовых внутренних переменных нецелесообразно;
- наличие флагов [4] и умолчаний значений внутренних и выходных переменных в операторных вершинах, которые затрудняют чтение схем алгоритмов ввиду необходимости помнить предысторию, особенно в тех случаях, когда значения переменных в этих вершинах изменяются в зависимости от путей, по которым можно "попасть" в рассматриваемую вершину;
- проверка в условных вершинах значений обычно только одиночных двоичных переменных, что приводит к громоздкости схем алгоритмов;
- связь операторных вершин через условные вершины, затрудняющая внесение изменений, так как модификация условий перехода между двумя операторными вершинами влияет на условия переходов в другие вершины.

При применении схем алгоритмов, в большинстве случаев переход от алгоритмизации к программированию для сложных задач логического управления представляет большую

проблему. Это объясняется тем, что обычно процесс алгоритмизации почти никогда не завершается тем, чем положено – созданием алгоритма в математическом смысле, который, по определению, должен однозначно выполняться любым Вычислителем. Обычно этот процесс заканчивается лишь некоторой ″картинкой″, называемой алгоритмом, которую в той или иной степени приходится додумывать при программировании (например, структурировать схему алгоритма или вводить безусловные переходы в неструктурированную программу). В этой ситуации либо Разработчик должен сам программировать, либо Программист должен знать особенности технологического процесса, либо они вместе должны при испытаниях устранять неминуемые ошибки традиционного проектирования программ.

Рассмотренный язык используется фирмой "Опто" (США) для программирования ПЛК "Mistic"[10]. Применение схем алгоритмов в форме диаграмм *Несси-Шнейдермана* [11] практически не устраняет указанных недостатков.

Логические схемы алгоритмов. Логические схемы алгоритмов, предложенные А.А. Ляпуновым [12], являются строчной формой записи линеаризованных схем алгоритмов. Они образованы буквами, которые соответствуют условным, безусловным и операторным вершинам линеаризованных схем алгоритмов, и пронумерованными стрелками, указывающими переходы, которые осуществляются при невыполнении условий.

Логические схемы алгоритмов обеспечивают компактность описания, но ненаглядны и весьма трудно строятся и читаются.

1.2. Нетрадиционные языки описания алгоритмов логического управления

Язык SDL. Идеи теории автоматов нашли свое отражение при разработке Международной комиссией по телефонии и телеграфии графического языка спецификации и описания алгоритмов – *SDL* – диаграмм (*Specification and Description Language*) [13], которые по внешнему виду напоминают схемы алгоритмов, но отличаются от последних введением в них состояний в явном виде. Недостатки *SDL* состоят в том, что они весьма громоздки и соответствуют только одному классу автоматов – автоматам Мили.

Р – схемы. Еще одна модель, базирующаяся на использовании автоматов Мили, была предложена И.В.Вельбицким [14] и носит название *Р-схемы (R-chart). Р-схема* – нагруженный по дугам ориентированный граф, изображаемый с помощью вертикальных и горизонтальных линий и состоящий из структур, каждая из которых имеет только один вход и один выход. Схемы этого класса содержат по два типа (один из которых специальный) вершин и дуг и один тип соединительных линий. *Р-схемы* образуются за счет трех типов соединений – последовательного, параллельного и вложенного.

Этот язык позволяет более компактно по сравнению со схемами алгоритмов отражать структуру последних, однако применение нестандартных обозначений и только одного типа автоматных моделей ограничивает использование таких схем.

Сети Петри и графы операций. Для описания сложных, в том числе параллельных, процессов, в 1962 г. К. Петри [15,16] была предложена графовая модель, названная его именем. Эта модель состоит из вершин двух типов – позиций и переходов, связанных между собой дугами, причем две вершины одного типа не могут быть соединены непосредственно. Для отражения динамики в сеть вводятся маркеры (метки), размещаемые в позициях. Если все позиции, связанные входящими дугами с некоторым переходом, маркированы, то этот переход срабатывает, и маркеры переходят в позиции, соединенные исходящими дугами с рассматриваемым переходом. Для целей управления С.А. Юдицким [17] было предложено применять только безопасные и живые сети Петри. В безопасных сетях Петри в позициях не может быть более

одного маркера, а в живых сетях Петри – имеется возможность срабатывания любого перехода. Назовем указанный класс сетей – управляющими сети Петри, а такие сети, в которых каждый переход имеет только одну входящую и одну исходящую дугу – автоматными сетями Петри.

В качестве модели для описания алгоритмов управления С.А. Юдицкий [31] предложил использовать графы операций, являющиеся управляющими сетями Петри, в которых позиции помечены значениями выходных переменных, а переходы – значениями входных переменных. Для описания иерархически построенных алгоритмов им было предложено применять системы вложенных графов операций.

Достоинство графов операций состоит в возможности описания в наглядной графической форме, в том числе и в виде одной компоненты, сложных алгоритмов управления, обладающих параллелизмом, а их ограничения и недостатки заключаются в том, что:

- параллельные процессы в большинстве случаев должны быть синхронизированы, в то время как для многих алгоритмов логического управления это не требуется;
- кодирование позиций и выходных переменных производится раздельно. При этом для графов операций, построенных на базе автоматных сетей Петри, используется только модель автомата Мура и невозможно применение других автоматных моделей. Это резко ограничивает изобразительные возможности графа операций;
- для кодирования позиций могут использоваться только двоичные коды. При этом число внутренних переменных (без учета их переобозначений) равно числу позиций, в том числе и для графов операций, построенных на базе автоматных сетях Петри. Отметим, что для этого класса графов операций при применении подхода, предлагаемого в настоящей работе, позиции могут быть закодированы одной многозначной переменной;
- при реализации система вложенных графов операций преобразуется в одну компоненту, в то время как при использовании предлагаемого подхода число компонент в описании и реализации может совпадать;
- позиции графов операций рекомендуется помечать не всеми значениями выходных переменных, а только теми из них, которые изменяются в соответствующей позиции

Применение умолчаний приводит к тому, что в общем случае сложности описаний алгоритма и его поведения различаются, что затрудняет чтение алгоритма и анализ всех его функциональных возможностей.

Язык "Графсет". Этот графический язык (*Grafcet* – de Graphe de Commande des Etapes et Transitions), разработанный в Центре космических исследований в Тулузе (Франция), применяется в настоящее время наряду с другими языками [18] такими фирмами, как, например, "Телемеханик" (Франция), "Сименс" (Германия), "Ален Бредли" (США), "Тошиба" (Япония), "Омрон" (Япония). Этот язык алгоритмизации при наличии транслятора является и языком программирования.

Язык "Графсет" отличается от языка графа операций, в основном, только формой изображения: квадраты вместо кружков для обозначения позиций и прямоугольники для записи значений выходных переменных, отсутствующие в графе операций. Поэтому все достоинства и недостатки этого языка сохраняются и в языке "Графсет". Созданы трансляторы с этого языка, по крайней мере, указанными выше фирмами для своих вычислительных устройств.

Одно из достоинств диаграмм ”Графсет” состоит в стандартизации их изображения. При этом диаграммы преимущественно располагаются в направлении сверху вниз. Это одновременно является и их недостатком, так как для целостного (гештальтного) восприятия ”картин” человеком более целесообразно их плоскостное изображение (как это имеет место в графах переходов), которое позволяет по этой причине отображать алгоритмы более компактно.

Язык ”Графсет” (несмотря на то, что он обладает и рядом других недостатков) входит в состав программного обеспечения *программируемых логических контроллеров*, выпускаемых ведущими в области автоматизации фирмами мира. Так, например, фирма "Сименс" обеспечивает возможность написания программ для своих контроллеров с помощью языка *STEP-5* (языки инструкций, лестничных и функциональных схем) [19], а также языка *S7-GRAF* (язык "Графсет") [31].

Следует отметить, что при наличии для одного и того же программируемого логического контроллера нескольких языков программирования Разработчики программ обычно применяют более традиционные для систем логического управления языки, такие как лестничные и функциональные схемы. Это во многом связано с недостаточностью научно-методического обеспечения использования управляющих графов для спецификации алгоритмов. Более того, в документации многих фирм лестничные и функциональные схемы обычно предлагается строить эвристически, без предварительного описания алгоритмов с помощью управляющих графов. При этом эффективность применения таких графов по сравнению, например, с лестничными схемами демонстрируется в отдельных случаях (фирма "Омрон") и только на примерах, без изложения метода формального перехода от управляющего графа к "схеме".

При этом для разных моделей контроллеров одной той же фирмы [20] предлагается использовать различные языки (для "младших" моделей лестничные схемы, а для старших – "Графсет”), в то время как единый язык спецификаций алгоритмов для всех моделей отсутствует.

Изложенное является вполне естественным, так как в рассматриваемой области до настоящего времени не установилась даже терминология. Так, например, под термином "Sequential Function Chart*" (SFC)* понимают как графы переходов, так и схемы алгоритмов (фирма "Опто") и диаграммы "Графсет" (фирма "Омрон"), а в документации фирмы "Телемеханик" отмечено, что указанные диаграммы известны, так же и под термином *SFC*. Фирма "Сименс", как отмечено выше, и вовсе применяет для тех же целей другой термин – "GRAF".

При этом отметим, что термин *SFC* не отражает главную отличительную особенность рассматриваемого языка – возможность представления в одном графе параллельных по состояниям процессов, так как слово "sequential" переводится на русский язык как "последовательный", в то время как из теории автоматов известно, что для описания последовательных (последовательностных) процессов может использоваться граф переходов детерминированного автомата, который не позволяет отображать параллельные по состояниям процессы.

Проблемно-ориентированные языки. Эти языки близки к естественным. Для целей логического управления разработано несколько проблемно-ориентированных языков, предназначенных для формального лингвистического описания алгоритмов рассматриваемого класса. Эти языки также называются [21] первичными, так как, по мнению авторов указанной работы, они прежде всего ориентированы на Заказчика и обладают развитыми изобразительными средствами и конструкциями, употребляемыми при задании условий работы на естественном языке.

Указанное достоинство этих языков влечет за собой ряд трудностей и недостатков:

- требуют от всех Участников процесса проектирования изучения синтаксиса и семантики нового, имеющего ограниченное распространение языка с достаточно большим числом конструкций;
- они построены на основе не математического, а естественного (например, русского) языка;
- обладают низкой наглядностью по сравнению с графами при отображении структуры, взаимодействия и динамики процессов;

- не позволяют формально проверять полноту, непротиворечивость, реализуемость и отсутствие генерации, а также выполнять оптимизирующие преобразования;
- требуют разработки многоуровневой системы трансляции, использующей различные типы языков, таких как базовые, автоматные, машинные;
- ориентированы на конкретный тип базового или автоматного языка и конкретный тип автоматов;
- сложно верифицируются и тестируются;
- их невозможно применять при отсутствии транслятора для используемого вычислительного устройства.

Из языков рассматриваемого класса наибольшее теоретическое обоснование получили следующие языки логического управления и их модификации – “Форум” [22], “Условие” [23], “Управление” [24], “Ярус” [25].

Например, конструкции первичного языка “Условие” сначала транслируются в базовый язык операторных схем параллельных алгоритмов с памятью, являющийся развитием языка схем алгоритмов, а затем в автоматный язык – язык функций возбуждения и выходов.

При использовании языка “Управление” в первичное лингвистическое описание (текст программы) вводятся по аналогии с разметкой схем алгоритмов, применяемой при построении автоматов [26], двоичные метки, трактуемые как состояния. По помеченному описанию строится граф переходов автомата Мили, на базе которого, в частности, решаются задачи минимизации числа состояний и параллельной декомпозиции.

Подход, наиболее близкий к предлагаемому в настоящей работе, был применен при разработке языка "Ярус". При этом О.П. Кузнецовым [27] для описания работы "пунктов" было предложено использовать автоматную модель, названную *графом переключений*, являющуюся графом переходов автомата Мили с умолчаниями не изменяющихся значений выходных переменных. Возможность сокращения числа вершин в этом графе по сравнению с классическими автоматными моделями привела к замене термина "состояние" на термин "ситуация", правда, с одновременным ухудшением читаемости графа ввиду появления зависимости от **“глубокой” предыстории** – зависимость не только от предыдущего состояния .

В отличие от изложенного подхода в предлагаемой технологии первичным и формализованным является не лингвистическое, а автоматное описание последовательностных процессов с помощью графов переходов. Тип, количество, способ кодирования и взаимосвязь графов в общем случае не фиксированы и определяются решаемой задачей. Понятность такого описания для Заказчика обеспечивается строгостью и простотой синтаксиса этого языка при описании статики, а самое главное, динамики процессов, а также обязательной разработкой схемы связей “управляющий автомат — объект управления”, которая определяет семантику каждой внешней переменной, используемой в графах переходов. Применение графов переходов без флагов и умолчаний позволяет непосредственно при описании процесса обеспечивать его корректность и использовать в дальнейшем в качестве теста для проверки формально построенной программы. За счет исключения флагов и умолчаний упрощается внесение изменений и устраняется зависимость от “глубокой” предыстории (в дальнейшем *“предыстория” – будущее зависит от настоящего и не зависит от прошлого*).

“Понятность” алгоритмов и программ, построенных по ним, еще более повышается, если устранить также и зависимость значений выходных переменных от значений входных переменных, что достигается при использовании графа автомата Мура, в котором значения выходных переменных зависят только от номера состояния, в котором автомат находится.

Граф переходов автомата этого типа либо строится непосредственно, либо получается в результате преобразования графа переходов, соответствующего автомату другого типа. Так,

например, по графу переходов автомата Мили может быть построен граф перехода автомата без выходного преобразователя, являющегося графом достижимых маркировок исходного графа, который, в свою очередь, весьма просто преобразуется в граф переходов автомата Мура.

При этом можно считать, что каждая вершина графа переходов соответствует одному состоянию автомата используемого типа, для которого граф строится, и одному состоянию оперативной памяти вычислителя, программно реализующего этот граф.

Однако, так как поведение автомата наиболее наглядно описывается не графом переходов, а соответствующим ему графом достижимых маркировок, то “реальное” число состояний автомата совпадает с их числом в этом графе или в эквивалентном ему графе переходов “классического” (без флагов и умолчаний) автомата Мура.

Таким образом, граф переходов в общем случае можно рассматривать как “закодированное” (число вершин в графе переходов может быть существенно меньше, чем число “реальных” состояний автомата) и поэтому весьма компактное описание поведения автомата. При этом наиболее “понятным” является графом переходов “классического” автомата Мура при многозначном кодировании его вершин. В этом случае граф переходов одновременно является и графом его достижимых маркировок. Именно граф переходов этого типа и предлагается применять в качестве основного языка спецификаций для задач логического управления.

В разрабатываемой технологии переход к лингвистическому описанию выполняется не при алгоритмизации, а только на этапе программирования и только при использовании алгоритмических языков. В этом случае в тексте программы должны быть по возможности сохранены **все** структурные особенности и свойства выбранной автоматной модели, что обеспечивается только при однозначном, а самое главное, **изоморфном** переходе от графа переходов, согласованного с Заказчиком, к программе. При этом программирование выполняется не по "мотивам" алгоритма, а строго математически, по принципу "один в один".

При применении предлагаемой технологии удается обеспечить соответствие между текстом программы и порядком ее выполнения и реализовать процедуру пошаговой детализации, как этого требует структурное проектирование (программирование) [28], а также использовать понятие *автомат* наряду с такими понятиями как *объект* и *класс*, характерными для объектно-ориентированного проектирования (программирования) [29].

Алгоритмические языки программирования. Из изложенного в предыдущем разделе следует, что алгоритмические языки как высокого, а тем более низкого уровня, целесообразно применять в задачах логического управления только на этапе изоморфного перехода от автоматного описания к тексту программ. Это объясняется тем, что в противном случае возникают многие из проблем, перечисленных выше применительно к использованию проблемно-ориентированных языков в качестве первичного описания алгоритмов.

2. Графы переходов как язык алгоритмизации

2.1. Стратегии синтеза алгоритмов логического управления

Возможны две стратегии построения алгоритмов логического управления. При применении первой из них считается, что известно словесное описание условий функционирования объекта и требуется построить алгоритм логического управления, обеспечивающий заданное поведение объекта.

Приведем в качестве примера фрагмент алгоритма логического управления клапаном, синтезированный указанным способом: "Для того чтобы клапан открылся, Вычислитель должен на вход открытия исполнительного механизма клапана выдать единичный сигнал".

При второй стратегии, учитывая информацию о состоянии (положении) объекта управления, строится алгоритм управления, который обеспечивает требуемое функционирование объекта. Например, "Если вычислитель выдает на вход открытия исполнительного механизма клапана единичный сигнал, то клапан открывается".

Первая стратегия "направлена" от объекта управления к вычислителю, а вторая – в обратную сторону – от вычислителя к объекту. Первая стратегия базируется на понятии "состояние", вторая – на понятии "событие".

Несмотря на то, что в настоящее время при создании алгоритмов управления, например, в форме схем алгоритмов или в виде продукций (секвенций) вида "если...то" обычно используется вторая стратегия, более естественной для рассматриваемого класса задач является первая из них. Это объясняется тем, что "состояние" по своей природе статично, а "событие" динамично. Поэтому **"управление по состояниям" является более целесообразным, чем "управление по событиям".**

Однако ни та, ни другая разновидность управления в общем случае не является исчерпывающей. **Только ”управление по состояниям и событиям”** при этом **является корректным**. Ввиду того, что оба понятия ”состояние” и ”событие” входят в понятие ”автомат”, то такой вид управления может быть назван **”автоматное управление”**, а его программная реализация – ”**автоматное программирование”.**

В предлагаемой в настоящей работе технологии *понятие "состояние" является первичным*, а *понятие "событие" – вторичным.*

Несмотря на то, что вторая стратегия приводит обычно к построению более компактных и быстрых программ, при отсутствии жестких ограничений на объем памяти и быстродействие, применение первой из них более естественно. Это объясняется тем, что такой подход соответствует основному **принципу управления**, применяемому в автоматизированных системах, который состоит в том, что при управлении *Пользователь (Пользователь) сначала определяет состояние объекта, а затем выполняет то или иное действие, порождающее возникновение события*.

При такой организации процесса управления для обеспечения простоты чтения и понимания алгоритмов и программ, они должны быть организованы также. *Весьма противоестественной является ситуация, когда управление организовано по одним принципам, а его программная реализация – по прямо противоположным*.

В рамках предлагаемой технологии управление и программирование должны быть автоматными, а **построение алгоритмов и программ должно начинаться с формирования дешифратора состояний, а не событий** [4].

При алгоритмизации состояние следует определять не по отдельным двоичным компонентам, а в целом, присваивая каждому состоянию десятичный номер, рассматриваемый как неделимая компонента описания.

Если каждому состоянию объекта сопоставить состояние управляющего автомата, которому, в свою очередь, сопоставить вершину графа переходов, а программу, реализующую граф переходов автомата, построить формально и изоморфно, то такая программа будет "понятной" не только Разработчику и Программисту, но и Заказчику, Проектанту (Технологу), Пользователю (Оператору) и Контролеру.

При этом необходимо отметить, что, несмотря на сложность построения модели объекта, она в большинстве случаев также может быть описана с помощью графов переходов. Поэтому в рамках предлагаемого подхода для проверки правильности разработанного алгоритма целесообразно выполнять моделирование комплекса "управляющий автомат – объект управления", используя для описания их поведения графы переходов. После этого алгоритм управления может и далее уточняться как на физической модели, так и на реальном объекте. Точность и детальность описания алгоритмов с помощью графов переходов резко повышают качество первоначальной алгоритмизации по сравнению с другими методами. Поэтому на объекте обычно требуется вносить сравнительно небольшое число изменений в разработанный алгоритм и реализующую его программу.

Факторы, ограничивающие широкое использование графов переходов в качестве языка алгоритмизации. Для устранения недостатков рассмотренных языков алгоритмизации предлагается применять в качестве такого языка графы переходов, предложенные более сорока лет назад для описания поведения автоматов с памятью. Графы переходов называют также диаграммами состояний или диаграммами изменений состояний.

Однако при синхронной аппаратной реализации автоматов этот язык применялся в основном для иллюстративных целей, так как в большинстве оптимизационных алгоритмов теории автоматов использовалось табличное представление графа переходов – таблицы переходов и таблицы переходов и выходов. Структура таких таблиц, требующая перечисления всех комбинаций значений всех входных переменных, резко ограничивает размерность решаемых с их помощью задач.

Другая проблема, ограничивающая применение графов переходов, была связана с тем, что при асинхронной схемной реализации из-за состязаний элементов памяти и произвольной дисциплины смены наборов входных переменных реальное поведение схемы может значительно отличаться от поведения модели (графа переходов), по которой схема строилась. Это в общем случае требует применения весьма трудоемкого противогоночного кодирования, связанного с избыточностью, часто неприемлемой при аппаратной реализации и, в особенности, релейно – контактной.

Еще одна причина состоит в том, что традиционно считалось, что графы переходов описывают только последовательностные алгоритмы логического управления, которые имеют весьма ограниченное использование в системах управления, для которых характерен параллелизм. Системы взаимосвязанных графов переходов это ограничение снимают.

Указанные трудности, а также традиции построения функциональных схем при аппаратной реализации и схем алгоритмов при программной реализации, видимо, явились причинами того,

что при программной реализации графы переходов до сих пор практически не применяются в качестве языка спецификаций процессов управления.

2.2. Графы переходов: сущность, термины и определения

Основным понятием, используемым в теории автоматов, является "внутреннее состояние автомата", которое в дальнейшем будем называть "состояние".

Это понятие, являющееся одним из основных в поведении человека (здоров – болен, сыт – голоден и т.д.), почему-то обычно не применяется в явном виде при алгоритмизации и программировании процессов управления (за исключением подхода, используемого в объектно-ориентированном анализе [30] и программных продуктах ”S7-Graph Technology Software”[31] и “Modicon State Language” [32]).

При применении других подходов внутреннее состояние Вычислителя либо не учитывается, либо не рассматривается как единое целое. При этом, как в событийно-управляемом программировании [33], проверяются лишь внешние события и выполняются действия, инициируемые этими событиями как, например, это имеет место в следующем описании алгоритма: ”Если стол накрыт, то Вычислитель должен идти обедать”. Такое описание реализуется автоматом без памяти, так как в этом случае и событие и действие являются наблюдаемыми только извне.

Из приведенного примера следует, что обычно для корректного описания алгоритма недостаточно внешней информации (”стол накрыт”), а необходимо знать также внутреннее состояние Вычислителя (сыт он или голоден). При этом описание преобразуется следующим образом: ”Если Вычислитель голоден и стол накрыт, то Вычислитель должен идти обедать”.

На первый взгляд, кажется, что произошло лишь количественное усложнение условия: вместо одной переменной стало две. Однако это не так – ситуация изменилась в принципе, так как в алгоритме появилась внутренняя переменная, которая должна находиться в памяти Вычислителя. Таким образом, наряду с ”комбинационной схемой” появляется внутренняя память, а ”схема” переходит в класс автоматов с памятью.

Применение графов переходов позволяет в явной (графической) форме **ввести понятие “состояние” в практику алгоритмизации и программирования** задач логического управления наряду с его использованием в теории конечных автоматов [34], теории линейных систем [35], теории марковских процессах [36] и в отдельных задачах практического [37] и теоретического [38] программирования.

Графы переходов позволяют в наглядной форме отразить динамику переходов автомата из одного состояния в другое при изменении входных воздействий с указанием значений всех выходных переменных, формируемых в каждом состоянии (для автоматов без выходного преобразователя или автоматов Мура) или во время каждого перехода (для автоматов Мили). Если в одном автомате используются оба способа формирования значений выходных переменных, то автомат называется “смешанным”.

Необходимо отметить, что одни и те же наборы значений выходных переменных бывает необходимо формировать в разных состояниях. Это требует введения дополнительных (промежуточных) переменных для *реализуемости автомата*, которая сводится к различению состояний.

Целесообразность применения автоматов состоит в том, что их *состояния декомпозируют все множество входных переменных на группы*, выделяя с помощью каждого состояния только то

подмножество входных переменных, которое определяет переходы из рассматриваемого состояния в соседние (смежные) состояния, в том числе и в самого себя. При этом входные переменные, не входящие в группу, определенную некоторым состоянием, не влияют на переходы из этого состояния в другие состояния – переходы из рассматриваемого состояния несущественно зависят (не зависят) от всех остальных переменных, которые не входят в группу. Это обеспечивает **возможность реализации с помощью графов переходов задач большой размерности.**

Находясь в некотором состоянии, автомат с памятью превращается в соответствующий автомат без памяти (комбинационный автомат), который по значениям входных переменных, "выбранных" этим состоянием, осуществляет выбор одного из смежных состояний, в состав которых входит и рассматриваемое.

Новое состояние "настраивает" автомат на реализацию в общем случае другого комбинационного автомата. Таким образом, автомат с памятью можно рассматривать в качестве многофункционального модуля, настраиваемого состояниями на реализацию в определенной последовательности различных ортогональных систем булевых формул, зависящих от различных групп входных переменных.

С другой стороны, автомат с памятью можно рассматривать как многофункциональный модуль, настраиваемый на реализацию с помощью входных переменных (значения которых в течение программного цикла обычно не изменяются) автономных (без входных переменных) автоматов.

Если существуют такие значения выбранных входных переменных, при которых автомат сохраняет свое состояние, то такое состояние называется устойчивым, и неустойчивым – в противном случае.

Состояниям автомата в графе переходов соответствуют вершины, а дугам между вершинами – переходы между состояниями (номенклатура составляющих графа переходов минимальна). При этом дуга, представляющая собой петлю, отражает сохранение состояния, в которой автомат находится до тех пор, пока выполняется условие, помечающее петлю. Отсутствие петли свидетельствует о неустойчивости вершины. Автомат может находиться в неустойчивой вершине только один программный цикл. Обычно дуги помечаются булевыми формулами от входных и временных (**X** и **T**) переменных. При этом единичные значения этих формул определяют выполнение переходов. Дуги, исходящие из неустойчивых вершин, помечаются единицами.

Значения выходных и временных (**Z** и **t**) переменных указываются в явном (битовом) виде в вершинах (для автоматов без выходного преобразователя и автоматов Мура), на дугах (для автоматов Мили) и в вершинах и дугах одновременно (для смешанных автоматов).

Двоичные переменные **t** управляют функциональными элементами задержки, а двоичные переменные **T** сигнализируют о срабатывании или несрабатывании этих элементов. При этом полагают, что **функциональные элементы задержки в состав автомата не входят** и являются для него одним из объектов управления. Комплекс "автомат – функциональные элементы задержки" образует единую компоненту, называемую "управляющий автомат".

При чтении графа переходов считается, что в каждый момент времени (за один программный цикл) выполняется не более одного перехода: ноль – если состояние сохраняется, и один – если совершается переход в смежное состояние, что поддерживается соответствующей программной реализацией.

Построение графов переходов. В этом разделе приводятся общие положения, связанные с построением графа переходов. В следующем разделе приведена методика их построения.

Пусть требуется построить граф переходов, описывающий поведение автомата с **n** двоичными входами и **m** двоичными выходами. Для задач большой размерности построение графа переходов обычно выполняется по словесному описанию условий работы объекта управления, в котором понятие "состояние", являющееся на этапе алгоритмизации математической абстракцией, естественно, не используется.

Однако если формализацию проводить с помощью графов переходов автоматов Мили, наиболее часто применяемых в литературе для описания примеров автоматов небольшой размерности, то это понятие (абстракцию) приходится вводить сразу для различения ситуаций, связанных с **изменениями** значений выходных переменных при одних и тех же значениях входных переменных. Построение графа переходов в "изменениях" порождает последующие трудности по их чтению (пониманию) и корректировке, так как при этом значения выходных переменных зависят не только от состояния, но и от значений входных переменных, существенных для рассматриваемого состояния.

Если же "состояние" определять как комбинацию значений **всех m** выходных переменных, причем одинаковые комбинации этих переменных рассматривать как различные состояния, то такое определение существенно менее абстрактно и более естественно, так как понятие "состояние", как бы в явном виде и не используется. При этом появляется возможность в **каждом** состоянии иметь информацию о значении **каждой** выходной переменной. Это является определяющим для простоты чтения и понимания графов переходов, а также их корректировки.

Каждая комбинация значений всех выходных переменных соответствует одной вершине графа переходов и помечает ее. Каждая вершина графа переходов соединяется дугами непосредственно с теми его вершинами, в которые должен "перейти" автомат при выполнении условий, помечающих дуги. В графе даже **соседние вершины могут быть помечены одинаково**. При этом каждому условию соответствует булева формула, а выполнению условия – равенство единице этой формулы на определенных входных наборах. Отметим, что каждой дуге соответствует одна булева формула, зависящая не от всех **n** входных переменных, а только от того их **подмножества**, которое семантически определяет переходы из рассматриваемой вершины в соседние. Это, как отмечалось выше, позволяет строить графы переходов для задач весьма большой размерности.

При такой методике построения заданные условия работы реализуются графом переходов автомата без выходного преобразователя с принудительным кодированием состояний. В дальнейшем, при необходимости (например, при наличии вершин, помеченных одинаково, и отсутствии различающих входных наборов), этот граф преобразуется в граф переходов автомата другого типа (граф переходов автомата Мура) или другого типа кодирования (граф переходов автомата без выходного преобразователя с принудительно-свободным кодированием состояний). При этом **структура первоначально построенного графа переходов не изменяется**. В отдельных случаях учет ограничений требует использования графов переходов смешанных автоматов. Это приводит к необходимости введения в граф переходов автомата без выходного преобразователя или в построенный по нему граф переходов автомата Мура значений выходных переменных (через дробь с булевой формулой, помечающей соответствующую дугу), формируемых на переходе.

При этом необходимо отметить, что граф переходов всегда может быть представлен в виде композиции, состоящей из системы булевых формул автомата без памяти (образованной формулами, помечающими дуги графа переходов) и графа переходов, дуги которого помечены одиночными буквами, каждая из которых заменяет соответствующую формулу.

Описание функционирования автоматов без памяти. Использование изложенной методики позволяет реализовать условия функционирования, не различая, соответствует ли им автомат с

памятью или автомат без памяти. Если для автоматов с памятью построение графа переходов при алгоритмизации является целесообразным, так как в явном виде отражает присущую автоматам этого класса зависимость набора значений выходных переменных, по крайней мере от состояния, то для автоматов без памяти, для которых такая зависимость отсутствует, а значения выходных переменных в рассматриваемый момент времени зависят только от значений набора входных переменных в тот же момент времени, в применении графов переходов нет необходимости. Поэтому, если удается определить, что граф переходов описывает автомат без памяти или может быть сведен к нему, то граф переходов целесообразно заменить, например, на систему булевых формул.

Свойства графов переходов. Одно из достоинств – графа переходов состоит в том, что они могут быть формально проверены на синтаксическую корректность (семантическая (смысловая) корректность, естественно, формально проверена быть не может).

Граф переходов считается корректным, если он:
- непротиворечив;
- полон;
- не содержит генерирующих контуров, отличных от петель;
- реализуем.

При этом считается, что непротиворечивость в графе переходов обеспечивается, в том случае, если в нем запрещены одновременные переходы по любым двум или более дугам, исходящим из одной вершины. При использовании системы взаимосвязанных графов переходов синхронизация процессов при необходимости осуществляется в головном графе.

Графы переходов (без флагов и умолчаний изменяющихся значений выходных переменных) в некотором смысле аналогичны параллельно-последовательным контактным схемам, для каждой из которых булева функция, описывающая ее структуру, одновременно задает и ее функционирование (поведение), в то время, как графы переходов, в которых указанные ограничения сняты, аналогичны мостиковым контактным схемам, для которых булева функция, описывающая поведение каждой из них, не задает ее структуру.

При этом, если в первом случае в графе переходов понятия ”вершина” и ”состояние” являются синонимами, то во втором – они не эквивалентны.

Графы переходов второго типа позволяют компактно описывать и реализовывать процессы, так как в противном случае пришлось бы строить и реализовывать графы переходов с большим числом состояний. При этом, однако, теряется важнейшее свойство графа переходов без флагов и умолчаний, состоящее в том, что число состояний в описании и реализации совпадает.

При формальном (и правильном) переходе от графа переходов к тексту программы это свойство при одних методах реализации может быть сохранено полностью, а при других – частично. Частичность сохранения этого свойства следует понимать в том смысле, что, например, при описании заданного графа переходов (в случае, когда число вершин в нем не равно величине два в степени) системой булевых функций, и обратном построении графа переходов по этой системе функций, в нем появятся вершины, отсутствующие в исходном задании. Однако, так как переходы из вершин заданного графа переходов в новые вершины отсутствуют, то, несмотря на наличие переходов из новых вершин в заданные, такая реализация является корректной.

При реализации графов переходов, например с помощью операторов *switch* языка *СИ*, указанное свойство ввиду их изоморфизма может быть сохранено полностью.

При неформальном переходе от графа переходов к программе поведение последней может отличаться от графа переходов, в том числе и таким образом, что, используя его в качестве теста для проверки программы, их несоответствие обнаружить, не удается. Это объясняется тем, что в графе переходов пометка практически каждого перехода не зависит от каких-либо переменных из всего множества входных переменных, которые в программе с ошибками могут стать существенными для рассматриваемого перехода. Например, если некоторый переход в графе переходов происходит при пометке *x4*, а в программе этому переходу соответствует пометка *x4&!x5*, то такую ошибку (без полного перебора или построения графа переходов по программе) обнаружить можно только случайно.

Это еще раз подтверждает высказывание **Э. Дейкстры** [5] о том, что *с помощью тестов можно обнаружить все новые и новые ошибки в программе, но нельзя доказать, что их в ней после тестирования не осталось*. Именно по этой причине в настоящей работе основное внимание уделяется построению ”понятных” Специалистам разного профиля алгоритмов и программ. Это должно позволить устранить в них многие ошибки в результате согласования на разных стадиях проектирования, включая и начальные. Формальность и изоморфность построения программы по ”понятной” спецификации, для которой также построен (и откорректирован в случае необходимости) граф достижимых маркировок, приводит к тому, что граф переходов может служить не только средством отладки, но и средством сертификации программ.

Непротиворечивость графа переходов (конъюнкция пометок любых двух дуг, исходящих из одной вершины, равна нулю) обеспечивается:

- при разновременном приходе "противоречивых" значений переменных;
- при работе с фронтами переменных;
- ортогонализацией (усложнением пометок) противоречивых дуг (например, при реализации по системе булевых формул);
- расстановкой приоритетов (учитывается порядок расположения команд в программе при реализации способом, отличным от построения системы булевых формул);
- "расщеплением" вершин с противоречивыми дугами (увеличение числа состояний автомата).

Полнота графа переходов (дизъюнкция пометок всех дуг, исходящих из вершины, равна единице) проверяется после обеспечения непротиворечивости. При реализации графа переходов с помощью системы булевых формул должны быть помечены все дуги, исходящие из каждой вершины, а при других вариантах реализации пометки петель для автомата без выходного преобразователя или автомата Мура могут умалчиваться. При этом предполагается, что пометка петли в вершине обеспечивает "полноту" последней.

В графе переходов существуют генерирующие контуры, если, по крайней мере, в одном из них конъюнкция пометок всех дуг, которые его образуют, не равна нулю. Устранение генерирующих контуров осуществляется теми же методами, что и устранение противоречивости (за исключением расстановки приоритетов).

Реализуемость графа переходов обеспечивается кодированием вершин, выполняемого для их различения.

Кодирование состояний автоматов. Для реализации графа переходов его вершины (состояния) должны иметь различные пометки (коды).

Если в графе переходов автомата без выходного преобразователя все вершины имеют различные пометки (значения выходных переменных), то эти же пометки (в целом или отдельные компоненты, их различающие) могут использоваться в качестве кодов состояний автомата. Этот способ кодирования будем называть принудительным.

Если в графе переходов автомата без выходного преобразователя пометки некоторых вершин совпадают, то для их различения вводится минимально необходимое число дополнительных (промежуточных, внутренних) переменных y_i, значения которых различают одинаковые вершины. Этот вид кодирования будем называть принудительно-свободным.

В автоматах Мура, Мили и смешанных автоматах применяется свободное кодирование, при котором коды вершин графа переходов выбираются независимо от значений выходных переменных, связанных с этими вершинами.

Из всех видов свободного кодирования при программной реализации автоматов наиболее целесообразно использовать многозначное (целочисленное) кодирование.

При этом **I**-му графу переходов в целом присваивается **одна** многозначная переменная Y_I, **j**-е значение которой, в свою очередь, присваивается **j**-й вершине графа переходов. Это обеспечивает реализацию алгоритмов **с минимально возможным числом дополнительных переменных**. Именно этот вид кодирования и обеспечивает наилучшее чтение программ. Другое достоинство этого вида кодирования состоит в том, что предыдущее значение многозначной переменной **нет необходимости сбрасывать принудительно**, так как происходит ее автоматический сброс при переходе к другому значению этой переменной. При этом отметим, что при наличии одной внутренней переменной в одном графе переходов, состязания элементов памяти отсутствуют, так как этой переменной не с чем "состязаться".

Более того, при корректной организации вычислительного процесса состязания “элементов” памяти отсутствуют при любом виде кодирования: программа может либо правильно, либо неправильно реализовывать заданный алгоритм, так как в отличие от асинхронной схемы она не может вести себя по разному в зависимости от реальных ”задержек элементов". Назначение порядка выполнения команд в программе является своего рода синхронизацией. Например, если в качестве языка программирования применяется язык функциональных схем и в базисе этого языка построена некоторая схема, то при одной нумерации (порядке реализации) элементов – схема будет иметь одно полностью детерминированное поведение, а при другой нумерации – другое также полностью детерминированное поведение. При этом ни то, ни другое поведение может не соответствовать желаемому.

Поэтому при программной реализации автоматов для любого вида кодирования состояний, использующего в том числе не соседние наборы переменных, при формальном и правильном переходе от графа переходов к программе она в "медленной тактности" (после завершения процесса однократного вычисления по ней) будет функционировать в соответствии с графом переходов, несмотря на то что в "быстрой тактности" (в ходе процесса однократного вычисления) значения переменных могут отличаться от желаемых.

При этом неприятности в отличие от асинхронных схем не возникают, так как промежуточные значения каждой переменной, вычисленные внутри программного цикла, фильтруются. Пусть, например, при непосредственном переходе из состояния графа переходов с кодом **00** в состояние с кодом **11** имеет место переключательный процесс **00 – 10 – 11**, в котором промежуточное значение **10** фильтруется. Переход через состояние с кодом **01** при программной реализации с помощью системы булевых формул в отличие от асинхронных схем невозможен, так как порядок изменения значений переменных однозначно определяется порядком расположения формул в системе.

Особенности использования графов переходов. При реализации алгоритма логического управления одним графом переходов автомата Мура или автомата без выходного преобразователя и **формальном** переходе к тексту программы по этому графу без флагов и умолчаний, являющемуся также графом достижимых маркировок, полностью описывающим

поведение автомата, граф переходов может служить также и тестом для проверки правильности программ.

Если программа в этом случае построена по графу переходов не только формально, но и **изоморфно** (обеспечена изобразительная эквивалентность между графом переходов и тестом программы), то тестирование может быть заменено сверкой ее текста с графом.

Если граф переходов содержит флаги, то для полного анализа его поведения должен строиться *граф достижимых маркировок*, который в дальнейшем может использоваться в качестве сертификационного теста программы.

Взаимодействие между графом переходов в системе взаимосвязанных графов переходов может осуществляться по входным, выходным, а, самое главное, внутренним переменным, кодирующим вершины графов, что обеспечивает большую наглядность и **исключает необходимость применения для этой цели дополнительных внутренних переменных.** При этом алгоритм управления может быть представлен в виде головного и вызываемых графов, а также в виде параллельно работающих компонент. Системы взаимосвязанных графов переходов могут быть построены и по принципу вложенности.

В вершинах (как и в диаграммах "Графсет") и на дугах графов переходов могут не только устанавливаться и сбрасываться двоичные переменные, но также могут запускаться процессы, описанные, как с помощью графов переходов, так и иным образом.

Для анализа поведения (всех функциональных возможностей) произвольной системы графов переходов, даже в случае, когда каждый из них не содержит флагов и умолчаний, должны строиться один (для случая, когда все графы переходов системы взаимосвязаны) или несколько (для случая существования в системе не связанных между собой групп графа переходов) графов достижимых маркировок.

Основные этапы алгоритмизации при использовании графов переходов. Разрабатывается схема связей “источники информации — управляющие автоматы — средства представления информации — исполнительные механизмы”.

Каждый управляющий автомат декомпозируется на автомат и функциональные элементы задержки. Это позволяет **исключить время из модели**, оставив только битовые переменные **t,** предназначенные для запуска функциональных элементов задержки (выходы автомата), и битовые переменные **T**, сигнализирующие о срабатывании этих элементов (входы автомата). При этом предыдущая схема преобразуется в схему связей “источники информации — автоматы — функциональные элементы задержки — средства представления информации — исполнительные механизмы ”.

При необходимости автомат эвристически декомпозируется на систему взаимосвязанных автоматов меньшей размерности. Декомпозиция может производиться по режимам, объектам или смешанным образом.

Построение схемы связей завершает стадию архитектурного (системного) проектирования.

Рассматривая **i-**й автомат вместе с управляемыми им функциональными элементами задержки в качестве **i-**го управляющего автомата, можно считать, что схема связей в этом случае содержит систему взаимосвязанных управляющих автоматов.

Для каждого автомата осуществляется выбор структурной модели (комбинационный автомат, автомат без выходного преобразователя, автомат Мура, автомат Мили, смешанный автомат и т.д.). Выполняется кодирование состояний автоматов с памятью.

Строя корректный граф переходов, однозначно соответствующий выбранным структурной модели и варианту кодирования состояний, для автомата с памятью, входящего в состав каждого управляющего автомата, и объединяя построенные графы в систему, получим систему взаимосвязанных графов переходов, которая является формальной спецификацией – алгоритмом управления. На этой стадии строятся также формальные спецификации для функциональных элементов и моделей объектов управления. Построение спецификаций завершает вторую стадию проектирования управляющей программы.

На третьей стадии осуществляется выбор, построение и оптимизация алгоритмических моделей, реализующих формальные спецификации, с учетом рода (первого и второго) принятой структурной модели каждого автомата с памятью, например, автомата Мура второго рода [34].

Предлагаемая технология включает методы формализованного перехода от графа переходов к различным типам алгоритмических моделей, основные из которых следующие:

- системы булевых формул;
- функциональные схемы;
- лестничные схемы;
- схемы алгоритмов без внутренних обратных связей.

При этом, естественно, что и граф переходов также являются алгоритмической моделью, перед программированием которой также необходимо знать род выбранной структурной модели.

Выбор той или иной алгоритмической модели зависит от используемого языка программирования. При этом для некоторых языков, например *СИ*, может применяться любая из перечисленных моделей, а для других, например функциональных схем, число таких моделей ограничивается одной.

После выбора алгоритмических моделей для реализации формальных спецификаций осуществляется переход к последней (четвертой) стадии предлагаемой технологии – программированию. На этой стадии после выбора языка программирования до написания программы для каждой алгоритмической модели должен осуществляться выбор программной модели, содержащий, в частности, перечень используемых операторов, например, операторов `switch`.

Программирование. При применении каждого языка программирования формально построенная по графу переходов (непосредственно или используя другие модели) программа может либо быть, либо не быть изоморфной по своей структуре графу переходов, по которому она строилась.

В первом случае **доказательство** эквивалентности программы с графом переходов, по которому она строилась, может проводиться их сопоставлением. При этом граф переходов всегда может быть восстановлен непосредственно по тексту программы без дополнительных вычислений.

Во втором случае читать программу трудно, но в этом и нет необходимости, так как она **формально** строится по графу переходов, который и следует читать. Проверка программы в этом случае может производиться, используя граф переходов в качестве теста, по "схеме": *настоящее состояние, вход – следующее состояние, выход*. Верификация, состоящая в построении графа переходов, в этом случае существенно более трудоемка и связана с вычислениями. Изоморфизм текста программы с графом переходов, естественно, обеспечивается за счет избыточности, использование которой невозможно при жестких ограничениях на объем памяти.

Главная особенность предлагаемых программных реализаций состоит в том, что **за один цикл работы в программе выполняется не более одного перехода в каждом графе переходов**. В противном случае, если, например, для некоторой вершины графа переходов автомата Мура условия, помечающие одну из входящих и одну из исходящих (кроме петли) дуг, выполняются, то значения выходных и внутренних переменных, которые должны быть сформированы в этой вершине, будут отфильтрованы (пропущены).

Название предлагаемой технологии порождено оператором `switch` языка СИ, так как ее использование позволяет наиболее просто переходить от построенного графа переходов к изоморфному по структуре тексту программы. Оператор `switch`, осуществляющий многовариантный (многозначный) выбор, обеспечивает в данном случае декомпозицию автомата по его внутренним состояниям.

Программная и методическая поддержка технологии. Возможность быстрого, безошибочного и изоморфного перехода от графа переходов к тексту программы на языке высокого уровня, например *СИ*, позволяет резко упростить отладку и моделирование комплекса "управляющие автоматы – объекты управления", описав не только автоматы, но и **модели объектов управления (**по компонентам и (или) режимам**)** с помощью графов переходов.

Для этого была разработана программная оболочка, позволяющая для системы вложенных графов переходов, состоящей из **N** графов, реализованных на языке *СИ*, изменять с помощью клавиатуры значения входных переменных. Оболочка также позволяет наблюдать на дисплее значения выходных, временных, а самое главное, **N** внутренних переменных (по **одной** внутренней переменной для каждого графа переходов), как в пошаговом, так и автоматическом режимах.

Возможность постоянного чтения на дисплее десятичного номера состояния каждого графа переходов в каждом программном цикле с помощью всего лишь одной многозначной переменной делает программу полностью **наблюдаемой** и **управляемой**. Это принципиально отличает предлагаемую технологию от известных, при использовании которых всегда имеется возможность вызова с помощью отладчика на дисплей любой переменной и слежения за изменениями ее значений. Однако при этом практически всегда остаются не ясными ответы на следующие вопросы:

- какие переменные следует применять в программе для обеспечения ее работоспособности и управляемости?
- сколько таких переменных (особенно внутренних) следует применять?
- что характеризует каждая переменная?
- какой набор переменных следует выводить на дисплей на каждом этапе отладки?
- какие переменные следует дополнительно ввести в программу и представить на дисплее, если на некотором этапе отладки ее работоспособность еще не обеспечена?

Решение этих вопросов для задач рассматриваемого класса резко упрощается, если уже на стадии разработки алгоритма и (или) его фрагментов регулярным образом ввести в них состояния, а не вводить нерегулярным образом отдельные переменные, отражающие компоненты состояний, в ходе всего процесса разработки программы.

Если в качестве языка программирования применяется алгоритмический язык высокого уровня, например СИ, то после построения алгоритма логического управления по предлагаемой технологии, включая его сертификацию и моделирование с помощью указанной оболочки, разработка программы завершается. При использовании других типов языков программирования после этого осуществляется формализованный (ручной или автоматический) переход к тексту программы на применяемом языке.

Например, для программируемых логических контроллеров "Autolog" фирмы "FF-Automation" Б.П. Кузнецовым и А. А. Шалыто разработан транслятор "язык *СИ* – язык *ALPro*", позволяющий по тексту структурированной программы, написанной по графу переходов на некотором подмножестве языка *СИ*, автоматически получать программы на языке инструкций *ALPro*.

При ограничениях на внутренние ресурсы программируемых логических контроллеров разработана методика "ручной" формальной реализации графа переходов в базисе языка *ALPro*. Разработаны также методики реализации графа переходов в базисе таких языков программирования, как лестничные и функциональные схемы. Одна из этих методик позволяет, в частности, строить функциональные схемы, изоморфные графу переходов, в базисе библиотечных элементов для системы "Selma – 2" [39].

Понятие “состояние” и теория управления. В шестидесятые годы был развит качественно новый подход к теории линейных систем, который стимулировался интересом к теории конечных автоматов, что, в свою очередь, привело к появлению новых идей и методов, концентрирующихся вокруг понятия "состояние". При этом метод, получивший название “метод пространства состояний” [35], начал играть центральную роль в теории управления при изучении линейных, нелинейных и дискретных (квантованных) систем.

Основное свойство состояния системы в момент времени $\mathbf{t_0}$ заключается в "отделении" будущего ($\mathbf{t} > \mathbf{t_0}$) от прошедшего ($\mathbf{t} < \mathbf{t_0}$) в том смысле, что состояние несет в себе всю информацию о прошлом системы, необходимую для определения реакции системы на любое входное воздействие, подаваемое в момент $\mathbf{t_0}$.

Для систем, описываемых дифференциальными уравнениями, знание определенного числа производных входа и выхода в момент времени $\mathbf{t_0}$ дает всю информацию о прошлом системы. Поэтому для подобных систем состояние в момент времени $\mathbf{t_0}$ является вектором, в качестве компонент которого используются производные различных порядков для входа и выхода, взятые в момент времени $\mathbf{t_0}$.

Отметим, что применение уравнений “вход-выход” для описания поведения систем с памятью является недостаточным, так как в этом случае приходится пользоваться отношением "вход-выход", а не соответствующим оператором. (Отношение – это множество упорядоченных пар "вход-выход", а оператор – отношение, в котором каждому значению входа соответствует единственное значение выхода). Необходимость использования операторов, а не отношений, приводит к применению уравнений "вход – состояние – выход" и уравнений "вход – состояние – следующее состояние", которые могут быть записаны, в том числе и в дифференциальной форме.

Одним из важнейших понятий, используемых в методе пространства состояний, является “измеримость”. В работе [35] *измеримым* называется такое состояние, которое экспериментатор может определить для каждого момента времени **t** либо непосредственно, либо через известные значения входного и выходного сигналов, не зная начального состояния. Другими важными понятиями, введенными Р. Калманом в работе [40], являются “управляемость” и “наблюдаемость”.

“Управляемость” означает, что, зная начальное состояние и матрицы, характеризующие рассматриваемую систему, можно найти вход, который переводит это состояние в начальное состояние за конечное время. Система управляема в том и только в том случае, если любое состояние достижимо из любого начального состояния.

Понятие "наблюдаемость" идентично понятию определимость начального состояния.

Из изложенного следует, что понятие "наблюдаемость", введенное в работе [1], эквивалентно понятию "измеримость" (по Заде).

Понятие "управляемость" (по Калману) эквивалентно понятию "сильная связность" в теории автоматов, введенному Муром [41].

Понятие "управляемость", применяемое в настоящей работе, является более широким и включает в себя наряду с указанным свойством, также и возможность корректного внесения изменений.

3. Методика построения графа переходов управляющего автомата, реализуемого программно

Изложение методики будем иллюстрировать примером построения графа переходов, реализующего алгоритм управления и контроля трехпозиционным клапаном (**Кл.**) с памятью с помощью трех кнопок без памяти. Наличие памяти в исполнительных механизмах объекта управления (клапана) позволяет снимать с них управляющие сигналы после того, как клапан откроется или закроется, сохраняя это положение.

Приведем словесное описание алгоритма управления клапаном.

1. При нажатии кнопки ***"Откр."*** клапан начинает открываться.
2. После его открытия срабатывает сигнализатор открытого положения, зажигается лампа "***Откр.***" и управляющий сигнал с клапана снимается.
3. При нажатии кнопки "***Закр.***" клапан начинает закрываться.
4. После его закрытия срабатывает сигнализатор закрытого положения, зажигается лампа ***"Закр."*** и управляющий сигнал с клапана снимается.
5. Если в течение трех секунд клапан не откроется или не закроется, то управляющий сигнал с клапана снимается и зажигается лампа контроля "***Неисправность***".
6. Сброс сигнала контроля осуществляется нажатием кнопки "***Разбл.***" (Разблокирока).

Методику построения графа переходов управляющего автомата представим в виде последовательности шагов.

1. Строится схема связей "источники информации — управляющий автомат — средства представления информации — исполнительные механизмы", являющаяся в общем случае схемой с обратными связями, как это принято в теории автоматического управления. Возможны варианты этой схемы, в которых, например, средства представления информации не применяются.

Каждая информационная связь в схеме помечается переменной. При этом считается, что входные переменные управляющего автомата принадлежат множеству двоичных переменных **X**, а его выходные переменные – множеству двоичных переменных **Z**.

Схема может содержать также краткие комментарии, поясняющие смысл используемых переменных и свойства органов управления, исполнительных механизмов и собственно объекта управления (рис.1).

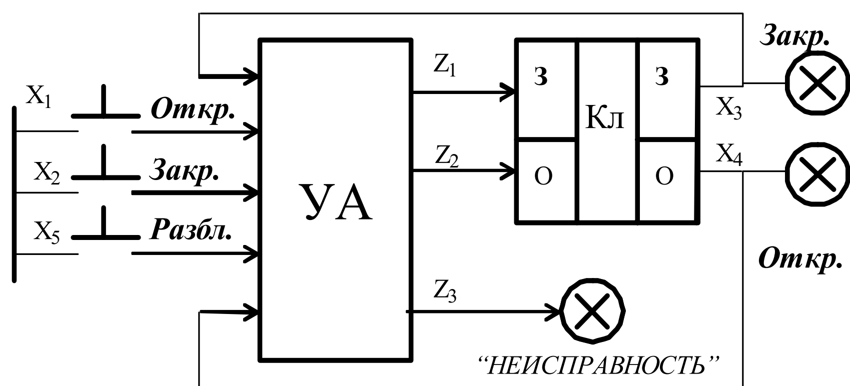

Рис. 1

2. Вводится понятие "состояние" объекта управления. Определяются и перечисляются (классифицируются) его состояния. Это формирует пространство состояний объекта. Состояния объекта управления могут быть как устойчивыми, так и неустойчивыми. Ввиду того, объекты управления при автоматизации технологических процессов обычно весьма инерционны по сравнению с программным циклом управляющего вычислительного устройства, то поэтому они не только в устойчивых, но и в неустойчивых состояниях могут находиться достаточно "долго". Например, для объекта клапан устойчивыми состояниями являются состояния "закрыт" и "открыт", а неустойчивыми – "открывается" и "закрывается" (рис.2).

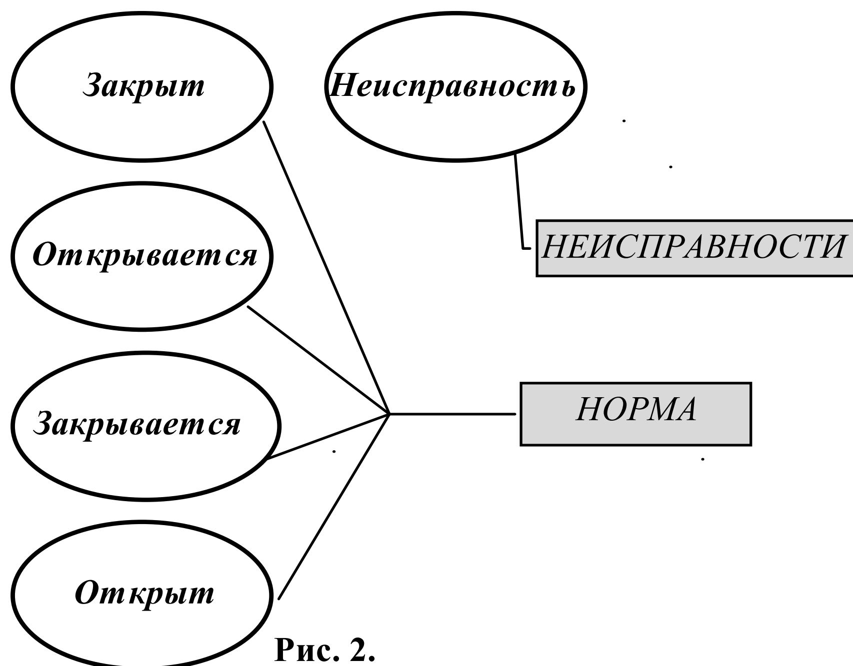

Рис. 2.

Таким образом, в предположении об исправной работе клапана в его графе переходов достаточно использовать только эти состояния.

В случае, когда требуется учесть возможные неисправности объекта, его *пространство состояний должно быть расширено.*

При этом если выбор числа "нормальных" состояний является в некотором смысле объективным, то выбор числа дополнительных состояний субъективен, и зависит от желания Разработчика алгоритма. С увеличением числа дополнительных состояний объекта "разрешающая" способность создаваемого алгоритма увеличивается. При этом, чем больше нюансов Разработчик желает отразить в графе объекта, а в дальнейшем и в алгоритме управления, тем больше состояний должно быть выделено. Предполагая, например, что неисправность клапана может состоять либо в его *неоткрытии*, либо в *незакрытии*, в граф объекта может быть введено лишь одно дополнительное состояние "неисправность". Если же принято решение отличить *неоткрытие* от *незакрытия*, то вместо одного состояния должно использоваться два состояния, каждое из которых соответствует "своей" неисправности. Таким образом, множество состояний объекта может состоять из двух подмножеств: "норма" и "неисправности" (рис.2).

3. Каждому состоянию объекта сопоставляется вершина в графе объекта. Вершины располагаются на плоскости и для идентификации нумеруются десятичными числами от **0** до **S – 1**, где **S –** число выделенных состояний объекта. При этом можно считать, что числа являются значениями одной многозначной переменной **S** (рис.3).

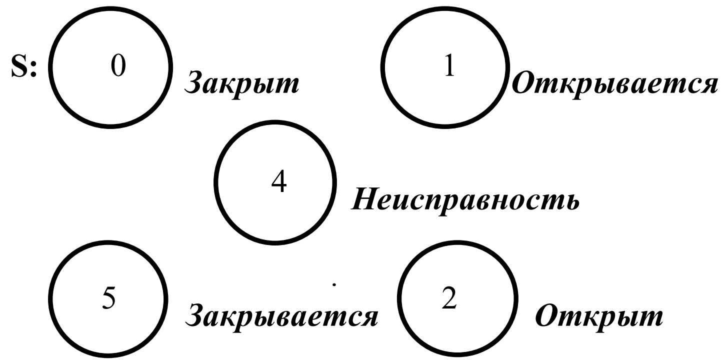

Рис.3

4. Каждая вершина графа объекта через дробь с кодирующей ее цифрой помечается кортежем значений двоичных переменных, являющихся подмножеством множества **X**, которые формируются сигнализаторами объекта в этом состоянии. Пометка вершин из подмножества ”норма” определяется свойствами объекта, а вершин из подмножества ”Неисправности” – решением Разработчика о том, какая информация характеризует соответствующее состояние. При этом различные вершины графа объекта могут быть помечены одинаковыми кортежами (рис.4).

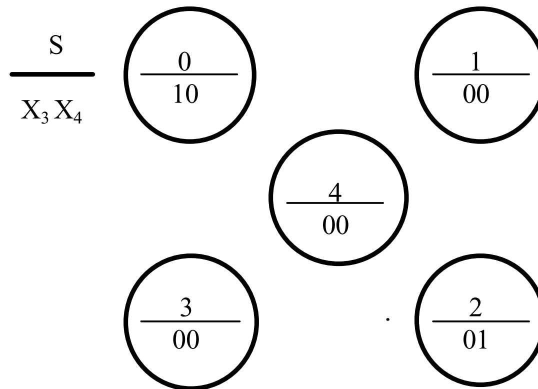

Рис.4

5. Определяются все допустимые переходы между состояниями объекта, что отражается введением соответствующих дуг в граф объекта. Дуги между вершинами из подмножества ”норма” вводятся в соответствии со свойствами объекта, а дуги между вершинами из подмножеств ”норма” и ”неисправности” вводятся Разработчиком в зависимости от принятых на втором и четвертом шагах решений об идентификации неисправностей и решений о том, что делать после их устранения. При этом для каждой вершины графа, соответствующей устойчивому состоянию объекта, вводится петля (рис.5).

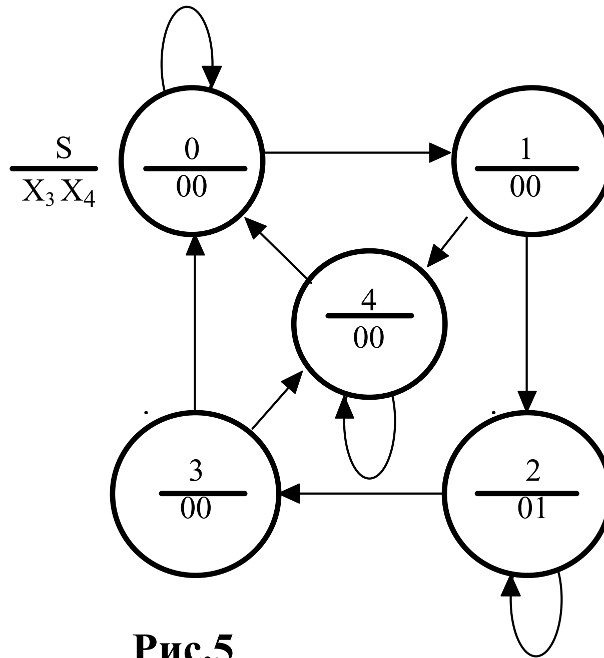

Рис.5

6. Каждая дуга и петля в графе объекта помечается конъюнкциями переменных или их инверсий из подмножества множества Z, которые соответствуют значениями переменных, подаваемых на входы исполнительных механизмов объекта и средств представления информации (рис.6). В этом графе в неустойчивых вершинах исходящие дуги могут быть помечены одинаково. На этом завершается построение графа объекта.

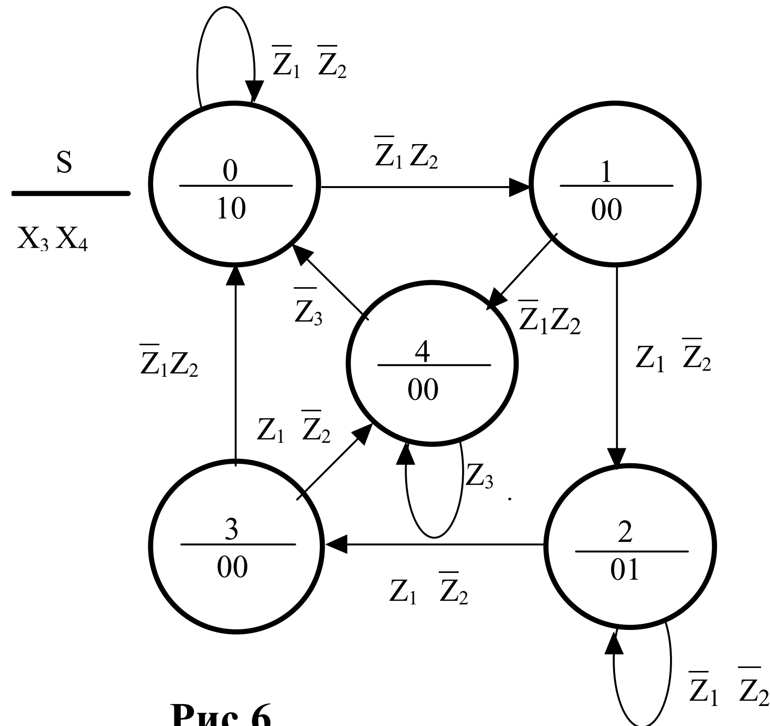

Рис.6

7. По графу объекта строится модель объекта, которую будем называть графом переходов функционирования модели (граф модели). В графе модели полностью сохраняется "скелет" графа объекта и пометки его вершин. В этом графе все вершины должны быть устойчивыми, и поэтому каждая из них должна иметь инцидентную ей петлю. В каждую вершину к значениям переменных из множества **X** добавляются значения двоичных переменных из множества **t**, управляющих функциональными элементами задержки, которые моделируют времена пребывания объекта в неустойчивых состояниях. Количество этих переменных может быть

сведено к одной, если можно сделать предположение о том, что все переходные процессы в объекте имеют одинаковую длительность.

Дуги графа модели, исходящие из вершин, соответствующих устойчивым вершинам графа объекта, помечаются прямыми или инверсными переменными из множества **Z**, входящими в состав конъюнкций, инициирующих соответствующие переходы в графе объекта.

Каждая дуга графа модели, исходящая из вершины, соответствующей неустойчивой вершине графа объекта, помечается конъюнкцией, состоящей из двоичной переменной из множества **T**, которая определяет факт срабатывания функционального элемента задержки, и минимального числа двоичных переменных или их инверсий из подмножества множества **X,** характеризующих состояние объекта в смежной устойчивой вершине.

При необходимости осуществляется пометка петель вершин графа модели за счет обеспечения полноты переходов из каждой вершины (рис.7).

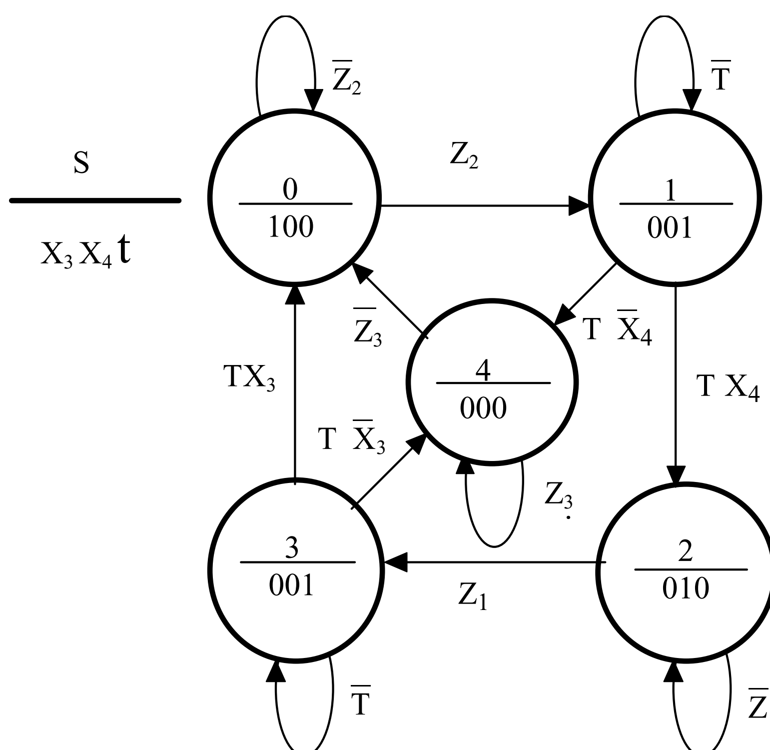

Рис.7

8. Построение графа переходов автомата начинается с анализа технического задания с целью определения необходимости применения функциональных элементов задержки в алгоритме управления. Если в этих элементах нет необходимости, то строится новая схема связей, отличающаяся от исходной заменой словосочетания "управляющий автомат" на слово "автомат."

В случае, когда в задании упоминаются временные задержки, строится новая схема связей, отличающаяся от исходной тем, что в ней управляющий автомат декомпозирован на автомат и функциональный элемент задержки, число которых равно числу различных временных задержек, упоминаемых в задании. При этом для автомата вводится два множества двоичных переменных **t** и **T**, описанных в предыдущем пункте, первое из которых для автомата является выходным, а второе – входным (рис.8).

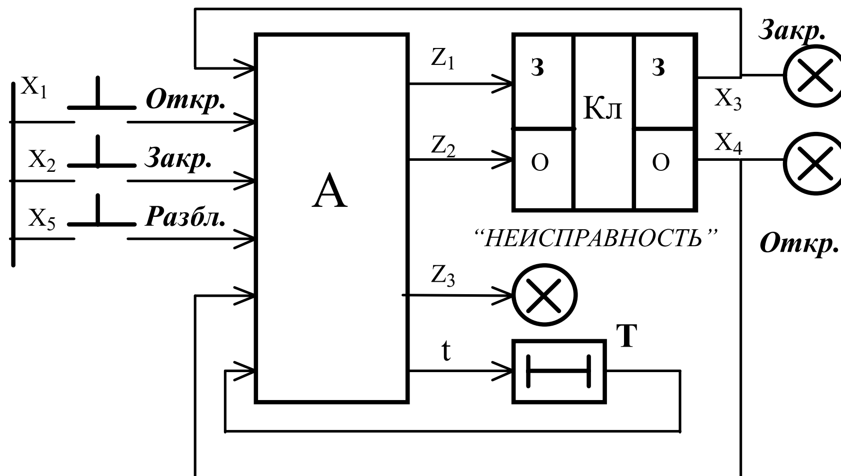

Рис. 8

Из изложенного следует, что в рамках предлагаемой методики функциональные элементы задержки для автомата рассматриваются в качестве внешних устройств наряду с объектами управления, рассматриваемыми совместно с их исполнительными механизмами и сигнализаторами.

Это позволяет при алгоритмизации процесса управления применять такую *математическую модель (граф переходов), в которой время в явном виде не используется.*

9. Каждому состоянию объекта сопоставляется состояние автомата, управляющего объектом. На плоскости изображаются S вершин графа переходов автомата. В качестве комментария (рис.9) у каждой вершины может быть написано название соответствующего ему состояния объекта управления.

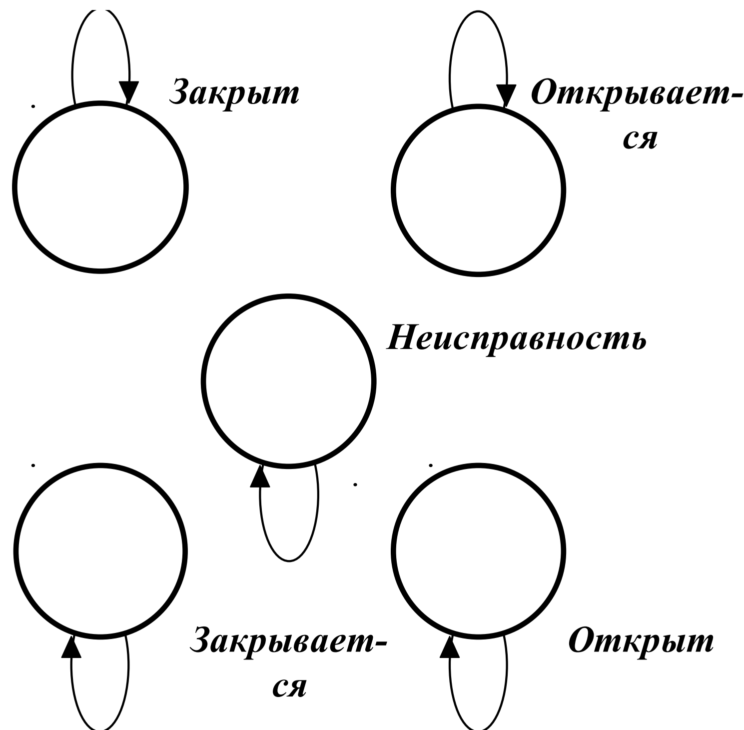

Рис.9

Так как в технологических процессах обычно используются весьма инерционные объекты управления, то, как устойчивым, так и неустойчивым состояниям объекта должны соответствовать только устойчивые состояния автомата управления. При этом каждому

состоянию автомата должна соответствовать вершина графа переходов, которой инцидентна петля.

10. Для каждой вершины графа переходов автомата (граф автомата) определяется кортеж значений всех его выходных переменных, образующих множество **Z**, который обеспечивает пребывание объекта в состоянии (устойчивом или неустойчивом), соответствующем рассматриваемому состоянию автомата.

При этом пометки вершин графа автомата, соответствующие устойчивым вершинам графа объекта, определяются пометками дуг последнего, а пометки вершин графа автомата, соответствующие неустойчивым вершинам графа объекта, определяются пометками входящих и исходящих дуг из рассматриваемой вершины графа объекта.

Найденные кортежи значений выходных переменных используются для пометки вершин графа автомата (рис.10).

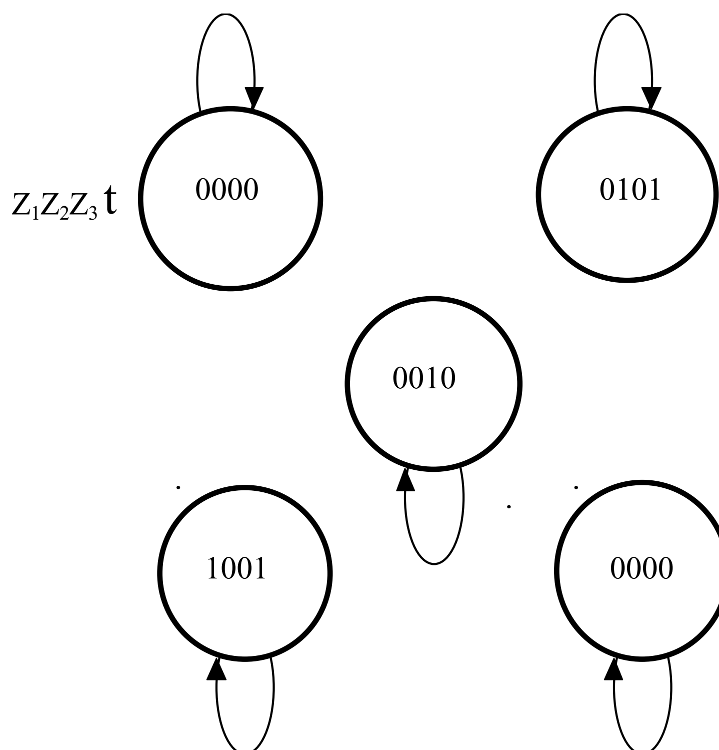

Рис.10

Если рассматриваемый автомат входит в состав управляющего автомата, то в кортежи, помечающие вершины, вводятся также значения всех двоичных переменных из множества **t**, управляющих функциональными элементами задержки.

11. Вершины графа автомата соединяются дугами в соответствии с соединением вершин в графе объекта (рис.11).

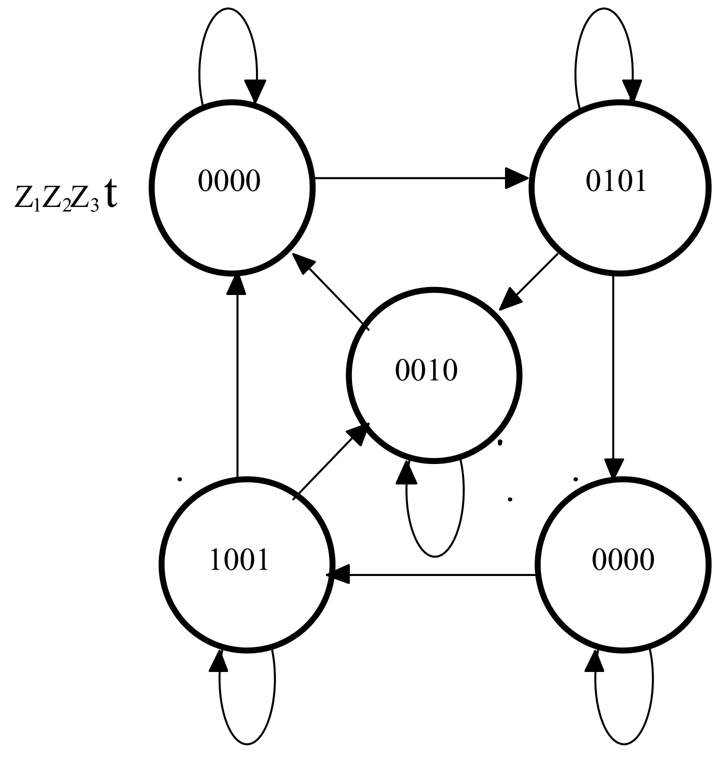

Рис.11

12. Каждая дуга графа автомата помечается булевой формулой, составленной только из тех переменных множества **X** и (или) множества **T**, равенство единице которой инициирует соответствующий переход в автомате (рис.12).

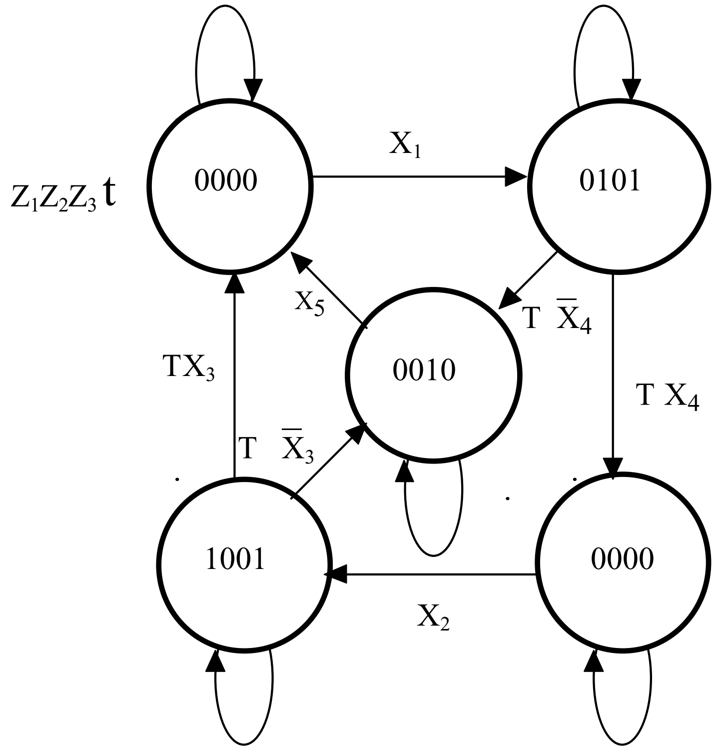

Рис. 12

13. В граф автомата могут быть введены также и дополнительные дуги, отсутствующие в графе объекта. Например, в граф автомата может быть введена дополнительная дуга для устранения возможного несоответствия начальных состояний автомата и объекта (рис.13).

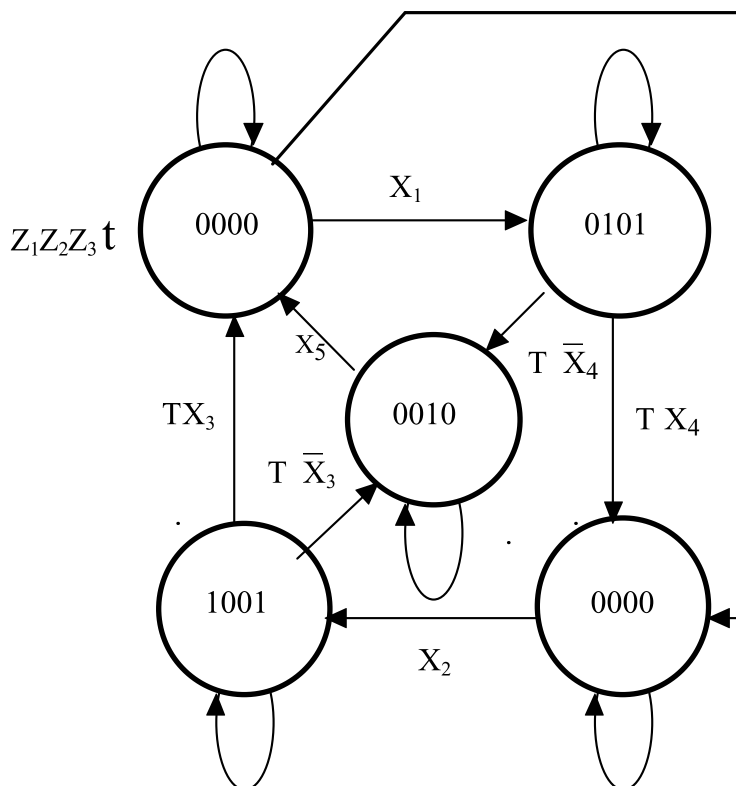

Рис.13

14. Если дополнительные дуги вводятся в граф автомата, то каждая из них должна быть помечена булевой формулой. Равенство единице значения этой формулы инициирует переход между вершинами, соединенными этой дугой, Например, несоответствие между состояниями автомата и объекта может быть устранено в результате перехода по введенной дуге при равенстве единице булевой формулы, зависящей от минимально возможного числа переменных, характеризующих исходное состояние объекта (рис.14).

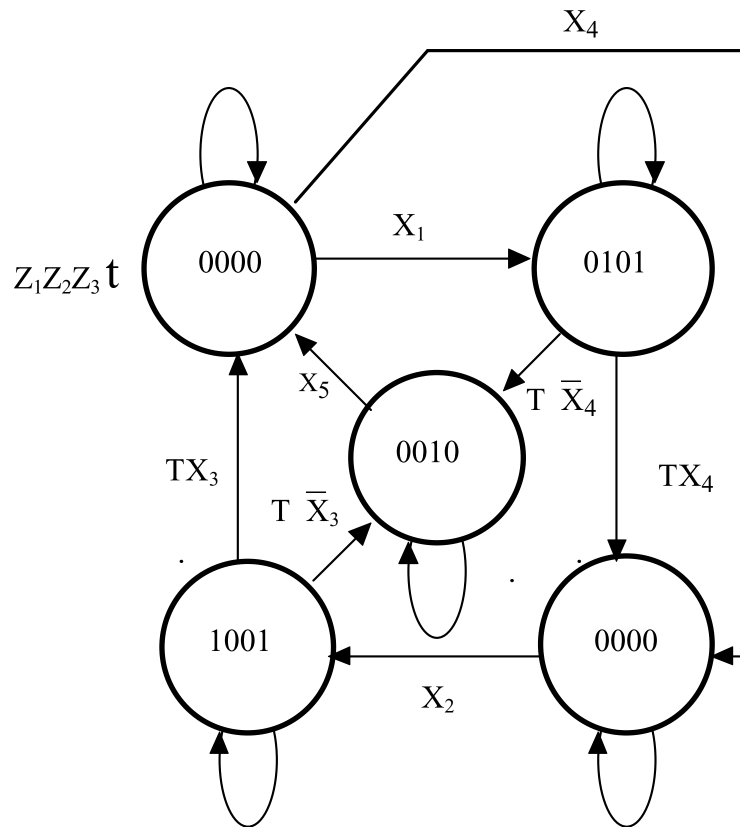

Рис. 14

15. Для каждой вершины графа автомата ортогонализацией или расстановкой приоритетов обеспечивается непротиворечивость переходов в другие вершины (рис.15).

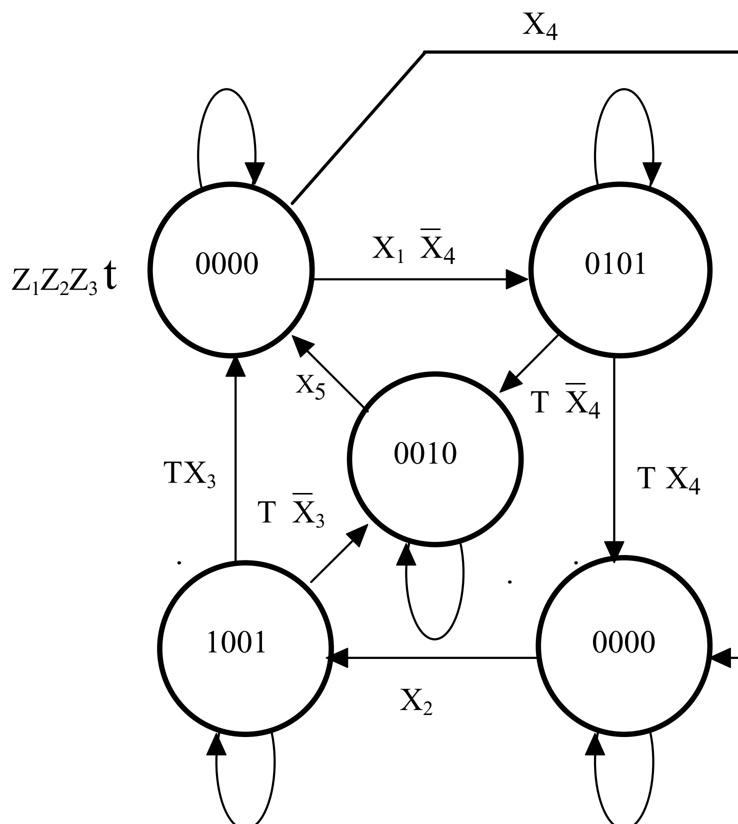

Рис. 15

16. При необходимости осуществляется пометка петель вершин графа автомата за счет обеспечения полноты переходов из каждой вершины (рис.16).

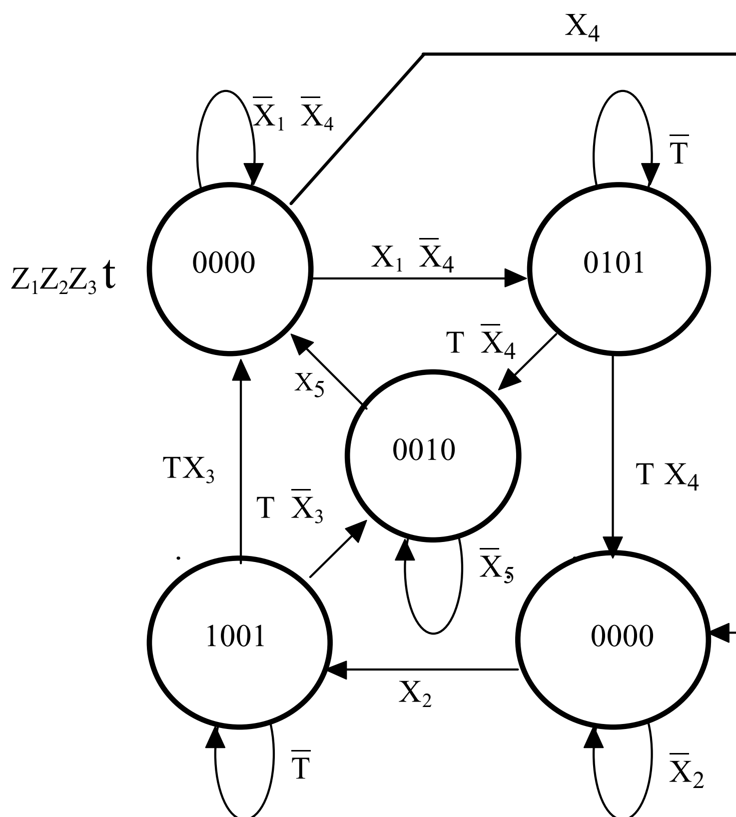

Рис. 16

17. Построенный граф автомата анализируется на наличие генерирующих контуров, отличных от петель, которые устраняются при их обнаружении (рис.17). В графах переходов, рассматриваемых в следующих разделах методики, предполагается, что выполненный анализ позволяет сделать вывод о том, что контуры в них не являются генерирующими.

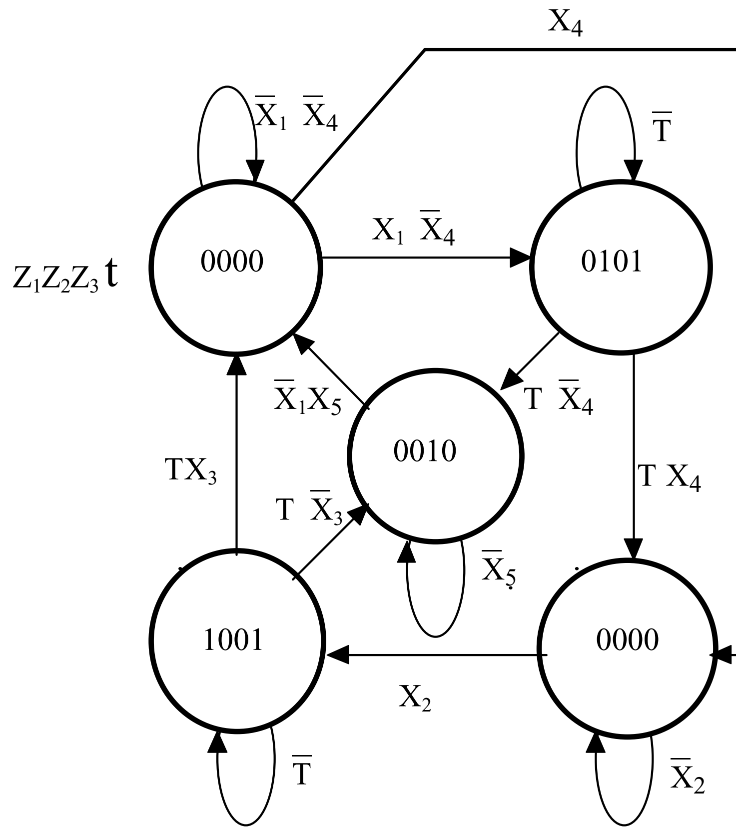

Рис. 17

18. Если в построенном графе автомата имеются вершины, помеченные одинаково, то для их различения должно использоваться кодирование.

Если в построенном графе автомата все вершины помечены различными кортежами выходных переменных, то эти кортежи могут непосредственно применяться в качестве кодов состояний автомата. Однако, и в этом случае бывает целесообразно закодировать вершины графа переходов, значениями переменной **Y**, принимающей значения от **0** до (**S-1**) – сохранить верхнюю пометку вершин графа объекта (рис.18)

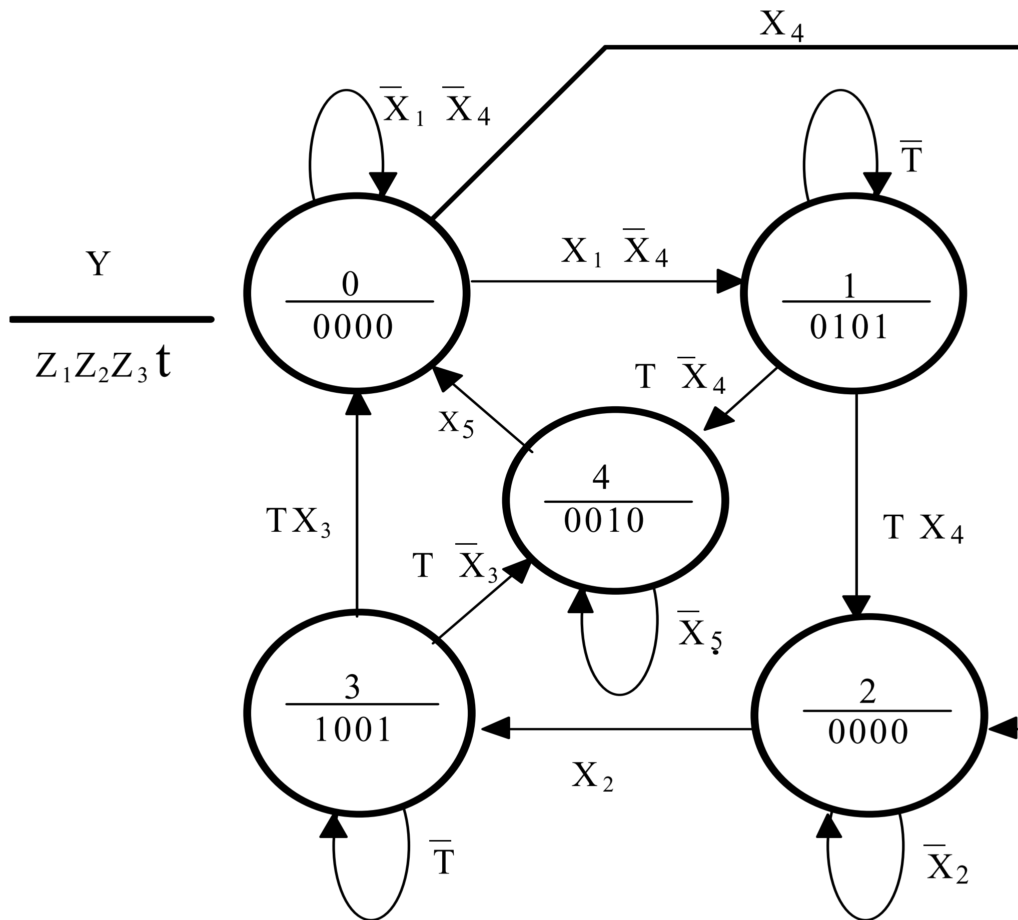

Рис. 18

На этом построение графа автомата завершено.

19. Наличие одинаковых наборов значений выходных переменных в различных вершинах графа автомата является необходимым (но недостаточным) условием для минимизации числа состояний в автомате и соответственно числа вершин в его графе.

В теории автоматов разработаны математические методы минимизации числа состояний автоматов. Однако эти методы не позволяют непосредственно работать с графами автоматов и являются даже для задач сравнительно небольшой размерности весьма трудоемкими. Это резко затрудняет их практическое использование.

Поэтому предлагается при необходимости выполнять эвристическую минимизацию числа состояний автомата в графической форме. При этом требуется совместить между собой, по крайней мере, некоторые из вершин графа автомата, помеченные одинаковыми кортежами значений выходных и временных переменных.

В новом графе устраняются противоречивость, неполнота и генерирующие контуры, если они появляются. Откорректированный граф автомата анализируется на возможность его использования в качестве управляющего для рассматриваемого объекта. Если этот граф семантически корректен, то на этом построение графа автомата завершается.

При этом необходимо отметить, что различные варианты устранения противоречивости и неполноты переходов в автомате могут приводить к различным графам переходов, соответствующих различным автоматам.

Таким образом, можно утверждать, что *могут существовать автоматы, число состояний в которых меньше, чем число состояний в управляемых ими объекте, так как различные состояния объекта могут "поддерживаться" одинаковыми значениями выходных переменных, формируемыми в одном и том же состоянии автомата.*

Если число состояний в автомате и объекте управления одинаково, то легко объяснить, почему целесообразно применять графы переходов при управлении. При управлении по состояниям, получая входное воздействие, автомат переходит в новое состояние и своими выходными воздействиями "перетаскивает" объект управления в соответствующее состояние. Все ясно и понятно.

Если управлять не по состояниями, а как-либо иначе, то указанное соответствие исчезает, и понимать процесс управления становится значительно сложнее.

Минимизация числа состояний автоматов бывает необходима только при жестких ограничениях на объем памяти и нецелесообразно при отсутствии таких ограничений.

Более того, при минимизации числа состояний структура графа автомата начинает "существенно" отличаться от структуры графа объекта, что, например, крайне неудобно, если требуется выполнять сигнализацию состояний объекта не от его сигнализаторов положения (рис.8), а используя построенный автомат.

Из рассмотрения рис.18 следует, что в графе переходов вершины с номерами "**0**"и "**2**" могут быть совмещены (рис.19).

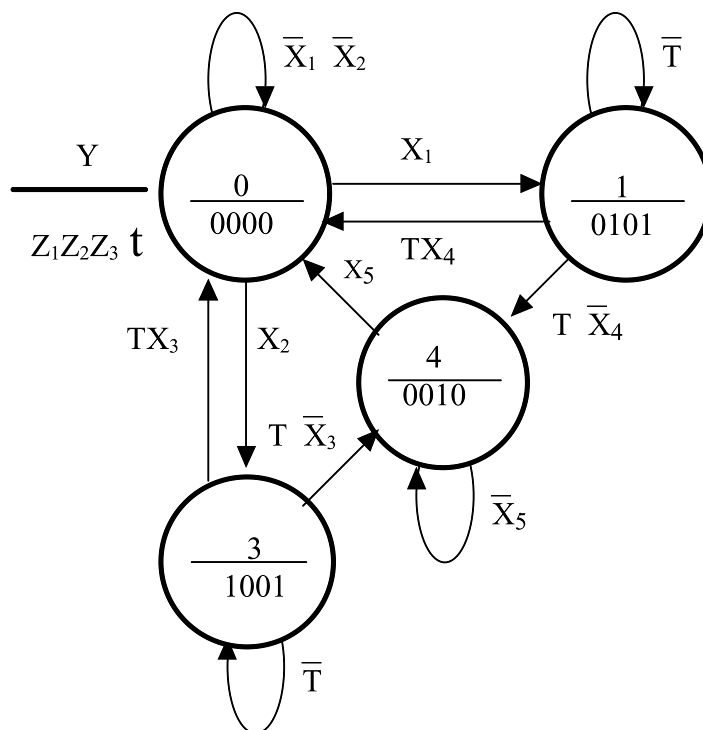

Рис. 19

В полученном графе в нулевой вершине, которая соответствует состояниям объекта "**Открыт**" и "**Закрыт",** имеет место противоречие при $\mathbf{X_1=X_2=1}$. Устранение этого противоречия может привести по желанию Разработчика к построению одного из трех графов переходов, в каждом из которых выполнена перенумерация вершины "**4**":

- граф с приоритетом сигнала закрытия (рис.20);

- граф с приоритетом сигнала открытия (рис.21);
- граф без приоритетов (рис.22).

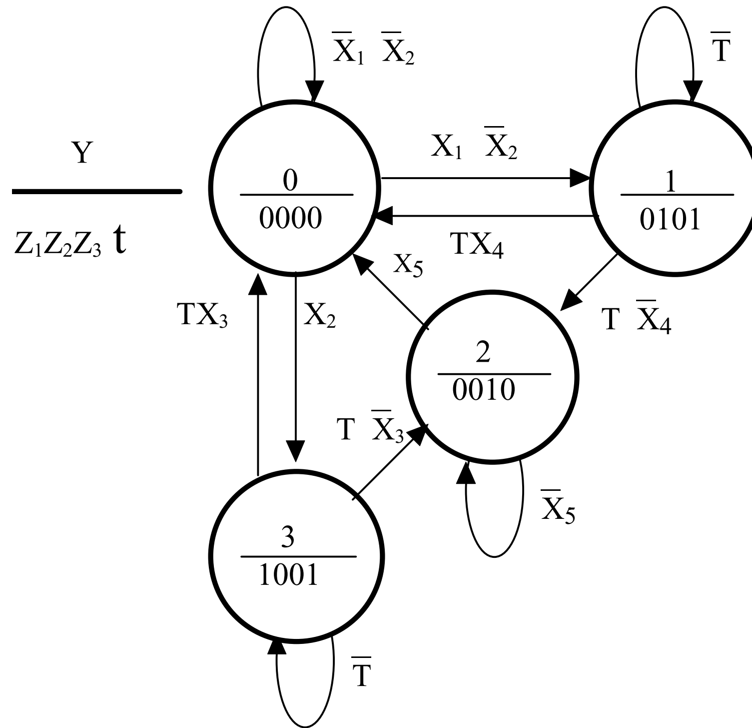

Рис. 20

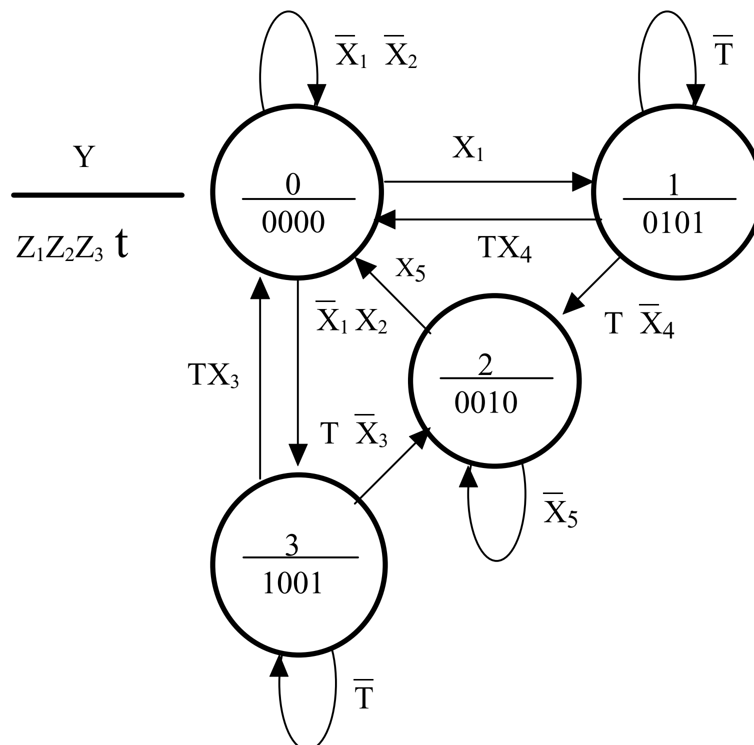

Рис. 21

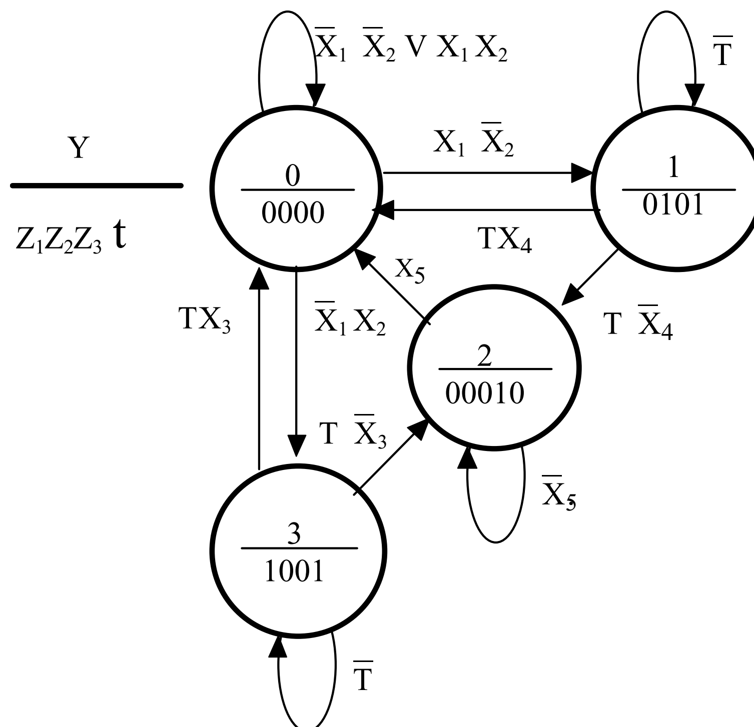

Рис. 22

Отметим, что минимизация числа состояний автомата может проводиться не только совмещением вершин его графа переходов, но и, например, за счет построения на его основе головного и вызываемого графов, последний из которых реализует однотипные фрагменты заданного графа.

20. Выше рассматривались состояния автомата, которые прямо (для автоматов без минимизации числа состояний) **или обобщенно** (для автоматов с минимизированным числом состояний) **были связаны с состояниями исправного или неисправного объекта управления.** Однако автоматы могут содержать также и такие состояния, которые с состояниями управляемого объекта не связаны, а их введение в граф переходов может определяться, например, необходимостью отражения в нем неправильных действий Оператора. Учет таких состояний требует корректировки схемы связей за счет введения в общем случае дополнительных органов управления, средств представления информации и функциональных элементов задержки (рис.23).

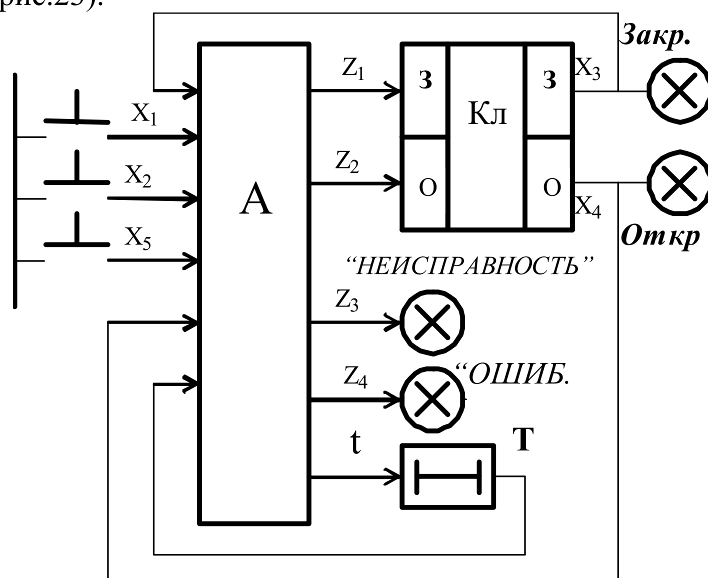

Рис. 23

На этом рисунке приведена откорректированная схема связей, в которую включена лампа сигнализации “***Ошиб.***”, предназначенная для фиксации такого события как одновременное нажатие двух управляющих кнопок $\mathbf{X_1}$ и $\mathbf{X_2}$. Сброс сигнализации осуществляется кнопкой $\mathbf{X_5}$.

Граф переходов, фиксирующий, в том числе, и указанное событие, приведен на рис. 24.

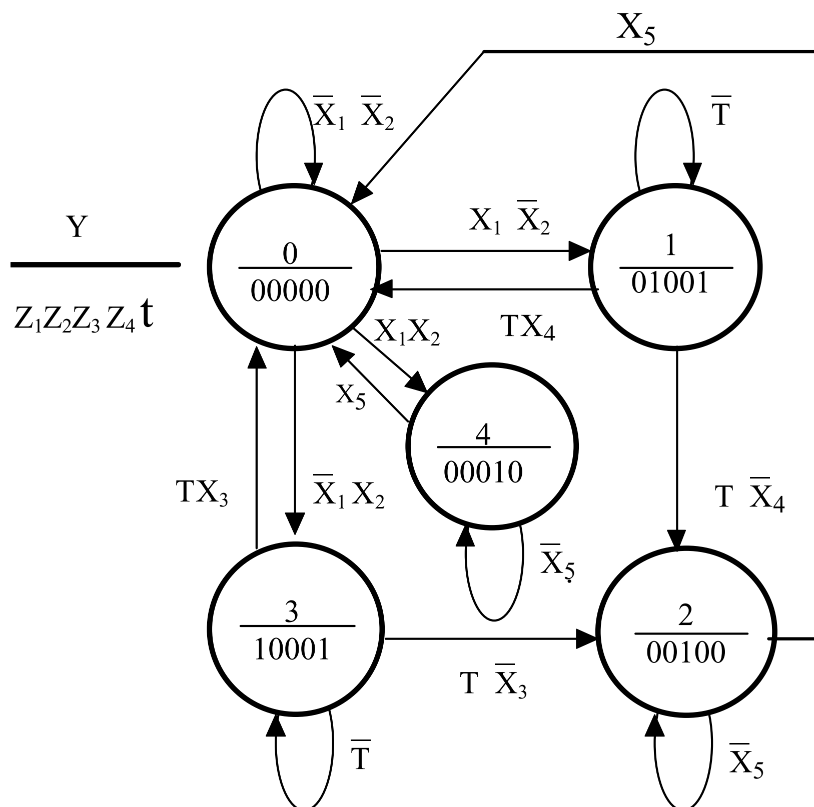

Рис.24

Построенная **схема связей и граф переходов автомата, входящего в эту схему, в исчерпывающей, однозначной и компактной форме определяют техническое задание на разработку программного обеспечения**. При этом словесное описание может использоваться только в качестве комментария (декларации о намерениях), так как **полностью** описать словами даже относительно простой граф переходов практически невозможно.

21.Изложенная методика отвечает на вопрос: “*откуда "берутся" состояния в управляющем автомате?*” Из ее рассмотрения следует, что если понятие "состояние" не применять, то при алгоритмизации возникают весьма существенные трудности, как это имеет место, например, при построении схем алгоритмов, которые обычно строятся не в результате анализа состояний автомата, а за счет проверки значений его отдельных переменных [4].

В заключение раздела отметим, что предложенная методика позволяет строить графы переходов для автоматов без выходного преобразователя или автоматов Мура, которые наиболее целесообразно использовать при логическом управлении такими технологическими объектами как, например, клапаны, насосы, вентиляторы, а также их совокупности, рассматриваемые в качестве единого объекта управления.

Отметим, что вопросы построения графов переходов для других классов автоматов, например автоматов Мили, которые в одном и том же состоянии могут "поддерживать" различные

состояния объекта управления за счет формирования различных значений выходных переменных, и систем графов переходов рассмотрены в работе [42].

Предложенная методика на первый взгляд кажется весьма традиционной для логического проектирования, однако многие введенные в нее аспекты, например, многозначное кодирование состояний, отличают ее от известных методик [43], что позволяет при использовании этого подхода решать задачи программного логического управления достаточно большой размерности.

Для придания завершенности настоящей работе реализуем граф переходов (рис.24) с помощью четырех программ, написанных на языке *СИ*, каждая из которых изоморфна этому графу. Все эти программы используют операторы `switch`. При этом первая и вторая программы соответствуют автомату Мура первого рода, а третья и четвертая – автомату Мура второго рода [42].

В первых двух программах новые значения временной и выходных переменных запаздывают на один программный цикл по сравнению с формированием нового значения переменной, кодирующей состояния автомата. В третьей и четвертой программах новые значения всех указанных переменных формируются в одном программном цикле.

Первая программа состоит из двух операторов `switch`, первый из которых формирует значения временной и выходных переменных в каждом состоянии, а второй – функцию переходов автомата:

```
switch (Y) {
    case 0:
        Z1=0;
        Z2=0;
        Z3=0;
        Z4=0;
        reset_time(1);
    break;
    case 1:
        /*Z1=0;*/
        Z2=1;
        /* Z3=0;
        Z4=0;*/
        _time(1,3);
    break;
    case 2:
        Z1=0;
        Z2=0;
        Z3=1;
        /*Z4=0;*/
        reset_time(1);
    break;
    case 3:
        Z1=1;
        /*Z2=0;
        Z3=0;
        Z4=0;*/
        _time(1,3);
    break;
```

```
    case 4:
        /*Z1=0;
        Z2=0;
        Z3=0;*/
        Z4=1;
        /* reset_time(1);*/
    break;
}
switch (Y) {
    case 0:
        if (X1&!X2)     Y=1;
        if (!X1&X2)     Y=3;
        if (X1&X2)      Y=4;
    break;
    case 1:
        if (T[1]&X4)    Y=0;
        if (T[1]&!X4)   Y=2;
    break;
    case 2:
        if (X5)          Y=0;
    break;
    case 3:
        if (T[1]&X3)     Y=0;
        if (T[1]&!X3)    Y=2;
    break;
    case 4:
        if (X5)          Y=0;
    break;
}
```

Вторая программа реализует заданный граф переходов с помощью одного оператора `switch`:

```
switch (Y) {
    case 0:
        Z1=0;
        Z2=0;
        Z3=0;
        Z4=0;
        reset_time(1);
        if (X1&!X2)     Y=1;
        if (!X1&X2)     Y=3;
        if (X1&X2)      Y=4;
    break;
    case 1:
        /*Z1=0;*/
        Z2=1;
        /* Z3=0;
        Z4=0;*/
        _time(1,3);
        if (T[1]&X4)    Y=0;
        if (T[1]&!X4)   Y=2;
```

```
    break;
    case 2:
        Z1=0;
        Z2=0;
        Z3=1;
        /*Z4=0;*/
        reset_time(1);
        if (X5)        Y=0;
    break;
    case 3:
        Z1=1;
        /*Z2=0;
        Z3=0;
        Z4=0;*/
        _time(1,3);
        if (T[1]&X3)   Y=0;
        if (T[1]&!X3)  Y=2;
    break;
    case 4:
        /*Z1=0;
        Z2=0;
        Z3=0;*/
        Z4=1;
        /*reset_time(1);*/
        if (X5)        Y=0;
    break;
}
```

Третья программа реализуется с помощью двух операторов `switch`:

```
switch (Y) {
    case 0:
        if (X1&!X2)    Y=1;
        if (!X1&X2)    Y=3;
        if (X1&X2)     Y=4;
    break;
    case 1:
        if (T[1]&X4)   Y=0;
        if (T[1]&!X4)   Y=2;
    break;
    case 2:
        if (X5)         Y=0;
    break;
    case 3:
        if (T[1]&X3)    Y=0;
        if (T[1]&!X3)   Y=2;
    break;
    case 4:
        if (X5)         Y=0;
    break;
}
```

```
switch (Y) {
    case 0:
        Z1=0;
        Z2=0;
        Z3=0;
        Z4=0;
        reset_time(1);
    break;
    case 1:
        /*Z1=0;*/
        Z2=1;
        /*Z3=0;
        Z4=0;*/
        _time(1,3);
    break;
    case 2:
        Z1=0;
        Z2=0;
        Z3=1;
        /*Z4=0;*/
        reset_time(1);
    break;
    case 3:
        Z1=1;
        /*Z2=0;
        Z3=0;
        Z4=0;*/
        _time(1,3);
    break;
    case 4:
        /*Z1=0;
        Z2=0;
        Z3=0;*/
        Z4=1;
        /*reset_time(1);*/
    break;
}
```

Четвертая программа, в которой не используются комментарии, реализуется одним оператором switch:

```
switch (Y) {
    case 0:
        if (X1&!X2)     {Y=1;Z1=0;Z2=1;Z3=0;Z4=0;    _time(1,3)};
        if (!X1&X2)     {Y=3;Z1=1;Z2=0;Z3=0;Z4=0;    _time(1,3)};
        if (X1&X2)      {Y=4;Z1=0;Z2=0;Z3=0;Z4=1;    reset_time(1)};
    break;
    case 1:
        if (T[1]&X4)    {Y=0; Z1=0;Z2=0;Z3=0;Z4=0;   reset_time(1)};
        if (T[1]&!X4)   {Y=2; Z1=0;Z2=0;Z3=1;Z4=0;   reset_time(1)};
    break;
```

```
    case 2:
        if (X5)         {Y=0; Z1=0;Z2=0;Z3=0;Z4=0;   reset_time(1)};
    break;
    case 3:
        if (T[1]&X3)    {Y=0; Z1=0;Z2=0;Z3=0;Z4=0;   reset_time(1)};
        if (T[1]&!X3)   {Y=2; Z1=0;Z2=0;Z3=1;Z4=0;   reset_time(1)};
    break;
    case 4:
        if (X5)         {Y=0; Z1=0;Z2=0;Z3=0;Z4=0;   reset_time(1)};
    break;
}
```

Комментарии в этих программах позволяют найти компромисс между их понятностью и сложностью реализации за счет сохранения в текстах программ и исключения из объектных кодов не изменяющихся при соответствующих переходах значений выходных переменных.

В этих программах применяются обращения к процедурам **_time(1,3)** и **reset_time(1)**. Первая процедура реализует функцию запуска первого функционального элемента задержки на 3 сек, а вторая – функцию его сброса. Программная реализация этих процедур приведена в работе [42]. Значения ноль и единица**,** вырабатываемые этими процедурами, присваиваются переменной **T[1].**

При реализации логико-вычислительных алгоритмов в состав операторов `switch` могут входить обращения и к другим видам процедур.

При реализации системы взаимосвязанных графов переходов взаимодействие графов может осуществляться за счет указания на их дугах условий нахождения других графов в соответствующих состояниях. Эти условия всегда доступны для каждого графа переходов, так как при их реализации с помощью операторов *switch,* в каждом графе *в каждом программном цикле* выполняется *не более одного перехода*.

Более подробно эти и другие вопросы, связанные, например, с параллелизмом и реализацией графов переходов в базисе других языков программирования, в том числе и таких как функциональные и лестничные схемы, рассмотрены в работе [42].

Заключение

Центральным понятием теории конечных автоматов является внутреннее состояние автомата (“состояние”).

Состояние несет в себе всю информацию о прошлом автомата, необходимую для определения его реакции на любое входное событие.

Л. Заде перенес это понятие на теорию линейных систем, создав метод пространства состояний.

Предлагается при использовании средств вычислительной техники для построения систем логического управления ввести **понятие “состояние”** в теорию и практику алгоритмизации и программирования этого класса систем.

Несомненно, что значения всех внутренних переменных, в качестве которых могут выступать и выходные переменные, в произвольно построенной программе в определенный момент являются **неявным заданием ее состояния**. Однако, ввиду того, что в программах обычно состояние, как понятие не определяется, а поэтому и не применяется, то при разработке и отладке программ обычно используют только отдельные (обычно битовые) неупорядоченно расположенные внутренние переменные, значения которых и определяют состояния. Это приводит к большим трудностям при создании качественного программного обеспечения.

Такая же ситуация возникает и в тех случаях, когда состояния определяются значениями некоторых входных, выходных и внутренних переменных.

Предлагаемая технология алгоритмизации и программирования **базируется на понятии “состояние”** и позволяет типизировать структуру алгоритмов и программ, задавая порядок введения, расположение и кодирование переменных, определяющих их состояния. В работе [4] рассмотрен вопрос о взаимном соответствии схем алгоритмов и графов переходов, последние из которых используются в предлагаемой технологии в качестве языка спецификаций.

Показано, какие структуры должны иметь схемы алгоритмов для обеспечения их изоморфизма с графами переходов и операторами `switch`.

Каждое **состояние позволяет декомпозировать множество входных переменных на подмножества (возможно, пересекающиеся), существенно влияющие на переходы в смежные состояния**, что обеспечивает возможность описания с помощью графов переходов автоматов с большим числом входов.

В программирование предлагается ввести понятие *наблюдаемость*, позволяющее рассматривать программу в качестве “белого ящика”, в котором все внутренние переменные, число которых минимально, доступны для наблюдения.

Показаны преимущества в части читаемости, понимаемости и возможности отражения динамики графов переходов по сравнению с другими языками спецификаций, используемыми в практике проектирования рассматриваемого класса систем.

Показано, что **при применении многозначного кодирования** состояний для каждого графа переходов вне зависимости от числа его вершин (состояний автомата) **достаточно использовать только одну внутреннюю переменную**, их кодирующую.

Программирование графов переходов с помощью операций `switch` или их аналогов, кроме обеспечения изоморфизма со спецификацией, обеспечивает также доступность каждого значения переменной, кодирующей состояния, для всех остальных графов переходов, входящих

в систему, что позволяет осуществлять **взаимодействие графов без введения дополнительных внутренних переменных.**

В работе [42] рассмотрены вопросы иерархического (в том числе вложенного) представления системы взаимосвязанных графов переходов. Предложен механизм описания с помощью системы взаимосвязанных графов переходов процессов параллельных по состояниям.

Разработаны методы формального и изоморфного перехода от спецификации к текстам программ на языках промышленных компьютеров (например, *ассемблер* и *СИ*) и программируемых логических контроллеров (*языки инструкций*, *лестничных* и *функциональных схем*). Формальность перехода позволяет для одного графа переходов использовать его в качестве полного теста для сертификации построенной программы, а изоморфность с графом переходов, позволяет проверять программу сверкой ее текста со спецификацией. В рамках предложенной технологии определены требования к представлению алгоритмов логического управления и программ в технической документации, их верификации и тестированию, например, в части создания методик проверки функционирования.

Предлагаемый подход позволяет Заказчику, Проектанту (Технологу), Разработчику, Программисту, Пользователю (Оператору) и Контролеру однозначно **понимать** (с точностью до бита, состояния и перехода), **что должно быть сделано, что делается и что сделано** в проекте, и тем самым решить для рассматриваемого класса задач проблему их взаимопонимания [44].

Это позволяет разделять работу и ответственность между Специалистами разных областей знаний, а также между организациями, что особенно важно **при работе с инофирмами, в частности, из-за наличия языкового барьера и неоднозначности понимания естественных языков.**

Предлагаемый подход позволяет также:

- использовать теорию автоматов при алгоритмизации и программировании процессов управления;
- первоначально описывать желаемое поведение управляющего “устройства”, а не его структуру, которая является вторичной и поэтому труднее читаемой и понимаемой;
- ввести в алгоритмизацию и программирование в качестве основного понятия “состояние”;
- ввести понятия “автоматное программирование” и “автоматное проектирование программ”;
- начинать построение алгоритмов и программ с формирования дешифратора состояний, а не событий;
- применять основные структурные модели теории автоматов и ввести новые;
- использовать в качестве языка алгоритмизации графы переходов и системы взаимосвязанных графов переходов;
- при построении графов переходов исключить зависимость от “глубокой” предыстории (как это имеет место, например, в теории марковских процессов);
- применять многозначное кодирование состояний для каждого графа переходов, и вне зависимости от числа его вершин использовать только одну внутреннюю переменную, их кодирующую;
- применять в качестве основной алгоритмической модели графы переходов автомата Мура;
- обеспечить реализацию таких свойств алгоритмов управления как композиция, декомпозиция, иерархичность, параллелизм, вызываемость и вложенность;
- иметь один язык спецификаций при различных языках программирования, в том числе и специализированных, которые применяются в программируемых логических контроллерах;

- проводить алгоритмизацию в результате взаимного общения Заказчика, Проектанта (Технолога) и Разработчика. При этом выдача Технического Задания превращается из однократного события с "бесконечными" последующими дополнениями в однократный процесс общения, завершающийся созданием графа переходов или системы взаимосвязанных графов переходов, в которых учтены все детали с точностью до каждого состояния, перехода и бита;
- применять графы переходов без флагов и умолчаний в качестве проверяющего теста и строить граф проверки или граф достижимых маркировок для других классов графов переходов или систем взаимосвязанных графов переходов с целью проверки их поведения;
- использовать методы формального и изоморфного перехода от спецификации к программам логического управления на различных языках программирования;
- при применении алгоритмических языков программирования высокого уровня проводить программирование с помощью операторов `switch` или их аналогов. Это кроме изоморфизма со спецификацией обеспечивает доступность каждого значения переменной, кодирующей состояния каждого графа переходов, для всех остальных графов, входящих в систему, и поэтому не требует введения дополнительных внутренних переменных для реализации взаимодействия графов в системе;
- ввести в программирование понятие "наблюдаемость", что обеспечивает возможность рассмотрения программы в качестве "белого ящика", в котором все внутренние переменные, число которых минимально, доступны для наблюдения;
- на ранних стадиях проектирования учесть все детали Технического Задания и продемонстрировать Заказчику, как оно понято;
- Участникам разработки общаться не традиционным путем в терминах технологического процесса (например, не "идет" режим экстренного пуска), а на промежуточном полностью формализованном языке (своего рода техническом эсперанто), на котором объясняться можно, например, следующим образом: "в третьем графе переходов, в пятой вершине, на четвертой позиции – изменить значение **0** на **1**". Это не вызовет разночтений, возникающих из-за неоднозначности понимания даже для одного естественного языка, а тем более для нескольких таких языков, в случае, когда Участники разработки представляют разные страны, и не требует привлечения Специалистов, знающих технологический процесс, для корректного внесения изменений;
- снять с Программиста необходимость знания особенностей технологического процесса, а с Разработчика – тонкостей программирования;
- Программисту функциональных задач ничего не додумывать за Заказчика, Проектанта (Технолога) и Разработчика, а только однозначно и формально реализовывать систему взаимосвязанных графов переходов в виде программ, что позволяет резко снизить требования к его квалификации, а в конечном счете, и вовсе отказаться от его услуг и автоматизировать процесс программирования или перейти к автопрограммированию Разработчиком. Последнее, однако, возможно только в случае, когда программирование является "открытым", что не всегда имеет место в особенности при работе с инофирмами или их подразделениями, занимающимися разработкой систем управления, а не только аппаратуры;
- оставлять понятные "следы" после завершения разработки. Это позволяет проводить модификацию программ новым людям, что при традиционном подходе чрезвычайно трудоемко (*"проще построить программу заново, чем разобраться в чужой программе")*. При этом необходимо отметить, что структурирование и комментарии указанную проблему решают лишь частично;
- упростить внесение изменений в спецификацию и программу и повысить их "надежность",
- сделать алгоритм управления инвариантным к используемым языкам программирования, что открывает возможность формирования и поддержания библиотек алгоритмов, записанных формально;

- Заказчику, Проектанту (Технологу) и Разработчику контролировать тексты функциональных программ, а не только результаты их выполнения, как это имеет место в большинстве случаев в настоящее время;
- устранить не равную “прочность” приемки аппаратуры и программ Заказчиком. В первом случае Заказчик кроме функционирования проверяет еще много других характеристик (например, качество печатных плат и их покрытий, качество пайки, номиналы и обозначения элементов), а во втором – все внимание уделяет только проверке функционирования и не исследует внутренняя организацию программ и технологию их построения.

Разработанный подход существенно дополняет международный стандарт IEC 1131-3 [45], распространяющийся на описания языков программирования программируемых логических контроллеров, позволяя при разных языках программирования иметь один язык спецификаций.

Появление в 1996 году программного продукта “S7-HiGraph Technology Software” для программируемых логических контроллеров фирмы “Сименс” [31], в котором в качестве языка программирования используются диаграммы состояний (другое название графов переходов), еще более укрепило уверенность авторf в правильности предложенной в настоящей работе методологии.

Предлагаемая технология впервые была предложена в работе [46] в 1991 году и успешно использована в НПО “Аврора” при разработке систем управления техническими средствами судов, которые построены, в частности, на базе средств вычислительной техники таких фирм как “ABB Stromberg” (Финляндия) [46], “Norcontrol” (Норвегия) [47], “FF-Automation” (Финляндия) [48].

В заключение отметим, что без использования технологии алгоритмизации и программирования, направленной на создание ”качественных” программ, невозможно решить проблему полноты их тестирования, также как нельзя обеспечить контролепригодность аппаратуры, если это свойство не закладывать при ее проектировании.

Предлагаемая технология может быть основой для повышения безопасности программного обеспечения [49] для систем логического управления.

Разработанная технология не исключая возможности применения других методов построения программного обеспечения “без ошибок”[50], существенно более конструктивна, так как позволяет начинать ”борьбу с ошибками” еще на стадии алгоритмизации.

Литература

1. *Шалыто А.А.* Программная реализация управляющих автоматов // Судостроительная промышленность. Сер. ”Автоматика и телемеханика”. 1991. Вып.13.
2. *Шалыто А.А.* Технология программной реализации алгоритмов логического управления как средство повышения живучести //Тезисы докладов научно-технической конференции ”Проблемы обеспечения живучести кораблей и судов”. НТО им. акад. А.Н.Крылова. СПб.:1992.
3. *Антипов В.В., Шалыто А.А.* Повышение безопасности алгоритмического и программного обеспечений систем логического управления техническими средствами // Сборник тезисов симпозиума “Энергетика-95”. СПб.: 1995.
4. *Шалыто А.А.* Использование граф – схем алгоритмов и графов переходов при программной реализации алгоритмов логического управления // Автоматика и телемеханика, 1996. № 6, 7.
5. *Дейкстра Э.* Дисциплина программирования. М.: Мир,1979.
6. *Бутаков Е.А.* Методы синтеза релейных устройств из пороговых элементов. М.: Энергия, 1970.
7. *Карповский М.Г., Москалев Э.С.* Спектральные методы анализа и синтеза дискретных устройств. М.: Энергия.1973.

8. *Артюхов В.Л., Кондратьев В.Н., Шалыто А.А.* Реализация булевых функций арифметическими полиномами // Автоматика и телемеханика,1988. № 4.
9. *Филд А., Харрисон П.* Функциональное программирование. М.: Мир,1993.
10. *Mistic Controller*. Opto 22. Booklet №1-800-321-OPTO.
11. *Nassi J., Shneiderman B.* Flowcharte Techniques for structured Programming.//Signal Not. 1973. № 8.
12. *Ляпунов А.А.* О логических схемах программ // Проблемы кибернетики. Вып.1. М.: Физматгиз. 1958
13. *Бардзинь Я.М., Калниньш А.А., Стродс Ю.Ф., Сыцко В.А.* Язык спецификаций SDL/PLUS/ и методика его использования. Рига: ЛГУ,1986.
14. *Вельбицкий И.В.* Технология программирования. Киев: Техника, 1984.
15. *Питерсон Д.* Теория сетей Петри и моделирование систем. М.: Мир, 1984.
16. *Котов В.Е.* Сети Петри. М.: Наука, 1984.
17. *Юдицкий С.А., Мачергут В.З.* Логическое управление дискретными процессами. М.: Машиностроение, 1992.
18. *Мишель Ж.* Программируемые контроллеры. Архитектура и применение. М.: Машиностроение, 1992.
19. *Бергер Г.* Программирование управляющих устройств на языке STEP-5. Том 1. Программирование основных функций. Сименс, 1982.
20. *Modicon Modsoft*. Руководство программиста.M-MSFT-001.Ped.E. Groupe Schneider.
21. *Гаврилов М.А., Девятков В.В., Пупырев Е.И.* Логическое проектирование дискретных автоматов. М.: Наука, 1977.
22. *Амбарцумян А.А., Искра С.В., Кривандина Н.Ю. и др*. Проблемно – ориентированный язык описания поведения систем логического управления ФОРУМ-М // Проектирование устройств логического управления. М.: Наука, 1984.
23. *Девятков В.В., Чичковский А.Б.* Условие – 82 – язык программно-логического управления //Автоматизация проектирования. Вып.2. М.: Машиностроение,1990.
24. *Горбатов В.А., Смирнов М.И., Хлытчев И.С.* Логическое управление распределенными системами. М.: Энергоиздат,1991.
25. *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") // Автоматика и телемеханика. 1985. № 6.
26. *Баранов С.И.* Синтез микропрограммных автоматов (граф – схемы и автоматы). Л.: Энергия, 1979.
27. *Кузнецов О.П.* Графы логических автоматов и их преобразования // Автоматика и телемеханика. 1975. № 9.
28. *Лингер Р., Миллс Х., Уитт С.* Теория и практика структурного программирования. М.: Мир, 1982.
29. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. Киев: Диалектика; М.: АО "ИВК", 1992.
30. *Shlaer S., Mellor S.* Object Lifecycle: Modeling the World in States. New Jersy: Prentice Hall.: Tuglewood Cliffs, 1992.
31. *SIMATIC.* Simatic S7 /M7/C7. Programmable Controllers: SIEMENS. Catalog ST70. 1996.
32. *Modicon Modsoft*. Руководство программиста. GM-MSFT-001. Peg.E. Groupe Schneider.
33. *Матчо Д., Фолкнер Д.* DELPHI. М.: Бином. 1995.
34. *Логика. Автоматы. Алгоритмы* // М.А.Айзерман, А.А.Таль, Л.И.Розоноэр и др. М.: Физматгиз, 1963
35. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний. М.: Наука, 1970.
36. *Гнеденко Б.В*. Курс теории вероятностей. М.: Наука, 1965.
37. *Касьянов В.Н., Поттосин И.В.* Методы построения трансляторов. Новосибирск: Наука, 1986.
38. Мартынюк В.В. Об анализе графа переходов для операторной схемы // Журн. вычисл. математики и мат.физики. 1965, № 2.

39. *“Селма-2”* Описание функциональных блоков. АББ Стромберг Драйвс, 1989.
40. *Kalman R.E.* On the General Theory of Control Systems. Proc.First Intern. Congr. Automatic Control, Moscow, 1960, vol. 1, London: Butterworth & Co.
41. *Мур Э.* Умозрительные эксперименты с последовательными машинами // Автоматы. М.: Изд-во иностр. лит., 1956.
42. *Шалыто А.А.* SWITCH – технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
43. *Ангер С.* Асинхронные последовательностные схемы. М.: Наука,1977.
44. *Искусственный интеллект в XXI веке.* // Новости искусственного интеллекта. 1995, № 4.
45. *Intrnational Standard IEC 1131 – 3.* Programmable controllers. Part.3. Programming languages // International Electrotecnical Commission, 1993.
46. *Project 15640.* AS21.DG1.CONTROL. АМИЕ.95564.12М. St.Petersburg. ASS “Avrora”. 1991.
47.. Warm & prelubrication logic. Generator Control Unit. Severnaya hull № 431. *Functional Description* Norcontrol, 1993.
48. *Autolog 32*. Руководство пользователя. FF-Futomation, 1990.
49. *Underwriters Laboratories* обновляет стандарт UL 1998 (Safety-Related Software) по безопасности программного обеспечения для ПЛК // Современные технологии автоматизации.1997. № 1.
50. *Бейбер Р.Л.* Програмное обеспечение без ошибок. М.: Дж. Уайли энд санз. Радио и связь.1996.