

ЖУРНАЛ "ИЗВЕСТИЯ АКАДЕМИИ НАУК. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ"  
Номер 6. Ноябрь-Декабрь 2000. С.63-81.

УДК 681.3.06 : 62-507

## **АВТОМАТНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММ. АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ ЗАДАЧ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ**

© 2000 г. А. А. Шалыто

Санкт-Петербург, Федеральный научно-производственный  
центр – ГУП "НПО "Аврора"",

С.-Петербургский гос. ин-т точной механики и оптики  
(техн. ун-т)

Поступила в редакцию 21.06.98 г.

Излагаются основы технологии алгоритмизации и программирования задач логического управления, обеспечивающей повышение "безопасности" программного обеспечения. Описываемая технология может быть названа технологией состояний или, более точно, автоматной технологией, а соответствующая область программирования – автоматным программированием.

**Введение.** В настоящей работе излагается технология алгоритмизации и программирования задач логического управления, названная переключательной технологией (**SWITCH-технология**).

Актуальность разработки такой технологии определяется, во-первых, необходимостью того, чтобы Заказчик, Технолог (Проектант), Разработчик, Программист, Оператор (Пользователь) 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 желателю изоморфному построению "хорошо понимаемых" алгоритмов и программ, позволяющих решать задачи рассматриваемого класса.

Эта проблема является актуальной и для других классов задач. Так, Э. Дейкстра во введении к [1] пишет: "Я знал, что 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 завершённой 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 логического управления 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 отражали лишь алгоритм управления и не содержали другой информации, характерной для принципиальных схем (например, обозначений разъемов, гасящих сопротивлений, устройств контроля и т.д.). Однако и эти схемы, весьма удобные для представления 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.

При программной реализации на базе аппаратуры "Selma-2" фирмы АВВ Stromberg (Финляндия) [2] в качестве как языка алгоритмизации, так и языка программирования, используются функциональные схемы, а для программируемых логических контроллеров "Autolog" фирмы "FF-Automation" (Финляндия) [3] в качестве языка программирования применяется язык инструкций ALPro, в то время как язык алгоритмизации не определен. Последний неопределен и для многих других типов программируемых логических контроллеров, таких, как, например, как "Melsec" фирмы "Mitsubishi Electric" (Япония) [4], языками программирования которых является язык инструкций и язык лестничных схем – язык релейно-контактных, дополненный большим числом вычислительных операций.

В настоящее время в качестве языков алгоритмизации в системах логического управления наиболее часто применяются лестничные и функциональные схемы и блок-схемы алгоритмов, называемые также граф-схемами алгоритмов или схемами алгоритмов, а в качестве языков программирования в зависимости от типов управляющих вычислительных устройств используются три типа языков: алгоритмические языки высокого уровня (например, Си, "Паскаль", ПЛ/М, "Форт"), алгоритмические языки низкого уровня (ассемблеры, языки инструкций) и специализированные языки (например, функциональные и лестничные схемы).

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

При логическом управлении могут использоваться традиционные (классические) пути формализации, как процедур управления, так и описаний объектов управления, что отличает этот класс задач, например от ситуационного управления [5].

Для ответственных технологических объектов системы логического и ситуационного управления могут применяться совместно.

## **1. Классические языки логического управления.**

1.1. Булевы функции, таблицы истинности и таблицы решений. В системах логического управления традиционно используются булевы функции и системы булевых функций, задаваемые в

форме таблиц истинности для полностью определенных функций и таблиц решений для неполностью 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.

1.2. Булевы 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 комбинационные схемы, так и автоматы с памятью 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 [8,9].

Основным ограничением на применение систем булевых формул для автоматов с памятью является их низкая наглядность.

1.3. 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, в то время, как они реализуются средствами вычислительной техники, позволяющими обрабатывать многозначные переменные; отсутствие указания значений выходных и внутренних переменных в схеме; трудоемкость их чтения (понимания) с целью получения исчерпывающего представления о реализованном с их помощью последовательном процессе; проблема выбора тестов для их полной проверки и сложность гарантированного внесения изменений.

При этом чтение функциональных схем заменяется вычислениями по 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, и составить целостное представление о поведении даже сравнительно небольшого фрагмента схемы при применении триггеров и обратных связей в нем. Так, например, при наличии трех взаимосвязанных триггеров 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, реализованных с помощью обратных связей (самоблокировок) и (или) триггеров, так как в этом случае необходимо проверять также и правильность порядка изменений переменных. Однако, несмотря на это, именно такие соотношения и применяются в настоящее время при создании методик проверки функционирования большинства систем логического управления, что не обеспечивает качественной их проверки, так как не позволяют анализировать все имеющиеся в схеме переходы между состояниями и, более того, они и не известны в силу того, что построение схем для этого класса систем обычно выполняется эвристически без использования понятия "состояние".

Функциональные схемы при их применении в качестве языка программирования, обладают всеми достоинствами декларативных языков функционального программирования [10], "основным из которых является функциональность (прозрачность по ссылкам), т.е. 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 этого 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, которое может привести к побочным эффектам, например к изменению значения функции при повторном ее вызове даже без изменения значений аргументов. Это приводит к тому, что такую 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) 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.

1.4. 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 строят обычно для описания "основного" режима, а алгоритм в целом отражается лишь в программе, которая по указанной причине строится по таким диаграммам во многом неформально.

1.5. Схемы алгоритмов. К достоинствам схем алгоритмов при их использовании в качестве языка алгоритмизации для систем логического управления относится возможность отражения с их помощью в явном виде последовательностей событий (определяемых значениями входных переменных) и реакций на их появление (представляемых в виде значений выходных переменных, в том числе и вычисляемых параллельно). Наличие двоичных значений переменных, записываемых в явном виде в операторных вершинах, резко упрощает понимание схем алгоритмов (называемых в этом случае автоматными) по сравнению с функциональными схемами.

К недостаткам схем алгоритмов относятся:

- применение в литературе двух видов автоматных схем алгоритмов, первый из которых характеризуется тем, что в схемах этого вида по умолчанию считают, что ввод входных переменных осуществляется в каждой условной вершине, а вывод значений выходных переменных – в каждой операторной вершине [11]. В схемах алгоритмов второго вида считают, что ввод входных переменных осуществляется в начале тела схемы алгоритмов, а вывод значений выходных переменных – в его конце. Использование в явном виде операторов "Ввод" и "Вывод" в таких схемах позволяет различать указанные разновидности схем;

- отсутствие требований к тому, что должна отражать схема алгоритма, а именно: алгоритм управления (схема алгоритма с внутренними обратными связями, но без внутренних переменных); алгоритм реализации алгоритма управления (схема алгоритма без

внутренних обратных связей); алгоритм, учитывающий свойства управляющих конструкций применяемого языка программирования (схема алгоритма, линейризованная и структурированная, возможно, специальным образом); алгоритм выполнения программы (схема алгоритма, в которой упоминаются компоненты процессора, например, аккумулятор);

– отсутствие требований к их организации (за исключением структурирования), обеспечивающих простоту "чтения";

– необходимость в общем случае их многократных преобразований с целью обеспечения возможности одновременного решения нескольких задач в одном управляющем вычислительном устройстве (расщепление) и учета свойств управляющих конструкций языка программирования (например, линейризация и структурирование);

– наличие внутренних (промежуточных) переменных, отсутствующих в "словесном алгоритме" логического управления, резко затрудняющих возможность чтения схем алгоритмов другими, отличными от Разработчика, Специалистами, и в особенности Заказчиком;

– применение обычно большого числа битовых внутренних переменных, каждую из которых приходится не только устанавливать, но и принудительно сбрасывать. Эти переменные характеризуют лишь отдельные компоненты состояний автомата, а его состояния в целом обычно не определяются. Использование этих переменных является естественным при аппаратной реализации алгоритмов, но при реализации алгоритмов с помощью языков программирования, позволяющих обрабатывать многозначные переменные, применение битовых внутренних переменных нецелесообразно;

– наличие флагов 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 такие 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, а оканчивается 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 все особенности технологического процесса, либо они вместе должны устранять неминуемые ошибки традиционного проектирования программ при испытаниях.

Остановимся более подробно на использовании умолчаний значений выходных переменных в операторных вершинах автоматных схем алгоритмов.

Автоматные схемы алгоритмов первого типа могут иметь две разновидности. Первая из них характеризуется тем, что в каждой

операторной вершине приводятся обозначения только тех выходных переменных, которые принимают в ней единичные значения, а всем остальным выходным переменным по умолчанию присваиваются нулевые значения. Для второй разновидности схем алгоритмов этого типа характерно, что в каждой операторной вершине приводятся обозначения только тех выходных переменных или их инверсий, которые принимают в ней единичные и нулевые значения соответственно, а для каждой из умалчиваемых переменных предполагается, что она сохраняет предыдущее значение.

Для автоматных схем алгоритмов второго типа в каждой операторной вершине в явном виде указываются обозначения тех выходных переменных, которые в этой вершине принимают константные значения, а также могут приводиться переобозначения тех выходных переменных, каждая из которых сохраняет в этой вершине предыдущее значение. Для каждой из умалчиваемых переменных предполагается сохранение ее предыдущего значения.

Для автоматных схем алгоритмов (кроме первой разновидности первого типа) имеется также возможность сохранения предыдущих значений всех выходных переменных на "проводах" – без применения операторных вершин в соответствующем контуре схемы алгоритма.

Возможность сохранения предыдущих значений выходных переменных в операторных вершинах или без использования таких вершин резко усложняет понимание схем алгоритмов.

Рассмотренный язык используется фирмой "Опто" (США) для программирования логических контроллеров "Mistic" [12]. Применение схем алгоритмов в форме диаграмм Несси-Шнейдермана [13] практически не устраняет указанных недостатков.

1.6. Логические схемы алгоритмов. Логические схемы алгоритмов, предложенные А.А. Ляпуновым [14], являются строчной формой записи линеаризованных схем алгоритмов и образованы буквами (которые соответствуют условным, безусловным и операторным вершинам линеаризованных схем алгоритмов) и пронумерованными стрелками, указывающими переходы, осуществляемые при не выполнении условий.

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

**2. Нетрадиционные языки описания алгоритмов логического управления.** 2.1. Язык SDL. Идеи теории автоматов нашли свое отражение при разработке Международной комиссией по телефонии и телеграфии графического языка спецификации и описания алгоритмов – SDL-диаграммы (Specification and Description Language) [15], 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 [11].

2.2. R-схемы. Еще одна модель, базирующаяся на использовании автоматов Мили, была предложена И.В. Вельбицким [16] и носит название R-схемы (R-chart). R-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. R-схемы образуются за счет трех типов соединений – последовательного, параллельного и вложенного.

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

2.3. Сети Петри и графы операций. Для описания сложных, в том числе параллельных процессов, в 1962 г. К. Петри [17] была предложена графовая модель, названная его именем, которая 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 переход срабатывает и маркеры переходят в позиции, связанные исходящими дугами с рассматриваемым переходом. Для целей управления С.А. Юдицким [18] было предложено применять только безопасные и живые сети Петри. В безопасной сети Петри в позициях не может быть более одного маркера, а в живых сетях Петри – имеется возможность срабатывания любого перехода. Назовем указанный класс сетей – управляющими сетями Петри, а сети Петри, 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 [18] 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 этого не требуется;

- для графов операций, построенных на базе автоматных сетей Петри, используется только модель автомата Мура [11] и невозможно применение других автоматных моделей, что резко ограничивает изобразительные возможности графов операций;

- для кодирования позиций могут использоваться только "единичные" коды. При этом число внутренних переменных (без учета их переобозначений) равно числу позиций, в том числе и для графов операций, построенных на базе автоматных сетей Петри, в то время как для этого класса графов при применении подхода, предлагаемого в настоящей работе, все 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.

2.4. Язык "Графсет". Этот графический язык, разработанный в Центре космических исследований в Тулузе (Франция), применяется в настоящее время наряду с другими языками [19] такими фирмами как, например, "Телемеханик" (Франция), "Сименс" (Германия), "Ален Бредли" (США), "Тошиба" (Япония), "Омрон" (Япония). Этот язык

алгоритмизации при наличии транслятора с него является и языком программирования.

Язык "Графсет" отличается от языка графов операций в основном только формой изображения: квадраты вместо кружков для обозначения позиций и прямоугольники для записи значений выходных переменных, отсутствующие в графах 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 управляющих 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 наличие ряда недостатков, указанных выше, 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 программируемых логических контроллеров с помощью языка STEP-5 (языки инструкций, лестничных и функциональных схем), а также языка S7-GRAF (язык "Графсет") [20,21].

Однако опыт показывает, что при наличии для одного и того же программируемого логического контроллера нескольких языков программирования разработчики обычно используют 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, а эффективность применения таких графов по сравнению, например, с лестничными схемами демонстрируется в отдельных случаях (фирма "Омрон") только на примерах, без изложения метода формального перехода от управляющего графа к "схеме".

При этом для разных моделей контроллеров одной той же фирмы [22] предлагается использовать различные языки (для "младших" моделей лестничные схемы, а для старших – "Графсет"), в то время как единый язык спецификаций алгоритмов для всех моделей не применяется.

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 "Графсет" (фирма "Омрон"), а в документации фирмы "Телемеханик" отмечено, что диаграммы "Графсет" известны, так же как SFC. Фирма "Сименс", как отмечено выше, применяет для тех же целей другой термин – "GRAF".

Более того, 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.



2.5. Проблемно – ориентированные языки, близкие к естественным. Для целей логического управления разработано несколько проблемно-ориентированных языков, предназначенных для формального лингвистического описания алгоритмов рассматриваемого класса. Эти языки называются [23] также 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, основные из которых следующие:

- они требуют от всех Участников процесса проектирования изучения синтаксиса и семантики нового, имеющего ограниченное распространение языка с достаточно большим числом конструкций;

- построены на основе не математического, а естественного (например, русского) языка;

- обладают низкой наглядностью "текстов" по сравнению с графами при отображении структуры, взаимодействия и динамики процессов;

- они не позволяют формально проверять полноту, непротиворечивость и отсутствие генерации, а также выполнять оптимизирующие преобразования;

- они 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;

- сложность верификации и отсутствие тестов для проверки правильности описания;

- сложность корректного внесения изменений;

- невозможность использования при отсутствии транслятора для применяемого управляющего вычислительного устройства.

Из языков рассматриваемого класса наибольшее теоретическое обоснование получили следующие языки логического управления и их модификации – Форум [24], Условие [25], Управление [26], Ярус [27].

Так, например, конструкции первичного языка "Условие" сначала транслируются в базовый язык операторных схем параллельных алгоритмов с памятью, являющийся развитием 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 для построения автомата [11] двоичные метки, трактуемые как состояния, и по помеченному описанию строится граф переходов автомата Мили, на базе которого, в частности, решаются задачи минимизации числа состояний и параллельной декомпозиции [26].

Подход, наиболее близкий к предлагаемому в настоящей работе, был применен при разработке языка "Ярус". При этом О.П. Кузнецовым [28] для описания работы "пунктов" было предложено использовать автоматную модель, названную графом переключений, являющуюся графом переходов автомата Мили с умолчаниями неизменяющихся значений выходных переменных. Возможность сокращения числа вершин в этом графе по сравнению с классическими автоматными моделями привело к замене термина "состояние" на термин "ситуация", правда, с одновременным ухудшением читаемости графа в виду появления зависимости от "глубокой" предыстории.

В отличие от изложенного в предлагаемой в настоящей работе технологии первичным и формализованным является не лингвистическое, а автоматное описание последовательностных процессов с помощью

графов переходов, тип, количество, способ кодирования и взаимосвязь которых 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 [29] 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 предыстории – будущее зависит от настоящего, но 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 графов переходов 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 автомата другого типа. Так, например, по графу переходов автомата Мили может быть построен граф переходов 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 в графе 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 [30], 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, если 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 компактное описание поведения автомата, то наиболее "понятным" является граф переходов "классического" автомата Мура при многозначном кодировании его вершин, который 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 по принципу "один в один".

При применении предлагаемой технологии удастся обеспечить соответствие между текстом программы и порядком ее выполнения и реализовать процедуру пошаговой детализации, как этого требует

структурное проектирование (программирование) [31], а также использовать понятия "объект" и "класс", как это принято в объектно-ориентированном проектировании (программировании) [32].

2.6. Алгоритмические языки программирования. Из изложенного выше 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, так как в противном случае возникают многие из проблем, перечисленных выше, применительно к использованию проблемно-ориентированных языков в качестве первичного описания.

**3. Графы переходов как язык спецификаций.** 3.1. Стратегии синтеза алгоритмов логического управления. Возможны две стратегии построения алгоритмов этого класса. При применении первой из них считается, что известен алгоритм функционирования объекта управления и требуется по нему синтезировать алгоритм логического управления, обеспечивающий заданное поведение объекта. Приведем в качестве примера фрагмент алгоритма управления клапаном, синтезированный указанным способом: "Для того, чтобы клапан открылся, вычислитель должен на вход открытия исполнительного механизма клапана подавать единичный сигнал".

При второй стратегии, учитывая информацию о состоянии (положении) объекта управления, строится алгоритм, 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", а вторая — на понятии "событие" ("входное воздействие").

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, по мнению автора, более естественной для рассматриваемого класса задач является первая из них, так как "состояние" по своей природе статично, а "событие" динамично, и поэтому "управление по состояниям" является более целесообразным, чем "управление по событиям" [33].

Однако ни та, ни другая разновидность управления в общем случае не является исчерпывающей. Только "управление по состояниям и событиям" при этом является корректным. Так как оба понятия "состояние" и "событие" входят в понятие "автомат", то такой вид управления может быть назван "автоматным управлением", а его программная реализация "автоматным программированием".

В предлагаемой в настоящей работе технологии понятие "состояние" является первичным, а понятие "событие" — вторичным, особенно учитывая тот факт, что события, формируемые объектом управления, определяются соответствующими состояниями последнего.

Несмотря на то, что вторая стратегия приводит обычно к построению более компактных и быстродействующих программ, при отсутствии жестких ограничений на объем памяти и быстродействие применение первой из них более естественно, так как соответствует основному принципу управления, применяемому в автоматизированных системах, состоящему в том, что при управлении Оператор 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 [29], так как при "управлении по событиям" возникают существенные проблемы с введением и корректным использованием внутренних переменных.

При алгоритмизации состояние должно определяться 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, Технологи, 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, и поэтому в рамках предлагаемого подхода для проверки правильности разработанного алгоритма целесообразно выполнять также моделирование комплекса "управляющий автомат – объект управления". После этого алгоритм управления может и далее уточняться как на физической модели, так и на реальном объекте. Точность и детальность описания алгоритма с помощью графов переходов резко повышают качество первоначальной алгоритмизации по сравнению с другими методами, и поэтому на объекте обычно требуется вносить сравнительно небольшое число изменений в разработанный алгоритм и реализующую его программу.

3.2. Факторы, ограничивающие широкое использование графов переходов в качестве языка алгоритмизации. Для устранения недостатков рассмотренных языков алгоритмизации предлагается использовать 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 графов переходов – таблицы переходов и таблицы переходов и выходов. Структура таких таблиц, требующая перечисления всех комбинаций значений всех входных переменных, резко уменьшает размерность решаемых с их помощью задач.

Другая проблема, ограничивающая применение графов переходов, была связана с тем, что при асинхронной схемной реализации систем логического управления 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, что требует в общем случае применения весьма трудоемкого противогоночного кодирования, связанного с избыточностью, часто неприемлемой при аппаратной и в особенности релейно-контактной реализации.

Еще одна из существующих причин состоит в том, что традиционно считалось, что графы переходов описывают только последовательные

алгоритмы, которые имеют весьма ограниченное использование в системах управления, для которых характерен параллелизм.

Указанные трудности, а также традиции построения функциональных схем при аппаратной и схем алгоритмов при программной реализации алгоритмов, видимо, явились причинами того, что графы переходов до сих пор практически не применяются в качестве языка спецификаций условий работы при их программной реализации.

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

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 программирования процессов управления (за исключением подхода, использованного в объектно-ориентированном анализе [34] и в программных продуктах "S7-NiGraph technology software" [21] и "Modicon State Language" [22]).

При применении других подходов внутреннее состояние Вычислителя обычно либо не учитывается, либо не рассматривается как единое целое. При этом, как в событийно-управляемом программировании, 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 и 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, ввиду того что в алгоритме появилась внутренняя переменная, которая должна находиться в памяти Вычислителя, так как не известно, когда ее опросят (видимо, когда накроют стол). Таким образом, наряду с "комбинационной схемой" появляется и 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 по крайней мере для задач логического управления наряду с его применением в теории конечных автоматов, теории линейных систем [35], марковских процессах [36] и в отдельных задачах практического [37] и теоретического [38] программирования. Графы переходов также 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 он называется "смешанным" (С-автомат). Если в нем одни и те же переменные применяются как в качестве входных, так и выходных переменных, то такой автомат называется "автоматом с флагами" [29].

Необходимо отметить, что одни и те же наборы значений выходных переменных могут формироваться в разных состояниях, что требует

введения дополнительных (промежуточных) переменных для различения этих состояний.

Состояния автомата классифицируют (декомпозируют) все входные переменные на группы, выделяя в каждом из состояний только то подмножество таких переменных, которое определяет требуемые 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 него в другие состояния, т.е. переходы из рассматриваемого состояния несущественно зависят (не зависят) от всех остальных переменных, не входящих 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, описывающих комбинационные схемы и 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 графов переходов минимальна). При этом дуга, представляющая собой петлю, отражает сохранение состояния, в котором 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. Единичные значения этих формул определяют возможность выполнения переходов. Обратим внимание на тот факт, что даже при наличии петли вершина может быть неустойчивой, если значения формул, помечающие одну из ее входных и одну из ее выходных дуг, не являющуюся петлей, одновременно равны единице.

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 будем полагать, что функциональные элементы задержки в состав автоматов не входят и являются для них одним из объектов управления. Комплекс "Автомат — функциональные элементы задержки" образует единую компоненту, называемую "управляющим автоматом".

При чтении графов переходов считается, что в каждый момент времени (за один программный цикл) выполняется не более одного перехода: ни одного, если состояние сохраняется, и один — если оно

не сохраняется, что поддерживается соответствующей программной реализацией.

В каждом сильносвязанном графе переходов, используемом для логического управления, выделяется одна начальная вершина, которая может совпадать с конечной.

3.4. Построение графов переходов. 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 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 (абстракция) приходится вводить сразу для различения ситуаций, связанных с **изменениями** значений выходных переменных при одних и тех же значениях входных переменных. Построение графов 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. При этом появляется возможность в **каждом** состоянии иметь информацию о значении **каждой** выходной переменной, что, по мнению автора, является определяющим для обеспечения в дальнейшем простоты чтения и понимания, а также корректировки графов переходов.

На начальном этапе построения графа переходов число вершин в нем должно быть выбрано так, чтобы каждая из них поддерживала одно из состояний объекта управления (включая состояния, соответствующие его неисправностям) или системы управления (например, состояния, соответствующие неправильным действиям Оператора). В дальнейшем при необходимости может быть сделана попытка минимизации числа вершин. При этом отметим, что в общем случае число состояний в автомате может быть меньше числа состояний объекта, так как одно и то же состояние автомата может поддерживать с помощью одинаковых значений выходных переменных разные состояния объекта (например, закрытое и открытое состояния клапана "с памятью" (его устойчивые состояния) могут поддерживаться как двумя, так и одним состоянием автомата с нулевыми значениями выходных переменных).

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, причем даже соседние вершины могут быть помечены одинаково. При этом условию соответствует булева формула, а выполнению условия – равенство этой формулы единице на определенных входных наборах, причем каждой дуге соответствует только 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, что 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 [29] кодированием состояний, который в дальнейшем в

случае необходимости, например при наличии вершин, помеченных одинаково, и отсутствии различающих входных наборов, преобразуется в граф переходов автомата другого типа с явным введением понятия "состояние" (граф переходов автомата Мура) или другого типа кодирования (граф переходов автомата без выходного преобразователя с принудительно-свободным [29] кодированием состояний), не изменяя структуру первоначально построенного графа переходов. В отдельных случаях учет ограничений требует использования графа переходов смешанного автомата, что 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.

3.5. Описание функционирования автоматов без памяти. Применение изложенной методики позволяет реализовать условия функционирования, не различая соответствует ли им автомат с памятью или 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 набора выходных переменных в рассматриваемый момент времени зависят только от набора входных переменных в тот же момент времени, в применении графов переходов нет необходимости. Поэтому если удастся определить, что граф переходов описывает автомат без памяти или может быть сведен к нему, то граф переходов целесообразно заменить, например, системой булевых формул.

3.6. Свойства графов переходов. Одно из достоинств графов переходов состоит в том, что они могут быть формально проверены на синтаксическую корректность (смысловая корректность, естественно, формально проверена быть не может). При этом граф переходов считается корректным, если он непротиворечив, полон, не содержит генерирующих контуров, отличных от петель.

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. Если одновременные переходы из одной вершины допустимы, то такие графы переходов, в каждом из которых допустимо одновременное существование нескольких "активных" вершин, называются графами переходов с параллелизмом. Графы переходов с параллелизмом отличаются от диаграмм "Графсет" только отсутствием возможности осуществления внутри одной компоненты синхронизации завершения параллельных процессов. При использовании системы взаимосвязанных графов переходов с параллелизмом и описании каждого алгоритма отдельным графом переходов (компонентой) синхронизация процессов при необходимости осуществляется в головном графе переходов. Поведение графов переходов с параллелизмом (определяемое графом достижимых маркировок, являющимся графом переходов между всеми возможными состояниями компоненты или системы компонент) отличается от его описания: граф переходов с параллелизмом делает "больше", чем описывает его структура.

В этом смысле графы переходов (без флагов и умолчаний зависящих от предыстории значений выходных переменных) аналогичны параллельно-



последовательным контактными схемам, для каждой из которых булева формула, описывающая ее структуру, одновременно задает и ее функционирование (поведение), а графы переходов с параллелизмом аналогичны мостиковым контактными схемам, для которых булева формула, описывающая поведение каждой из них, не задает ее структуру.

При этом, если в графе переходов этого типа понятия "вершина" и "состояние" являются синонимами, то для графов переходов с параллелизмом эти понятия не эквивалентны.

Эта особенность графов переходов с параллелизмом, позволяющая компактно и обзорно описывать и реализовывать некоторые классы параллельных процессов одной компонентой (так как в противном случае пришлось бы строить и реализовывать графы переходов с большим числом состояний), связана с потерей важнейшего свойства графов переходов без флагов и умолчаний. Это свойство состоит в том, что число вершин в таком графе переходов совпадает с числом состояний описываемого им автомата.

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") с помощью системы булевых формул и обратном построении графа переходов по этой системе, в новом графе появляются вершины, отсутствующие в исходном задании [39]. 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, с помощью конструкции switch языка Си, указанное свойство ввиду их изоморфизма может быть сохранено полностью [29].

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 удастся. 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 происходит при пометке  $x_4$ , а в программе этому переходу соответствует пометка  $x_4 \& !x_5$ , то такую ошибку без полного перебора или построения графа переходов по программе обнаружить можно только случайно. Это еще раз подтверждает высказывание Э. Дейкстры о том, что с помощью тестов можно обнаруживать 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 большое внимание уделяется построению "понятных" Специалистам разного профиля алгоритмов и программ, что должно позволить устранить многие ошибки в них 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 средством отладки, а средством сертификации программ.

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

– при разновременном приходе "противоречивых" значений переменных;

- при работе с фронтами переменных ("события");
- 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 автоматов без выходного преобразователя или автоматов Мура могут умалчиваться. При этом предполагается, что пометка петли в вершине обеспечивает "полноту" последней.

В графах 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).

3.7. 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 графов переходов выбираются независимо от значений выходных переменных, связанных с этими вершинами.

"Свобода" кодирования для этих типов автоматов при программной реализации состоит также и в том, что сами коды при выбранном виде кодирования в отличие от асинхронной аппаратной реализации могут присваиваться вершинам графов переходов произвольно.

Из всех видов свободного кодирования при программной реализации автоматов наиболее целесообразно использовать два вида кодирования: двоичное и многозначное (целочисленное).

В первом случае  $i$ -й вершине графа переходов присваивается одна двоичная переменная  $Y_i$ , принимающая единичное значение только в  $i$ -й вершине и нулевое во всех остальных вершинах.

Во втором случае  $I$ -му графу переходов в целом присваивается одна многозначная переменная  $YI$ ,  $j$ -е значение которой, в свою очередь, присваивается  $j$ -й вершине графа переходов, что 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 необходимо отметить, 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 одной 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 допустимости изменений значений входных переменных только в начале программного цикла при одном порядке срабатывания (нумерации) элементов она будет иметь одно полностью детерминированное поведение, а при другой нумерации – другое также полностью детерминированное поведение, причем ни то ни другое поведение может не соответствовать желаемому.

Поэтому при программной реализации для любого вида кодирования состояний, использующего в том числе несоседние наборы переменных, при формальном 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 значение "10" является промежуточным и "фильтруется". Переход "00-01-11" 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.

3.8. Особенности использования графов переходов. При реализации алгоритма одним графом переходов автомата Мура или автомата без выходного преобразователя и формальном переходе к тексту программы по графу переходов без флагов и умолчаний, являющемуся одновременно и графом достижимых маркировок, полностью описывающим поведение автомата, этот граф может служить также и тестом для проверки правильности программы. Если программа в этом случае построена по графу переходов не только формально, но и изоморфно (обеспечена изобразительная эквивалентность между графом переходов и текстом программы), то тестирование может быть заменено сверкой ее текста с графом переходов.

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

Взаимодействие между графами переходов в системе взаимосвязанных графов переходов может осуществляться по входным, выходным, а, самое главное, многозначным 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 в диаграммах "Графсет") и на дугах графов переходов могут не только устанавливаться и сбрасываться двоичные выходные переменные, но могут и запускаться процессы, описанные, например, как с помощью графов переходов, так и с помощью схем алгоритмов. При описании процесса, происходящего в некоторой вершине графа переходов, с помощью схемы алгоритма, он многократно выполняется, до тех пор пока автомат находится в этой вершине, и завершается (после окончания очередного прохода схемы алгоритма) при переходе графа переходов в новую вершину. В качестве примера такого процесса можно привести реализацию в вершинах графа переходов функциональных элементов задержки с помощью импульсной переменной, принимающей единичные значения один раз в секунду.

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 графов переходов) графов достижимых маркировок.

3.9. Основные этапы алгоритмизации при использовании графов переходов. Разрабатывается схема связей "источники информации – управляющий автомат – средства представления информации – исполнительные механизмы".

Управляющий автомат декомпозируется на автомат и функциональные элементы задержки, что 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 – автомат – 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 "источники информации – система взаимосвязанных автоматов – функциональные элементы задержки – средства представления информации – исполнительные механизмы" 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 последняя схема 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 задержки и моделей объектов управления. Построение спецификаций завершает вторую стадию проектирования управляющей программы.

На третьей стадии проектирования при необходимости осуществляется выбор, построение и оптимизация алгоритмических моделей, реализующих формальные спецификации, с учетом рода (первого или второго) принятой структурной модели каждого автомата с памятью, например автомата Мура второго рода [39].

Предлагаемая технология включает методы формализованного перехода от графов переходов к различным типам алгоритмических моделей, например к таким, как системы булевых формул, функциональные схемы, лестничные схемы, схемы алгоритмов без внутренних обратных связей.

При этом, естественно, что и графы переходов также являются алгоритмической моделью, перед программированием которой 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 для каждой алгоритмической модели должен осуществляться выбор программной модели, содержащей в частности перечень используемых операторов.

Развернутый пример, иллюстрирующий использование предлагаемого подхода, приведен в [40].

3.10. 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 производиться 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 между текстом программы и 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 программный цикл в программе выполняется не более одного перехода в графе переходов. В противном случае, если 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 конструкцией **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.

3.11. 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 управляющего автомата, а при необходимости и комплекса "управляющий автомат – объект управления", описав не только автомат, но и модель объекта управления (по компонентам и/или режимам) с помощью графов переходов.

Под руководством автора разработана программная оболочка, позволяющая на персональном компьютере для системы взаимосвязанных графов переходов, состоящей из N графов переходов, реализованных на языке Си, изменять с помощью клавиатуры значения входных переменных и наблюдать на дисплее значения выходных, временных, а самое главное, N внутренних переменных (всего по одной внутренней переменной для каждого графа переходов) в пошаговом 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**, что принципиально отличает предлагаемую технологию от традиционных технологий программирования, при использовании которых всегда имеются возможности 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 её 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 беспорядочно 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 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 (синтез) от построенной программы к тексту программы на применяемом языке.

Например, для программируемых логических контроллеров "Autolog" фирмы "FF-Automation" Б.П. Кузнецовым (НПО "Аврора") совместно с автором разработан транслятор "язык Си – язык ALPro", позволяющий по тексту структурированной программы, написанной по графу переходов на некотором подмножестве языка Си, автоматически получать программу на языке инструкций ALPro с выбранными до трансляции управляющими конструкциями условным переходом или шаговым регистром [3].

При ограничениях на внутренние ресурсы программируемых логических контроллеров автором разработана методика "ручной" формальной реализации графов переходов в базе языка ALPro. Разработаны также методики реализации графов переходов в базе таких языков программирования, как лестничные и функциональные схемы [39]. Одна из этих методик позволяет, в частности, строить функциональные схемы, изоморфные графы переходов, в базе библиотечных элементов для системы "Selma-2" [41].

#### **4. Выводы.** Предлагаемый подход позволяет:

- использовать теорию конечных детерминированных 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 понятие "состояние", начиная алгоритмизацию с определения числа состояний;
- ввести понятие "автоматное программирование" и "автоматное проектирование программ";
- построение алгоритмов и программ начинать 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 по состояниям и выходам, а по возможности и зависимость значений выходных переменных автоматов с памятью от значений входных переменных;
- 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 автоматов Мура, у которых коды состояний и значения выходов принципиально разделены, а значения выходных переменных в каждом состоянии не зависят от входных переменных, что упрощает внесение изменений;
- 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, применяемых в программируемых логических контроллерах;
- проводить алгоритмизацию в результате взаимного общения Заказчика, 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;
- в качестве сертификационного теста применять графы переходов без флагов и умолчаний, у которых число вершин совпадает с числом состояний автомата, и строить граф проверки или граф достижимых маркировок для других классов графов переходов или систем взаимосвязанных графов переходов с целью проверки их поведения, что

позволяет заменить тестирование программ анализом их функциональных возможностей;

– ввести формальный критерий понятности различных форм описаний автоматов с памятью – изоморфизм каждого из них с соответствующим графом переходов без флагов и умолчаний. При этом автомат, заданный таким графом переходов, может быть назван "абсолютно белым ящиком", а автомат, заданный в любой другой форме – "относительно белым ящиком". Автомат, для которого известно максимальное число состояний в его минимальной форме, который может быть распознан на основе анализа "входо-выходных" последовательностей, назван в [42] "относительно черным ящиком", а автомат, о внутреннем содержании которого ничего не известно, не распознаваемый указанным образом – "абсолютно черным ящиком". Таким образом, если понятие "состояние" при алгоритмизации и программировании не вводится, то в результате тестирования алгоритмов и программ с помощью входо-выходных последовательностей в общем случае не удастся распознать автоматы, которые они реализуют;

– 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 конструкций switch (в том числе вложенных) или им аналогичных, что кроме изоморфизма со спецификацией обеспечивает доступность каждого значения многозначной 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;

– ввести в программирование понятие "наблюдаемость", что обеспечивает возможность рассмотрения программы в качестве "абсолютно белого ящика", в котором все внутренние переменные, число которых минимально, доступны для наблюдения;

– проводить проверку программ в два этапа: сначала по номерам состояний проверить наличие всех переходов, имеющих в графе переходов, а затем в статике в каждом состоянии проверить значения выходных переменных;

– Участникам разработки (Заказчику, Технологию (Проекту), Разработчику, Программисту, Оператору (Пользователю) и Контролеру) однозначно и полностью понимать, что должно быть сделано, что делается и что сделано в функциональной части программно реализуемого проекта, т.е. решить для рассматриваемого класса задач проблему взаимопонимания;

– на ранних стадиях проектирования учесть все детали Технического Задания и продемонстрировать Заказчику как оно понято;

– разделить работу, а самое главное, ответственность между Заказчиком, Технологом, Разработчиком и Программистом. Это особенно важно, когда указанные Специалисты представляют разные организации, а тем более страны, так как в противном случае возникают существенные языковые, а, в конечном счете, и экономические проблемы;

– 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", что



не вызывает 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 [43];

- снять с Программиста необходимость знания особенностей технологического процесса, а с Разработчика – тонкостей программирования;

- Программисту функциональных задач ничего не додумывать за Заказчика, Технолога и Разработчика, а только однозначно и формально реализовывать систему графов переходов в виде программы, 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 возможно, однако, только в том случае, когда программирование является для Разработчика "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 алгоритмы управления инвариантными к используемым языкам программирования, что открывает возможность формирования и поддержания библиотек алгоритмов, записанных строго формально;

- Заказчику, 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 Контролером, так как в первом случае им кроме функционирования проверяется много других характеристик (например, качество печатных плат и их покрытий, качество пайки, номиналы и обозначения элементов), а во втором – все внимание уделяется только проверке функционирования и не исследуются внутренняя организация программ и технология их построения.

#### **5. Аппробация.** Предлагаемый подход, в частности, использован:

- при создании системы управления дизель-генератором ДГР-2А 500\*500 судна проекта 15640 на базе аппаратуры "Selma-2". Программирование выполнено НПО "Аврора" на языке функциональных блоков [2, 41];

- при создании совместно с фирмой "Norcontrol" (Норвегия) системы управления дизель-генератором того же типа для судов проекта 15760. Программирование выполнено фирмой на языке ПЛ/М [43];

- при создании комплексной системы управления техническими средствами для судна проекта 17310 на базе программируемых логических контроллеров "Autolog". Программирование выполнено НПО "Аврора" на языке инструкций ALPro вручную (для общесудовых систем) и автоматически с помощью транслятора (для систем управления вспомогательными механизмами главного двигателя).

**Заключение.** В течение всего времени пока создавалась предложенная и внедренная автором в 1991 году

SWITCH-технология [41,44], складывалась весьма странная ситуация, состоящая в том, что ни одна (за исключением [22]) из ведущих в области автоматизации фирм мира не предлагала технологий алгоритмизации и/или программирования на базе графов переходов, являющихся основой рассматриваемой методологии.

Только в 1996 году эта ситуация существенно изменилась – фирма "Сименс" в дополнение к своим программным продуктам разработала новый продукт, названный "S7-HiGraph technology software", который позволяет использовать в качестве языка программирования диаграммы состояний (state diagram), что является другим названием графов переходов. При этом по описанию на этом языке автоматически генерируется исполняемый (только на программируемых логических контроллерах этой фирмы) код.

По мнению фирмы "Сименс", описание на таком языке не только подходит для Программиста программируемых логических контроллеров, но также понятно Инженеру-механику, Инженеру по запуску оборудования и Инженеру по обслуживанию [21].

Особенно странным в этой ситуации является то, что подобный продукт мог быть разработан, например и 15 лет назад, так как диаграммы состояний, в свою очередь, были предложены более сорока лет назад, однако, всемирное увлечение сетями Петри, которые обеспечивают возможность описания параллельных процессов одной компонентой (функциональной единицей) и являются поэтому основой языка "Графсет" (S7-Graph technology software), видимо, психологически не позволило этого сделать.

Автор надеется, что после появления указанного продукта будет иметь место эффект "домино", и в ближайшее время многие фирмы мира создадут аналогичные продукты.

При этом книга [39], подробно описывающая предлагаемую технологию, в которой графы переходов и системы взаимосвязанных графов переходов предлагается использовать не только в качестве языка программирования, но и в качестве языка спецификаций задач логического управления при применении любых других языков программирования, например Си или Си++, может стать полезным пособием по использованию управляющих графов при алгоритмизации и программировании для широкого класса Пользователей промышленных компьютеров и программируемых логических контроллеров, например [45], в том числе и в случаях отсутствия трансляторов с этого языка спецификаций.

Излагаемая технология – существенное дополнение Международного стандарта IEC 1131-3 [46], в котором описываются языки программирования программируемых логических контроллеров, но не излагаются методы алгоритмизации и программирования, которые подробно описаны в [39].

Предлагаемая технология может стать основой для повышения безопасности программного обеспечения систем логического управления [47]. Эта технология, не исключая других методов построения программного обеспечения "без ошибок" [48], существенно более конструктивна, так как позволяет начинать "борьбу с ошибками" еще на стадии алгоритмизации.

Использование пентады (состояние – независимость от "глубокой" предыстории – система взаимосвязанных графов переходов – многозначное кодирование – конструкция switch (или ее аналог в любом языке программирования, например, в языке функциональных схем)) обеспечивают наглядность, структурность, вызываемость, вложенность, иерархичность, управляемость и наблюдаемость программ, а также их изоморфизм (изобразительную эквивалентность) со спецификациями, по которым они формально строятся. Это позволяет Заказчику, Технологию

(Проектанту), Разработчику, Программисту, Оператору (Пользователю) и Контролеру однозначно понимать друг друга и точно знать, что должно быть сделано, что делается и что сделано в программно реализуемом проекте. Это позволяет также разделять работу и ответственность, легко и корректно вносить изменения в алгоритмы и программы.

Таким образом можно утверждать, что по аналогии с теорией переключательных схем и теорией вероятностей в настоящей работе предлагается вместо "мостиковых и немарковских" программ строить "параллельно-последовательные и марковские" программы, которые хорошо читаются и понимаются.

В заключение отметим, что в [49] было предложено использовать графы переходов при программной реализации алгоритмов логического управления технологическими процессами. Однако изложенный в этой работе подход был недостаточно "изящным", что не позволило ему найти широкое практическое применение.

Применение систем взаимосвязанных графов переходов было рассмотрено в работах В.В. Руднева (например, [50]). Однако теоретический характер этих работ и использование двоичных переменных для связи графов ограничили применение этой модели на практике.

Автор надеется, что судьба предлагаемого подхода в части его практического использования будет более успешной.

#### СПИСОК ЛИТЕРАТУРЫ

1. Дейкстра Э. Дисциплина программирования. М.: Мир, 1979.
2. "Селма-2". Описание функциональных блоков. АБВ Стромберг Драйвс, 1989.
3. Autolog 32. Руководство пользователя. FF-Automation.
4. Programmable Controller. MELSEC - A. Programming Manual. Type A CPU. Common Instructions. Mitsubishi Electric.
5. Поспелов Д.А. Ситуационное управление. Теория и практика. М.: Наука, 1986.
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. Малюгин В.Д. Реализация булевых функций арифметическими полиномами //Автоматика и телемеханика. 1982. N4.
9. Артюхов В.Л., Кондратьев В.Н., Шальто А.А. Реализация булевых функций арифметическими полиномами //Автоматика и телемеханика. 1988, N4.
10. Филд А., Харрисон П. Функциональное программирование. М.: Мир, 1993.
11. Баранов С.И. Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.
12. Mystic Controller. Opto. Booklet N1-800-321-OPTO.
13. Nassi J., Shnenderman B. Flowcharte Techiques for Structured Programming //SIGPLAN Not. 1973. N8.
14. Ляпунов А.А. О логических схемах программ //Проблемы кибернетики. Вып.1. М.: Физматгиз, 1958.
15. Бардзинь Я.М., Калниньш А.А., Стродс Ю.Ф., Сыцко В.А. Язык спецификаций SDL /PLUS/ и методика его использования. Рига: ЛГУ, 1986.
16. Вельбицкий И.В. Технология программирования. Киев: Техника, 1984.
17. Питерсон Д. Теория сетей Петри и моделирование систем. М.: Мир, 1984.
18. Юдицкий С.А., Мачергут В.З. Логическое управление дискретными процессами. М.: Машиностроение, 1987.
19. Мишель Ж. Программируемые контроллеры. Архитектура и применение. М.: Машиностроение, 1992.
20. Бергер Г. Программирование управляющих устройств на языке STEP-5. Том 1. Программирование основных функций. Сименс. 1982.
21. SIMATIC. Sematic S7/M7/C7. Programmable Controllers. SIEMENS. Catalog ST 70. 1996.

22. Modicon Modsoft. Руководство программиста. GM-MSFT-001. Ред.Е. 1993.
23. Гаврилов М.А., Девятков В.В., Пупырев Е.И. Логическое проектирование дискретных автоматов. М.: Наука, 1977.
24. Амбарцумян А.А., Искра С.А., Кривандина Н.Ю. и др. Проблемно-ориентированный язык описания поведения систем логического управления //Проектирование устройств логического управления. М.: Наука. 1984.
25. Девятков В.В., Чичковский А.Б. Условие-82 - язык программно-логического управления //Автоматизация проектирования. Вып.2. М.: Машиностроение, 1990.
26. Горбатов В.А., Смирнов М.И., Хлытчев И.С. Логическое управление распределенными системами. М.: Энергоиздат, 1991.
27. Кузнецов О.П., Шипилина Л.Б., Марковский А.В. и др. Проблемы разработки языков логического программирования и их реализация на микроЭВМ (на примере языка "Ярус-2") //Автоматика и телемеханика. 1985. №6.
28. Кузнецов О.П. Графы логических автоматов и их преобразования //Автоматика и телемеханика. 1975. №9.
29. Шалыто А.А. Использование граф-схем алгоритмов и графов переходов при программной реализации алгоритмов логического управления //Автоматика и телемеханика. 1996. №6,7.
30. Аперидические автоматы. /В.И. Варшавский, Л.Я. Розенблюм, В.В. Мараховский и др. М.: Наука, 1976.
31. Лингер Р., Миллс Х., Уитт С. Теория и практика структурного программирования. М.: Мир, 1982.
32. Буч Г. Объектно-ориентированное проектирование с примерами применения. Киев: Диалектика; М.: АО "ИВК", 1992.
33. Матчо Д., Фолкнер Д. DELPHI. М.: БИНОМ, 1995.
34. Шлеер С., Меллор С. Объектно-ориентированный анализ. Моделирование мира в состояниях. Киев: Диалектика, 1994.
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 Н.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. Т.5. №2.
39. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб., Наука, 1998.
40. Шалыто А.А., Антипов В.В. Алгоритмизация и программирование задач логического управления. СПб.: Моринтех, 1996.
41. Project 15640. AS 21.DG1. CONTROL. АМИЕ. 95564.12М. St.Petersburg. ASS "Aurora", 1991.
42. Трахтенброт Б.А., Бардзинь Я.М. Конечные автоматы. Поведение и синтез. М.: Наука, 1970.
43. Functional Description. Warm-up & prelubrication logic. Generator Control Unit. Severnaya hull N431. Norcontrol, 1993.
44. Шалыто А.А. Программная реализация управляющих автоматов // Судостроит. пром-ть. Сер. Автоматика и телемеханика. 1991. Вып.13.
45. Modicon-Telemecanique. TSX Micro industrial programmable controllers. Groupe Schneider, Catalog, 1996, March.
46. International Standart IEC 1131-3. Programmable controllers. Part 3. Programming languages // International Electrotechnical Commission, 1993.
47. Underwriters Laboratories обновляет стандарт по безопасности программного обеспечения для ПЛК //Современные технологии автоматизации. 1997. №1.
48. Бейбер Р.Л. Программное обеспечение без ошибок. Дж. Уайли энд санз. Радио и связь. 1996.
49. Горбатов В.А., Кафаров В.В., Павлов П.Г. Логическое управление технологическими процессами. М.: Энергия, 1978.
50. Руднев В.В. Система взаимосвязанных графов и моделирование дискретных процессов //Автоматика и телемеханика. 1984. №9.