Глава 2

Архитектурное проектирование систем логического управления

Одним из классов систем управления являются системы логического управления (СЛУ).

Специфика СЛУ состоит в том, что в них входные X и выходные Z переменные могут принимать только два значения — 0 и 1.

При проектировании таких систем одним из важнейших вопросов является построение управляющих автоматов. Применение микропроцессорных средств в системах этого класса приводит к необходимости разработки методов программной реализации таких автоматов. На базе этих методов может быть создана технология алгоритмизации и программирования задач логического управления.

Существует большое число работ, посвященных отдельным вопросам построения СЛУ, в том числе и управляющих автоматов [1—58]. Однако вопрос о взаимосвязи различных подходов и создании на их основе единой концепции построения этого класса автоматов в должной мере в настоящее время не решен.

В предлагаемой работе на основе методов, разработанных автором, в том числе и в соавторстве, а также известных методов делается попытка решения указанного вопроса.

В работе обоснован выбор графов переходов и их систем в качестве языка алгоритмизации, изложены методы их построения по другим формам задания и методы обратного перехода. При этом рассматриваются такие языки, как булевы формулы, таблицы решений, граф-схемы алгоритмов, логические схемы алгоритмов, сети Петри, графы операций, диаграммы «Графсет» и другие.

Разработаны методы программной реализации первичных описаний в. базисе языков программирования высокого уровня, например СИ, языков низкого уровня — инструкций и ассемблеров, а также специализированных языков — лестничных и функциональных схем.

Эти подходы концептуально объединены в единую технологию алгоритмизации и программирования задач логического управления, названную SWITCH-технологией.

2.1. Объекты и системы логического управления

Объектами управления (ОУ) для СЛУ являются клапаны, вентиляторы, электродвигатели, заслонки, захлопки и т. д.

СЛУ воздействуют на ОУ не непосредственно, а через исполнительные механизмы (ИМ) — магнитные пускатели, электрогидропреобразователи, электропневмопреобразователи и т. д. Поэтому при построении таких систем предполагается, что ИМ входят в состав ОУ.

Объекты управления в зависимости от своих свойств (например, наличие или отсутствие внутренней (распределенной [2]) памяти в ИМ) обладают различным поведением, которое перед построением системы должно быть описано.

Информация о работе ОУ поступает в СЛУ от сигнализаторов положения, состояния и параметров. При этом необходимо отметить, что сигнализаторы параметров состоят из двух составляющих — датчика и вторичного прибора, осуществляющего сравнение аналогового сигнала от датчика с заданной установкой и формирование двоичного сигнала при превышении установки.

Таким образом, до построения системы должна быть построена подробная технологическая схема объекта с указанием всех точек контроля и управления.

На рис. 2.1 в общем виде изображен объект управления с указанием всех входных Z] и выходных X_i его переменных.

Проектирование системы целесообразно начинать с разработки схемы связей «СЛУ—ОУ» (рис. 2.2).

СЛУ в общем случае состоит из ряда составляющих: средств представления информации (СПИ), органов управления (ОРУ), управляющего автомата (УА), выходных усилителей, источников питания и т. д.

Для построения модели управляющего автомата из перечисленных компонентов основное значение имеют СПИ и ОРУ, а остальные составляющие в дальнейшем рассматриваться не будут. Указанные компоненты (ОРУ, УА и СПИ) образуют ядро СЛУ (рис. 2.3).

Средства представления информации. В качестве СПИ используются лампы сигнализации, табло, мнемосхемы, видеоконтрольные устройства и т. д. В зависимости от требований, предъявляемых к системе логического управления, необходимо осуществить выбор СПИ.

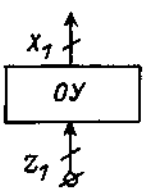

Рис. 2.1

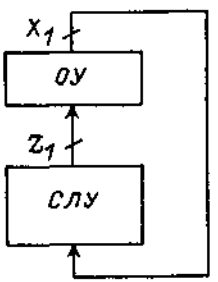

Рис. 2.2

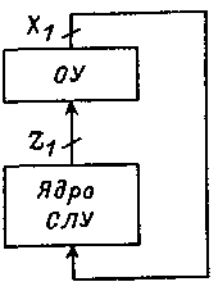

Рис. 2.3

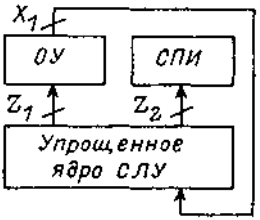

Рис. 2.4

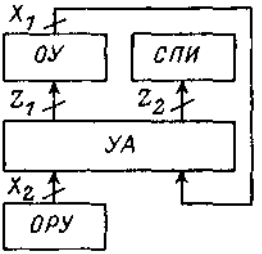

Рис. 2.5

Для целей настоящей работы применительно к СПИ важно лишь то, что они при построении управляющего автомата выступают наряду с исполнительными механизмами в качестве приемников информации (ПИ). Выделяя СПИ из ядра системы и вводя переменные Z_2, сформируем упрощенное ядро СЛУ (рис. 2.4).

Органы управления. В качестве ОРУ в системах логического управления используются ключи, тумблеры, кнопки (как с памятью, так и без памяти) и т. д. В зависимости от требований, предъявляемых к СЛУ, осуществляется выбор ОРУ. Необходимо отметить, что при программной реализации в качестве ОРУ обычно используются кнопки без памяти.

Для настоящей работы применительно к ОРУ важно лишь то, что при построении управляющего автомата они выступают наряду с сигнализаторами объекта управления в качестве источников информации (ИИ). Выделяя ОРУ из упрощенного ядра СЛУ и вводя переменные X_2, сформируем управляющий автомат (рис. 2.5). Построенная схема представляет собой схему связей «ИИ—У А—СПИ—ИМ», так как УА непосредственно с ОУ не взаимодействует.

2.2. Модели управляющих автоматов

В общем случае управляющий автомат состоит из автомата и функциональных элементов задержки.

При этом автомат рассматривается не как временной объект, а в качестве замкнутого объекта, изменяющего свои состояния (осуществляющего переходы между состояниями) под влиянием тех или иных событий, объединяемых в общем случае в булевы формулы, равенство единице каждой из которых инициирует соответствующий переход между состояниями, включая сохранение состояния.

В зависимости от размещения ФЭЗ возможны две модели управляющих автоматов. Первая из них используется в тех случаях, когда ФЭЗ непосредственно входят в контур управления (рис. 2.6). Более универсальной и широко применяемой является вторая модель, в которой выходы ФЭЗ для автомата рассматриваются в качестве источников, а их входы — в качестве приемников информации наряду с ИМ и СПИ (рис. 2.7).

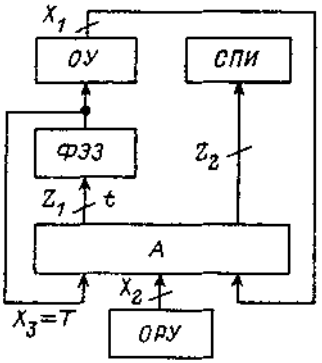

Рис. 2.6

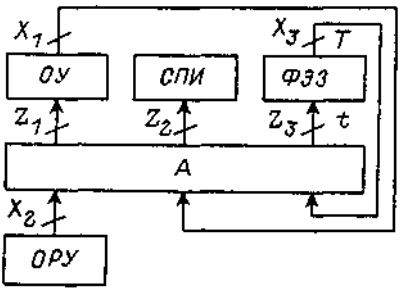

Рис. 2.7

В результате строится схема связей «ИИ—А—ФЭЗ—СПИ—ИМ».

В большинстве случаев построение автомата в виде единого компонента либо затруднительно, либо нецелесообразно.

Поэтому при необходимости автомат декомпозируют в общем случае на систему взаимосвязанных автоматов (СВА), взаимодействующих между собой по принципу вызываемости (гл. 8) или принципу вложенности (гл. 12).

Декомпозиция автомата может производиться по объектам управления, по режимам или смешанным образом. При этом получающаяся система взаимосвязанных автоматов может быть в общем случае иерархической.

Этап архитектурного (системного) проектирования завершается построением схемы связей «ИИ—СВА—ФЭЗ—СПИ—ИМ», которая совместно с описанием свойств источников и приемников информации, используемых в ней, является основой для дальнейшего проектирования.

Эта схема наряду с формализованным описанием поведения автоматов в наглядной форме (в виде графов переходов и графов достижимых маркировок) позволяет отказаться от словесного описания работы каждого автомата в технической документации, оставляя его в случае необходимости лишь в виде комментариев, так как в полном объеме словесное описание является весьма громоздким и практически нечитаемым, а также не позволяет оценить его полноту и непротиворечивость. При этом в документации в словесной форме должны быть изложены лишь правила чтения моделей поведения автоматов.

2.3. Выбор языка спецификаций

Техническое задание на проектирование СЛУ должно содержать описание алгоритмов логического управления, сигнализации и контроля в словесной или иной форме.

В техническом задании на построение системы некоторые ее функции, например в части контроля работы объекта управления и Оператора, обычно не задаются. Это приводит к необходимости расширения исход-

ного описания (разд. 4.4.9). Дополнение задания должно быть согласовано с Заказчиком.

Задание в словесной форме обычно является логически неполным, иногда противоречивым, а главное, в ряде случаев двусмысленным. Задания часто содержат техницизмы, аналогичные выражению «косой пошел с косой» [34], которые требуют дополнительных разъяснений.

В силу недостаточной формализованности исходного задания в такой форме при испытаниях системы возникают проблемы с доказательством правильности реализованных алгоритмов.

Изложенное приводит к необходимости создания по исходному заданию спецификации [102] в форме, устраняющей указанные недостатки и обеспечивающей простоту ее согласования с Заказчиком. Кроме того, выбранный язык спецификаций должен обеспечить весьма простой и формализованный переход к программной реализации, причем спецификация должна быть отражена в структуре программы таким образом, чтобы имелась возможность «видеть» ее в тексте программы.

При этом возникает важный вопрос о чтении и понимании спецификаций и программ. Предположим, что задан некоторый текст программы, например на языке ассемблер, который реализует систему булевых формул, описывающую работу автомата с памятью. Прочесть эту программу в традиционном понимании этого термина — это также и понять, как выполняются команды программы, и выяснить, реализует ли она заданную систему. Это можно назвать «пониманием программы в малом», так как, поняв, как работает программа, остается неясным, как «выглядит» процесс, который она реализует. Ситуация в этом смысле практически не улучшается, если программа, реализующая автомат в виде системы булевых формул, написана на языке высокого уровня или изображена в виде функциональной или лестничной схемы. Эта ситуация напоминает случай, когда человек может прочесть текст на иностранном языке, но плохо или совсем не понимает его содержание.

В настоящей работе термин «прочесть программу» трактуется более широко, т. е. при чтении программы ее необходимо понять настолько, чтобы непосредственно по ее тексту, без дополнительных математических преобразований можно было восстановить спецификацию в форме, описывающей процесс в динамике. Такую трактовку термина «понимание» будем называть «пониманием в большом».

Рассмотрим более подробно по сравнению с гл. 1 языки спецификаций алгоритмов логического управления, получившие наибольшее распространение в практике проектирования систем логического управления.

2.3.1. Функциональные схемы

Наиболее широко используемым в настоящее время языком спецификаций в системах логического управления является язык функциональных схем, который обычно применяется при аппаратной реализации алгоритмов. Этот язык широко используется также и в качестве языка программирования в программируемых логических контроллерах и управ-

ляющих системах, таких, например, как система «Selma-2», описанная в [60].

Доступность и традиционность языка функциональных схем для инженеров, не являющихся программистами, создают иллюзию простоты формализации и программирования. При этом возникают по крайней мере два вопроса:

— откуда «берутся» функциональные схемы?

— как читать и проверять такие схемы для автоматов, использующих триггеры и обратные связи?

Обычно считается, что одним из основных достоинств традиционных языков программирования ПЛК (функциональных и лестничных схем) является то, что они позволяют легко перейти от «старых» схем с «жесткой» логикой к программируемым средствам.

Однако непосредственное программирование по «старым» схемам, несмотря на, казалось бы, простоту и удобство, при программной реализации не всегда рационально. Это объясняется тем, что «жесткая» логика сдерживает уровень автоматизации, так как усложнение алгоритма управления в этом случае приводит к пропорциональному росту объема аппаратуры. Эта зависимость существенно снижается при программной реализации алгоритмов, что позволяет значительно поднять уровень автоматизации. При этом достаточно «прозрачные» ФС за счет возможности усложнения реализуемого алгоритма практически при тех же аппаратных затратах преобразуются в весьма сложные, которые трудно читаются и понимаются. Кроме того, аппаратная асинхронная реализация управляющих автоматов, характерная для многих систем управления, требует их усложнения для обеспечения работоспособности, например в части устранения состязаний, что приводит к избыточности по сравнению со схемами, отражающими только алгоритм управления и достаточными для программной реализации.

Имеются три подхода к построению ФС при такой реализации:

— построение спецификации и формальный синтез по ней схемы управляющего автомата (при этом выполнять анализ построенной схемы нет необходимости);

— эвристический синтез схемы управляющего автомата, формальный и семантический анализ ее с целью исследования всех функциональных возможностей и корректировка в случае необходимости построенной в результате анализа спецификации с последующим формальным синтезом по ней схемы управляющего автомата;

— эвристический синтез схемы управляющего автомата и тестирование, подтверждающее выполнение технического задания, но не позволяющее выполнить верификацию построенного автомата.

Если первые два подхода дают при программной реализации гарантию правильности построения автомата, то третий подход этого не обеспечивает. Однако в настоящее время в практике проектирования автоматов для СЛУ именно этот подход и является наиболее распространенным, что является весьма опасным в случае ответственных объектов управления.

Еще одним вопросом, близко примыкающим к рассмотренному, является вопрос об эквивалентности различных форм представления одного

и того же алгоритма логического управления или о формальном переходе от одной формы представления к другой. Рассмотрим в качестве примера две задачи:

1. Пусть имеются ФС и ГСА, построенные эвристически одним и тем же человеком. Доказать, что они эквивалентны.

2. Пусть имеется ГСА. Требуется построить по ней эквивалентную ФС.

Умение практически и непосредственно решать подобные задачи (несмотря на алгоритмическую неразрешимость проблемы распознавания эквивалентности алгоритмов в общем случае [268]) позволяет переносить отлаженные алгоритмы управления с одних программируемых средств на другие, а не разрабатывать алгоритм каждый раз сначала в новой форме. Имеется и другой подход к решению этих задач, развиваемый в настоящей работе, а именно: выбирается такой язык спецификаций, описание на котором позволяет осуществить формальный переход к указанным формам представления и выполнить при необходимости обратный переход. При этом считается, что описания, представленные в разных формах, эквивалентны, если они сводятся к одному и тому же описанию на языке спецификаций.

Решение указанных задач весьма актуально, если их постановка и программирование осуществляются разными людьми или разными организациями. При этом обычно после разработки программы Пользователю демонстрируют ее работу, в то время как текст программы ему не передается. Это не позволяет Пользователю быть полностью уверенным в правильности реализации алгоритма и в отсутствии в реализации чего-либо лишнего и делает его целиком зависимым от Программиста и его квалификации. Естественно, что передача текста программы проблему не решает, так как прочесть и понять чужую программу весьма сложно даже при наличии комментариев.

По мнению автора, все перечисленные вопросы могут быть решены или по крайней мере резко упрощены, если общение Заказчика, Технолога, Разработчика, Программиста, Пользователя и Контролера при создании СЛУ будет осуществляться не с помощью естественных языков или функциональных схем, а на основе графов переходов без флагов и умолчаний, каждый из которых в отличие от указанных схем отражает не структуру автомата, а его поведение в однозначно трактуемой форме.

Смысл использования ГП при анализе функциональных схем состоит в том, что к статической конструкции, задаваемой схемой, добавляется модель ее поведения. При этом процесс чтения резко упрощается, так как в этом случае при изменении значений входных переменных необходимо вычислить не всю систему булевых формул, описывающую схему, а лишь сравнительно простую ортогональную СБФ, задающую переходы из рассматриваемой вершины во все смежные вершины, и, определив новую вершину, прочесть значения выходных переменных, указанные в ней или на соответствующей дуге ГП. Использование ГП в качестве промежуточной формы представления алгоритмов логического управления позволяет решить весьма важные для практики вопросы эквивалентности различных форм их представления.

Переходя к более подробному рассмотрению вопроса об анализе и проверке ФС, необходимо отметить, что их анализ обычно проводится в основном лишь на тех наборах входных переменных, которые оговорены в техническом задании. При этом, однако, если задание на разработку управляющего автомата может быть не полностью определенным, то схема или программа всегда полностью определены и поэтому они так или иначе реагируют на любой входной набор, в том числе и не заданный в техническом задании. Однако в силу того что при построении схемы доопределение условий работы обычно осуществляется без анализа функционирования на исходно не заданных наборах, а по другим критериям, например по критерию минимальности «аппаратурных» затрат, то в случае, если такие наборы поступят на вход схемы, она будет себя вести для Пользователя в некотором смысле непредсказуемо.

Реакция на некоторые входные наборы может быть не определена ввиду следующих причин: 1) из-за «недоговоренности» технического задания и отсутствия вопросов Заказчику со стороны Разработчика; 2) из-за безразличности реакции на эти наборы; 3) из-за невозможности их появления, так как на их взаимосвязь могут быть наложены различные ограничения, в том числе и законами физики.

Последняя из указанных причин является самой неприятной, так как невозможность появления тех или иных значений переменных обычно определяется для «нормального» режима функционирования и может не рассматриваться для других режимов. Пусть, например, имеется цистерна, содержащая два сигнализатора (нижнего (x_1) и верхнего (x_2) уровней), срабатывающих в случае, если вода превышает соответствующий уровень. При этом на «ровном киле» определены следующие входные наборы: $x_1 = x_2 = 0$ (вода ниже нижнего уровня); $x_1 = 1$, $x_2 = 0$ (средний уровень); $x_1 = x_2 = 1$ (вода выше верхнего уровня). Ситуация $x_1 = 0$, $x_2 = 1$ при исправных сигнализаторах считается невозможной, что позволяет проводить доопределение на этом наборе, например по критерию упрощения сложности реализации управляющего автомата. Однако при крене или дифференте указанная ситуация может возникнуть, и построенный таким образом автомат выдаст реакцию, принятую при его доопределении, которая может привести в том числе и к аварийной ситуации.

Другая особенность функциональных схем состоит в том, что анализ их работы без привлечения других моделей весьма затруднителен, так как они являются лишь «схемами» вычислений и не содержат в явном виде результатов вычислений в виде наборов битов. Поэтому при традиционном анализе таких схем приходится в «голове» проводить вычисления, однако не для всех состояний, число которых обычно неизвестно, и тем более не для всех входных наборов. Тем самым все функциональные возможности схемы даже для ее автора остаются обычно не исследованными. Это затрудняет согласование схем и делает неполным их описание в технической документации. При этом сами схемы являются весьма «опасными» в эксплуатации.

ФС, построенные эвристически, весьма сложны для тестирования, так как по схеме трудно определить, какие входы влияют на переход из рассматриваемого состояния в следующее. При этом подать все входные наборы во всех состояниях практически невозможно, а если подать только

те тесты, которые показывают, что схема удовлетворяет заданным требованиям, то отсутствует уверенность в том, что схема не делает ничего лишнего.

Эта же проблема возникает и при чтении ФС, так как по ней не ясно, в какой последовательности необходимо подавать входные переменные для получения требуемых выходных реакций, и поэтому эту информацию приходится «держать в голове» [202]. Кроме того, в функциональные схемы весьма трудно вносить изменения.

Языки лестничных и релейно-контактных схем [59], а также булевых формул [40] в качестве языков спецификаций обладают теми же недостатками, а последний — еще и низкой наглядностью.

2.3.2. Граф-схемы алгоритмов

При программной реализации в качестве языка спецификаций автоматов часто используются граф-схемы алгоритмов [33]. ГСА в настоящее время недостаточно стандартизованы.

Для задач рассматриваемого класса они могут содержать в операторных вершинах присвоения выходным переменным или булевых формул, или констант 0 и 1. В первом случае ГСА весьма трудно читаются, а во втором — возможны две разновидности: вершина содержит или все, или только изменяющиеся выходные переменные. В случае если каждая вершина содержит все переменные, то те из них, которые принимают значения, равные нулю, могут не указываться по умолчанию [16]. Это должно оговариваться в описании ГСА, что, к сожалению, обычно не выполняется и приводит к неоднозначности в понимании их работы.

Условные вершины могут быть с двумя выходами и со многими выходами. В первом случае эти вершины могут содержать либо отдельные переменные, либо отдельные булевы формулы, а во втором — они помечаются системами булевых формул. Наиболее просто читаются ГСА с условными вершинами, содержащими только одиночные переменные.

ГСА отличаются также и числом операторов «Ввод» и «Вывод». Если при аппаратной реализации «Ввод» предполагается в каждой условной вершине с входными переменными, а «Вывод» — в каждой операторной вершине с выходными переменными [16], то при программной реализации обычно требуется, чтобы ГСА содержала лишь по одному указанному оператору. Из изложенного следует, что если при аппаратной реализации в ГСА допустимы возвраты назад, то при программной реализации при выполнении вычислителем более чем одного алгоритма должна существовать лишь одна внешняя обратная связь.

Если ГСА, содержащая только изменяющиеся выходные переменные, не имеет внутренних обратных связей и внутренних переменных, то в случае, когда на каждом пути от входа до выхода граф-схемы в операторных вершинах каждой выходной переменной присваивается некоторое значение, эта ГСА реализует обычно автомат без памяти, а когда хотя бы на одном пути от входа к выходу граф-схемы по крайней мере одна из выходных переменных не упоминается в операторных вершинах, расположенных на этом пути, эта ГСА реализует автомат с памятью.

При программной реализации ГСА, содержащая внутренние обратные связи, обычно должна быть преобразована, возможно, за счет введения дополнительных внутренних переменных таким образом, чтобы после преобразования она содержала лишь внешнюю обратную связь.

При построении ГСА отсутствует соглашение о порядке расположения условных вершин, поименованных входными, выходными и внутренними (в том числе и временными) переменными.

Кроме недостаточной стандартизации ГСА обладают также и рядом других недостатков, среди которых необходимо отметить:

— сложность тестирования по всем путям, без подсчета и последующей проверки каждого из которых невозможно верифицировать ГСА [100];

— сложность «чтения» и внесения изменений в ГСА в случае, если в операторных вершинах не указываются значения всех выходных переменных;

— наличие внутренних переменных, «неинтересных» Пользователю;

— использование обычно только битовых внутренних переменных, делающих ГСА весьма громоздкими даже для сравнительно простых алгоритмов;

— реализация функциональных элементов задержки в составе ГСА, что делает эту модель не автоматной (причинно-следственной), а временной.

В гл. 13 показано, как устранить указанные недостатки и как переходить от ГСА к ГП и обратно.

2.3.3. Графы переходов

Указанные недостатки отсутствуют у графов, построенных определенным образом и называемых графами переходов.

Рассмотрим однокомпонентный направленный граф, в котором отмечена начальная вершина, являющаяся конечной. Этот граф должен обладать тем свойством, что из каждой его вершины существует по крайней мере один путь в начальную вершину. Для упрощения чтения графа он должен быть максимально планарным.

Каждая дуга и каждая вершина в графе входит по крайней мере в один контур, содержащий не менее двух вершин. Контур, содержащий одну вершину и одну дугу, будем называть (как и дугу в этом случае) петлей. В графе могут существовать два типа контуров: контуры, которые содержат начальную вершину, и контуры, которые эту вершину не содержат.

Если граф содержит контуры только первого типа, то каждому из них соответствует путь (отличный от петли) из начальной вершины в саму себя без самопересечений. Если граф содержит также и контуры второго типа, то в нем существует по крайней мере один самопересекающийся путь из начальной вершины в саму себя.

Сопоставим в описанном графе его вершинам внутренние состояния автомата, называемые в дальнейшем состояниями, а его дугам — допустимые переходы между состояниями, включая переходы состояний в самих себя, называемые сохранением состояний. Так как автомат в каждый дискретный момент времени может находиться только в одном состоянии (одной вершине графа), то отметим вершину, в которой

«находится» автомат, маркером, символизирующим активность вершины и сохранение состояния. Тогда переходу автомата из одного состояния в другое соответствует переход маркера из одной вершины графа в другую его вершину, смежную с исходной.

Для отображения условий перехода и сохранения состояния каждая дуга и петля при традиционном подходе [16] помечаются дизъюнкцией тех наборов (всех входных переменных), при появлении которых осуществляется переход или сохраняется состояние. Это резко ограничивает размерность задач, описываемых с помощью рассматриваемых графов. Для устранения этого ограничения каждую дугу и петлю целесообразно помечать булевой формулой, содержащей обозначения только тех входных переменных, от которых переход зависит, и не содержащей обозначений всех остальных переменных, от которых рассматриваемый переход не зависит.

Граф может содержать два типа вершин — с петлями и без петель. Вершины первого типа называются устойчивыми, а второго типа — неустойчивыми.

Устойчивая вершина является активной до тех пор, пока выполняется условие, помечающее петлю. Это условие может зависеть от переменных, помечающих дуги, входящие в вершину, но в общем случае может зависеть от других переменных, определенные значения которых и поддерживают активность вершины.

Неустойчивая вершина не может поддерживаться входными переменными длительное время активной, а является таковой лишь один «такт» (программный цикл). Из неустойчивой вершины может исходить одна или несколько дуг в другие вершины. В первом случае дуга помечается единицей, свидетельствующей о том, что переход из этой вершины не зависит ни от одной входной переменной, т. е. выполняется безусловно. При этом отметим, что при программной реализации каждая вершина графа, в том числе и неустойчивая, на каждом пути не должна фильтроваться и должна быть доступна для наблюдения в конце программного цикла его «обработки».

Граф, обладающий описанными выше свойствами, будем называть графом переходов.

Каждому пути в ГП из начальной вершины в саму себя, проходящему через несколько вершин, соответствует последовательный процесс изменения активности вершин. Поэтому ГП можно считать компактной формой отображения некоторой совокупности процессов. При этом каждому несамопересекающемуся пути соответствует один процесс, а каждому самопересекающемуся пути — бесконечное множество. Последнее связано с возможностью прохождения неограниченного числа раз контуров, не содержащих начальную вершину. Из изложенного следует, что ГП, содержащий только контуры, проходящие через начальную вершину, реализует фиксированное число процессов.

Если при прохождении контуров графа переходов, как содержащих начальную вершину, так и не содержащих ее, возникает генерация, связанная с выполнением условий на всех дугах, образующих контур, то такой граф некорректен и должен быть откорректирован, исключив в нем возможность одновременного выполнения указанных условий.

Если каждой вершине присвоить десятичный номер, то изменение активности вершин сводится к наблюдению за изменением номеров вершин. При этом каждый процесс характеризуется номенклатурой, количеством, порядком взаимного расположения и длительностью появления номеров, а также видом и порядком взаимного расположения булевых формул, помечающих дуги и петли. Можно утверждать, что в этом случае автомат наблюдаем по внутренним состояниям.

Если каждую вершину ГП пометить кортежем, состоящим из требуемых значений всех выходных переменных, то процесс будет наблюдаем по выходам, а каждое значение каждой выходной переменной не будет зависеть от предыстории. При этом длительность формирования кортежей значений выходных переменных определяется временем пребывания в вершине, т. е. временем активности вершины, совпадающим со временем выполнения условия, помечающего петлю. В этом случае смежные вершины при просмотре в одном направлении могут быть связаны только одной дугой, помеченной одной булевой формулой, а каждая вершина может иметь только одну петлю, также помеченную одной такой формулой. Наряду с выходными переменными в кортеж могут входить также и значения переменных, управляющих функциональными элементами задержки и являющихся для автомата выходными, а для управляющего автомата внутренними.

При этом можно утверждать, что при таком подходе число состояний автомата не может быть меньше числа различных кортежей значений его выходов, так как каждый из них формируется по крайней мере в одном из состояний, а некоторые кортежи могут формироваться в нескольких состояниях. Из изложенного следует, что процесс по выходам наблюдается (понимается) существенно «хуже», чем по состояниям, так как в первом случае приходится следить за кортежами бит, которые могут быть достаточно длинными и одинаковыми, а во втором — только за одной многозначной переменной, кодирующей состояния (вершины), все значения которой различны. При этом если в статике в каждой вершине проверить соответствие между номером вершины и кортежем значений выходных переменных, который должен формироваться в рассматриваемой вершине, то в динамике можно осуществлять проверку, следя только за значением одной многозначной переменной, предыдущее значение которой сбрасывается автоматически.

Если каждую дугу и петлю ГП пометить наряду с булевой формулой (через дробь с ней) соответствующим кортежем значений выходных переменных (не помечая этими кортежами вершины), то и в этом случае процесс будет наблюдаем как по внутренним состояниям, так и по выходам. При этом, однако, соответствие между номером состояния и кортежем в отличие от предыдущего случая неоднозначно, так как значения выходных переменных могут зависеть и от значений входных переменных. Это приводит к тому, что в ГП смежные вершины могут быть соединены несколькими однонаправленными дугами с различной пометкой, а вершины могут быть связаны с несколькими петлями, также помеченными различно. При замене в ГП нескольких дуг одной дугой, а нескольких петель одной петлей исходные пометки сохраняются.

Длительность формирования значений выходных переменных на дугах ГП равна времени перехода (программного цикла), а время формирования этих значений на петлях — времени выполнения условия, помечающего соответствующую петлю. Так как в этом случае в отличие от предыдущего с одной вершиной могут быть связаны более одной петли, то время активности вершины может быть больше времени формирования значений выходных переменных, помечающих одну из петель.

Кроме рассмотренных вариантов построения ГП возможен также и случай смешанного расположения кортежей в них — в вершинах, на дугах и петлях.

Выше ГП был определен как однокомпонентный граф, однако в общем случае синтаксически корректной является также и система из нескольких компонент, каждая из которых является корректным ГП со своей начальной вершиной и своей многозначной переменной для кодирования номеров вершин. При этом состояние системы определяется кортежем значений всех введенных многозначных переменных. Система ГП позволяет описывать параллельно протекающие, вызываемые и вложенные процессы.

ГП с описанными выше свойствами ориентированы на программную реализацию алгоритмов логического управления. При применении ГП для решения других классов задач, например вычислимости функций на машинах Тьюринга [257], свойства ГП могут отличаться от описанных. Так, для этого класса задач «замкнутость» графа не является необходимой (вместо циклической достаточно однократной реализации), а вместо кортежей значений выходных переменных на дугах графа могут указываться значения одной многозначной выходной переменной (влево, вправо, записать, стереть), предыдущее значение которой в отличие от двоичных переменных принудительно не требуется сбрасывать. В этом случае состояния машины связаны со значениями выходной переменной неоднозначно, и поэтому при проверке алгоритма в динамике следить только за значением многозначной переменной состояния далеко недостаточно. При этом компактное описание алгоритма с помощью ГП даже со сравнительно небольшим числом вершин, дуг и петель может вызвать очень сложное поведение. Поэтому анализ такого графа является весьма трудоемким, а доказательство его правильности при произвольных значениях входных переменных из области определения может быть алгоритмически неразрешимой или трудноразрешимой задачей [303]. Построение же ГП для этих классов задач, даже в тех случаях, когда оно возможно, и вовсе становится искусством, предмет которого бывает приятно «созерцать», но непонятно, как его создать самому. Естественно, что такие проблемы могут возникать и при создании сложных алгоритмов логического управления, однако для многих задач этого класса однозначность связи состояний со значениями всех выходов и отсутствие умолчаний этих значений упрощает решение указанных проблем за счет определенного изоморфизма структуры ГП и его поведения.

Возвращаясь к рассмотрению свойств ГП, отметим, что для них кроме перечисленных возможны еще два вида некорректностей — неполнота и противоречивость.

Из изложенного выше следует, что каждой вершине ГП соответствует СБФ, состоящая из формул $f_1, f_2, \ldots, f_m$, являющихся пометками исходящих из вершины дуг и петель. Эта система зависит от «существенных» для этой вершины входных переменных, среди которых могут быть и переменные, сигнализирующие о срабатывании функциональных элементов задержки.

В силу того что для каждого набора входных переменных должно быть определено следующее состояние, то по крайней мере одна из булевых формул рассматриваемой системы должна быть равна единице. В математической форме это записывается следующим образом: $f_1 \vee f_2 \vee \ldots \vee f_m = 1$. Назовем это соотношение условием полноты переходов для рассматриваемой вершины.

Проектируемый автомат должен быть детерминированным, т. е. из рассматриваемого состояния он должен переходить только в одно следующее состояние. Это возможно, если все формулы рассматриваемой системы взаимно ортогональны: $f_1 \& f_2 = 0$; $f_1 \& f_3 = 0$; ... ; $f_{m-1} \& f_m = 0$. Назовем эти соотношения условием непротиворечивости для рассматриваемой вершины.

Автомат, для каждой вершины которого выполняются оба рассмотренных условия, назовем полным и непротиворечивым.

Приведем без доказательства две теоремы.

Теорема 1. Пусть некоторая вершина ГП имеет m исходящих дуг, $m-1$ из которых помечены булевыми формулами: $f_1, f_2, \ldots, f_{m-1}$, причем $f_1 \& f_2 = 0, f_1 \& f_{m-1} = 0, \ldots, f_{m-2} \& f_{m-1} = 0$, тогда $f_m = f_1 \vee f_2 \vee \ldots \vee f_{m-1} = \bar{f}_1 \& \bar{f}_2 \& \ldots \& \bar{f}_{m-1}$.

Теорема 2. Пусть для некоторой полной и непротиворечивой вершины ГП пометка первой исходящей дуги изменена введением формулы f_{m+1}, тогда для сохранения указанных условий пометки остальных исходящих из этой вершины дуг также должны быть изменены, причем

если $f_1 \vee f_{m+1}$, то $f_2 \& \bar{f}_{m+1}, \ldots, f_m \& \bar{f}_{m+1}$;
если $f_1 \vee \bar{f}_{m+1}$, то $f_2 \& f_{m+1}, \ldots, f_m \& f_{m+1}$;
если $f_1 \& f_{m+1}$, то $f_2 \& f_{m+1}, \ldots, f_j \& \bar{f}_{m+1}, f_m \& f_{m+1}$;
если $f_1 \& \bar{f}_{m+1}$, то $f_2 \& \bar{f}_{m+1}, \ldots, f_j \& f_{m+1}, f_m \& \bar{f}_{m+1}$.

Завершая описание свойств ГП, отметим еще раз, что коды состояний автомата всегда записываются в вершинах ГП, а значения выходных переменных в зависимости от структурной модели автомата записываются либо также в его вершинах, либо на дугах через дробь с соответствующей булевой формулой.

Перечислим основные преимущества, которые могут обеспечить графы переходов. Они позволяют в наглядной форме описывать поведение автоматов, отражая динамику переходов из состояния в состояние; обеспечивают возможность рассмотрения только рабочих состояний автоматов; появляется возможность по формальным признакам обеспечить полноту и непротиворечивость; ГП позволяют отмечать каждую их дугу булевой формулой тех переменных, которые обеспечивают рассматрива-

емый переход, и не учитывать все остальные переменные, используемые в них; позволяют применять в условиях переходов не минтермы, а булевы формулы, в том числе в скобочных формах произвольной глубины, что гарантирует компактность описания; обеспечивают возможность в каждой вершине ГП для определения следующей вершины и значений выходов рассматривать ортогональную систему, образованную лишь формулами, помечающими исходящие из этой вершины дуги; позволяют читать двоичные значения всех выходных переменных; обеспечивают возможность при применении многозначного кодирования значности s (s — число вершин в ГП) сведения числа внутренних переменных в каждом графе к одной; многозначность кодирования внутренних переменных резко упрощает взаимодействие графов переходов во взаимосвязанной системе, указывая значения этих переменных на дугах графов; графы переходов позволяют для их изоморфного отображения в тексты программ на языках высокого уровня использовать конструкцию, аналогичную конструкции switch языка СИ, обеспечивающую наблюдаемость, управляемость, удобство модификации и структурируемость; позволяют применять граф достижимых маркировок в качестве графа проверки (теста), так как программно реализованный автомат в отличие от аппаратного выполняет (при правильном переходе от ГП к программе) лишь то, что описывает ГП, и ничего лишнего (за исключением, быть может, вершин, которые не имеют входящих дуг, связанных с «ядром» графа).

Описание заданного алгоритма с помощью ГП существенно более компактно и поэтому более обозримо по сравнению с реализациями на основе ГСА и ФС. Это весьма важно для восприятия алгоритма человеком, для которого желательно целостное («гештальтное») осознание картины, явления или процесса.

Система взаимосвязанных ГП в отличие от одного ГП без флагов и умолчаний может породить нежелательные переходы, однако применение графа переходов проверки (ГППР), построение которого предлагается автором, позволяет по крайней мере теоретически устранить указанный недостаток.

ГППР, представленный, например, в форме таблицы «состояние, вход—следующее состояние, выход», должен включаться в методику проверки функционирования потому, что именно он позволяет проверить функционирование автомата с памятью, а не таблица «вход—выход», применение которой целесообразно лишь для комбинационных схем. Другая возможность проверки обеспечивается построением графа достижимых маркировок (разд. 5.4.6).

По мнению автора, ГП совместно со схемой связей УА с объектом управления и органами управления должны включаться и в другую техническую и программную документацию, так как полностью описать словами работу даже сравнительно простых управляющих автоматов практически невозможно, так как такое описание становится чрезвычайно громоздким и нечитаемым. Поэтому на практике обычно описывают не всю схему или программу, а лишь некоторое ее ядро, в то время как все функциональные возможности остаются неописанными. Например, в документации может встретиться такой фрагмент описания: «При нажатии первой кнопки клапан открывается, а при нажатии второй — закры-

вается». При этом ничего не сказано, что должно происходить в случаях, когда обе кнопки не нажаты или нажаты одновременно. В полном и непротиворечивом ГП функционирование в доступной форме отражено полностью и текст становится ненужным. После завершения архитектурного проектирования и выбора языка спецификаций можно продолжить процесс алгоритмизации, переходя, как и при аппаратной реализации, к логическому проектированию автоматов, в ходе которого осуществляется:

— выбор структурной модели каждого автомата системы (разд. 3.1 и 3.2);

— выбор вида кодирования состояний каждого автомата (разд. 3.3);

— построение корректного ГП без флагов и умолчаний для каждого автомата, однозначно соответствующего выбранной структурной модели и виду кодирования его состояний;

— построение графа достижимых маркировок, позволяющего выполнить полный анализ поведения системы ГП (разд. 5.4.6);

— выбор и построение алгоритмических моделей, реализующих построенные ГП (гл. 4), с учетом рода выбранной структурной модели каждого автомата.