«О многом,— молвил Морж, — пришла пора поговорить».

«Девочки любят картинки, — сказала Алиса».
Л. Кэрролл. «Алиса в Стране чудес».

Глава 1
Языки описания алгоритмов логического управления

В настоящей работе разрабатывается технология алгоритмизации и программирования задач логического управления, названная SWITCH-технологией.

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 желательно 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 [227] 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 завершенной программы».

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

В рамках разработанной технологии предлагается использовать два уровня языков — языки алгоритмизации (ЯА), или спецификации (языки общения), и языки программирования (языки реализации). Языки этих классов могут как совпадать (при наличии транслятора с ЯА), так и различаться между собой.

Так, например, при аппаратной реализации систем логического управления в базисе релейно-контактных схем в качестве языка алгоритмизации обычно применялись функциональные схемы (ФС), а в качестве языка реализации — собственно релейно-контактные схемы (РКС). Однако плохая «читаемость» как ФС, так и РКС 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 алгоритмов на базе аппаратуры «Selma-2» фирмы «ABB Stromberg» (Финляндия) в качестве как языка алгоритмизации, так и языка программирования используются функциональные схемы, а для программируемых логических контроллеров (ПЛК) «Autolog» фирмы «FF—Automation» (Финляндия) в качестве языка программирования применяется язык инструкций ALPro, в то время как язык алгоритмизации не определен. Последний не определен и для многих других типов ПЛК, таких, например, как контроллеры «Melsec» фирмы «Mitsubishi Electric» (Япония), языками программирования которых является язык инструкций и язык лестничных схем — язык релейно-контактных схем, 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 (ГСА), 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 (языки инструкций, ассемблеры) и специализированные языки (функциональные и лестничные схемы).

Ниже обосновывается целесообразность применения в качестве языка спецификаций для описания алгоритмов логического управления 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 модификации традиционных (классических) путей формализации как процедур управления, так и описаний объектов управления, что отличает этот класс задач, например, от ситуационного управления [281].

Для ответственных технологических объектов системы логического и ситуационного управления могут применяться совместно [282].

1.1. Классические языки логического управления

Булевы функции, таблицы истинности и таблицы решений. В системах логического управления традиционно используются булевы функции (БФУ) и системы БФУ (СБФУ), задаваемые в форме таблиц истинности (ТИ) для полностью определенных функций и таблиц решений (ТР) для неполностью определенных функций. При этом ТИ, описывающие автоматы с памятью, носят название кодированных таблиц переходов, или кодированных таблиц переходов и выходов.

Применение ТИ ограничивается задачами небольшой размерности, а применение ТР — в основном автоматами без памяти — комбинационными схемами. Табличное задание автоматов с памятью ненаглядно.

Булевы формулы и другие аналитические формы представления алгоритмов логического управления. Аналитической формой представления булевых функций являются булевы формулы (БФ) и системы БФ (СБФ), которые позволяют описывать как комбинационные схемы, так и автоматы с памятью большой размерности. СБФ могут быть изоморфно реализованы лестничными или функциональными схемами. Иногда используются также и другие аналитические формы представления булевых функций, например пороговые [22, 54], спектральные [35] или арифметические [78, 80—82].

Основным ограничением на применение СБФ для автоматов с памятью является их низкая наглядность.

Функциональные схемы. К достоинствам ФС 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 последовательностном 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 и не позволяет анализировать все имеющиеся в схеме переходы между состояниями. Более того, эти переходы и неизвестны 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 [178], «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.

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 [267].

Граф-схемы алгоритмов. К достоинствам ГСА 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 ГСА (называемых в этом случае автоматными) по сравнению с функциональными схемами.

К недостаткам ГСА относятся:

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

— отсутствие требований к тому, что должна отражать граф-схема, а именно: алгоритм управления (ГСА с внутренними обратными связями, но без внутренних переменных); алгоритм реализации алгоритма управления (ГСА без внутренних обратных связей); алгоритм, учитывающий свойства управляющих конструкций применяемого языка программирования (ГСА, линеаризованная и структурированная, возможно, специальным образом); алгоритм выполнения программы (ГСА, 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 (расцикливание) и учета свойств управляющих конструкций языка программирования (например, линеаризация и структурирование);

— наличие внутренних (промежуточных) переменных, отсутствующих в «словесном алгоритме» логического управления, резко затрудняющих возможность чтения ГСА 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 ГСА 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 ГСА;

— 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 ГСА 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 ГСА 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 графсхем.

Автоматные ГСА 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 граф-схем 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 ГСА (кроме первой разновидности первого 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 ГСА второго типа имеется также возможность сохранения предыдущих значений всех выходных переменных на «проводах» — без применения операторных вершин в соответствующем контуре граф-схемы.

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

Рассмотренный язык используется, например, фирмой «Опто» (США) для программирования ПЛК «Mistic» [234]. Применение ГСА в форме

диаграмм Несси—Шнейдермана [228] практически не устраняет указанных недостатков.

Логические схемы алгоритмов. ЛСА, предложенные А. А. Ляпуновым [24], являются строчной формой записи линеаризованных ГСА (ЛГСА). Они образованы буквами, которые соответствуют условным, безусловным и операторным вершинам ЛГСА, и пронумерованными стрелками, указывающими переходы, осуществляемые при невыполнении условий.

ЛСА обеспечивают компактность описания, но ненаглядны и весьма трудно строятся и читаются.

1.2. Нетрадиционные языки описания алгоритмов логического управления

Язык SDL. Идеи теории автоматов нашли свое отражение при разработке Международной комиссией по телефонии и телеграфии графического языка спецификации и описания алгоритмов — SDL-диаграмм (Specification and Description Language) [189, 359], которые по внешнему виду напоминают ГСА, 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 [180].

Р-схемы. Еще одна модель, базирующаяся на использовании автоматов Мили, была предложена И. В. Вельбицким [99, 203] и носит название Р-схемы (*R*-chart).

Р-схема — нагруженный по дугам ориентированный граф, изображаемый с помощью вертикальных и горизонтальных линий и состоящий из структур, каждая из которых имеет только один вход и один выход. Схемы этого класса содержат по два типа (один из которых специальный) вершин и дуг и один тип соединительных линий. Р-схемы образуются за счет трех типов соединений — последовательного, параллельного и вложенного.

Этот язык позволяет более компактно по сравнению с ГСА отражать структуру алгоритмов, однако применение нетрадиционных обозначений и только одного типа автоматных моделей ограничивает использование таких схем.

Сети Петри и графы операций. Для описания сложных, в том числе параллельных, процессов в 1962 г. К. Петри [145, 146] 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 [31] было предложено применять только безопасные и живые сети Петри (СП). В безопасной СП в позициях не может быть более одного маркера, а в живых СП — имеется возможность срабатывания любого перехода. Назовем указанный класс СП

управляющими сетями Петри, а сети Петри, в которых каждый переход имеет только одну входящую и одну исходящую дуги, — автоматными сетями Петри.

В качестве модели для описания алгоритмов управления С. А. Юдицкий [31] предложил использовать графы операций (ГО), 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 ГО.

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 ГО, построенных на базе автоматных сетей Петри, используется только модель автомата Мура [181] и невозможно применение других автоматных моделей, что резко ограничивает изобразительные возможности графов операций;

— для кодирования позиций могут использоваться только двоичные коды. При этом число внутренних переменных (без учета их переобозначений) равно числу позиций, в том числе и для графов операций, построенных на базе автоматных сетей Петри, в то время как для этого класса графов при применении подхода, предлагаемого в настоящей работе, все позиции могут быть закодированы лишь 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 реализованного алгоритма и анализ всех его функциональных возможностей.

Язык «Графсет». Этот графический язык, разработанный в Центре космических исследований в Тулузе (Франция), применяется в настоящее время наряду с другими языками [221] такими фирмами, как например «Телемеханик» (Франция), «Сименс» (Германия), «Ален Бредли» (США), «Тошиба» (Япония), «Омрон» (Япония), «Модикон» (США). Этот язык алгоритмизации при наличии транслятора с него является также 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 и ряда других недостатков, рассмотренных в гл. 12 и разд. 19.3, как отмечено выше, входит в состав программного обеспечения ПЛК, выпускаемых ведущими в области автоматизации фирмами мира. Так, например, фирма «Сименс» обеспечивает возможность написания программ для своих ПЛК с помощью языка STEP-5 (языки инструкций, лестничных и функциональных схем), а также языка S7-GRAF (язык «Графсет») [229, 305].

Однако опыт показывает, что при наличии для одного и того же ПЛК 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 и той же фирмы могут использоваться 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 ГСА (фирма «Опто»), и «Графсет» (фирмы «Омрон» и «Модикон»), 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.

Проблемно-ориентированные языки, близкие к естественным. Для целей логического управления разработано несколько проблемно-ориен-

тированных языков, предназначенных для формального лингвистического описания алгоритмов рассматриваемого класса. Эти языки называются [11] 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» [62], «Условие» [170], «Управление» [23], «Ярус» [61, 155].

Так, например, конструкции первичного языка «Условие» сначала транслируются в базовый язык операторных схем параллельных алгоритмов с памятью, являющийся развитием ГСА, 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 ГСА для построения автомата [16] двоичные метки, трактуемые как состояния, и по помеченному описанию строится граф переходов (ГП) автомата Мили, на базе которого, в частности, решаются задачи минимизации числа состояний (разд. 4.4.5) и параллельной декомпозиции.

Подход, наиболее близкий к предлагаемому в настоящей работе, был применен при разработке языка «Ярус». При этом О. П. Кузнецовым [120] для описания работы «пунктов» было предложено использовать автоматную модель, названную графом переключений, являющуюся ГП автомата Мили с умолчаниями неизменяющихся значений выходных переменных. Возможность сокращения числа вершин в этом графе по сравнению с классическими автоматными моделями привела к замене

термина «состояние» на термин «ситуация», правда, с одновременным ухудшением читаемости графа в виду появления зависимости от более глубокой предыстории.

В отличие от изложенного в предлагаемой 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 (УА)—объект управления (ОУ)», которая определяет семантику каждой внешней переменной, используемой в ГП. Применение ГП без флагов и умолчаний [275] 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.

«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 ГП 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 ГП автомата Мили может быть построен ГП автомата без выходного преобразователя, являющегося графом достижимых маркировок исходного графа, который в свою очередь весьма просто преобразуется в ГП автомата Мура.

При этом можно считать, что каждая вершина в ГП 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 ГП, а соответствующим ему графом достижимых маркировок, то «реальное» число состояний автомата совпадает с их числом в этом графе или в эквивалентном ему ГП «классического» (без флагов и умолчаний) автомата Мура.

Таким образом, если ГП в общем случае можно рассматривать как' «закодированное» (число вершин в ГП может быть существенно меньше, чем число «реальных» состояний автомата) и поэтому компактное описание поведения автомата, то наиболее «понятным» является ГП «классического» автомата Мура при многозначном кодировании его вершин, который одновременно является и графом его достижимых маркировок. Именно ГП 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 (формальном), а самое главное, изоморфном переходе от ГП, согласованного с Заказчиком, к программе. При этом программирование выполняется не по «мотивам» алгоритма, а по принципу «один в один».

При применении предлагаемой технологии удается обеспечить соответствие между текстом программы и порядком ее выполнения и реализовать процедуру пошаговой детализации, как этого требует структурное проектирование (программирование) [104], а также использовать понятия «объект» и «класс», как это принято в объектно-ориентированном проектировании (программировании) [226].

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

1.3. Графы переходов как язык спецификаций

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 подает 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».

Несмотря на то что в настоящее время при создании алгоритмов управления, например в форме граф-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».

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.

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 так же. 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. 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 ГП, а программу, реализующую граф переходов автомата, построить формально и изоморфно, то такая программа будет «понятной» не только Разработчику и Программисту, но и Заказчику, Технологу, Оператору и Контролеру.

При этом необходимо отметить, что, несмотря на сложность построения модели объекта [358], 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 «УА—ОУ». 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 [267].

Факторы, ограничивающие широкое использование графов переходов в качестве языка алгоритмизации. Для устранения недостатков рассмотренных языков алгоритмизации предлагается использовать в ка-

честве такого языка графы переходов, предложенные более сорока лет назад для описания поведения автоматов с памятью [181]. Графы переходов называют также диаграммами состояний [305], 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 ГП — 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 ГП, 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 ГП 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 граф-схем при программной реализации алгоритмов, видимо, явились причинами того, что ГП до сих пор практически не применяются в качестве языка спецификаций условий работы при программной реализации.

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 программировании процессов управления (за исключением подхода, используемого в объектно-ориентированном анализе [250] и в программных продуктах «S7-HiGraph technology software» [305] и «Modicon State Language» [299]).

При применении других подходов внутреннее состояние Вычислителя либо не учитывается, либо не рассматривается как единое целое. При этом, как в событийно-управляемом программировании [280], 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. Таким образом, наряду с «комбинационной схемой» появляется и внутренняя память, а «схема» переходит в класс автоматов с памятью.

Отметим также, что когда при использовании традиционного подхода говорят о том, что программа, реализованная одной компонентой, установлена в начальное состояние, то на самом деле при этом обычно имеют ввиду инициализацию эвристически введенных в нее внутренних переменных, а не установку начального значения переменной, описывающей «состояния» программы в целом.

Применение ГП 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 [43], теории линейных систем [258], марковских процессах [295] и в отдельных задачах практического [264] и теоретического [265] программирования. ГП 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 (АБВП) или автоматов Мура (АМ)) или во время каждого перехода (для автоматов Мили (АМИ)).

Если в одном автомате используются оба способа формирования значений выходных переменных, то он называется «смешанным» (С-автомат (СА)). Если в нем одни и те же переменные применяются как в качестве входных, так и выходных переменных, то такой автомат называется «автоматом с флагами».

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 ГП 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 ГП соответствуют вершины, а дугам между вершинами — переходы между состояниями (номенклатура составляющих ГП 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 (А) 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, помечающих 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 АБВП и АМ), на дугах (для АМИ) и в вершинах и дугах одновременно (для СА).

Двоичные переменные *t* управляют функциональными элементами задержки (ФЭЗ), а двоичные переменные *T* сигнализируют о срабатывании или несрабатывании этих элементов. При этом будем полагать, что ФЭЗ в состав автоматов не входят и являются для них одним из объектов управления. Комплекс «А—ФЭЗ» образует единую компоненту, называемую «управляющим автоматом».

При чтении ГП считается, что в каждый момент времени (за один программный цикл) выполняется не более одного перехода: ноль — если состояние сохраняется и один — если оно не сохраняется, что поддерживается соответствующей программной реализацией.

Построение графов переходов. Пусть требуется построить ГП, описывающий поведение автомата с *n* двоичными входами и *m* двоичными выходами. Для задач большой размерности построение ГП 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 ГП автоматов Мили, наиболее часто применяемых в литературе для описания примеров автоматов небольшой размерности, то это понятие (абстракцию) приходится вводить сразу для различения ситуаций, связанных с изменени-

ям и значений выходных переменных при одних и тех же значениях входных переменных. Построение ГП в «изменениях» порождает последующие трудности по их чтению (пониманию) и корректировке, так как при этом значения выходных переменных зависят не только от состояния, но и от значений входных переменных, существенных для рассматриваемого состояния.

Если же «состояние» определять как комбинацию значений всех m выходных переменных, причем одинаковые комбинации этих переменных рассматривать как различные состояния, то такое определение существенно менее абстрактно и более естественно, так как понятие «состояние» как бы в явном виде не используется. 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 ГП.

Каждая комбинация значений всех выходных переменных соответствует одной вершине ГП и помечает ее. Каждая вершина ГП 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 ГП для задач весьма большой размерности.

При такой методике построения (разд. 4.4.9) заданные условия работы реализуются графом переходов автомата без выходного преобразователя с принудительным кодированием состояний, который в дальнейшем в случае необходимости, например при наличии вершин, помеченных одинаково, и отсутствии различающих входных наборов, преобразуется в ГП автомата другого типа с явным введением понятия «состояние» (ГП автомата Мура) или другого типа кодирования (ГП автомата без выходного преобразователя с принудительно-свободным кодированием состояний),не изменяя структуру первоначально построенного графа переходов. В отдельных случаях учет ограничений требует использования ГП смешанного автомата, что приводит к необходимости введения в ГП автомата без выходного преобразователя или в построенный по нему ГП автомата Мура значений выходных переменных (через дробь с булевой формулой, помечающей соответствующую дугу), формируемых на переходе.

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

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

Свойства графов переходов. Одно из достоинств ГП состоит в том, что они могут быть формально проверены на синтаксическую корректность (семантическая (смысловая) корректность, естественно, формально проверена быть не может). При этом ГП считается корректным, если он:

— непротиворечив;
— полон;
— не содержит генерирующих контуров, отличных от петель.

При этом считается, что непротиворечивость в ГП 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 ГП, в каждом из которых допустимо одновременное существование нескольких «активных» вершин, называются графами переходов с параллелизмом (ГПП). ГПП отличаются от диаграмм «Графсет» только отсутствием возможности осуществления внутри одной компоненты синхронизации завершения параллельных процессов. При использовании системы взаимосвязанных ГП и описании каждого алгоритма отдельным ГП (компонентой) синхронизация процессов при необходимости осуществляется **в** головном графе переходов. Поведение ГПП (определяемое графом достижимых маркировок [14], являющимся графом переходов между всеми возможными состояниями компоненты или системы компонент) отличается от его описания: ГПП делает «больше», чем описывает его структура.

В этом смысле ГП (без флагов и умолчаний изменяющихся в зависимости от предыстории значений выходных переменных в одной вершине или на одной дуге графа) аналогичны параллельно-последовательным контактным схемам, для каждой из которых булева формула, описывающая ее структуру, одновременно задает и ее функционирование (поведение), а ГПП аналогичны мостиковым контактным схемам, для которых булева формула, описывающая поведение каждой из них, не задает ее структуру.

При этом если в ГП этого типа понятия «вершина» и «состояние» являются синонимами, то для ГПП эти понятия не эквивалентны.

Эта особенность ГПП, позволяющая компактно и обозримо описывать и реализовывать некоторые классы параллельных процессов одной компонентой (так как в противном случае пришлось бы строить и реализовывать ГП с большим числом состояний), связана с потерей важнейшего свойства ГП без флагов и умолчаний, состоящего в том, что он может

быть реализован так, что в графе, описывающем поведение «реализации», число вершин по сравнению с исходным ГП не изменяется.

При формальном (и правильном) переходе от ГП 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 ГП (в случае, когда число вершин в нем не равно величине два в степени) с помощью СБФ и обратном построении ГП по этой системе, в новом графе появляются вершины, отсутствующие в исходном задании. Однако так как переходы из вершин заданного ГП в новые вершины отсутствуют, то, несмотря на наличие переходов из новых вершин в заданные, такая реализация является корректной.

При реализации ГП, например с помощью конструкции s w i t c h языка СИ, указанное свойство ввиду их изоморфизма может быть сохранено полностью.

При неформальном переходе от ГП к программе поведение последней может отличаться от ГП, в том числе и таким образом, что, используя его в качестве теста для проверки программы, их несоответствие обнаружить не удается. Это объясняется тем, что в ГП 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 ГП происходит при пометке x_4, а в программе этому переходу соответствует пометка $x_4\&\bar{x}_5$,[1] то такую ошибку без полного перебора или построения ГП 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 построен (и откорректирован в случае необходимости) граф достижимых маркировок, приводят к тому, что он 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;

— ортогонализацией (усложнением пометок) противоречивых дуг (например, при реализации по СБФ);

— расстановкой приоритетов (учитывается порядок расположения команд в программе при реализации способом, отличным от построения СБФ);

[1] В настоящей работе для обозначения логической операции «инверсия» используется либо символ «-», либо символ «!».

— «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 ГП с помощью СБФ 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 ГП 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).

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

Если в ГП АБВП 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 ГП АБВП 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 ГП 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 ГП 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 ГП 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 ГП в целом присваивается одна многозначная переменная *YI, j*-е значение которой в свою очередь присваивается *j*-й вершине ГП, 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 ГП 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» (разд. 5.4.6). 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 ГП к программе она в «медленной тактности» (после завершения процесса однократного вычисления по ней) будет функционировать в соответствии с ГП, 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 ГП с кодом «00» в состояние с кодом «11» имеет место переключательный процесс «00—10—11», в котором промежуточное значение «10» фильтруется. Переход через состояние с кодом «01» при программной реализации с помощью СБФ в отличие от асинхронных схем невозможен, так как порядок изменения значений переменных однозначно определяется порядком расположения формул в системе.

Особенности использования графов переходов. При реализации алгоритма одним ГП автомата Мура или автомата без выходного преобразователя и формальном переходе к тексту программы по этому 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 ГП не только формально, но и изоморфно, то тестирование может быть заменено сверкой ее текста с ГП.

Если ГП 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 ГП в системе взаимосвязанных ГП (СВГП) может осуществляться по входным, выходным, а самое главное, внутрен-

ним 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. СВГП могут быть построены и по принципу вложенности.

В вершинах (как в диаграммах «Графсет») и на дугах ГП могут не только устанавливаться и сбрасываться двоичные переменные, но могут и запускаться процессы, описанные, например, как с помощью ГП, так и с помощью ГСА. При описании процесса, происходящего в некоторой вершине ГП, с помощью ГСА он многократно выполняется, до тех пор пока автомат находится в этой вершине, и завершается (после окончания очередного прохода ГСА) при переходе ГП в новую вершину. В качестве примера такого процесса можно привести реализацию в вершинах графа переходов ФЭЗ 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 ГП системы взаимосвязаны) или несколько (для случая существования в системе не связанных между собой групп ГП) графов достижимых маркировок.

Основные этапы алгоритмизации при использовании ГП. Разрабатывается схема связей «источники информации (ИИ)—управляющий автомат—средства представления информации (СПИ)—исполнительные механизмы (ИМ)».

Управляющий автомат декомпозируется на автомат и ФЭЗ, что позволяет исключить время из модели, оставив только битовые переменные *t,* предназначенные для запуска ФЭЗ (выходы автомата), и битовые переменные *T,* сигнализирующие о срабатывании этих элементов (входы автомата). При этом предыдущая схема преобразуется в схему связей «ИИ—А—ФЭЗ—СПИ—ИМ».

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

Построение схемы связей «ИИ—СВА—ФЭЗ—СПИ—ИМ» 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 СВГП, которая

является формальной спецификацией — алгоритмом управления. На этой стадии строятся также формальные спецификации для ФЭЗ и моделей объектов управления. Построение спецификаций завершает вторую стадию проектирования управляющей программы.

На третьей стадии проектирования 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 [3].

Предлагаемая технология включает методы формализованного перехода от ГП к различным типам алгоритмических моделей, основные из которых следующие: системы булевых формул; функциональные схемы; лестничные схемы; ГСА без внутренних обратных связей.

При этом естественно, что и ГП 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.

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

В первом случае доказательство эквивалентности программы с ГП, по которому она строилась, может производиться их сопоставлением. При этом ГП 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 ГП, который и следует читать. Проверка программы в этом случае может производиться, используя ГП в качестве теста, по «схеме»: настоящее состояние, вход—следующее состояние, выход. Верификация, состоящая в построении ГП, в этом случае существенно более трудоемка и связана с вычислениями. Изоморфизм между текстом программы и ГП, естественно, обеспечивается за счет избыточности, использование которой невозможно при жестких ограничениях на объем памяти.

Главная особенность предлагаемых в настоящей работе программных реализаций состоит в том, что за один программный цикл в программе выполняется не более одного перехода в ГП, так как в противном случае, если, например, для некоторой вершины ГП автомата Мура условия, помечающие одну из входящих и одну из

исходящих (кроме петли) дуг, выполняются, 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 ГП к изоморфному по структуре тексту программы. Оператор `switch`, осуществляющий многовариантный (многозначный) выбор, обеспечивает в данном случае декомпозицию автомата по его внутренним состояниям.

Программная и методическая поддержка технологии. Возможность быстрого, безошибочного и изоморфного перехода от ГП к тексту программы на языке высокого уровня, например СИ, позволяет резко упростить отладку и моделирование управляющего автомата, а при необходимости и комплекса «УА—ОУ», описав не только автомат, но и модель объекта управления (по компонентам и(или) режимам) с помощью ГП.

Под руководством автора разработана программная оболочка, позволяющая на персональной электронной вычислительной машине (ПЭВМ) для СВГП, 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 ГП) в пошаговом и автоматическом режимах. Возможность постоянного чтения на дисплее десятичного номера состояния каждого ГП 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 неясными ответы на следующие вопросы:

— какие переменные следует применять в программе для обеспечения ее работоспособности и управляемости?

— сколько таких переменных (особенно внутренних) следует применять?

— что характеризует каждая переменная?

— какой набор переменных следует выводить на дисплей на каждом этапе отладки?

— какие переменные следует дополнительно ввести в программу и представить на дисплее, если на некотором этапе отладки ее работоспособность еще не обеспечена?

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

Если в качестве языка программирования применяется алгоритмический язык высокого уровня, например СИ, то после построения алгоритма

но 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 ПЛК «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 ГП на некотором подмножестве языка СИ, автоматически получать программу на языке инструкций ALPro с выбранными до трансляции управляющими конструкциями — условным переходом или шаговым регистром.

При ограничениях на внутренние ресурсы ПЛК автором разработана методика «ручной» формальной реализации ГП в базисе языка ALPro (гл. 14). Разработаны также методики реализации ГП в базисе таких языков программирования, как функциональные (гл. 17) и лестничные (гл. 18) схемы. Одна из этих методик позволяет, в частности, строить функциональные схемы, изоморфные ГП, в базисе библиотечных элементов для системы «Selma-2» [60] (разд. 4.2.3).

Понятие «состояние» и теория управления. В шестидесятые годы был развит качественно новый подход к теории линейных систем, который стимулировался интересом к теории конечных автоматов, что в свою очередь привело к появлению новых идей и методов, концентрирующихся вокруг понятия «состояние». При этом метод, получивший название «метод пространства состояний» [258], начал играть центральную роль в теории управления при изучении линейных, нелинейных и дискретных (квантованных) систем [350—357].

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

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

Отметим, что применение уравнений «вход—выход» для описания поведения систем с памятью является недостаточным, так как в этом случае приходится пользоваться отношением «вход—выход», а не соответствующим оператором. (Отношение есть множество упорядоченных пар «вход—выход», в то время как оператор есть отношение, в котором каждому значению входа соответствует единственное значение выхода).

Необходимость использования операторов, а не отношений приводит к применению уравнений «вход—состояние—выход» и уравнений «вход—состояние—следующее состояние», которые могут быть записаны в том числе и в дифференциальной форме.

Одним из важнейших понятий, используемых в методе пространства состояний, является «измеримость». Под измеримым понимается [258] такое состояние, которое экспериментатор может определить для каждого t либо непосредственно, либо через известные значения входного и выходного сигналов, не зная начального состояния.

Другими важными понятиями, введенными Р. Калманом [259], являются «управляемость» и «наблюдаемость».

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

Понятие «наблюдаемость» идентично понятию «определимость начального состояния».

Из изложенного следует, что понятие «наблюдаемость», вводимое в настоящей работе, эквивалентно понятию «измеримость» (по Заде).

Понятие «управляемость» (по Калману) эквивалентно понятию «сильная связность» в теории автоматов, введенному Муром [181]. Автомат называется сильно связанным, если для любой упорядоченной пары его состояний (Y_i, Y_j) существует последовательность входных сигналов, которая переводит автомат из состояния Y_i- в состояние Y_j. Этот термин используется, ввиду того что ГП любого такого автомата является связным графом. Обратное неверно, так как существуют связные ГП, для любой пары состояний которых (Y_i, Y_j) невозможен переход из состояния Y_i в состояние Y_j.

Понятие «управляемость», применяемое в настоящей работе, является более широким и включает в себя наряду с изложенным и возможность корректного внесения изменений.

Выводы. Предлагаемый подход позволяет:

— использовать теорию автоматов при алгоритмизации и программировании процессов управления;

— первоначально описывать желаемое поведение управляющего «устройства», а не его структуру, которая является вторичной и поэтому труднее читаемой и понимаемой;

— ввести в алгоритмизацию и программирование в качестве основного понятие «состояние»;

— ввести понятия «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 (как это имеет место, например, в марковских процессах), 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;

— 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 ГП или системы взаимосвязанных ГП, в которых учтены все детали с точностью до каждого состояния, перехода и бита;

— применять ГП без флагов и умолчаний в качестве сертификационного теста и строить граф проверки или граф достижимых маркировок для других классов графов переходов или систем взаимосвязанных графов переходов с целью проверки их поведения;

— использовать методы формального и изоморфного перехода от спецификации к программам логического управления на различных языках программирования;

— при применении алгоритмических языков программирования высокого уровня проводить программирование с помощью конструкций switch (в том числе вложенных) или им аналогичных, что кроме изоморфизма со спецификацией обеспечивает доступность каждого значения переменной, кодирующей состояния каждого графа переходов, для всех остальных графов, входящих в систему, и поэтому не требует введения дополнительных внутренних переменных для реализации взаимодействия графов в системе;

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

— Участникам разработки (Заказчику, Технологу (Проектанту), 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 их взаимопонимания [340];

— 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 нетрадиционным 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 ГП, 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 [267];

— 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 ГП 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.

Апробация. Подход, в частности, использован:

— при создании совместно с фирмой «Norcontrol» (Норвегия) системы управления дизель-генератором ДГР-2А 500*500 судна проекта 15760 [267]. Программирование выполнено фирмой на языке ПЛ/М;

— при создании системы управления дизель-генератором того же типа судна проекта 15640 на базе аппаратуры «Selma-2». Программирование выполнено НПО «Аврора» на языке функциональных блоков [269];

— при создании комплексной системы управления техническими средствами для судна проекта 17310 на базе ПЛК «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 писалась эта книга, в которой излагается предложенная и внедренная автором в 1991 году SWITCH-технология [167], [269], которая может быть также названа STATE-технология, или более точно AUTOMATON-технология, складывалась весьма странная ситуация, состоящая в том, что ни одна из ведущих в области автоматизации фирм мира (за исключением [299], 1993 г.) не предлагала технологий алгоритмизации и (или) программирования на базе графов переходов, являющихся основой рассматриваемой в книге методологии.

Только в 1996 году эта ситуация 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 diagrams»), что является другим названием графов переходов. При этом по описанию на этом языке автоматически генерируется исполняемый (только на ПЛК этой фирмы) код.

По мнению фирмы «Сименс», описание на таком языке не только подходит для Программиста ПЛК, но также понятно Инженеру-механику, Инженеру по запуску оборудования и Инженеру по обслуживанию [305].

Особенно странным в этой ситуации является то, что подобный продукт мог быть разработан, например, и пятнадцать лет назад, так как диаграммы состояний в свою очередь были предложены более сорока лет назад. Однако всемирное увлечение сетями Петри, которые обеспечивают возможность описания параллельно-последовательных процессов одной компонентой (функциональной единицей [305]) и являются поэтому основой языка «Графсет» (S7-Graph technology software [305]), 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 настоящая книга, в которой графы 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 лестничных и функциональных схем, 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, в том числе и в случаях отсутствия трансляторов с этого языка спецификаций.

Предлагаемая технология может применяться на этапе алгоритмизации даже при использовании в качестве языка программирования «state diagrams», так как в руководствах по их применению [299, 305] ничего не говорится о том, что необходимо сделать для обеспечения «хорошей понимаемости» ГП (например, построить схему связей «УА—ОУ» и

исключить зависимость от «глубокой» предыстории). Это особенно важно, так как при использовании этого языка программирования для полного описания даже одного ГП приходится применять несколько экранов, на одном из которых изображается образ ГП, а на других — условия переходов и выполняемые действия, что не позволяет «охватить описание одним взглядом».