Книжная полка

Автоматное программирование

Поликарпова Н. И.,
Шалыто А. А.
Издательство Питер
ISBN 978-5-388-00692-9
2009 год
167 стр.

В книге рассматривается автоматное программирование — подход к разработке программных систем со сложным поведением, основанный на модели автоматизированного объекта управления (расширении конечного автомата). Предлагаемый подход позволяет создавать качественное программное обеспечение для ответственных систем, охватывая все этапы его жизненного цикла и поддерживая его спецификацию, проектирование, реализацию, тестирование, верификацию и документирование.

Книга предназначена для специалистов в области программирования, информатики, вычислительной техники и систем управления, а также аспирантов и студентов, обучающихся по специальностям «Прикладная математика и информатика», «Управление и информатика в технических системах» и «Вычислительные машины, системы, комплексы и сети».

Ниже вы можете ознакомиться с отрывкам из этой книги, предоставленными ее авторами.

1.1. Парадигма автоматного программирования

★★☆ **Ключевые слова: конечные автоматы, автоматное программирование.**

Для того чтобы лучше разобраться в основных концепциях автоматного программирования, рассмотрим сначала один из абстрактных вычислителей, широко применяемых в теории формальных языков — *машину Тьюринга* [12, 13]. Эта абстрактная машина была предложена А. Тьюрингом в 1936 г. в качестве формального определения понятия «алгоритм». Тезис Черча-Тьюринга [13] гласит, что все, что можно «вычислить», «запрограммировать» или «распознать» в любом смысле (из формально определенных в настоящее время), можно вычислить, запрограммировать или распознать с помощью подходящей машины Тьюринга.

Машина Тьюринга состоит из двух частей: устройства управления и запоминающего устройства — ленты (рис. 1.5). Лента содержит бесконечное число ячеек, в которых могут быть записаны символы некоторого конечного алфавита. В каждый момент времени на одной из ячеек ленты установлена головка чтения-записи, позволяющая устройству управления считывать или записывать символ в этой ячейке.

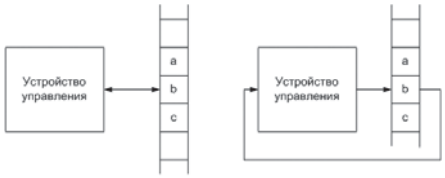

Рис. 1.5. Машина Тьюринга: традиционное изображение (слева) и изображение в традициях теории управления (справа)

Устройство управления представляет собой конечный автомат. У него имеется единственное входное воздействие: символ, считанный с ленты — и два выходных воздействия: символ, записываемый на ленту, и указание головке сдвинуться на одну ячейку в ту или иную сторону, либо остаться на месте.

Тезис Черча-Тьюринга означает, что в терминах операций машины Тьюринга можно записать все те же программы, что и на любом существующем языке программирования.

Как же программировать на машине Тьюринга? Пусть, например, необходимо реализовать функцию *инкремент* (увеличение целого числа на единицу). Пусть исходное число записано на ленте в двоичном виде слева направо, во всех остальных ячейках находится пустой символ ('blank') и головка указывает на самый старший разряд числа. Тогда для увеличения числа на единицу можно предложить следующий алгоритм:

1. Двигаться вправо, пока не встретится пустой символ.
2. Сдвинуться на одну ячейку влево.
3. Пока в текущей ячейке находится символ '1', изменять его на '0' и двигаться влево.
4. Если в текущей ячейке находится '0' или 'blank', записать в ячейку '1' и завершить работу.

Этот алгоритм необходимо «ввести» («закодировать») в устройство управления машины Тьюринга. Другими словами, необходимо задать состояния, а также функции переходов и выходов ее управляющего автомата. Удобный и наглядный способ сделать это предоставляют *графы переходов* автоматов, иначе называемые диаграммами переходов. Подробнее язык графов переходов будет обсуждаться в разд. 2.2, а пока достаточно знать, что вершины в этом графе соответствуют состояниям автомата, а дуги — переходам между состояниями. Каждая дуга помечается *условием перехода* (значениями входных воздействий, которые инициируют этот переход) и *действием на переходе* (значениями выходных воздействий).

На рис. 1.6 представлен граф переходов управляющего автомата машины Тьюринга, реализующей функцию инкремент. Здесь символ ‘b’ — сокращение от blank, символ ‘*’ на месте записываемого символа означает «Записать тот же самый символ, который был считан». Команды головке обозначаются стрелками (стрелка вниз означает «Остаться на месте»). В метке перехода над чертой записывается его условие, а под чертой — действие.

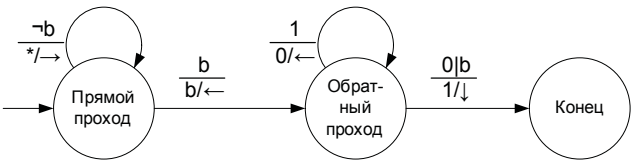

Рис. 1.6. Увеличение числа на единицу с помощью машины Тьюринга

Отметим, что в графе переходов обозначены имена состояний управляющего автомата. Эти имена отражают смысл состояния и являются кратким описанием поведения машины в этом состоянии.

Итак, управляющий автомат машины Тьюринга, реализующей функцию инкремент, имеет три состояния. Сколько же состояний у этой машины в целом? Ее действия в каждый момент времени полностью определяются совокупностью состояния управляющего автомата, строки на ленте и положения головки. Отметим, что символы на ленте для устройства управления представляют собой входные воздействия, однако, относительно машины в целом они не являются входными (внешними), а формируют часть внутреннего состояния вычислителя. Всевозможных строк на ленте, как и положений головки, бесконечно много, поэтому и у машины Тьюринга бесконечное число состояний.

Однако, если задуматься, состояния управляющего устройства и состояния ленты имеют принципиально различные значения. В приведенном примере оказалось, что для того чтобы задать алгоритм для машины Тьюринга, достаточно описать ее поведение в каждом из трех состояний управляющего автомата (рис. 1.6). Нам не потребовалось задавать реакцию машины для каждой из бесконечного числа возможных входных строк. Неформально можно сказать, что состояния управляющего автомата определяют *действия* машины, а состояние ленты — лишь *результат* этих действий.

Примечание:

Здесь уместна аналогия из объектно-ориентированного программирования. Вызывая компонент (метод) некоторого класса, клиент указывает имя компонента и, если требуется, его фактические аргументы. Различных компонентов в классе обычно всего несколько, в то время как различных аргументов может быть необозримо много. При этом имя компонента определяет *действие* (алгоритм вычислений), а значения аргументов — только *результат действия* (результат вычислений).

Теперь очевидно, что состояния устройства управления и состояния ленты — совершенно разные понятия с точки зрения программирования на машине Тьюринга, и смешивать их не стоит. Первые следует явно перечислять, отображать на графе переходов, описывать алгоритм поведения в каждом из них. Вторые в программе в явном виде не участвуют, построить граф переходов между ними невозможно, а если бы это и удалось, то для понимания программы такой граф был бы бесполезен. Первые можно назвать качественными состояниями машины, а вторые — количественными. В автоматном программировании для этих двух классов состояний приняты термины, заимствованные из теории управления. Состояния автомата называются *управляющими*, а состояния ленты — *вычислительными* .

Примечание:

В автоматном программировании (и, в частности, в этой книге), говоря без уточнения о *состоянии* некоторой автоматной модели, сущности или системы со сложным поведением, подразумевают управляющее состояние. Если речь идет о вычислительном состоянии, это оговаривается особо.

Примечание:

Понятие «состояние», так же как и деление на управляющие и вычислительные состояния, не является «чужеродным» для программирования. Традиционно под состоянием программы подразумевают множество текущих значений всех используемых в ней переменных [14, 15]. И хотя число переменных, равно как и число потенциальных значений каждой переменной, можно считать конечным, получающееся пространство состояний оказывается необозримо большим.

Однако не все исследователи трактуют понятие состояния программы столь прямолинейно. Так, *А. Дж. Перлис* в 1966 г. [16] предложил в описания языка, среды и правил вычислений включать состояния, которые могут подвергаться мониторингу во время исполнения, позволяя диагностировать программы, не нарушая их целостности. В этом же году *Э. Дейкстра* [17] предложил ввести так называемые переменные состояния, с помощью которых можно описывать состояния системы в любой момент времени (Дейкстра использовал для этих целей целочисленные переменные). При этом им были поставлены вопросы о том, какие состояния должны вводиться, как много значений должны иметь переменные состояния и что эти значения должны означать. Он предложил сначала определять набор подходящих состояний, а лишь затем строить программу. По мнению Э. Дейкстры, диаграммы переходов между состояниями могут оказаться мощным средством для проверки программ. Это обеспечивает поддержку его идеи о том, что программы должны быть с самого начала составлены правильно, а не отлаживаться до тех пор, пока они не станут правильными.

Понятия управляющих и вычислительных состояний применимы не только к машине Тьюринга, но и к любой сущности со сложным поведением. Однако, если в машине Тьюринга отличить управляющие состояния от вычислительных не составляет труда (поскольку она по определению состоит из устройства управления и ленты), то для произвольной сущности *явное выделение управляющих состояний* — сложная задача. Об этом уже упоминалось в разд. 1.1. Причина сложности состоит в том, что различия между управляющими и вычислительными состояниями в общем случае трудно формализовать. Неформально основные различия сформулированы в табл. 1.1.

Таблица 1.1. Управляющие и вычислительные состояния

Управляющие состояния	Вычислительные состояния
Их число не очень велико	Их число либо бесконечно, либо конечно, но очень велико
Каждое из них имеет вполне определенный смысл и качественно отличается от других	Большинство из них не имеет смысла и отличается от остальных лишь количественно
Они определяют действия, которые совершает сущность	Они непосредственно определяют лишь результаты действий

Представьте себе, что в машине Тьюринга не было бы управляющего автомата. Алгоритм ее работы может быть закодирован на ленте в виде последовательности команд. Машина точно также продолжала бы вести себя по-разному на разных шагах алгоритма, однако, из ее описания это было бы уже не ясно. Логика ее поведения была бы потеряна среди не столь существенных деталей, а управляющие состояния смешались бы с вычислительными. Программировать на такой машине и разбираться в уже существующих программах стало бы практически невозможно.

Переход от ленты, головки и простейших команд к языкам высокого уровня, конечно, упрощает программирование, но при реализации сущностей со сложным поведением полностью проблемы не решает. Вспомните электронные часы из разд. 1.1. Точно так же, как в нашей «воображаемой» машине, состоящей только из ленты, в листинге 1.1 логика затеряна среди деталей. Опыт рассмотрения машины Тьюринга подсказывает, что для того, чтобы сделать программу простой и понятной, необходимо *явно выделить управляющие состояния* (идентифицировать их, дать им имена) и *описать поведение сущности в каждом из них*.

Например, при реализации электронных часов с будильником можно выделить три управляющих состояния: «Будильник выключен», «Установка времени будильника» и

«Будильник включен». В каждом из этих состояний реакция будильника на нажатие любой кнопки будет однозначной и специфической.

Как отмечено выше, в случае с машиной Тьюринга выделение управляющих состояний тривиально, так как логика в ней априори вынесена в отдельное устройство — управляющий автомат. Подобная идея используется при построении систем автоматизации, в которых всегда выделяют управляющие устройства и управляемые объекты. Следуя этой концепции, сущность со сложным поведением естественно разделить на две части:

- управляющую часть, ответственную за логику поведения — выбор выполняемых действий, зависящий от текущего состояния и входного воздействия, а также за переход в новое состояние;
- управляемую часть, ответственную за выполнение действий, выбранных для выполнения управляющей частью, и, возможно, за формирование некоторых компонентов входных воздействий для управляющей части — *обратных связей*.

В соответствии с традицией теории управления, управляемая часть здесь и далее называется *объектом управления*, а управляющая часть — *системой управления*. Поскольку для реализации управляющей части используются автоматы, то она часто называется *управляющим автоматом* или просто *автоматом*.

После разделения сущности со сложным поведением на объект управления и автомат реализовать ее уже несложно, а главное, ее реализация становится понятной и удобной для модификации. Вся логика поведения сущности сосредоточена в управляющем автомате. Объект управления, в свою очередь, обладает *простым поведением* (а следовательно, может быть легко реализован традиционными «неавтоматными» методами). Он не обрабатывает непосредственно входные воздействия от внешней среды, а только получает от автомата команды совершить те или иные действия. При этом каждая команда всегда вызывает одно и то же действие (это и есть определение простого поведения).

Таким образом, в соответствии с автоматным подходом, сущности со сложным поведением следует представлять в виде *автоматизированных объектов управления* — так в теории управления называют объект управления, интегрированный с системой управления в одно устройство.

Парадигма автоматного программирования состоит в представлении сущностей со сложным поведением в виде автоматизированных объектов управления

1.4.3. Автоматы в программировании

Рассмотрев два типа автоматов: абстрактные, применяемые в теории формальных языков, и структурные, которые используются в логическом управлении — попытаемся обобщить полученные сведения, указать сходства и различия автоматных моделей и выделить те их черты, которые важны при применении автоматов в программировании.

Цель этого раздела — построить модель автоматизированного объекта управления (для краткости называемого просто *автоматизированным объектом*, АО). Это понятие уже было введено неформально (разд. 1.3) как совокупность управляющего автомата и объекта управления. Предпосылкой для создания этой концепции была необходимость проектирования и реализации систем со сложным поведением. Теперь попробуем придти к понятию автоматизированного объекта управления с другой стороны: путем обобщения традиционных автоматных моделей.

Вспомним абстрактные модели автоматов, рассмотренные в разд. 1.4.1. Как отмечалось выше, все они имеют похожую структуру: состоят из устройства управления (представляющего собой ДКА с выходом) и хранилища данных того или иного вида (лента, магазин). Для теории формальных языков вид хранилища и набор элементарных операций с данными имеют решающее значение: они определяют вычислительную мощность машины. При моделировании и высокоуровневой программной реализации сущностей со сложным поведением, удобнее заменить конкретное хранилище данных объектом управления (ОУ), множество (*вычислительных*) состояний которого может быть любым и определяется спецификой решаемой задачи (рис. 1.20).

Рис. 1.20. Слева направо: автомат с магазинной памятью, автомат с ленточной памятью (машина Тьюринга), автомат с произвольной памятью (автоматизированный объект)

Вместо ограниченного набора элементарных операций с данными будем использовать произвольные *запросы* и *команды*. В соответствии с традицией теории формальных языков такой вычислитель можно было бы назвать *автоматом с произвольной памятью*.

Примечание:

В объектно-ориентированном программировании *запросами* (или *чистыми функциями*) называются компоненты класса, возвращающие значение и не имеющие побочных эффектов. Такие компоненты не изменяют вычислительные состояния объекта, но позволяют получить некоторую информацию об этих состояниях. Другой тип компонентов класса (команды) напротив, предназначен для изменения состояний объектов.

Важная черта устройства управления всех абстрактных машин — его конечность. Управляющий автомат не только имеет конечное число состояний, но, кроме того, реализуемые им функции переходов и выходов оперируют исключительно конечными множествами. Именно это свойство позволяет описывать логику поведения машины явно: в виде таблицы или графа переходов. Поэтому свойство конечности устройства управления необходимо сохранить при построении модели автоматизированного объекта. Более того, это свойство целесообразно усилить следующим неформальным требованием: число управляющих состояний, входных и выходных воздействий должно быть небольшим *(обозримым)*. Управляющий автомат с тысячей состояний, безусловно, является конечным, однако, изобразить его граф переходов практически невозможно, что сводит на нет преимущества явного выделения управляющих состояний.

Напротив, число вычислительных состояний может быть сколь угодно большим (при переходе от модели к программной реализации оно с необходимостью станет конечным, однако, может остаться необозримым). В процессе работы управляющему автомату требуется получать информацию о вычислительном состоянии и изменять его. Однако в силу свойства конечности автомат не может напрямую считывать и записывать вычислительное состояние. Для этого и требуются операции объекта управления. Небольшое число запросов, возвращающих конечнозначные результаты, позволяет автомату получать информацию о вычислительных состояниях, которую он способен обработать. Небольшое число команд используется для «косвенного» изменения вычислительных состояний.

Операции с памятью в традиционных абстрактных вычислителях также можно считать командами и запросами. Рассмотрим, например, автомат с магазинной памятью. Его множество вычислительных состояний бесконечно: это множество всех возможных конфигураций стека. Для управления стеком автомат использует один запрос top, возвращающий символ на вершине стека, и две команды pop и push, первая из которых снимает символ со стека, а вторая заталкивает символ в стек .

Примечание:

Push можно считать как одной командой, аргументом которой является заталкиваемый символ, так и набором команд без аргументов — по одной для каждого символа магазинного алфавита. Это не имеет значения, поскольку магазинный алфавит конечен.

В вопросах, связанных с программированием, важна не только структура АО, но и особенности процесса его работы. Работа всех рассмотренных автоматных моделей разбита на такты. За такт они успевают считать входное воздействие, вычислить функции переходов и выходов и обновить значения выходных и внутренних переменных.

Некоторые из рассмотренных моделей (например, ДКА и детерминированный МП-автомат) начинают следующий такт работы только в том случае, если получают на вход очередной символ. Их работа всегда заканчивается по достижении конца входной строки. Обобщая такое поведение, введем понятие пассивной автоматной модели: каждый такт ее работы инициируется внешней средой, а по окончании такта управление передается обратно этой среде.

У таких автоматных моделей, как машина Тьюринга и структурные автоматы, напротив, после окончания текущего такта немедленно начинается следующий. Процесс работы продолжается до тех пор, пока автомат может совершить очередной переход . Такие автоматные модели можно назвать активными, поскольку, будучи однажды запущенными, они не нуждаются в дальнейших «стимулах» для продолжения работы.

Примечание:

Такой критерий завершения работы используется при описании машин Тьюринга. При спецификации поведения программных систем в целях наглядности удобно выделять на графах переходов специальные конечные (или завершающие) состояния. Переход в любое из таких состояний приводит к немедленному завершению работы автомата.

Примечание:

На самом деле, при аппаратной реализации синхронных структурных автоматных моделей начало нового такта инициируется *тактовым генератором*. Однако предполагается, что он является «внутренним» для автомата по сравнению с внешней средой, подающей входные воздействия. Поэтому, с точки зрения среды, такой автомат является активным.

Если активная автоматная модель в качестве входного воздействия считывает вычислительное состояние внешней среды, то для пассивной характерно событийное взаимодействие: среда сама (асинхронно) сигнализирует о своем изменении, вызывая автоматную модель с некоторым событием.

В программировании целесообразно использовать как активные, так и пассивные автоматные модели в зависимости от решаемой задачи. В пассивной модели, в общем случае, лишь некоторые компоненты входного воздействия являются событиями, а остальные представляют собой «традиционные» входные переменные: их значения опрашиваются самим автоматом, а их изменения не инициируют начало такта.

Вернемся к абстрактным вычислителям. Заметим, что в некоторых из них (таких как ДКА) автомат взаимодействует только с внешней средой, получая от нее входные символы. В других (например, в машине Тьюринга) — автомат общается лишь со своим объектом управления, или, в терминах теории абстрактных автоматов, со своей дополнительной памятью. В третьих (таких как МП-автомат) — устройство управления взаимодействует и с внешней средой и с объектом управления, причем от среды оно получает лишь входные воздействия, тогда как взаимодействие с объектом управления имеет двунаправленный характер. При построении модели автоматизированного объекта целесообразно использовать третий вариант как наиболее общий (рис. 1.21).

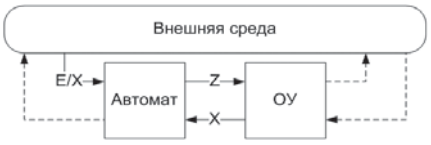

Рис. 1.21. Взаимодействие компонентов модели автоматизированного объекта

На этом рисунке сплошными стрелками обозначены традиционные и наиболее типичные для программных реализаций виды взаимодействия между автоматом, объектом управления и внешней средой. Автомат получает входные воздействия как со стороны среды, так и от объекта управления. В событийных системах часть или все компоненты входного воздействия со стороны среды могут быть событиями (множество событий обозначено на рисунке буквой E). Входное воздействие со стороны объекта управления формирует в модели *обратную связь* (от управляемого объекта к управляющему). Это воздействие может отсутствовать, тогда модель является *разомкнутой* — так в теории управления называются системы управления без обратной связи [23]. В противном случае модель называется замкнутой.

Автомат, в свою очередь, воздействует на объект управления.

Пунктирными стрелками обозначены менее распространенные, хотя и возможные, варианты взаимодействия. Так, автомат может оказывать выходное воздействие и на внешнюю среду. Однако таких связей обычно можно избежать, включив все управляемые автоматом сущности в состав его объекта управления. Отметим, что в программировании, в общем случае, различие между объектом управления и внешней средой носит скорее концептуальный, а не формальный характер. Создавая модель системы со сложным поведением, *разработчик производит ее декомпозицию на автоматизированные объекты*, определяя тем самым объект управления каждого автомата. В целях минимизации связей между модулями программной системы целесообразно проводить декомпозицию таким образом, чтобы автомат оказывал выходные воздействия только на собственный объект управления.

Кроме того, объект управления может взаимодействовать с внешней средой напрямую.

Напомним, что в абстрактных автоматных моделях входные и выходные воздействия обычно представляют собой символы некоторого конечного алфавита или цепочки таких символов, а в структурных моделях — битовые строки заданной длины. В программировании на вид входных и выходных воздействий нет ограничений: это могут быть символы, числа, строки, множества, последовательности, произвольные объекты — все зависит от специфики поставленной задачи и инструментов, используемых для ее решения.

Кроме того, могут различаться способы передачи входных воздействий автомату и интерпретации выходных воздействий в объекте управления.

Если по назначению сущность близка к традиционной системе управления, то представление входных и выходных воздействий битовыми строками будет удобным по тем же причинам, что и для структурных автоматных моделей. Однако интерпретация этого представления может быть различной. В примере со счетным триггером (разд. 1.4.2) каждое из двух значений выходной переменной соответствовало определенному вычислительному состоянию: включенной или выключенной лампочке. В программировании чаще используется другая интерпретация: каждой выходной переменной сопоставляется определенное *изменение* вычислительного состояния (действие, команда). При этом единица обозначает наличие действия, а ноль — его отсутствие. В этом случае вектору из нулей соответствует отсутствие каких-либо команд. Такой вид выходного воздействия может привести к недетерминизму в том случае, если результат зависит от последовательности выполнения команд. Поэтому в качестве выходного воздействия вместо множества команд часто используется последовательность команд.

При рассмотрении всевозможных деталей использования автоматных моделей в программировании становится ясно, что выбрать одну конкретную модель, подходящую для всех задач, невозможно. При программной реализации сущностей со сложным поведением применение могут найти активные и пассивные, разомкнутые и замкнутые модели, различные формы представления и интерпретации входных и выходных воздействий. Модель автоматизированного объекта управления должна быть применима для любой сущности со сложным поведением, и поэтому целесообразно сформулировать ее довольно абстрактно. Примеры программной реализации сущностей со сложным поведением, которые будут приведены в последующих главах, являются конкретными воплощениями этой модели.

Итак, приведем формальное определение автоматизированного объекта управления.

Пара (A, O), состоящая из управляющего автомата и объекта управления, называется *автоматизированным объектом управления*.

Управляющий автомат представляет собой шестерку $(X, Y, Z, y_0, \varphi, \delta)$, где $X = X_E \times X_O$ — конечное множество входных воздействий, причем каждое входное воздействие x состоит из компоненты x_E, порождаемой внешней средой, и компоненты x_O, порождаемой объектом управления; Y — конечное множество управляющих состояний; Z — конечное множество выходных воздействий; $y_0 \in Y$ — начальное состояние; $\varphi = \varphi' \times \varphi''$ — функция выходов (выходных воздействий), состоящая, в общем случае, из двух компонент: функции выходных воздействий в состояниях $\varphi': Y \to Z$ и функции выходных воздействий на переходах $\varphi'': X \times Y \to Z$; $\delta: X \times Y \to Y$ — функция переходов.

Объект управления — это тройка (V, f_q, f_c), где V — потенциально бесконечное множество вычислительных состояний (или значений), $f_q: V \to X_O$ — функция, сопоставляющая входное воздействие вычислительному состоянию, $f_c: Z \times V \to V$ —функция, изменяющая вычислительное состояние в зависимости от выходного воздействия.

Функции f_q и f_c являются математическими эквивалентами набора запросов и набора команд соответственно.

Графическое представление описанной модели приведено на рис. 1.22.

Таким образом, с позиций теории формальных языков автоматизированный объект — это автомат с произвольной памятью. По вычислительной мощности он, в общем случае, эквивалентен машине Тьюринга. Формально, для обеспечения эквивалентности необходимо потребовать, чтобы запросы и команды объекта управления являлись *вычислимыми* функциями (их можно было вычислить с помощью машины Тьюринга). В определении это свойство подразумевается.

Рис. 1.22. Автоматизированный объект управления

Таким образом, с позиций теории формальных языков автоматизированный объект — это автомат с произвольной памятью. По вычислительной мощности он, в общем случае, эквивалентен машине Тьюринга. Формально, для обеспечения эквивалентности необходимо потребовать, чтобы запросы и команды объекта управления являлись *вычислимыми* функциями (их можно было вычислить с помощью машины Тьюринга). В определении это свойство подразумевается.

В разомкнутой модели автоматизированного объекта входные воздействия поступают только от внешней среды ($X = X_E$), а запросы объекта управления отсутствуют (рис. 1.23). Такой автоматизированный объект по вычислительной мощности эквивалентен ДКА: его объект управления уже не является дополнительной памятью, а скорее представляет собой разновидность выходной ленты.

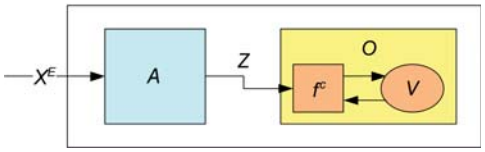

Рис. 1.23. Разомкнутый автоматизированный объект управления

В терминах теории логического управления автоматизированный объект — это, в общем случае, замкнутая система, управляемая автоматом первого рода с выходным преобразователем, в качестве которого может использоваться автомат Мура, Мили или смешанный автомат.

Как было упомянуто выше, автоматизированный объект эквивалентен по вычислительной мощности машине Тьюринга. Иначе говоря, для любого АО можно построить машину Тьюринга, которая решает ту же задачу, и наоборот. С одной стороны, из этого свойства следует важное достоинство автоматного подхода: с помощью автоматизированного объекта можно описать любой алгоритм, любую программу, которая только может быть выполнена компьютером. С другой стороны, возникает вопрос: зачем было изобретать автоматизированный объект вместо того, чтобы выбрать на роль модели сущности со сложным поведением более простую и столь же мощную машину Тьюринга?

Ответ приходит сам собой, если вспомнить пример с машиной Тьюринга, реализующей функцию инкремент (разд. 1.3). Для вычисления простейшей функции понадобился управляющий автомат из трех состояний. Автомат машины Тьюринга, выполняющей умножение двух чисел [12] (преобразование строки «0^n10^m1» в строку «0^{nm}»), содержит уже 12 состояний! Такая модель не только не упрощает описа-

ние сложного поведения, но и значительно усложняет описание простого.

Программирование на машине Тьюринга (или *тьюрингово программирование* [25]) чрезвычайно сложно и непрактично по той причине, что набор элементарных операций этой машины с дополнительной памятью очень ограничен. Поэтому автомату приходится не только управлять, но и *выполнять не свойственную ему функцию вычисления*. Переход к модели автоматизированного объекта управления позволяет использовать в качестве элементарных операций произвольные запросы и команды. При этом на управляющий автомат ложится лишь часть ответственности по реализации алгоритма: та, что связана с логикой (управлением) — то, для чего автоматы, собственно, и предназначены.

Можно сказать, что при программировании на машине Тьюринга любое поведение (кроме алгоритма, состоящего из единственного шага, на котором необходимо записать символ и сдвинуть головку) является сложным. В автоматном программировании грань между простым и сложным поведением определяется разработчиком. Умножение двух чисел может быть сложным, если вы собираетесь эмулировать и визуализировать счеты. Построение самобалансирующегося дерева [26] может быть элементарной операцией, если в вашем распоряжении есть соответствующая библиотека, предоставляющая готовую функцию. Таким образом, переход от тьюрингова программирования к автоматному состоит в повышении уровня абстракции операций с памятью, причем этот уровень при автоматном подходе не фиксирован, а зависит от решаемой задачи.

Более того, низкоуровневый автоматизированный объект может быть инкапсулирован в объекте управления, существующем на более высоком уровне абстракции. Благодаря такому вложению автоматизированных объектов автоматное программирование поддерживает концепцию выделения уровней абстракции, распространенную в современной методологии разработки ПО.

Выбор уровня абстракции элементарных операций при моделировании сущности со сложным поведением определяет разделение поведения на логику и семантику, а состояний на управляющие и вычислительные. Это и есть наиболее творческий и нетривиальный шаг в автоматном подходе к разработке ПО. Выбор чересчур простых элементарных операций приводит к разрастанию и усложнению автомата, логика становится менее понятной и перегруженной деталями, которые эффективнее было бы реализовать в объекте управления. Машина Тьюринга — крайний случай такого «злоупотребления логикой».

Напротив, выбор слишком абстрактных запросов и команд ведет к усложнению их реализации, перегруженности флагами и условными конструкциями. Вырожденный случай такого «злоупотребления семантикой» — использование традиционных (неавтоматных) стилей программирования, где любое поведение считается простым (рис. 1.24).

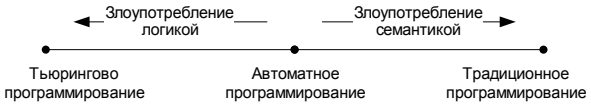

Рис. 1.24. Золотая середина в разделении поведения на логику и семантику

Нахождение компромисса между сложностью автомата и сложностью операций объекта управления, примирение тьюрингова программирования с традиционным и есть «миссия» автоматного подхода в мире разработки программного обеспечения

Защитники традиционных парадигм программирования могут возразить, что борьба со сложностью программ осуществляется там путем декомпозиции: функциональной (в процедурном программировании) или объектной (при объектно-ориентированном подходе). При наличии сложной логики декомпозиция лишь распределяет ее по различным компонентам системы, но не делает ее явной и, конечно же, не устраняет ее. Таким образом, декомпозиция отчасти выполняет одну функцию автоматного подхода — упрощение операций, однако совершенно не справляется с другой — формированием у разработчика целостной картины поведения сущности.

С другой стороны, борьба со сложностью автоматов оставалась (и отчасти, остается до сих пор) одной из главных задач автоматного программирования. Эта задача по-разному решается в процедурной и объектно-ориентированной разновидностях парадигмы. Два различных решения подробно обсуждаются во второй и третьей главах этой книги. Пока отметим лишь то, что объектно-ориентированный подход поддерживает понятие автоматизированного объекта управления более непосредственно. Поэтому борьба со сложностью как автомата, так и объекта управления, осуществляются при этом подходе гораздо проще и эффективнее.

```
def runTest(label, controlResult, test)
{
  def sw = Stopwatch.StartNew();
  def result = test();
  def time = sw.Elapsed;
  WriteLine(«{0, 20} ({1}) время выполнения: {2}»,
    label, if (result.Equals(controlResult)) «OK» else «Fail», time);
}

def rnd = System.Random(1);
def buffer : array[byte] = array(size);
rnd.NextBytes(buffer);

def foreachTest = () => {
    mutable sum = 0L;
    foreach (b in buffer)
      when (b % 10 == 0)
        sum += b;
    sum;
  };

def controlResult = foreachTest();

runTest(«foreach», controlResult, foreachTest);

runTest(«for», controlResult, () => {
  mutable sum = 0L;
  for (mutable i = 0; i < buffer.Length; i++)
  {
    def b = buffer[i];
    when (b % 10 == 0)
      sum += b;
  }
  sum;
});

runTest(«Fold(array)», controlResult, () =>
  buffer.Fold(0L, (b : byte, acc : long) => if (b % 10 == 0) acc + b else acc)
);

runTest(«Fold(IEnumerable)», controlResult, () =>
  (buffer : IEnumerable[byte]).Fold(0L, (b : byte, acc : long) =>
    if (b % 10 == 0) acc + b else acc)
);
```

продолжение на стр. 72