Лекция 4 Тестирование программы как чёрного ящика

Основная идея в тестировании системы как черного ящика состоит в том, что все материалы, которые доступны тестировщику, - требования на систему, описывающие ее поведение, и сама система, работать с которой он может, только подавая на ее входы некоторые внешние воздействия и наблюдая на выходах некоторый результат. Все внутренние особенности реализации системы скрыты от тестировщика, - таким образом, система представляет собой "черный ящик", правильность поведения которого по отношению к требованиям и предстоит проверить.

С точки зрения программного кода черный ящик может представлять с собой набор классов (или модулей) с известными внешними интерфейсами, но недоступными исходными текстами.

Основная задача тестировщика для данного метода тестирования состоит в последовательной проверке соответствия поведения системы требованиям. Кроме того, тестировщик должен проверить работу системы в критических ситуациях - что происходит в случае подачи неверных входных значений. В идеальной ситуации все варианты критических ситуаций должны быть описаны в требованиях на систему и тестировщику остается только придумывать конкретные проверки этих требований. Однако в реальности в результате тестирования обычно выявляется два типа проблем системы.

1.Несоответствие поведения системы требованиям

2.Неадекватное поведение системы в ситуациях, не предусмотренных требованиями.

Отчеты об обоих типах проблем документируются и передаются разработчикам. При этом проблемы первого типа обычно вызывают изменение программного кода, гораздо реже - изменение требований. Изменение требований в данном случае может потребоваться из-за их противоречивости (несколько разных требований описывают разные модели поведения системы в одной и той же самой ситуации) или некорректности (требования не соответствуют действительности).

Проблемы второго типа однозначно требуют изменения требований ввиду их неполноты - в требованиях явно пропущена ситуация, приводящая к неадекватному поведению системы. При этом под неадекватным поведением может пониматься как полный крах системы, так и вообще любое поведение, не описанное в требованиях.

Тестирование черного ящика называют также тестированием по требованиям, т.к. это единственный источник информации для построения тест-плана.

Основная статья: Чёрный ящик

Под «чёрным ящиком» понимается объект исследования, внутреннее устройство которого неизвестно. Понятие «чёрный ящик» предложено У. Р. Эшби. В кибернетике оно позволяет изучать поведение систем, то есть их реакций на разнообразные внешние воздействия и в то же время абстрагироваться от их внутреннего устройства.

Манипулируя только лишь со входами и выходами, можно проводить определённые исследования. На практике всегда возникает вопрос, насколько гомоморфизм «чёрного» ящика отражает адекватность его изучаемой модели, то есть как полно в модели отражаются основные свойства оригинала.

Описание любой системы управления во времени характеризуется картиной последовательности её состояний в процессе движения к стоящей перед нею цели. Преобразование в системе управления может быть либо взаимно-однозначным и тогда оно называется изоморфным, либо только однозначным, в одну сторону. В таком случае преобразование называют гомоморфным.

«Чёрный» ящик представляет собой сложную гомоморфную модель кибернетической системы, в которой соблюдается разнообразие. Он только тогда является удовлетворительной моделью системы, когда содержит такое количество информации, которое отражает разнообразие системы. Можно предположить, что чем большее число возмущений действует на входы модели системы, тем большее разнообразие должен иметь регулятор.

В настоящее время известны два вида «чёрных» ящиков. К первому виду относят любой «чёрный» ящик, который может рассматриваться как автомат, называемый конечным или бесконечным. Поведение таких «чёрных» ящиков известно. Ко второму виду относятся такие «чёрные» ящики, поведение которых может быть наблюдаемо только в эксперименте. В таком случае в явной или неявной форме высказывается гипотеза о предсказуемости поведения «чёрного» ящика в вероятностном смысле. Без предварительной гипотезы невозможно любое обобщение, или, как говорят, невозможно сделать индуктивное заключение на основе экспериментов с «чёрным» ящиком. Для обозначения модели «чёрного» ящика Н. Винером предложено понятие «белого» ящика. «Белый» ящик состоит из известных компонентов, то есть известных X, Y, δ, λ. Его содержимое специально подбирается для реализации той же зависимости выхода от входа, что и у соответствующего «чёрного» ящика. В процессе проводимых исследований и при обобщениях, выдвижении гипотез и установления закономерностей возникает необходимость корректировки организации «белого» ящика и смены моделей. В связи с этим при моделировании исследователь должен обязательно многократно обращаться к схеме отношений «чёрный» — «белый» ящик.

Для науки метод «чёрный» ящик имеет весьма большое значение. С его помощью в науке были сделаны очень многие выдающиеся открытия. Например, учёный Гарвей ещё в XVII веке предугадал строение сердца. Он моделировал работу сердца насосом, позаимствовав идеи из совершенно другой области современных ему знаний — гидравлики. Практическая ценность метода «чёрный» ящик заключается во-первых, в возможности исследования очень сложных динамических систем, и, во-вторых, в возможности замены одного «ящика» другим. Окружающая действительность и биология дают массу примеров выявления строения систем методом «чёрного» ящика.

Принципы тестирования чёрного ящика

В этом методе программа рассматривается как чёрный ящик. Целью тестирования ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Для обнаружения всех ошибок в программе необходимо выполнить исчерпывающее тестирование, то есть тестирование на всевозможных наборах данных. Для большинства программ такое невозможно, поэтому применяют разумное тестирование, при котором тестирование программы ограничивается небольшим подмножеством всевозможных наборов данных. При этом необходимо выбирать наиболее подходящие подмножества, подмножества с наивысшей вероятностью обнаружения ошибок.

Свойства правильно выбранного теста

Уменьшает более чем на одно число других тестов, которые должны быть разработаны для разумного тестирования.

Покрывает значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибки до и после ограниченного множества тестов.

Приёмы тестирования чёрного ящика

Эквивалентное разбиение.

Анализ граничных значений.

Анализ причинно-следственных связей.

Предположение об ошибке.

Рассмотрим подробнее каждый из этих методов:

Эквивалентное разбиение

Основу метода составляют два положения:

Исходные данные необходимо разбить на конечное число классов эквивалентности. В одном классе эквивалентности содержатся такие тесты, что если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку.

Каждый тест должен включать, по возможности, максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

Разработка тестов этим методом осуществляется в два этапа: выделение классов эквивалентности и построение теста.

Классы эквивалентности выделяются путём выбора каждого входного условия, которые берутся с помощью технического задания или спецификации и разбиваются на две и более группы. Для этого используется следующая таблица:

Входное условие Правильные классы эквивалентности Неправильные классы эквивалентности

' ' '

Заметим, что различают два типа классов эквивалентности: *правильные классы эквивалентности*, представляющие правильные входные данные программы, и *неправильные классы эквивалентности*, представляющие все другие возможные состояния условий (т. е. ошибочные входные значения). Таким образом, придерживаются одного из принципов тестирования о необходимости сосредоточивать внимание на неправильных или неожиданных условиях.

Выделение классов эквивалентности является эвристическим способом, однако существует ряд правил:

Если входное условие описывает область значений, например «Целое число принимает значение от 0 до 999», то существует один правильный класс эквивалентности и два неправильных.

Если входное условие описывает число значений, например «Число строк во входном файле лежит в интервале (1..6)», то также существует один правильный класс и два неправильных.

Если входное условие описывает множество входных значений, то определяется количество правильных классов, равное количеству элементов в множестве входных значений. Если входное условие описывает ситуацию «должно быть», например «Первый символ должен быть заглавным», тогда один класс правильный и один неправильный.

Если есть основание считать, что элементы внутри одного класса эквивалентности могут программой трактоваться по-разному, необходимо разбить данный класс на подклассы.

1. Если входное условие описывает *область* значений (например, «целое данное может принимать значения от 1 до 99»), то определяются один правильный класс эквивалентности (1 ≤ значение целого данного ≤ 99) и два неправильных (значение целого данного <1 и значение целого данного >99).
2. Если входное условие описывает *число* значений (например, «в автомобиле могут ехать от одного до шести человек»), то определяются один правильный класс эквивалентности и два неправильных (ни одного и более шести человек).
3. Если входное условие описывает *множество* входных значений и есть основание полагать, что каждое значение программа трактует особо (например, «известны должности ИНЖЕНЕР, ТЕХНИК, НАЧАЛЬНИК ЦЕХА, ДИРЕКТОР»), то определяется правильный класс эквивалентности для каждого значения и один неправильный класс эквивалентности (например, «БУХГАЛТЕР»).
4. Если входное условие описывает ситуацию «должно быть» (например, «первым символом идентификатора должна быть буква»), то определяется один правильный класс эквивалентности (первый символ – буква) и один неправильный (первый символ – не буква).
5. Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности.

На этом шаге тестирующий на основе таблицы должен составить тесты, покрывающие собой все правильные и неправильные классы эквивалентности. При этом составитель должен минимизировать общее число тестов.

Определение тестов:

Каждому классу эквивалентности присваивается уникальный номер.

Если ещё остались не включённые в тесты правильные классы, то пишутся тесты, которые покрывают максимально возможное количество классов.

Если остались не включённые в тесты неправильные классы, то пишут тесты, которые покрывают только один класс.

Анализ граничных значений

Граничные условия — это ситуации, возникающие на высших и нижних границах входных классов эквивалентности.

Анализ граничных значений отличается от эквивалентного разбиения следующим:

Выбор любого элемента в классе эквивалентности в качестве представительного осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.

При разработке тестов рассматриваются не только входные значения (пространство входов), но и выходные (пространство выходов).

Метод требует определённой степени творчества и специализации в рассматриваемой задаче.

Существует несколько правил:

Построить тесты с неправильными входными данными для ситуации незначительного выхода за границы области значений. Если входные значения должны быть в интервале [-1.0 .. +1.0], проверяем −1.0, 1.0, −1.000001, 1.000001.

Обязательно писать тесты для минимальной и максимальной границы диапазона.

Использовать первые два правила для каждого из входных значений (использовать пункт 2 для всех выходных значений).

Если вход и выход программы представляет упорядоченное множество, сосредоточить внимание на первом и последнем элементах списка.

Анализ граничных значений, если он применён правильно, позволяет обнаружить большое число ошибок. Однако определение этих границ для каждой задачи может являться отдельной трудной задачей. Также этот метод не проверяет комбинации входных значений.

Анализ причинно-следственных связей

Этапы построения теста:

Спецификация разбивается на рабочие участки.

В спецификации определяются множество причин и следствий. Под причиной понимается отдельное входное условие или класс эквивалентности. Следствие представляет собой выходное условие или преобразование системы. Здесь каждой причине и следствию присваивается номер.

На основе анализа семантического (смыслового) содержания спецификации строится таблица истинности, в которой последовательно перебираются всевозможные комбинации причин и определяются следствия для каждой комбинации причин.

Таблица снабжается примечаниями, задающими ограничения и описывающими комбинации, которые невозможны. Недостатком этого подхода является плохое исследование граничных условий.

Предположение об ошибке

Тестировщик с большим опытом выискивает ошибки без всяких методов, но при этом он подсознательно использует метод предположения об ошибке. Данный метод в значительной степени основан на интуиции. Основная идея метода состоит в том, чтобы составить список, который перечисляет возможные ошибки и ситуации, в которых эти ошибки могли проявиться. Потом на основе списка составляются тесты.

хороший тест имеет приемлемую вероят-
ность обнаружения ошибки и что исчерпывающее входное тестирование программы невозможно. Следовательно, тестирование программы ограничивается использованием небольшого подмножества всех возможных входных данных. Тогда, конечно, хотелось бы выбрать для тестирования самое подходящее подмножество (т. е. подмножество с наивысшей вероятностью обнаружения большинства ошибок).

Правильно выбранный тест этого подмножества должен обладать двумя свойствами:

• уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для достижения заранее определенной цели «приемлемого» тестирования;

• покрывать значительную часть других возможных тестов, что в не-

которой степени свидетельствует о наличии или отсутствии оши до и после применения этого ограниченного множества значений

входных данных.

Указанные свойства, несмотря на их кажущееся подобие, описывают два различных положения. Во-первых, каждый тест должен включать столько различных входных условий, сколько это возможно, с тем, чтобы минимизировать общее число необходимых тестов. Во-вторых, необходимо пытаться разбить входную область программы на конечное число классов эквивалентности так, чтобы можно было предположить (конечно, не абсолютно уверенно), что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Иными словами, если один тест класса эквивалентности обнаруживает ошибку, то следует ожидать, что и все другие тесты этого класса эквивалентности будут обнаруживать ту же самую ошибку. Наоборот, если тест не обнаруживает ошибки, то следует ожидать, что ни один тест этого класса эквивалентности не будет обнаруживать ошибки (в том случае, когда некоторое подмножество класса эквивалентности не попадает в пределы любого другого класса эквивалентности, так как классы эквивалентности могут пересекаться).

Эти два положения составляют основу методологии тестирования по принципу черного ящика, известной как эквивалентное разбиение. Второе положение используется для разработки набора «интересных» условий, которые должны быть протестированы, а первое – для разработки минимального набора тестов, покрывающих эти условия.

Примером класса эквивалентности для программы о треугольнике (см. § 1.1) является набор «трех равных чисел, имеющих целые значения, большие нуля». Определяя этот набор как класс эквивалентности, устанавливают, что если ошибка не обнаружена некоторым тестом данного набора, то маловероятно, что она будет обнаружена другим тестом набора. Иными словами, в этом случае время тестирования лучше затратить на что-нибудь другое (на тестирование других классов эквивалентности).

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

1) выделение классов эквивалентности;

2) построение тестов.

3.2.1.1. Выделение классов эквивалентности

Классы эквивалентности выделяются путем выбора каждого входного условия (обычно это предложение или фраза в спецификации) и разбиением его на две или более групп. Для проведения этой операции используют таблицу, изображенную на рис. 5.

Входные условия Правильные классы эквивалентности

Неправильные классы эквивалентности

Рис. 5. Форма таблицы для перечисления классов эквивалентности

Если задаться входными или внешними условиями, то выделение классов эквивалентности представляет собой в значительной степени эвристический процесс. При этом существует ряд правил:

Этот процесс ниже будет кратко проиллюстрирован.

3.2.1.2. Построение тестов

Второй шаг заключается в использовании классов эквивалентности для построения тестов. Этот процесс включает в себя:__ 2. Проектирование новых тестов, каждый из которых покрывает как

можно большее число непокрытых правильных классов эквивалентности, до тех пор пока все правильные классы эквивалентности не будут покрыты (только не общими) тестами.
3. Запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы эквивалентности не будут покрыты тестами.
Причина покрытия неправильных классов эквивалентности индивидуальными тестами состоит в том, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами. Например, спецификация устанавливает «тип книги при поиске (ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА, ПРОГРАММИРОВАНИЕ или ОБЩИЙ) и количество (1-999)». Тогда тест
XYZ 0
отображает два ошибочных условия (неправильный тип книги и количество) и, вероятно, не будет осуществлять проверку количества, так как программа может ответить: «XYZ – несуществующий тип книги» и не проверять остальную часть входных данных.
3.2.1.3. Пример
Предположим, что при разработке интерпретатора для подмножества языка Бейсик требуется протестировать синтаксическую проверку оператора DIM [1]. Спецификация приведена ниже. (Этот оператор не является полным оператором DIM Бейсика; спецификация была значительно сокращена, что позволило сделать ее «учебным примером». Не следует думать, что тестирование реальных программ так же легко, как в этом примере.) В спецификации элементы, написанные латинскими буквами, обозначают синтаксические единицы, которые в реальных операторах должны быть заменены соответствующими значениями, в квадратные скобки заключены необязательные элементы, многоточие показывает, что предшествующий ему элемент может быть повторен подряд несколько раз.
Оператор DIM используется для определения массивов, форма оператора DIM:
DIM *ad*[,*ad*]…,
где *ad* есть описатель массива в форме
n(*d*[,*d*]...),
n – символическое имя массива, а *d* – индекс массива. Символические имена могут содержать от одного до шести символов – букв или цифр, причем первой должна быть буква. Допускается от одного до семи индексов. Форма индекса
[*lb* :] *ub*,
где *lb* и *ub* задают нижнюю и верхнюю границы индекса массива. Граница может быть либо константой, принимающей значения от -65534 до 65535, либо целой переменной (без индексов). Если *lb* не определена, то предполагается, что она равна единице. Значение *ub* должно быть больше или равно *lb.* Если *lb* определена, то она может иметь отрицательное, нулевое или положительное значение. Как и все операторы, оператор DIM может быть продолжен на нескольких строках. (Конец спецификации.)
Первый шаг заключается в том, чтобы идентифицировать входные условия и по ним определить классы эквивалентности (табл. 1). Классы эквивалентности в таблице обозначены числами в круглых скобках.

Следующий шаг – построение теста, покрывающего один или более правильных классов эквивалентности. Например, тест

DIM A(2)

покрывает классы 1, 4, 7, 10, 12, 15, 24, 28, 29 и 40 (см. табл. 1). Далее определяются один или более тестов, покрывающих оставшиеся правильные классы эквивалентности. Так, тест

DIM A12345(I, 9, J4XXXX.65535, 1, KLM,
X 100), BBB (-65534:100, 0:1000, 10:10, I:65535)

покрывает оставшиеся классы. Перечислим неправильные классы эквивалентности и соответствующие им тесты:

(3) DIM (21) DIM C(I.,10)
(5) DIM (10) (23) DIM C(10,1J)
(6) DIM A234567(2) (25) DIM D(-65535:1)
(9) DIM A.1(2) (26) DIM D (65536)
(11) DIM 1A(10) (31) DIM D(4:3)
(13) DIM B (37) DIM D(A(2):4)
(14) DIM B (4,4,4,4,4,4,4,4) (38) DIM D(.:4)
(17) DIM B(4,A(2))

Эти классы эквивалентности покрываются 18 тестами. Можно, при желании, сравнить данные тесты с набором тестов, полученным каким-либо специальным методом.

Хотя эквивалентное разбиение значительно лучше случайного выбора тестов, оно все же имеет недостатки (т. е. пропускает определенные типы высокоэффективных тестов). Следующие два метода – анализ граничных значений и использование функциональных диаграмм (диаграмм причинно-следственных связей cause-effect graphing) – свободны от многих недостатков, присущих эквивалентному разбиению.

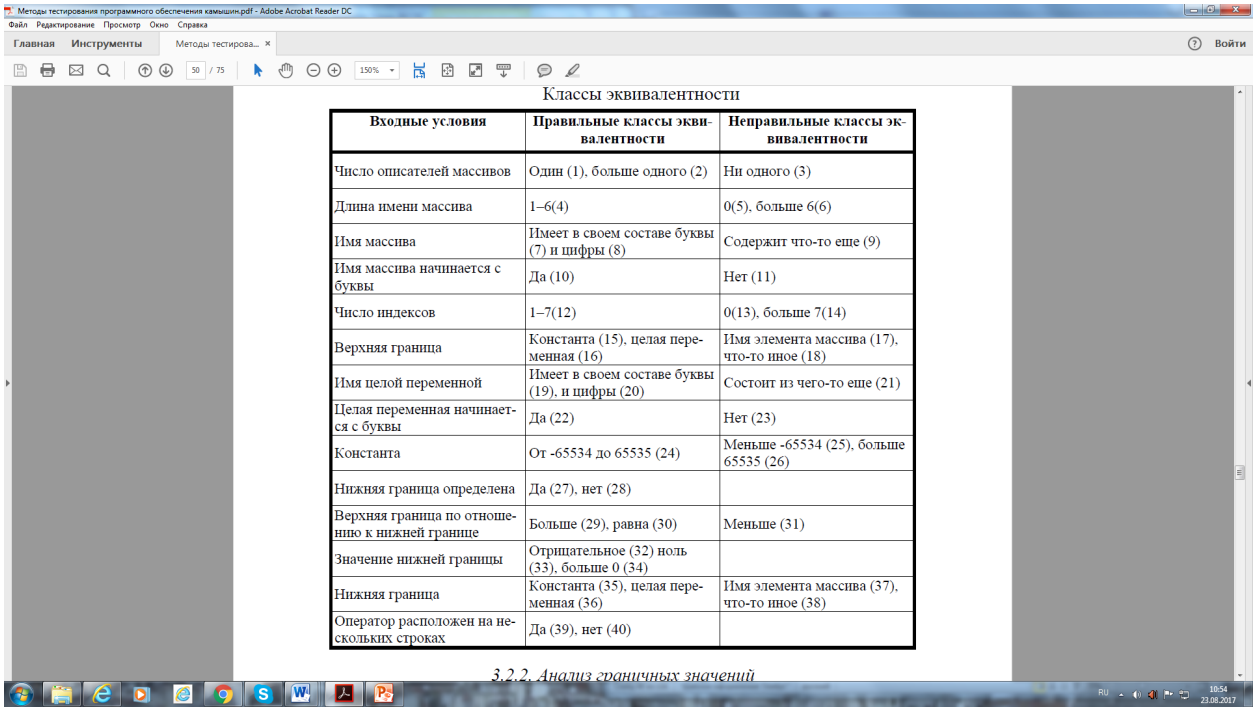
Классы эквивалентности

Входные условия Правильные классы эквивалентности

Неправильные классы эквивалентности

Число описателей массивов Один (1), больше одного (2) Ни одного (3)
Длина имени массива 1–6(4) 0(5), больше 6(6)
Имя массива Имеет в своем составе буквы
(7) и цифры (8) Содержит что-то еще (9)
Имя массива начинается с
буквы Да (10) Нет (11)
Число индексов 1–7(12) 0(13), больше 7(14)
Верхняя граница Константа (15), целая переменная (16)
Имя элемента массива (17),
что-то иное (18)
Имя целой переменной Имеет в своем составе буквы
(19), и цифры (20) Состоит из чего-то еще (21)
Целая переменная начинается с буквы Да (22) Нет (23)
Константа От -65534 до 65535 (24) Меньше -65534 (25), больше
65535 (26)
Нижняя граница определена Да (27), нет (28)
Верхняя граница по отношению к нижней границе Больше (29), равна (30) Меньше (31)
Значение нижней границы Отрицательное (32) ноль
(33), больше 0 (34)
Нижняя граница Константа (35), целая переменная (36)

Имя элемента массива (37),
что-то иное (38)
Оператор расположен на не-
скольких строках Да (39), нет (40)



Классы эквивалентности

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Число описателей массивов	Один (1), больше одного (2)	Ни одного (3)
Длина имени массива	1–6(4)	0(5), больше 6(6)
Имя массива	Имеет в своем составе буквы (7) и цифры (8)	Содержит что-то еще (9)
Имя массива начинается с буквы	Да (10)	Нет (11)
Число индексов	1–7(12)	0(13), больше 7(14)
Верхняя граница	Константа (15), целая переменная (16)	Имя элемента массива (17), что-то иное (18)
Имя целой переменной	Имеет в своем составе буквы (19), и цифры (20)	Состоит из чего-то еще (21)
Целая переменная начинается с буквы	Да (22)	Нет (23)
Константа	От -65534 до 65535 (24)	Меньше -65534 (25), больше 65535 (26)
Нижняя граница определена	Да (27), нет (28)	
Верхняя граница по отношению к нижней границе	Больше (29), равна (30)	Меньше (31)
Значение нижней границы	Отрицательное (32) ноль (33), больше 0 (34)	
Нижняя граница	Константа (35), целая переменная (36)	Имя элемента массива (37), что-то иное (38)
Оператор расположен на нескольких строках	Да (39), нет (40)	

3.2.2. Анализ граничных значений

3.2.2. Анализ граничных значений

Как показывает опыт, тесты, исследующие *граничные условия,* приносят большую пользу, чем тесты, которые их не исследуют. **Граничные условия** – это ситуации, возникающие непосредственно на, выше или ниже границ входных и выходных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения в двух отношениях:

1. Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.
2. При разработке тестов рассматривают не только входные условия (пространство входов), но и *пространство результатов* (т. е. выходные классы эквивалентности).

Достаточно трудно описать принимаемые решения при анализе граничных значений, так как это требует определенной степени творчества и специализации в рассматриваемой проблеме. (Следовательно, анализ граничных значений, как и многие другие аспекты тестирования, в значительной мере основывается на способностях человеческого интеллекта.) Тем не менее существует несколько общих правил этого метода.

1. Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений. Например, если правильная область входных значений есть от –1.0 до +1.0, то нужно написать тесты для ситуаций –1.0, 1.0, –1.001 и 1.001.
2. Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих значений, если входное условие удовлетворяет дискретному ряду значений. Например, если входной файл может содержать от 1 до 255 записей, то получить тесты для 0, 1, 255 и 256 записей.

3. Использовать первое правило для каждого выходного условия. Например, если программа вычисляет ежемесячный расход и если минимум расхода составляет $0.00, а максимум – $1165.25, то построить тесты, которые вызывают расходы с $0.00 и $1165.25. Кроме того, построить, если это возможно, тесты, которые вызывают отрицательный расход и расход больше 1165.25 дол. Заметим, что важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей (например, при рассмотрении подпрограммы вычисления синуса). Не всегда также можно получить результат вне выходной области, но тем не менее стоит рассмотреть эту возможность.
4. Использовать второе правило для каждого выходного условия. Например, если система информационного поиска отображает на экране наиболее релевантные статьи в зависимости от входного запроса, но никак не более четырех рефератов, то построить тесты, такие, чтобы программа отображала нуль, один и четыре реферата, и тест, который мог бы вызвать выполнение программы с ошибочным отображением пяти рефератов.
5. Если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества.
6. Попробовать свои силы в поиске других граничных условий.

Чтобы проиллюстрировать необходимость анализа граничных значений, можно использовать программу анализа треугольника, приведенную в первой главе. Для задания треугольника входные значения должны__ быть целыми положительными числами, и сумма любых двух из них должна быть больше третьего. Если определены эквивалентные разбиения, то целесообразно определить одно разбиение, в котором это условие выполняется, и другое, в котором сумма двух целых не больше третьего. Следовательно, двумя возможными тестами являются 3–4–5 и 1–2–4. Тем не менее, здесь есть вероятность пропуска ошибки. Иными словами, если выражение в программе было закодировано как $A + B \geq C$ вместо $A + B > C$, то программа ошибочно сообщала бы нам, что числа 1–2–3 представляют правильный равносторонний треугольник. Таким образом, **существенное различие** между анализом граничных значений и эквивалентным разбиением заключается в том, что анализ граничных значений исследует ситуации, возникающие *на* и *вблизи границ эквивалентных разбиений*.

В качестве примера применения метода анализа граничных значений рассмотрим следующую спецификацию программы [1].

Пусть имеется программа или модуль, которая сортирует различную информацию об экзаменах. Входом программы является файл, названный results.txt, который содержит 80-символьные записи. Первая запись представляет название; ее содержание используется как заголовок каждого выходного отчета. Следующее множество записей описывает правильные ответы на экзамене. Каждая запись этого множества содержит «2» в качестве последнего символа. В первой записи в колонках 1–3 задается число ответов (оно принимает значения от 1 до 999). Колонки 10–59 включают сведения о правильных ответах на вопросы с номерами 1–50 (любой символ воспринимается как ответ). Последующие записи содер-

жат в колонках 10–59 сведения о правильных ответах на вопросы с номерами 51–100, 101–150 и т. д. Третье множество записей описывает ответы каждого студента; любая запись этого набора имеет число «3» в восьмидесятой колонке. Для каждого студента первая запись в колонках 1–9 содержит его имя или номер (любые символы); в колонках 10–59 помещены сведения о результатах ответов студентов на вопросы с номерами 1–50. Если в тесте предусмотрено более чем 50 вопросов, то последующие записи для студента описывают ответы 51–100, 101–150 и т. д. в колонках 10–59. Максимальное число студентов – 200. Форматы входных записей показаны на рис. 6.

Выходными записями являются:

1) отчет, упорядоченный в лексикографическом порядке идентификаторов студентов и показывающий качество ответов каждого студента (процент правильных ответов) и его ранг;

2) аналогичный отчет, но упорядоченный по качеству;

3) отчет, показывающий среднее значение, математическое ожидание (медиану) и дисперсию (среднеквадратическое отклонение) качества ответов;__ 4) отчет, упорядоченный по номерам вопросов и показывающий процент студентов, отвечающих правильно на каждый вопрос.

(Конец спецификации)

Название
Число Правильные ответы 1-50
вопросов 2
1 80
1 9 10 59 60 79 80
Правильные ответы 50-100 2
1 9 10 59 60 79 80
Идентификатор Ответы студента 1-50
студента 3
1 9 10 59 60 79 80
Ответы студента 50-100 3
1 9 10 59 60 79 80
Идентификатор Ответы студента 1-50
студента 3
1 9 10 59 60 79 80

Рис. 6. Структуры входных записей для программы

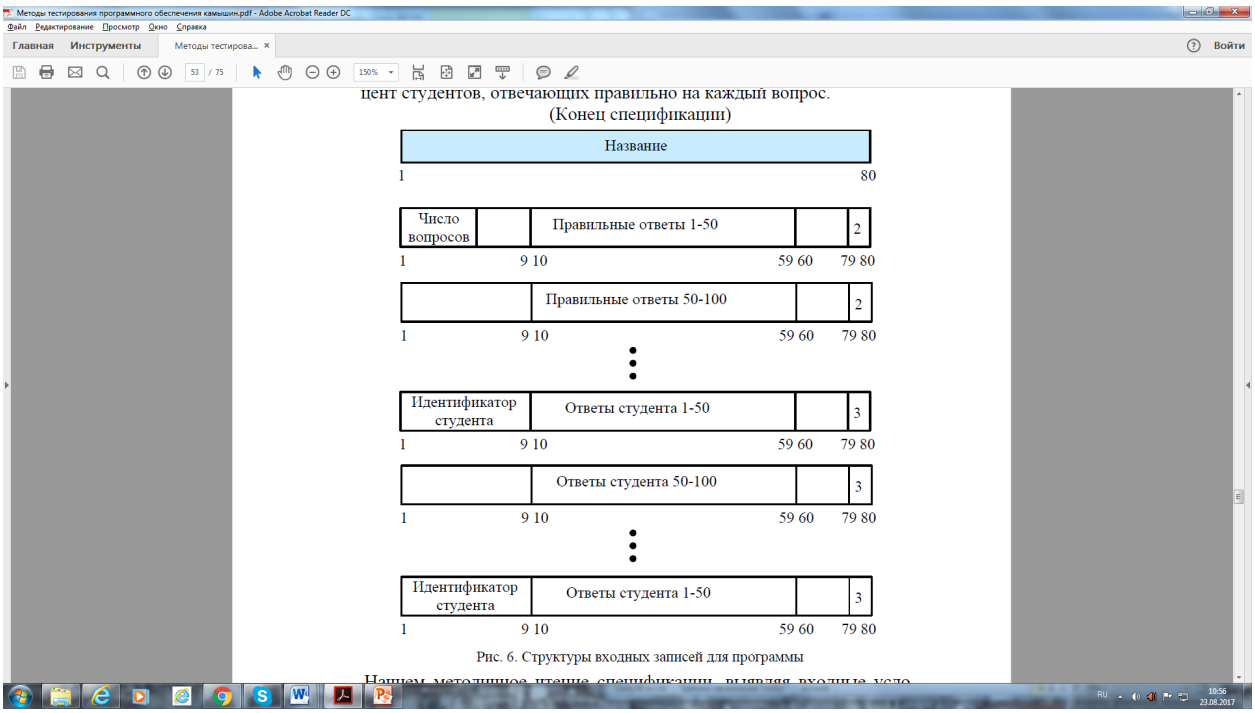


Рис. 6. Структуры входных записей для программы

Начнем методичное чтение спецификации, выявляя входные условия. Первое граничное входное условие есть пустой входной файл. Второе входное условие – карта (запись) названия; граничными условиями являются отсутствие карты названия, самое короткое и самое длинное названия. Следующими входными условиями служат наличие записей о правильных ответах и наличие поля числа вопросов в первой записи ответов. 1–999 не является классом эквивалентности для числа вопросов, так как для каждого подмножества из 50 записей может иметь место что-либо специфическое (т. е. необходимо много записей). Приемлемое разбиение вопросов на классы эквивалентности представляет разбиение на два подмножества: 1–50 и 51–999. Следовательно, необходимы тесты, где поле числа вопросов принимает значения 0, 1, 50, 51 и 999. Эти тесты покрывают большинство граничных условий для записей о правильных от- ветах; однако существуют три более интересные ситуации – отсутствие записей об ответах, наличие записей об ответах типа «много ответов на один вопрос» и наличие записей об ответах типа «мало ответов на один вопрос» (например, число вопросов – 50, и имеются три записи об ответах в первом случае и одна запись об ответах во втором). Таким образом, определены следующие тесты:

1. Пустой входной файл.
2. Отсутствует запись названия.
3. Название длиной в один символ.
4. Название длиной в 80 символов.
5. Экзамен из одного вопроса.
6. Экзамен из 50 вопросов.
7. Экзамен из 51 вопроса.
8. Экзамен из 999 вопросов.
9. 0 вопросов на экзамене.
10. Поле числа вопросов имеет нечисловые значения.
11. После записи названия нет записей о правильных ответах.
12. Имеются записи типа «много правильных ответов на один вопрос».

13. Имеются записи типа «мало правильных ответов на один вопрос». Следующие входные условия относятся к ответам студентов. Тестами граничных значений в этом случае, по-видимому, должны быть:
14. 0 студентов.
15. 1 студент.
16. 200 студентов.
17. 201 студент.
18. Есть одна запись об ответе студента, но существуют две записи о правильных ответах.
19. Запись об ответе вышеупомянутого студента первая в файле.
20. Запись об ответе вышеупомянутого студента последняя в файле.
21. Есть две записи об ответах студента, но существует только одна запись о правильном ответе.
22. Запись об ответах вышеупомянутого студента первая в файле.
23. Запись об ответах вышеупомянутого студента последняя в файле.
Можно также получить набор тестов для проверки выходных границ, хотя некоторые из выходных границ (например, пустой отчет 1) покрываются приведенными тестами. Граничными условиями для отчетов 1 и 2 являются:
0 студентов (так же, как тест 14);
1 студент (так же, как тест 15);
200 студентов (так же, как тест 16).
24. Оценки качества ответов всех студентов одинаковы.
25. Оценки качества ответов всех студентов различны.
26. Оценки качества ответов некоторых, но не всех студентов одинаковы (для проверки правильности вычисления рангов).
27. Студент получает оценку качества ответа 0.
28. Студент получает оценку качества ответа 100.
29. Студент имеет идентификатор наименьшей возможной длины (для проверки правильности упорядочения).
30. Студент имеет идентификатор наибольшей возможной длины.
31. Число студентов таково, что отчет имеет размер, несколько больший одной страницы (для того чтобы посмотреть случай печати на другой странице).
32. Число студентов таково, что отчет располагается на одной странице.
Граничные условия отчета 3 (среднее значение, медиана, среднеквадратическое отклонение).
33. Среднее значение максимально (качество ответов всех студентов наивысшее).
34. Среднее значение равно 0 (качество ответов всех студентов равно 0).
35. Среднеквадратическое отклонение равно своему максимуму (один студент получает оценку 0, а другой – 100).
36. Среднеквадратическое отклонение равно 0 (все студенты получают одну и ту же оценку).
Тесты 33 и 34 покрывают и границы медианы. Другой полезный тест описывает ситуацию, где существует 0 студентов (проверка деления на 0 при вычислении математического ожидания), но он идентичен тесту 14.
Проверка отчета 4 дает следующие тесты граничных значений:
37. Все студенты отвечают правильно на первый вопрос.
38. Все студенты неправильно отвечают на первый вопрос.

39. Все студенты правильно отвечают на последний вопрос.
40. Все студенты отвечают на последний вопрос неправильно.
41. Число вопросов таково, что размер отчета несколько больше одной страницы.
42. Число вопросов таково, что отчет располагается на одной странице.

Опытный тестировщик, вероятно, согласится с той точкой зрения, что многие из этих 42 тестов позволяют выявить наличие общих ошибок, которые могут быть сделаны при разработке данной программы. Кроме того, большинство этих ошибок, вероятно, не было бы обнаружено, если бы использовался метод случайной генерации тестов или специальный метод генерации тестов. Анализ граничных значений, если он применен правильно, является одним из наиболее полезных методов проектирования тестов. Однако он часто оказывается неэффективным из-за того, что внешне выглядит простым. Необходимо понимать, что граничные условия могут быть едва уловимы и, следовательно, определение их связано с большими трудностями.__ *3.2.3. Применение функциональных диаграмм*

Одним из недостатков анализа граничных значений и эквивалентного разбиения является то, что они не исследуют *комбинаций* входных условий. Например, пусть программа из приведенного выше примера не выполняется, если произведение числа вопросов и числа студентов превышает некоторый предел (например, объем памяти). Такая ошибка не обязательно будет обнаружена тестированием граничных значений. Тестирование комбинаций входных условий – непростая задача, поскольку даже при построенном эквивалентном разбиении входных условий число комбинаций обычно астрономически велико. Если нет систематического способа выбора подмножества входных условий, то, как правило, выбирается произвольное подмножество, приводящее к неэффективному тесту.

Метод функциональных диаграмм или диаграмм причинно-следственных связей [1] помогает систематически выбирать высокорезультативные тесты. Он дает полезный побочный эффект, так как позволяет обнаруживать неполноту и неоднозначность исходных спецификаций. Функциональная диаграмма представляет собой формальный язык, на который транслируется спецификация, написанная на естественном языке. Диаграмме можно сопоставить цифровую логическую цепь (комбинаторную логическую сеть), но для ее описания используется более простая нотация (форма записи), чем обычная форма записи, принятая в электронике. Для уяснения метода функциональных диаграмм вовсе не обязательно знание электроники, но желательно понимание булевской логики (т. е. логических операторов *и, или* и *не*). Построение тестов этим методом осуществляется в несколько этапов.

1. Спецификация разбивается на «рабочие» участки. Это связано с тем, что функциональные диаграммы становятся слишком громоздкими при применении данного метода к большим спецификациям. Например, когда тестируется система разделения времени, рабочим участком может быть спецификация отдельной команды. При тестировании компилятора в качестве рабочего участка можно рассматривать каждый отдельный оператор языка программирования.
2. В спецификации определяются причины и следствия. *Причина* есть отдельное входное условие или класс эквивалентности входных ус-

ловий. *Следствие* есть выходное условие или преобразование системы (остаточное действие, которое входное условие оказывает на состояние программы или системы). Например, если сообщение программы приводит к обновлению основного файла, то изменение в нем и является преобразованием системы; подтверждающее сообщение было бы выходным условием. Причины и следствия определяют- ся путем последовательного (слово за словом) чтения спецификации.
При этом выделяются слова или фразы, которые описывают причины и следствия. Каждым причине и следствию приписывается отдельный номер.
3. Анализируется семантическое содержание спецификации, которая преобразуется в булевский граф, связывающий причины и следствия. Это и есть функциональная диаграмма.
4. Диаграмма снабжается примечаниями, задающими ограничения и описывающими комбинации причин и (или) следствий, которые являются невозможными из-за синтаксических или внешних ограничений.
5. Путем методического прослеживания состояний условий диаграммы она преобразуется в таблицу решений с ограниченными входами. Каждый столбец таблицы решений соответствует тесту.
6. Столбцы таблицы решений преобразуются в тесты.
Базовые символы для записи функциональных диаграмм показаны на рис. 7. Каждый узел диаграммы может находиться в двух состояниях – 0 или 1; 0 обозначает состояние «отсутствует», а 1 – «присутствует». Функция *тождество* устанавливает, что если значение *a* есть 1, то и значение *b* есть 1; в противном случае значение *b* есть 0. Функция *не* устанавливает, что если *a* есть 1, то *b* есть 0; в противном случае *b* есть 1. Функция *или* устанавливает, что если *a,* или *b,* или *c* есть 1, то *d* есть 1; в противном случае *d* есть 0. Функция *и* устанавливает, что если и *a*, и *b* есть 1, то и *c* есть 1; в противном случае *c* есть 0. Последние две функции разрешают иметь любое число входов.
Для иллюстрации изложенного рассмотрим диаграмму, отображающую спецификацию: символ в колонке 1 должен быть буквой «A» или «B», а в колонке 2 – цифрой. В этом случае файл обновляется. Если первый символ неправильный, то выдается сообщение X12, а если второй символ неправильный – сообщение X13.

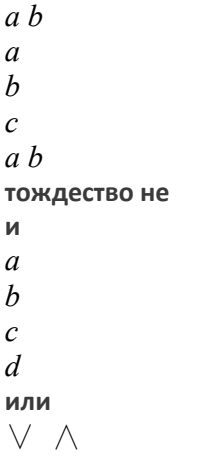
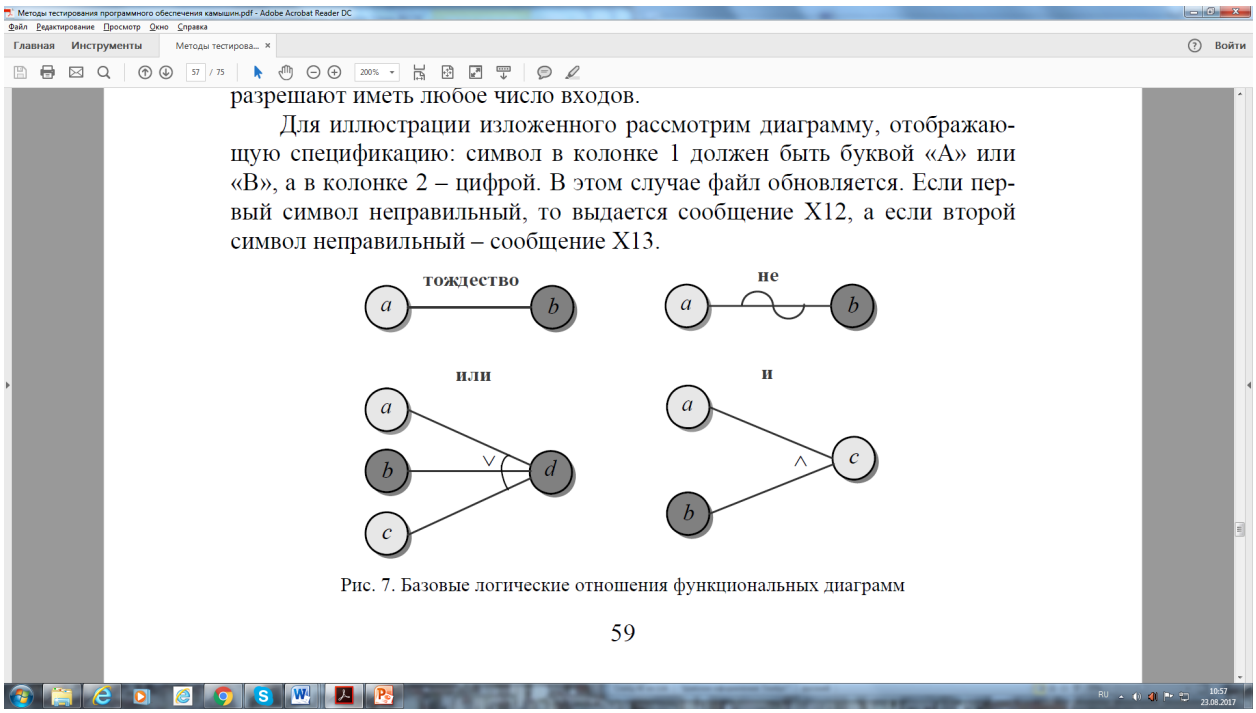


Рис. 7. Базовые логические отношения функциональных диаграмм


разрешают иметь любое число входов.

Для иллюстрации изложенного рассмотрим диаграмму, отображающую спецификацию: символ в колонке 1 должен быть буквой «A» или «B», а в колонке 2 – цифрой. В этом случае файл обновляется. Если первый символ неправильный, то выдается сообщение X12, а если второй символ неправильный – сообщение X13.

Рис. 7. Базовые логические отношения функциональных диаграмм

59

Причинами являются: 1 – символ «A» в колонке 1; 2 – символ «B» в колонке 1; 3 – цифра в колонке 2; а **следствиями:** 70 – файл обновляется; 71 – выдается сообщение X12; 72 – выдается сообщение X13. Функциональная диаграмма показана на рис. 8. Отметим, что здесь создан промежуточный узел 11. Следует убедиться в том, что диаграмма действительно отображает данную спецификацию, задавая причинам все возможные значения и проверяя, принимают ли при этом следствия правильные значения. Рядом показана эквивалентная логическая схема. Хотя диаграмма, показанная на рис. 8, отображает спецификацию, она содержит невозможную комбинацию причин – причины 1 и 2 не могут быть установлены в 1 одновременно. В большинстве программ определенные комбинации причин невозможны из-за синтаксических или внешних ограничений (например, символ не может принимать значения «A» и «B» одновременно).

3
1
2
11
71
70
∨
∧
72
71
70
72
3
1
2

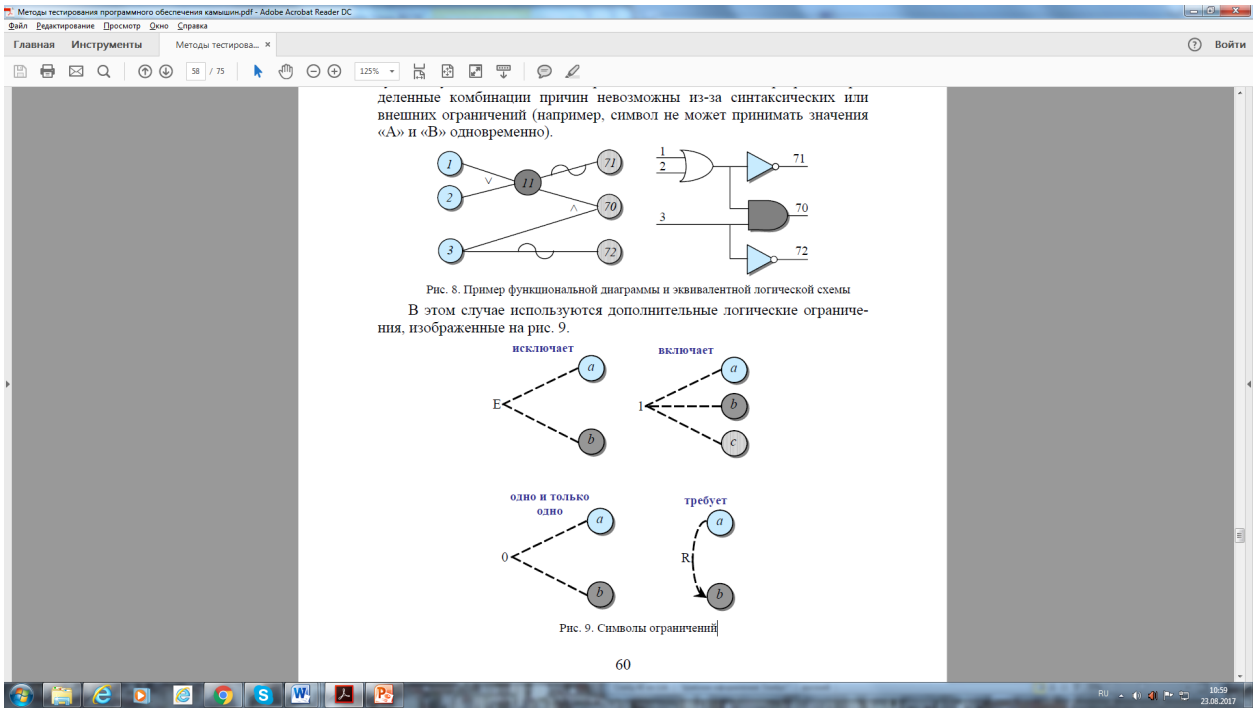
Рис. 8. Пример функциональной диаграммы и эквивалентной логической схемы

В этом случае используются дополнительные логические ограничения, изображенные на рис. 9.

a
b

c
включает
1
a
b
исключает
E
a
b
одно и только
одно
0
a
b
R
требует

Рис. 9. Символы ограничений



хотя бы одна из величин – a или b – принимает значение 1 (a и b не могут принимать значение 1 одновременно). Ограничение I устанавливает, что, по крайней мере, одна из величин a, b или c всегда должна быть равной 1 (a, b и c не могут принимать значение 0 одновременно). Ограничение 0 устанавливает, что одна и только одна из величин a или b должна быть равна 1. Ограничение R устанавливает, что если a принимает значение 1, то и b должна принимать значение 1 (т. е. невозможно, чтобы a была равна 1, а b – 0).

Часто возникает необходимость в ограничениях для следствий. Ограничение M на рис. 10 устанавливает, что если следствие a имеет значение 1, то следствие b должно принять значение 0.

a
b
M скрывает

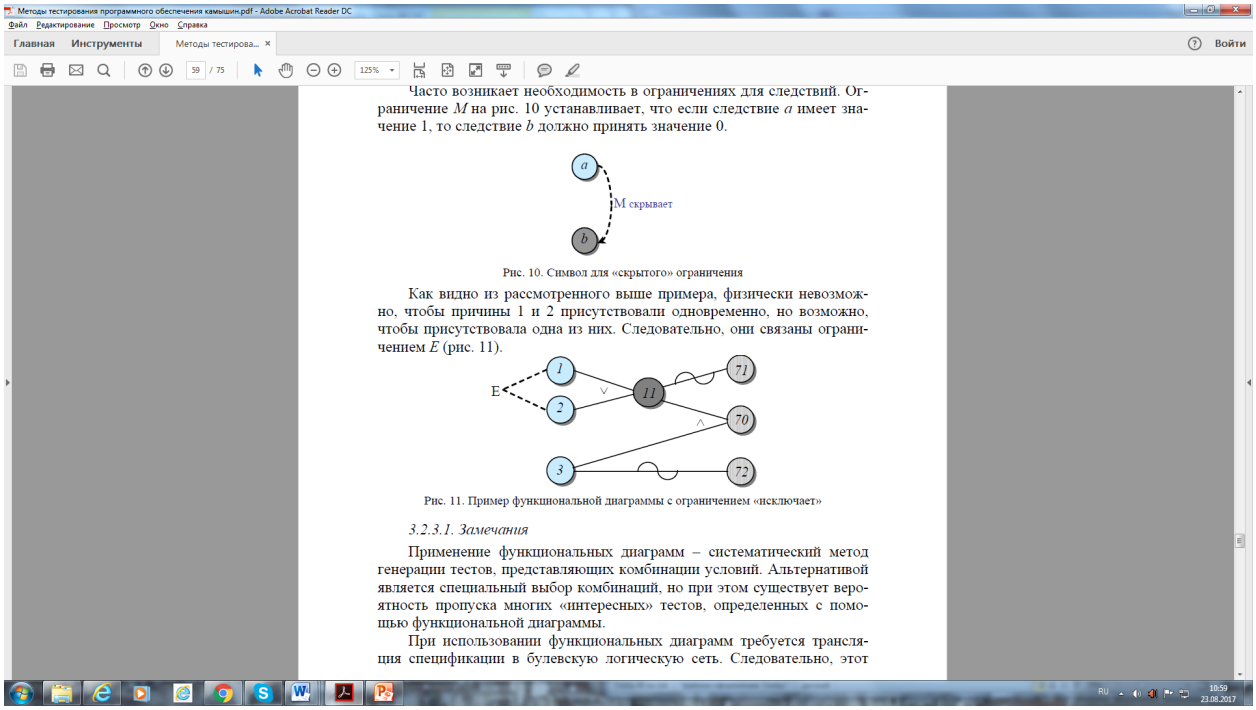
Рис. 10. Символ для «скрытого» ограничения

Как видно из рассмотренного выше примера, физически невозможно, чтобы причины 1 и 2 присутствовали одновременно, но возможно,

чтобы присутствовала одна из них. Следовательно, они связаны ограничением E (рис. 11).

3
1
2
11
71
70
∨
∧
72
E

Рис. 11. Пример функциональной диаграммы с ограничением «исключает»



3.2.3.1. Замечания

Применение функциональных диаграмм – систематический метод генерации тестов, представляющих комбинации условий. Альтернативой является специальный выбор комбинаций, но при этом существует вероятность пропуска многих «интересных» тестов, определенных с помощью функциональной диаграммы.

При использовании функциональных диаграмм требуется трансляция спецификации в булевскую логическую сеть. Следовательно, этот метод открывает перспективы ее применения и дополнительные возможности спецификаций. Действительно, разработка функциональных диаграмм есть хороший способ обнаружения неполноты и неоднозначности в исходных спецификациях.

Метод функциональных диаграмм позволяет построить набор полезных тестов, однако его применение обычно не обеспечивает построение *всех* полезных тестов, которые могут быть определены. Кроме того, функциональная диаграмма неадекватно исследует граничные условия. Конечно, в процессе работы с функциональными диаграммами можно попробовать покрыть граничные условия. Однако при этом граф сущест-

венно усложняется, и число тестов становится чрезвычайно большим. Поэтому лучше отделить анализ граничных значений от метода функциональных диаграмм.

Поскольку функциональная диаграмма дает только направление в выборе определенных значений операндов, граничные условия могут входить в полученные из нее тесты.

Наиболее трудным при реализации метода является преобразование диаграммы в таблицу решений. Это преобразование представляет собой алгоритмический процесс. Следовательно, его можно автоматизировать посредством написания соответствующей программы. Фирма IBM имеет ряд таких программ, но не поставляет их.