Использование конечных автоматов для тестирования программ.

И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин[1]

Институт Системного Программирования РАН

e-mail: {igor,kos,kuliamin}@ispras.ru

Рассматривается применение теории конечных автоматов к проблеме тестирования программ. Проблема сводится к тестированию конечного автомата. Описывается тестирование автоматов по графам состояний, фактор-графы, тестирование автоматов по фактор-графам и способы построения фактор-графов.

Введение

Автоматы являются широко используемой моделью аппаратных и программных объектов. Основное отличие автомата от чисто функциональной зависимости заключается в том, что значения выходных параметров зависят не только от значений входных параметров, но и от состояния объекта. Концепция автомата очень близка таким программным понятиям, как абстрактные типы данных и объекты классов, являясь их математической моделью. Например, в объектно-ориентированном программировании (ООП) объект класса можно рассматривать как автомат: состояние автомата — это состояние объекта; входной символ — операция в объекте с некоторым набором значений входных параметров; выходной символ — набор значений выходных параметров.

Абстракция автомата применяется как для проектирования, так и для анализа проектных решений. Настоящая статья посвящена проблемам, возникающим при тестировании программных систем, которые рассматриваются как конечные автоматы, при этом для краткости мы будем такое тестирование называть «тестированием конечного автомата».

Задача автоматизации тестирования распадается на две относительно независимые части: генерация тестовых воздействий и автоматическая проверка результатов тестового воздействия, что обычно выполняется программой-оракулом. В последнее время проблема создания оракулов обычно трактуется как проблема генерации оракулов по формальным спецификациям. Этой теме посвящено довольно много работ (см. [1], [3], [6], [7]) и здесь мы не будем на этом останавливаться. Для наших целей достаточно знать, что оракул автомата для любой допустимой пары <состояние, входной символ> может определить правильность перехода в новое состояние (правильность функции перехода) и правильность полученного выходного символа (правильность функции выхода). Задача тестирования конечного автомата может решаться в предположении, что состояние конечного автомата доступно или не доступно непосредственному наблюдению со стороны тестовой системы. Если состояние не доступно (автомат как «чёрный ящик»), то тестовая система должна

[1] Данная работа поддержана грантами РФФИ № 96-01-01277 и № 99-01-00207

вести некоторое «абстрактное состояние», моделирующее реальное состояние автомата. В такой ситуации оракул вместо проверки функции перехода в новое реальное состояние вычисляет новое абстрактное состояние. Соответствие реального и абстрактного состояний устанавливается косвенным путём при проверках в оракуле функции выхода в процессе тестирования (см. [3]).

Естественным критерием полноты тестового покрытия при тестировании автоматов является покрытие всех переходов автомата (допустимых пар <состояние, входной символ>). Для выполнения этого критерия необходимо генерировать все требуемые тестовые воздействия во всех состояниях автомата. Тем самым, задача генерации распадается на задачу «обхода» всех состояний автомата и задачу «перебора» тестовых воздействий для каждого состояния. Заметим, что для решения этих задач не используется функция выхода автомата.

Одним из распространённых способов представления автомата является представление его в виде графа состояний (или графа переходов), вершины которого соответствуют состояниям автомата, а дуги — допустимым переходам. В терминах теории графов задача покрытия всех переходов автомата формулируется как задача обхода графа, то есть, прохождения по маршруту, содержащему все дуги графа.

Существуют две основные проблемы, связанные с обходом графа состояний автомата: недетерминизм и слишком большой размер графа.

Недетерминированный автомат — это такой автомат, в котором функция перехода неоднозначна: одной паре <состояние, входной символ> соответствуют несколько дуг в графе. Поскольку выбор той или иной из этих дуг не детерминирован тестовым воздействием, во время тестирования невозможно гарантировать обход графа состояний (проход по всем дугам и, тем самым, возможно, по всем состояниям). Заметим, что неоднозначность функции выхода не создаёт дополнительных проблем: оракул требует только, чтобы выполнялся некоторый предикат от состояния, входного и выходного символов.

> **Замечание**: Недетерминизм моделирующего автомата не обязательно означает реальный недетерминизм моделируемого объекта. Во многих случаях недетерминизм модели является следствием естественного абстрагирования от деталей реализации. Например, при запросе памяти нас может не интересовать алгоритм распределения памяти; важно лишь, что выделяемый фрагмент памяти не должен пересекаться ни с одним из уже занятых фрагментов. Спецификация часто неоднозначно определяет результат выполнения операции, поскольку, как правило, описывает не алгоритм вычисления результата, а лишь требования, которым результат должен удовлетворять.

Слишком большой размер графа приводит, естественно, к слишком долгому его обходу (времени тестирования).

Существует общий подход к решению обеих указанных выше проблем, основанный на введении классов эквивалентности вершин и дуг графа. Критерий покрытия всех дуг и вершин ослабляется до критерия покрытия всех классов эквивалентности. На этих классах эквивалентности строится фактор-граф, который и обходится при тестировании. Гомоморфность (по отношению инцидентности вершин и дуг) фактор-графа исходному графу состояний автомата существенно используется в алгоритме тестирования. При подходя-

щем задании классов эквивалентности фактор-граф может стать детерминированным, а его размер — резко уменьшиться.

Примером разбиения множества дуг на классы эквивалентности может служить понятие операции в объекте ООП. Если на доменах операций задаются некоторые предикаты, разбивающие домены операций на поддомены, то мы получаем более «детальное» разбиение.

Заметим, что, эквивалентность по операциям может рассматриваться как эквивалентность входных символов: эквивалентны все *вызовы* одной операции с различными параметрами. Однако, поддомен операции — это в общем случае предикат на входных параметрах *и* состоянии объекта. Поэтому следует говорить об эквивалентности переходов автомата (дуг графа состояний), а не входных символах.

Мы будем рассматривать конечные автоматы с сильно связными графами состояний. Автоматы, соответствующие объектам ООП, обладают этими свойствами (сильная связность легко достигается добавлением операций конструкторов и деструкторов объектов). Такие автоматы обладают важным для тестирования свойством: после любого перехода остаётся возможность попасть в любое состояние и протестировать любой переход из него.

В следующих разделах мы рассмотрим графы состояний автоматов, тестирование автоматов по таким графам, фактор-графы, тестирование автоматов по фактор-графам и способы построения фактор-графов.

Граф состояний автомата

В дальнейшем изложении мы будем часто пользоваться следующими понятиями и обозначениями без дополнительных пояснений:

- эквивалентность на множестве **A** — рефлексивное, симметричное и транзитивное бинарное отношение $\Sigma \subseteq \mathbf{A}\times\mathbf{A}$; обозначается большой греческой буквой;
- эквивалентность Σ на множестве **A** индуцирует разбиение **A** на классы Σ-эквивалентности — Σ-классы; обозначается $\mathbf{A}/\Sigma$; обратно: любое разбиение **A** на непересекающиеся классы индуцирует соответствующее отношение эквивалентности;
- разбиение $\mathbf{A}/\Sigma$ индуцирует отображение, обозначаемое соответствующей малой греческой буквой, $\sigma:\mathbf{A}\to\mathbf{A}/\Sigma$; обратно: любое отображение индуцирует разбиение области определения и соответствующую эквивалентность на ней;
- вложенность $\subseteq$, пересечение $\cap$, дополнение $\neg$ эквивалентностей понимается в обычном теоретико-множественном смысле (как подмножеств декартового произведения).

Ориентированный граф $G=(V,E,\lambda,\rho)$ задаётся двумя непересекающимися множествами: множеством **вершин V** и множеством **дуг E**, и двумя **функциями инцидентности** $\lambda:\mathbf{E}\to\mathbf{V}$ и $\rho:\mathbf{E}\to\mathbf{V}$, определяющими для каждой дуги, соответственно, её начальную вершину (**начало**) и её конечную вершину (**конец**). Граф конечен, если множества **V** и **E** конечны.

Функции инцидентности λ и ρ определяют **отношение смежности дуг** Ω:

$\forall e_1,e_2 \in E \quad e_1 \Omega e_2 \equiv \rho(e_1)=\lambda(e_2)$.

Будем говорить, что на графе $G=(V,E,\lambda,\rho)$ задана **раскраска** (X,χ), если задано некоторое множество X, которое будем называть **алфавитом** раскраски, и отображение χ дуг графа на это множество, то есть, $\chi: E \to X$. Раскраску будем называть **правильной**, если кратные дуги отображаются в разные элементы (символы) из X:

$\forall e_1,e_2 \in E \quad \lambda(e_1) = \lambda(e_2)\ \&\ \rho(e_1) = \rho(e_2) \Rightarrow \chi(e_1) \neq \chi(e_2)$.

В правильно раскрашенном графе любая дуга e однозначно определяется тройкой $(x,v,v`)$, где $x=\chi(e)$, $v=\lambda(e)$, $v`=\rho(e)$.

Граф с произвольной заданной эквивалентностью дуг Σ будем называть **Σ-детерминированным**, если все дуги, выходящие из одной вершины, Σ-неэквивалентны:

$\forall e_1,e_2 \in E \quad \lambda(e_1) = \lambda(e_2) \Rightarrow e_1 \neg\Sigma\ e_2$.

Очевидно, что отображение $\sigma: E \to E/\Sigma$ определяет правильную раскраску с алфавитом E/Σ.

Маршрутом P называется последовательность смежных дуг графа $e_0,...,e_t$ такая, что $e_{i-1}\ \Omega\ e_i$ при $1 \le i \le t$. Если на графе определена правильная раскраска (X,χ), то маршрут можно задавать как последовательность троек $(x_0,v_0,v_0`),...,(x_t,v_t,v_t`)$, где $x_i=\chi(e_i)$, $v_i=\lambda(e_i)$, $v_i`=\rho(e_i)$ для каждого $0 \le i \le t$ и $e_{i-1}\ \Omega\ e_i$ для каждого $1 \le i \le t$. Маршрут P X-детерминированного графа может быть задан начальной вершиной v_0 и последовательностью символов алфавита (словом в алфавите X) $x_0,..,x_t$: $P=(v_0,x_0,..,x_t)$. **Обходом** ориентированного графа называется маршрут, содержащий все дуги графа. Для сильно связных конечных графов обход всегда существует и может начинаться с любой вершины.

Автомат $A=(X,V,Y,\varphi,\psi,v_0)$ задаётся как совокупность шести объектов:

- множества X, называемого входным алфавитом;
- множества V, называемого множеством состояний автомата;
- множества Y, называемого выходным алфавитом;
- соответствия $\varphi \subseteq (X \times V) \times V$, называемого функцией переходов;
- соответствия $\psi \subseteq (X \times V) \times Y$, имеющего ту же область определения, что φ ($\mathrm{Dom}\,\psi = \mathrm{Dom}\,\varphi$), называемого функцией выходов;
- состояния $v_0 \in V$, называемого начальным состоянием.

Автомат **конечен**, если множества X, V, Y конечны.

Автомат называется **детерминированным**, если функция переходов однозначна:
$\forall (x,v) \in \mathrm{Dom}\,\varphi\ \exists!\ v` \in V \quad \varphi((x,v),v`)$.
В этом случае мы будем записывать $v`=\varphi(x,v)$.

> **Замечание**: Часто детерминированность автомата понимают как однозначность обеих функций: функции переходов и функции выходов. Однако, для наших целей

обхода графа состояний нам достаточно однозначности функции переходов.

Автомату $\mathbf{A=(X,V,Y,\varphi,\psi,v_0)}$ соответствует граф его состояний $\mathbf{G=(V,E,\lambda,\rho)}$ с правильной раскраской $\mathbf{(X,\chi)}$:

- $\mathbf{E = \{ (x,v,v`) \mid x \in X \ \& \ v \in V \ \& \ v` \in V \ \& \ \varphi((x,v),v`) \}}$
- $\mathbf{\lambda((x,v,v`)) = v}$
- $\mathbf{\rho((x,v,v`)) = v`}$
- $\mathbf{\chi((x,v,v`)) = x}$

χ-эквивалентные дуги, имеющие общее начало, назовём Δ-эквивалентными. Δ-класс, в отличие от дуги **(x,v,v`)**, однозначно определяется парой **(x,v)**. Если автомат детерминирован, то граф его состояний χ-детерминирован и, очевидно, Δ-детерминирован, причём каждая дуга Δ-эквивалентна только самой себе (Δ-класс состоит из одной дуги).

Тестирование автомата по графу его состояний

Тестирование детерминированного автомата **A** на основе обхода $\mathbf{P=(v_0,x_0,..,x_t)}$ графа его состояний проводится по следующему алгоритму $\mathbb{A}\mathbf{(A,P)}$:

1. На **i**-м шагу алгоритма (вначале **i=0**) мы находимся в состоянии $\mathbf{v_i}$ и нам нужно пройти по дуге $\mathbf{(x_i,v_i,v_{i+1})}$. Поскольку автомат детерминирован, такая дуга однозначно определяется парой $\mathbf{(x_i,v_i)}$.
2. Подаём на вход автомата символ $\mathbf{x_i}$ и применяем оракул автомата.
3. Если оракул выдаёт отрицательный вердикт, тестирование заканчивается по обнаруженной ошибке. Иначе, получаем от оракула новое состояние $\mathbf{v_{i+1}}$.
4. Если **i<t**, то увеличиваем **i** на 1 и переходим к п. 1. Иначе тестирование считается нормально законченным.

Время тестирования определяется длиной **t** обхода **P** графа. Длина оптимального обхода для графов, содержащих **n** вершин и **m** дуг, имеет, как известно, порядок **nm**.

Гомоморфный граф

Гомоморфизм графов и фактор-граф

Гомоморфизмом $\mathbf{(\xi,\pi)}$ графа **G** на граф **G*** называется пара сюръективных отображений вершин ξ и дуг π, сохраняющих функции инцидентности λ и ρ:

$\mathbf{\forall e \in E \quad \xi(\lambda(e)) = \lambda(\pi(e)) \ \& \ \xi(\rho(e)) = \rho(\pi(e))}$.

Гомоморфизм $\mathbf{(\xi,\pi)}$ графа **G** на граф **G*** индуцирует конгруэнцию $\mathbf{(\Xi,\Pi)}$ на **G**, которая, в свою очередь, определяет фактор-граф $\mathbf{G/(\Xi,\Pi)}$, изоморфный **G***, и естественный гомоморфизм **G** на $\mathbf{G/(\Xi,\Pi)}$. Поэтому в дальнейшем для удобства изложения мы будем рассматривать, как правило, гомоморфизм графа на фактор-граф, сохраняя те же обозначе-

ния: **G***, ξ, π. Однако, в алгоритме тестирования используется гомоморфизм **G** на любой **G***, а не обязательно фактор-граф.

Раскраска **(X*,χ*)** на фактор-графе **G*** индуцирует раскраску **(X*,θ)** на графе **G**, где $\theta=\chi^*\pi$. Если раскраска **(X*,χ*)** правильная, то для каждой дуги **e**$\in$**E** тройка **(θ(e), ξ(λ(e)), ξ(ρ(e)))** однозначно определяет фактор-дугу **e***=π**(e)**. Алфавит **X*** будем называть обобщённым алфавитом, чтобы отличить от алфавита **X**, символами которого раскрашены дуги графа состояний **G**. Отметим, что обобщённый алфавит **X***, вообще говоря, не является фактор-алфавитом, то есть, разбиением алфавита **X**.

Обратно: пусть на графе **G** заданы эквивалентность вершин Ξ и раскраска дуг **(X*,θ)**, индуцирующая соответствующую эквивалентность дуг Θ. Эквивалентность Ξ индуцирует естественную эквивалентность дуг $\Xi^{\sim}$: две дуги $\Xi^{\sim}$-эквивалентны, если их начала Ξ-эквивалентны и их концы Ξ-эквивалентны. Пересечение эквивалентностей $\Theta^{\sim}=\Theta\cap\Xi^{\sim}$ как раз и порождает разбиение дуг на фактор-дуги: **E***=**E**/$\Theta^{\sim}$, то есть, $\Pi=\Theta^{\sim}$. Мы будем говорить о гомоморфизме **(ξ,θ)**, имея в виду индуцируемый гомоморфизм **(ξ,$\theta^{\sim}$)**.

Если на графе **G=(V,E,λ,ρ)** заданы только эквивалентности вершин Ξ и дуг Θ, фактор-граф **G*=(V,E,λ,ρ,** Ξ,Θ**)=(V*,E*,λ,ρ)** можно определить независимым образом:

- фактор-вершина **v***$\in$**V*** — это множество Ξ-эквивалентных вершин;
- фактор-дуга **e***$\in$**E*** — это множество Θ-эквивалентных дуг, начала которых Ξ-эквивалентны и концы которых тоже Ξ-эквивалентны;
- если **e*** — фактор-дуга, то λ**(e*)** есть та фактор-вершина, которой принадлежат начала всех дуг из **e***;
- если **e*** — фактор-дуга, то ρ**(e*)** есть та фактор-вершина, которой принадлежат концы всех дуг из **e***.

Для **X*=X/**Θ раскраска **(X*,θ)** индуцирует правильную раскраску фактор-графа **(X*,θ*)**: θ**(e) = x*** $\Rightarrow$ $\theta^*(\theta^{\sim}$**(e)) = x***.

> **<u>Замечание</u>**: При подходящем задании раскраски фактор-дуг обобщёнными *выходными* символами фактор-графу можно поставить в соответствие обобщённый автомат. В некотором смысле тестирование исходного автомата по фактор-графу можно рассматривать как тестирование соответствующего обобщённого автомата.

Пример фактор-графа

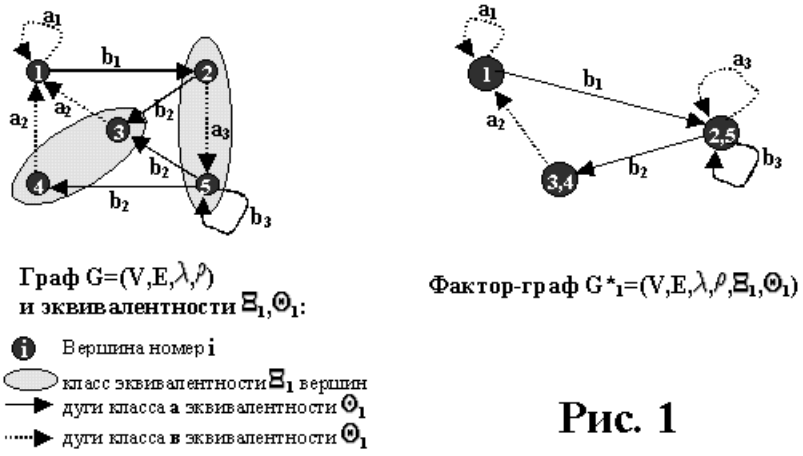


Рис. 1

Детерминированный фактор-граф

Для произвольной эквивалентности Σ на множестве дуг графа **G** фактор-граф **G*=(V,E,λ,ρ,Ξ,Θ)** назовём **Σ-детерминированным**, если Σ-эквивалентные дуги, имеющие Ξ-эквивалентные начала, принадлежат одной фактор-дуге, то есть, Θ-эквивалентны и имеют Ξ-эквивалентные концы. Соответствующий гомоморфизм графов будем называть **Σ-**детерминированным.

Нас будут интересовать Δ-детерминированность и Θ-детерминированность фактор-графов. Заметим, что Θ-детерминированность фактор-графа совпадает с его Θ*-детерминированностью как графа, то есть, образ Θ-детерминированного гомоморфизма Θ*-детерминирован.

Критерием Δ-детерминированности является условие $\Delta \subseteq \Theta^{\sim}$. Из $\Delta \subseteq \Theta^{\sim}$ и $\Theta^{\sim} \subseteq \Theta$ следует $\Delta \subseteq \Theta$. Заметим, что из Δ-детерминированности не следует χ-детерминированность.

Критерием Θ-детерминированности является условие:

$\forall e_1,e_2 \in E \;\; \lambda(e_1)\,\Xi\,\lambda(e_2) \;\; \& \;\; \rho(e_1) \neg\Xi\, \rho(e_2) \;\; \Rightarrow \;\; e_1 \neg\Theta\, e_2$.

В Θ-детерминированном фактор-графе фактор-дуга **(θ*(e*),λ(e*),ρ(e*))** однозначно определяется парой **(θ*(e*),λ(e*))**.

Например, на рис. 1 в исходном графе G_1 две дуги b_2, ведущие из вершины ❺, Δ-эквивалентны (эти дуги соответствуют одному и тому же символу b_2), и граф G_1 Δ-недетерминирован. Однако, фактор-граф G_1* Δ-детерминирован, хотя и Θ_1-недетерминирован, поскольку две Θ_1-эквивалентные дуги b_2 и b_3 имеют Ξ_1-эквивалентные начала (фактор-вершина ❷,❺), но Ξ_1-неэквивалентные концы (❸,❹ и ❷,❺).

Просто **детерминированным** фактор-графом будем называть фактор-граф, который одновременно Δ-детерминирован и Θ-детерминирован. Соответствующий гомоморфизм графов будем называть детерминированным гомоморфизмом.

Вполне определённый фактор-граф

Поскольку фактор-граф **G*** является гомоморфным образом исходного графа **G**, любой маршрут **P** в графе **G** отображается на некоторый маршрут **P*** в фактор-графе **G***. Если **P** содержит, быть может, не все вершины и дуги, но проходит по всем классам эквивалентностей Ξ и $\tilde{\Theta}$ вершин и дуг, то **P*** является обходом фактор-графа **G***. Именно маршрут **P** и будет реально проходиться при тестировании, а завершение обхода **P*** является критерием полноты тестирования. Поскольку фактор-граф **G*** содержит меньше вершин и дуг, чем граф **G**, длина обхода **P*** (и, значит, длина **P**) меньше длины обхода графа **G**, то есть, время тестирования по фактор-графу меньше, чем по исходному.

Однако, не любой маршрут **P*** фактор-графа **G*** обязан быть образом маршрута графа **G**. Здесь возникает **проблема смежности дуг**: паре смежных фактор-дуг $\mathbf{e_1^*}$ и $\mathbf{e_2^*}$, соседних в обходе **P***, могут соответствовать в графе **G** такие пары дуг $\mathbf{e_1 \in e_1^*}$ и $\mathbf{e_2 \in e_2^*}$, которые не являются смежными.

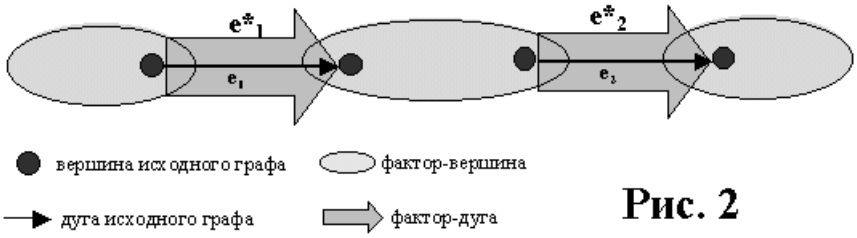


Рис. 2

Существует несколько подходов к решению этой проблемы.

Один подход заключается в том, чтобы каждый раз, когда нужно последовательно пройти по несмежным дугам $\mathbf{e_1}$ и $\mathbf{e_2}$, используется путь в графе **G**, ведущий из конца дуги $\mathbf{e_1}$ в начало дуги $\mathbf{e_2}$ (см. [4]). При этом, конечно, проходится некоторый циклический путь в фактор-графе **G***. Тем самым, вместо обхода **P*** мы фактически выполняем более длинный обход $\mathbf{P_1^*}$, который уже является гомоморфным образом **P**. Этот подход применим для любого фактор-графа (если исходный граф сильно связен). Однако, длина $\mathbf{P_1^*}$ (оценка времени тестирования) может оказаться настолько больше длины **P***, что она приблизится к длине обхода исходного графа **G**, и вся выгода от использования фактор-графа вместо исходного будет потеряна.

Другой подход заключается в том, чтобы рассматривать только такие фактор-графы, для которых выполняется требование **жёсткой смежности дуг**: если фактор-дуги $\mathbf{e_1^*}$ и $\mathbf{e_2^*}$ смежны и $\mathbf{e_1 \in e_1^*}$, то существует $\mathbf{e_2 \in e_2^*}$ такая, что $\mathbf{e_1}$ и $\mathbf{e_2}$ тоже смежны. В этом случае каждый обход **P*** фактор-графа является образом хотя бы одного маршрута **P** в исходном графе. Тем самым время тестирования определяется длиной **P***, которая может быть существенно меньше длины обхода исходного графа. Именно этот подход рассматривается в данной статье.

Фактор-дугу **e*** назовём **вполне определённой**, если множество вершин, являющихся началами дуг $\mathbf{e \in e^*}$ совпадает с фактор-вершиной, являющейся началом **e***: $\{\lambda(\mathbf{e}) \mid \mathbf{e \in e^*}\} = \lambda(\mathbf{e^*})$. Фактор-граф вполне определён, а Θ — вполне определённая эквивалентность дуг, если все фактор-дуги вполне определены. Соответствующий гомоморфизм будем называть вполне определённым гомоморфизмом. Для сильно связного графа **G** жёсткая смежность дуг фактор-графа **G*** эквивалентна его вполне определённости.

Например, для фактор-графа $\mathbf{G_1^*}$ на рис. 2 фактор-дуги $\mathbf{a_1}$, $\mathbf{b_1}$, $\mathbf{a_2}$, $\mathbf{b_2}$ являются вполне определёнными, а фактор-дуги $\mathbf{a_3}$ и $\mathbf{b_3}$ — не вполне определённые.

В терминах гомоморфизмов графов жёсткую смежность дуг и вполне определённость можно понимать следующим образом.

Гомоморфизм (алгебраических систем) τ: $\mathbf{A \to A^*}$ назовём **жёстким** (слева) по отношению Σ, если:

$$\forall x \in A \; \forall y^* \in A^* \quad \tau(x) \, \Sigma y^* \quad \Rightarrow \quad \exists y \in A \quad \tau(y)=y^* \;\&\; x \, \Sigma \, y.$$

Жёсткая смежность дуг означает жёсткость гомоморфизма графов по отношению Ω смежности дуг, а вполне определённость означает жёсткость гомоморфизма графов по функции инцидентности $v=\lambda(e)$ (рассматриваемой как отношение $v \; \lambda \; e$). Заметим, что, если инцидентность λ рассматривать как отношение $e \; \lambda \; v$, то соответствующее свойство выполняется при любом гомоморфизме графов.

Тестирование автомата по гомоморфному графу

Гомоморфизм графов и функция вычисления символа

Пусть задан детерминированный вполне определённый гомоморфизм (ξ,θ) графа **G** состояний автомата **A** на граф **G***. Из вполне определённости следует, что любой обход **P*** графа **G*** является образом некоторого маршрута **P** в исходном графе **G**. Из Θ-детерминизма следует, что обход **P*** может быть задан начальной вершиной v^*_0 графа **G*** и последовательностью обобщённых символов (обобщённое слово) $x^*_0,..,x^*_t$. В сильно связном графе обход может начинаться с любой вершины; выберем в качестве начальной вершины обхода **P*** образ начального состояния v_0 автомата **A**, то есть, $\xi(v_0)$. Тогда обход может быть задан как $P^*=(v_0,x^*_0,..,x^*_t)$.

В алгоритме тестирования используется **функция вычисления символа**, которая для дуги **(x*,v*,v*`)** графа **G\*** по обобщённому символу **x\*** и состоянию автомата **v** из прообраза **v\*** вычисляет (входной) символ **x** автомата так, что любая дуга **(x,v,v`)** гарантированно принадлежит прообразу **(x*,v*,v*`)**. Для этого нужно:

- Во-первых, чтобы пара $(x^*,\xi(v))$ определяла одну дугу **(x*,v*,v*`)**, что гарантируется Θ-детерминированностью гомоморфизма.
- Во-вторых, чтобы для любого **v** из прообраза **v*** существовала дуга **(x,v,v`)**, отображаемая в $(x^*,\xi(v))$, что гарантируется вполне определённостью гомоморфизма.
- В-третьих, чтобы весь Δ-класс **(x,v)**, которому принадлежит такая дуга **(x,v,v`)**, отображался в одну дугу $(x^*,\xi(v))$, что гарантируется Δ-детерминированностью гомоморфизма.

При этих условиях функция вычисления символа находит хотя бы одно решение уравнения $\theta(x,v)= x^*$ относительно **x** .

> **Замечание**: Функцию вычисления символа можно реализовать как функцию перебора символов **x** с проверкой $\theta(x,v)=x^*$.

Таким образом, алгоритм тестирования использует, фактически, только заданные отображения ξ и θ, то есть, гомоморфизм исходного графа **G** на граф **G***, и заданный обход **P*** графа **G***.

Алгоритм тестирования

Пусть для автомата **A** задан детерминированный и вполне определённый гомоморфизм (ξ,θ) графа состояний $G=(V,E,\lambda,\rho)$ на граф $G^*=(V^*,E^*,\lambda,\rho)$. Тестирование автомата на основе обхода $P^*=(v_0,x^*_0,..,x^*_t)$ графа **G*** проводится по следующему алгоритму $\mathbb{A}^*(A,P^*)$:

1. На **i**-м шагу алгоритма (в начале **i=0**) мы находимся в состоянии v_i и нам нужно пройти по дуге **e*** графа **G***, однозначно определяемой парой $(x^*_i,\xi(v_i))$. С помощью функции вычисления символа определяем x_i, являющееся решением уравнения $\theta(x_i,v_i)=x^*_i$.

2. Подаём на вход автомата символ x_i и применяем оракул автомата.

3. Если оракул выдаёт отрицательный вердикт, тестирование заканчивается по обнаруженной ошибке. Иначе, получаем от оракула новое состояние v_{i+1} такое, что $\xi(v_{i+1})=\rho(e^*)$.

4. Если $i<t$, то увеличиваем **i** на 1 и переходим к п. 1. Иначе тестирование считается нормально законченным.

В процессе тестирования мы совершаем обход $P^*=(v_0,x^*_0,..,x^*_t)$ графа **G*** и при этом проходим маршрут $P=(v_0,x_0,..,x_t)$ в графе состояний автомата **A**.

Пример тестирования по гомоморфному графу

Рассмотрим фактор-граф $G^*_1=(V,E,\lambda,\rho,\Xi_1,\Theta_1)$ на рис. 2. Этот фактор-граф детерминирован, но не вполне определён. Модифицируем отношение Ξ_1 так, чтобы получился вполне определённый фактор-граф $G^*_2=(V,E,\lambda,\rho,\Xi_2,\Theta_1)$. Для этого достаточно считать вершины **2** и **5** неэквивалентными.

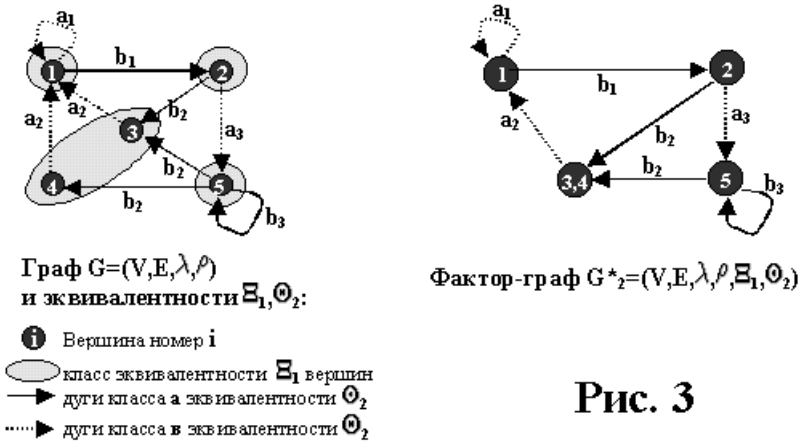


Рис. 3

Существуют несколько обходов фактор-графа G^*_2. Например, обход P^*_2:

❶-a_1→❶-b_1→❷-b_2→❸❹-a_2→❶-b_1→❷-a_3→❺-b_3→❺-b_2→❸❹

Обходу $\mathbf{P^*_2}$ фактор-графа $\mathbf{G^*_2}$ соответствуют два возможных маршрута $\mathbf{P_{21}}$ и $\mathbf{P_{22}}$ в исходном графе **G**:

- $\mathbf{P_{21}}$ = ❶-a_1→❶-b_1→❷-b_2→❸-a_2→❶-b_1→❷-a_3→❺-b_3→❺-b_2→❸
- $\mathbf{P_{22}}$ = ❶-a_1→❶-b_1→❷-b_2→❸-a_2→❶-b_1→❷-a_3→❺-b_3→❺-b_2→❹

Эти маршруты отличаются последним переходом из вершины **5** в вершину **3** или **4** по одной из дуг $\mathbf{b_2}$ одного Δ-класса. Тем самым выбор $\mathbf{P_{21}}$ или $\mathbf{P_{22}}$ недетерминирован. Заметим, что в обоих случаях обходятся не все дуги исходного графа (не обходятся дуги ❹-a_2→❶ и одна из дуг Δ-класса ❺-b_2→❹ или ❺-b_2→❸). Кроме того, маршрут $\mathbf{P_{21}}$ не содержит вершину ❹.

Построение обхода графа в процессе тестирования

Общая задача построения обхода графа хорошо известна. Здесь мы рассмотрим вопрос о построении обхода графа в процессе тестирования.

Тестирование автомата по графу его состояний

Существует инструмент, реализующий универсальный алгоритм $\mathbb{A}$**(A,∅)** тестирования любого автомата **A**, граф состояний которого детерминирован, конечен и сильно связен, с одновременным автоматическим построением обхода графа состояний [3]. В отличие от алгоритма $\mathbb{A}$**(A,P)** для алгоритма $\mathbb{A}$**(A,∅)** не требуется задавать обход **P** графа (обход строится автоматически). Более того, алгоритм $\mathbb{A}$**(A,∅)** не использует никаких знаний об автомате и графе его состояний, кроме тех, которые задаются следующими параметрами алгоритма:

- значение начального состояния автомата;
- операция сравнения состояний на равенство;
- итератор входных символов;
- оракул автомата.

Итератор входных символов по паре <состояние, входной символ> выдаёт следующий входной символ, допустимый для данного состояния:

$\mathbf{It : V \times X \cup \{\varnothing\} \to X \cup \{\varnothing\}}$

Для получения начального входного символа указывается "пустой" входной символ ∅, добавляемый к входному алфавиту. Итератор должен гарантировать, что $\forall \mathbf{v} \in \mathbf{V}$ последовательность **It(v,∅), It(v,It(v,∅)), ...** пробегает все допустимые в состоянии **v** входные символы. В конце этой последовательности итератор возвращает пустой входной символ ∅.

Замечание: Итератор может перебирать все входные символы, а не только допус-

тимые для данного состояния. В этом случае используется отдельная **функция проверки допустимости** для фильтрации недопустимых в данном состоянии входных символов. В терминах формальных спецификаций, проверка допустимости — это проверка предусловия операции.

О допустимых состояниях автомата (вершинах графа) алгоритм узнаёт тогда, когда после подачи на вход автомата некоторого допустимого входного символа получает от оракула значение состояния, в которое переходит автомат.

Обход графа, который автоматически строится алгоритмом $\mathbb{A}$**(A,∅)**, имеет длину того же порядка, что и оптимальный обход графа: произведение числа вершин на число дуг.

Тестирование автомата по гомоморфному графу

Существует модификация этого алгоритма для тестирования по гомоморфному графу — $\mathbb{A}$***(A,∅)**. Гомоморфизм **(*ξ*,*θ*)** графа состояний **G** автомата **A** на граф **G*** должен быть детерминированным и вполне определённым, а граф **G*** конечным и сильно связным. В отличие от алгоритма $\mathbb{A}$***(A,P*)**, алгоритм $\mathbb{A}$***(A,∅)** использует только заданные отображения *ξ* и *θ*, то есть, заданный гомоморфизм исходного графа **G** на граф **G***, и не требует задания обхода **P*** графа **G*** (обход строится автоматически). Более того, алгоритм $\mathbb{A}$***(A,∅)** не использует никаких знаний об автомате, графе его состояний и гомоморфном графе, кроме тех, которые задаются следующими параметрами алгоритма:

- значение начального состояния автомата;
- отображение *ξ*:**V**→**V*** состояния графа **G** в состояние графа **G***;
- итератор обобщённых входных символов (итератор по **X***);
- функция вычисления символа (которая, в свою очередь, зависит только от отображений *ξ* и *θ*);
- оракул автомата.

Вместо отображения *ξ* можно использовать двуместный предикат на множестве состояний **V**, реализующий Ξ-эквивалентность (аналогично тому, как в алгоритме $\mathbb{A}$**(A,∅)** используется операция сравнения состояний на равенство).

Обход гомоморфного графа **G***, который автоматически строится алгоритмом $\mathbb{A}$***(A,∅)**, имеет длину того же порядка, что и оптимальный обход **G***: произведение числа его вершин на число его дуг.

Построение детерминированных вполне определённых фактор-графов

Если на основе некоторого критерия тестового покрытия ввести эквивалентности вершин Ξ и дуг Θ графа **G=(V,E,*λ*,*ρ*)** состояний автомата, то введенные эквивалентности определяют следующее **граничное** требование: неэквивалентные вершины и дуги мы *должны* различать при тестировании. Однако, это не означает, что мы не можем проводить более детального различения вершин и дуг, то есть, использовать эквивалентности Ξ`⊆Ξ и

$\Theta` \subseteq \Theta$. Отчасти это уже происходит при тестировании по гомоморфному графу, поскольку дуги мы различаем, фактически, не по эквивалентности Θ, а по эквивалентности $\Theta^{\sim} \subseteq \Theta$. В общем случае граничное требование выполняется на любом графе, вложенном в **G***, если вложенность графов определить через отображения гомоморфизма или, что то же самое, вложенность определяющих его эквивалентностей Ξ и $\Theta^{\sim}$: $\mathbf{G^{*}`} \subseteq \mathbf{G^{*}} \equiv \Xi` \subseteq \Xi$ & $\Theta`^{\sim} \subseteq \Theta^{\sim}$. (Заметим, что $\Theta` \subseteq \Theta \Rightarrow \Theta`^{\sim} \subseteq \Theta^{\sim}$, но обратное не верно.)

Поэтому, если для заданных **начальных** эквивалентностей Ξ и Θ фактор-граф $\mathbf{G^{*}=(V,E,\lambda,\rho, \Xi,\Theta)}$ оказался Δ-недетерминированным, и/или Θ-недетерминированным, и/или не вполне определённым, мы можем попытаться построить эквивалентности $\Xi_{\Delta fd}$ и $\Theta_{\Delta fd}$ такие, что $\Xi_{\Delta fd} \subseteq \Xi$, $\Theta_{\Delta fd} \subseteq \Theta$, и фактор-граф $\mathbf{G^{*}_{\Delta fd}=(V,E,\lambda,\rho, \Xi_{\Delta fd},\Theta_{\Delta fd})}$ детерминирован и вполне определён.

Пусть фактор-граф $\mathbf{G^{*}_{\Delta f}=(V,E,\lambda,\rho, \Xi_{\Delta f},\Theta_{\Delta f})}$ Δ-детерминирован и вполне определён, но не является $\Theta_{\Delta f}$-детерминированным. В этом случае мы можем выбрать любую эквивалентность $\Theta_{\Delta fd}$ в диапазоне $\Theta_{\Delta f}^{\sim} \subseteq \Theta_{\Delta fd} \subseteq \Theta_{\Delta f}$, удовлетворяющую критерию $\Theta_{\Delta fd}$-детерминированности:
$\forall \mathbf{e_1,e_2} \in \mathbf{E}\ \lambda(\mathbf{e_1})\ \Xi_{\Delta f}\ \lambda(\mathbf{e_2})$ & $\rho(\mathbf{e_1}) \neg \Xi_{\Delta f}\ \rho(\mathbf{e_2}) \Rightarrow \mathbf{e_1} \neg \Theta_{\Delta fd}\ \mathbf{e_2}$.

Очевидно, что такая эквивалентность существует, например, $\Theta_{\Delta fd} = \Theta_{\Delta f}^{\sim}$. Вообще, все такие эквивалентности $\Theta_{\Delta fd}$ легко получить, определяя всевозможные подразбиения множеств фактор-дуг, выходящих из одной фактор-вершины, и $\Theta_{\Delta f}$*-эквивалентных.

Поэтому ниже мы решаем проблему построения Δ-детерминированного и вполне определённого фактор-графа $\mathbf{G^{*}_{\Delta f}=(V,E,\lambda,\rho, \Xi_{\Delta f},\Theta_{\Delta f})}$. Поскольку вложенный фактор-граф имеет не меньшее (при строгой вложенности — большее) число вершин и дуг, естественно строить наибольшие (по отношению вложенности) эквивалентности $\Xi_{\Delta fmax}$ и $\Theta_{\Delta fmax}$.

Критерий Δ-детерминированности фактор-графа

Очевидно, что если $\mathbf{G^{*}`} \subseteq \mathbf{G^{*}}$ и **G*** $\Delta$-недетерминирован, то $\mathbf{G^{*}`}$ также Δ-недетерминирован: $\Theta`^{\sim} \subseteq \Theta^{\sim}$ & $\neg(\Delta \subseteq \Theta^{\sim}) \Rightarrow \neg(\Delta \subseteq \Theta`^{\sim})$.

Поскольку $\Theta^{\sim} = \Theta \cap \Xi^{\sim}$, условие Δ-детерминизма $\Delta \subseteq \Theta^{\sim}$ означает выполнение двух условий вложенности: $\Delta \subseteq \Theta$ & $\Delta \subseteq \Xi^{\sim}$. Покажем, что существует такая минимальная эквивалентность вершин $\Xi_{\Delta min}$, которая необходима и достаточна для того, чтобы $\Delta \subseteq \Xi^{\sim}$, то есть, $\Delta \subseteq \Xi^{\sim} \equiv \Xi_{\Delta min} \subseteq \Xi$.

Алгоритм 1: Построение эквивалентности $\Xi_{\Delta min}$:

1. объявляем $\Xi_{\Delta min}$-эквивалентными концы Δ-эквивалентных дуг;
2. полученное рефлексивное и симметричное отношение транзитивно замыкаем до требуемой эквивалентности.

Таким образом, критерий Δ-детерминированности формулируется как выполнение двух условий вложенности: $\Delta \subseteq \Theta$ & $\Xi_{\Delta min} \subseteq \Xi$.

Это условие выполнено для всех фактор-графов в диапазоне по отношению вложенности

$[G^*_{\Delta min}, G^*_{max}]$, где $G^*_{\Delta min}=(V,E,\lambda,\rho, \Xi_{\Delta min},\Delta)$, $G^*_{max}=(V,E,\lambda,\rho, \Xi_{max},\Theta_{max})$, Ξ_{max} и Θ_{max} — **максимальные** эквивалентности вершин и дуг (все вершины эквивалентны и все дуги эквивалентны). Граф G^*_{max} содержит одну фактор-вершину и одну фактор-дугу — петлю.

Рис. 4

Построение вполне определённого фактор-графа

Очевидно, что для любого фактор-графа $G^*=(V,E,\lambda,\rho, \Xi,\Theta)$ существует вложенный в него вполне определённый фактор-граф. Примером может служить сам граф состояний **G**. Мы построим максимальный (по отношению вложенности) вполне определённый фактор-граф $G^*_{fmax} \subseteq G^*$. При этом мы изменим только эквивалентность вершин, то есть, $G^*_{fmax}=(V,E,\lambda,\rho, \Xi_{fmax},\Theta)$, где $\Xi_{fmax} \subseteq \Xi$.

Алгоритм 2: Построение фактор-графа $G^*_{fmax}=(V,E,\lambda,\rho, \Xi_{fmax},\Theta)$.

Алгоритм начинает с фактор-графа $G^*_0=G^*=(V,E,\lambda,\rho, \Xi,\Theta)$ и состоит из последовательности однотипных шагов. Каждый **i**-ый шаг алгоритма преобразует фактор-граф G^*_{i-1} в фактор-граф G^*_i. Назовём **условием вложенности** следующее условие: каждый вполне определённый фактор-граф, вложенный в G^*, вложен в G^*_i. Для того, чтобы доказать, что алгоритм строит фактор-граф G^*_{fmax}, достаточно доказать, что алгоритм удовлетворяет следующим требованиям:

1. В начале работы алгоритма условие вложенности выполнено для фактор-графа G_0^* (что очевидно).
2. Каждый **i**-ый шаг алгоритма сохраняет условие вложенности для фактор-графа G^*_i, если это условие выполнено в начале шага (для фактор-графа G^*_{i-1}).
3. Алгоритм заканчивается после **i**-ого шага, если G^*_i вполне определённый.
4. Алгоритм заканчивается за конечное число шагов.

Шаг **i** алгоритма заключается в следующем:

> Если $G^*_{i-1}=(V,E,\lambda,\rho, \Xi_{i-1},\Theta)$ — вполне определённый фактор-граф, то алгоритм заканчивается (выполняя требование 3). Иначе, существует не вполне определённая фактор-дуга $e^* \in E/\Theta_{i-1}^{\sim}$, то есть, $\{ \lambda(e) \mid e \in e^* \} \neq \lambda(e^*)$. Разобьём фактор-вершину $v^* = \lambda(e^*)$ на два подмножества: $v^*_1 = \{ \lambda(e) \mid e \in e^* \}$ и $v^*_0 = v^* \backslash v^*_1$. Тем самым, мы получаем новую эквивалентность Ξ_i и новый фактор-граф $G^*_i=(V,E,\lambda,\rho, \Xi_i,\Theta)$:
>
> $$\forall v_1,v_2 \in V \quad v_1 \,\Xi_i\, v_2 \equiv (v_1 \,\Xi_{i-1}\, v_2) \;\&\; \neg(v_1 \in v^*_0 \;\&\; v_2 \in v^*_1 \vee v_1 \in v^*_1 \;\&\; v_2 \in v^*_0)$$

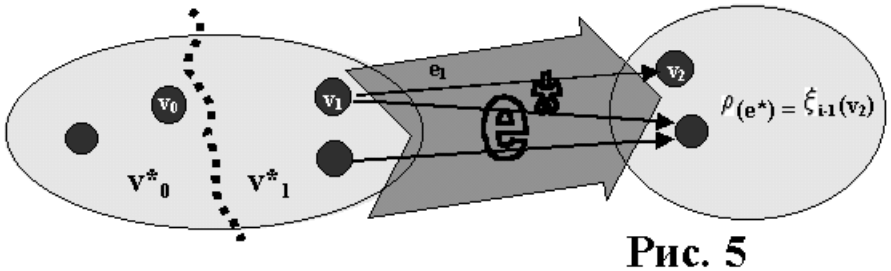


Рис. 5

Нам нужно доказать требование 2: если условие вложенности было выполнено для $\mathbf{G^*_{i-1}}$, то оно выполнено для $\mathbf{G^*_i}$.

Рассмотрим любые две вершины $\mathbf{v_1 \in v^*_1}$ и $\mathbf{v_0 \in v^*_0}$. Для $\mathbf{v_1}$ существует дуга $\mathbf{e_1 \in e^*}$, начинающаяся в $\mathbf{v_1}$ и заканчивающаяся в $\mathbf{v_2 \in \rho(e^*)}$. Очевидно, что $\mathbf{\rho(e^*)=\xi_{i-1}(v_2)}$. Поскольку для $\mathbf{G^*_{i-1}}$ выполнено условие вложенности, в любом вполне определённом фактор-графе $\mathbf{G^*_f}$ для вершины $\mathbf{v_2}$ должно выполняться $\mathbf{\xi_f(v_2) \subseteq \xi_{i-1}(v_2)}$. Из вершины $\mathbf{v_0}$ не выходит ни одной дуги из множества $\mathbf{e^*}$. Поскольку $\mathbf{e^*}$ содержит, по определению фактор-дуги, все Θ-эквивалентные дуги, ведущие из $\mathbf{v^*}$ в $\mathbf{\rho(e^*)}$, из $\mathbf{v_0}$ не выходит ни одной дуги Θ-эквивалентной дуге $\mathbf{e_1}$ и ведущей в $\mathbf{\rho(e^*)=\xi_{i-1}(v_2)}$, тем более, в его подмножество $\mathbf{\xi_f(v_2)}$. Итак, из $\mathbf{v_1}$ ведёт в $\mathbf{\xi_f(v_2)}$ дуга $\mathbf{e_1}$, а из $\mathbf{v_0}$ не ведёт в $\mathbf{\xi_f(v_2)}$ ни одной дуги Θ-эквивалентной дуге $\mathbf{e_1}$. Следовательно, по определению вполне определённости, вершины $\mathbf{v_1}$ и $\mathbf{v_0}$ не могут принадлежать одной фактор-вершине фактор-графа $\mathbf{G^*_f}$.

Таким образом, любая фактор-вершина любого вполне определённого фактор-графа $\mathbf{G^*_f \subseteq G^*_{i-1}}$, вложенная в $\mathbf{v^*}$, должна быть вложена в $\mathbf{v^*_1}$ или в $\mathbf{v^*_0}$. Иными словами, разбиение фактор-вершины $\mathbf{v^*}$ на две фактор-вершины $\mathbf{v^*_1}$ и $\mathbf{v^*_0}$ не разбивает ни одной фактор-вершины любого вполне определённого фактор-графа $\mathbf{G^*_f \subseteq G_{i-1}}^*$. Тем самым, условие вложенности сохраняется.

Нам осталось доказать требование 4: алгоритм заканчивается за конечное число шагов. Действительно, каждый шаг алгоритма увеличивает число фактор-вершин фактор-графа, а это число ограничено сверху числом вершин графа $\mathbf{G}$ (граф конечен).

На этом доказательство выполнения требований 1-4 закончено.

Все вполне определённые фактор-графы располагаются в диапазоне $\mathbf{[G,G^*_{max}]}$, однако, не все фактор-графы в этом диапазоне вполне определены. Тем не менее, алгоритм 2 показывает, что для любого фактор-графа в диапазоне $\mathbf{[G,G^*_{max}]}$ можно построить вложенный в него вполне определённый фактор-граф $\mathbf{G^*_{fmax}}$.

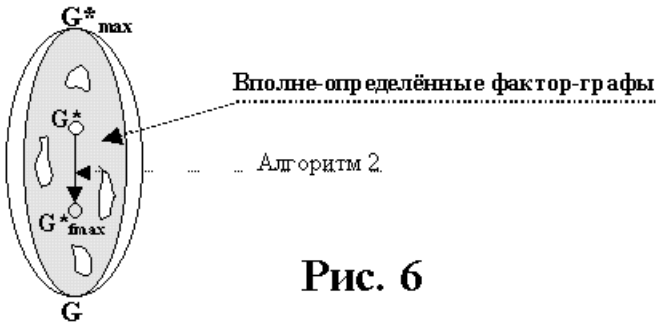


Рис. 6

Критерий существования и алгоритм построения Δ-детерминированного и вполне определённого фактор-графа

Суммируя предыдущие подразделы, можно сказать, что не для всякого фактор-графа **G*** заданного графа **G** существует Δ-детерминированный и вполне определённый фактор-граф $\mathbf{G^*_{\Delta f}} \subseteq \mathbf{G^*}$. Алгоритм 2 строит максимальный вполне определённый фактор-граф $\mathbf{G^*_{fmax}} \subseteq \mathbf{G^*}$, который может быть проверен на Δ-детерминизм по критерию $\Delta \subseteq \Theta_{fmax}$ & $\Xi_{\Delta min} \subseteq \Xi_{fmax}$, где $\Xi_{\Delta min}$ строится из Ξ алгоритмом 1. Таким образом, если фактор-графы $\mathbf{G^*_{\Delta f}}$ существуют, то $\mathbf{G^*_{fmax}}$ — максимальный среди них, иначе $\mathbf{G^*_{fmax}}$ Δ-недетерминирован.

С другой стороны, для всякого графа $\mathbf{G=(V,E,\lambda,\rho)}$ можно найти такие эквивалентности $\Xi_{\Delta f}$ и $\Theta_{\Delta f}$, что фактор-граф $\mathbf{G^*_{\Delta f}=(V,E,\lambda,\rho,\ \Xi_{\Delta f},\Theta_{\Delta f})}$ Δ-детерминирован и вполне определён. Примером может служить фактор-граф $\mathbf{G^*_{max}=(V,E,\lambda,\rho,\ \Xi_{max},\Theta_{max})}$ (все вершины эквивалентны и все дуги эквивалентны).

Легко показать, что для всякого Δ-детерминированного и вполне определённого фактор-графа $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta f},\Theta_{\Delta f})}$ фактор-граф $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta f},\Theta_{max})}$ также является Δ-детерминированным и вполне определённым фактор-графом. Таким образом, множество всех Δ-детерминированных и вполне определённых фактор-графов можно описать в два этапа:

- Описывается множество Ξ эквивалентностей вершин $\Xi_{\Delta f}$, для которых $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta f},\Theta_{max})}$ Δ-детерминирован и вполне определён.
- Для $\Xi_{\Delta f} \in \Xi$ описывается множество $\Theta(\Xi_{\Delta f})$ эквивалентностей дуг $\Theta_{\Delta f}$, для которых фактор-граф $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta f},\Theta_{\Delta f})}$ Δ-детерминирован и вполне определён.

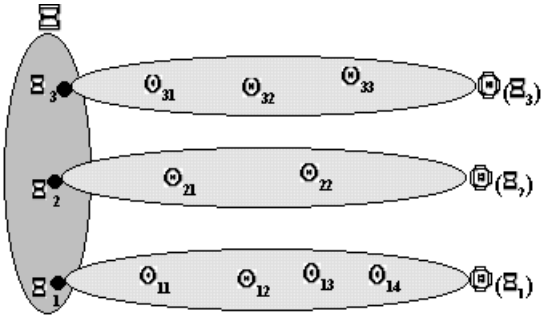
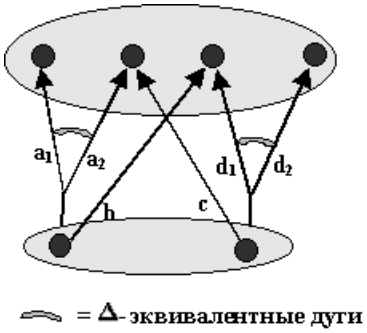


Рис. 7

Множество Ξ располагается в диапазоне $\mathbf{[\Xi_{\Delta min},\ \Xi_{max}]}$. Если $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta min},\Theta_{max})}$ не вполне определён, то диапазон открыт снизу: $\mathbf{(\ \Xi_{\Delta min},\ \Xi_{max}]}$. Для всех Ξ в этом диапазоне фактор-граф $\mathbf{(V,E,\lambda,\rho,\ \Xi,\Theta_{max})}$ Δ-детерминирован, но не обязательно вполне определён. Для любого Ξ_Δ в этом диапазоне алгоритм 2 строит максимальный $\Xi_{fmax} \subseteq \Xi_\Delta$, для которого фактор-граф $\mathbf{(V,E,\lambda,\rho,\ \Xi_{fmax},\Theta_{max})}$ вполне определён. Однако, Ξ_{fmax} может оказаться за пределами диапазона — $\Xi_{\Delta min}$ не вложен в Ξ_{fmax} — и, тем самым, $\mathbf{(V,E,\lambda,\rho,\ \Xi_{fmax},\Theta_{max})}$ Δ-недетерминирован.

Эквивалентность вершин $\Xi_{\Delta f} \in \Xi$ индуцирует эквивалентность дуг $\Xi_{\Delta f}^{\sim}$, определяющую некоторое базовое разбиение дуг, при котором фактор-граф $\mathbf{(V,E,\lambda,\rho,\ \Xi_{\Delta f},\ \Xi_{\Delta f}^{\sim})}$ Δ-детерминирован и вполне определён. Эквивалентность дуг $\Theta_{\Delta f} \in \Theta(\Xi_{\Delta f})$ задаёт для каж-

дой базовой дуги **e*** такое её разбиение, при котором не нарушается **Δ**-детерминизм и вполне определённость. **Δ**-детерминизм не нарушается, если при разбиении **e*** не разбиваются вложенные в неё **Δ**-классы. Вполне определённость не нарушается, если в каждый класс разбиения **e*** входит хотя бы одна дуга, ведущая из каждой вершины $v \in \lambda(e^*)$.



Разбиения фактор-дуги, сохраняющие Δ-детерминизм и вполне-определённость:

- **(a1,a2,c) (b,d1,d2)**
- **(a1,a2,d1,d2) (b,c)**

Рис. 8

Для $\Theta_{\Delta f} \in \mathcal{O}(\Xi_{\Delta f})$ можно рассматривать эквивалентности $\Theta_{\Delta fd}$, удовлетворяющие условиям:

- $\Theta_{\Delta f}^{\sim} \subseteq \Theta_{\Delta fd} \subseteq \Theta_{\Delta f}$,
- $\forall e_1, e_2 \in E \;\; \lambda(e_1)\, \Xi_{\Delta f}\, \lambda(e_2) \;\; \& \;\; \rho(e_1) \neg \Xi_{\Delta f}\, \rho(e_2) \;\; \Rightarrow \;\; e_1 \neg \Theta_{\Delta fd}\, e_2$
  (критерий $\Theta_{\Delta fd}$-детерминированности).

Все такие эквивалентности $\Theta_{\Delta fd}$ получаются с помощью всевозможных подразбиений множеств фактор-дуг, выходящих из одной фактор-вершины, и $\Theta_{\Delta f}^*$-эквивалентных. Соответствующие фактор-графы будут детерминированными и вполне определёнными.

Пример графа состояний автомата и его фактор-графов

В качестве примера рассмотрим продажу билетов на авиарейсы. Определены две операции: **продажа билета** и **возврат билета**.

Операция **продажа билета** имеет один входной параметр — номер рейса, и возвращает номер одного из свободных билетов на этот рейс. В случае если на заказываемый рейс все билеты проданы, может быть выдан билет в ту же точку назначения на более поздний рейс. Операция имеет предусловие: на заказываемый или более поздние рейсы имеются непроданные билеты. Если имеется несколько непроданных билетов, удовлетворяющих заказу, операция недетерминирована, поскольку не специфицируется, какой именно из этих билетов будет продан.

Операция **возврат билета** имеет один параметр — номер возвращаемого билета. Операция имеет предусловие: билет должен быть ранее проданным билетом. В результате выполнения операции билет поступает в свободную продажу.

Для удобства изложения ограничимся двумя рейсами в одну точку назначения и двухместными самолётами. Билеты нумеруются: **0**,**1** — рейс **1**; **2**,**3** — рейс **2**. Граф состояний **G** изображён на рис. 9. Этот граф **Δ**-недетерминирован.

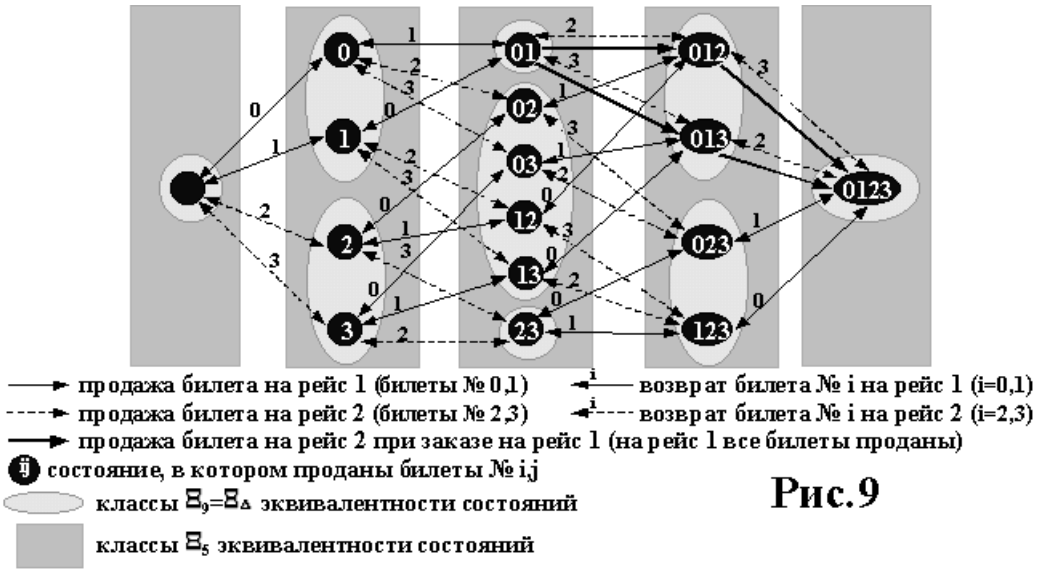


Рис.9

Множество Ξ состоит из трёх эквивалентностей: $\Xi_9 = \Xi_{\Delta min}$ (9 фактор-состояний), $\Xi_1 = \Xi_{max}$ (все состояния эквивалентны — 1 фактор-состояние) и промежуточной эквивалентности Ξ_5 (5 фактор-состояний), которая получается из Ξ_9 объединением Ξ_9-классов, расположенных на рис. 9 на одной вертикали.

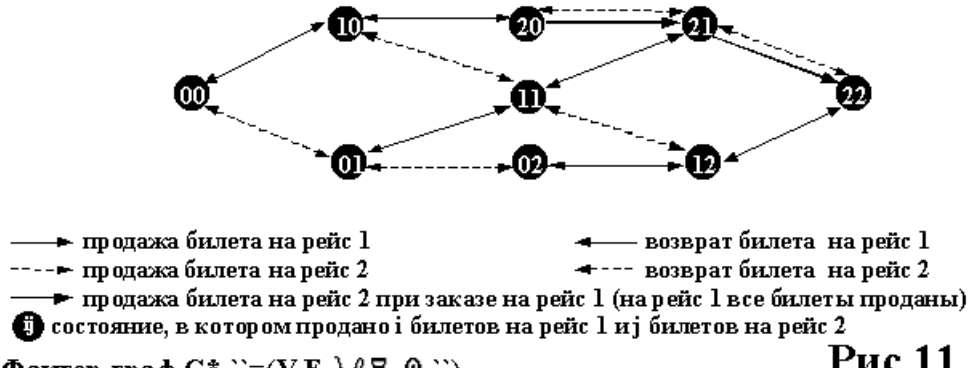
Множество $\Theta(\Xi_1)$ состоит из единственной эквивалентности дуг $\Theta_1 = \Theta_{max}$, при которой мы не различаем операции (все дуги эквивалентны — одна фактор-дуга — петля). Этот случай детерминированного и вполне определённого фактор-графа неинтересен для тестирования.

Эквивалентность состояний Ξ_5 строится алгоритмом 2, если начальная эквивалентность дуг Θ_5 различает операции (продажа или возврат билетов), но не различает поддомены операций. Фактор-состояние определяется числом проданных билетов (на оба рейса в сумме). Фактор-граф $G^*_5=(V,E,\lambda,\rho, \Xi_5,\Theta_5)$ детерминирован и вполне определён.

Рис.10

Эквивалентность состояний $\Xi_9=\Xi_{\Delta min}$ строится алгоритмом 1. Обобщённое состояние определяется парой чисел: число проданных билетов на рейс 1 и на рейс 2. Ξ_9 строится также алгоритмом 2, если начальная эквивалентность дуг Θ_9 различает операции (продажа или возврат билетов) и два поддомена операции продажи, определяемые параметром операции: заказ билета на рейс 1 или рейс 2. Фактор-граф $G^*_9=(V,E,\lambda,\rho, \Xi_9,\Theta_9)$ Δ-детерминирован и вполне определён, но Θ_9-недетерминирован. Дело в том, что возврат билета в любом фактор-состоянии, кроме начального (не продано ни одного билета), переводит в разные фактор-состояния в зависимости от того, билет на какой рейс возвраща-

ется. Естественно ввести подэквивалентность $\Theta_9` \subseteq \Theta_9$, которая различала бы два поддомена операции возврата билетов: возврат билета на рейс 1 и возврат билета на рейс 2. В этом случае обобщённый алфавит состоит из четырёх символов: продажа или возврат билета на тот или иной рейс. Фактор-граф $G^*_9`=(V,E,\lambda,\rho, \Xi_9,\Theta_9`)$ детерминирован и вполне определён. Также будет детерминирован и вполне определён фактор-граф $G^*_9``=(V,E,\lambda,\rho, \Xi_9,\Theta_9``)$, в котором различается ещё один поддомен операции продажи: продажа билетов на рейс 2 при заказе билетов на рейс 1 (когда все билеты на более ранний рейс 1 проданы).



Фактор-граф $G^*_9``=(V,E,\lambda,\rho,\Xi_9,\Theta_9``)$ **Рис.11**

Сравнение с аналогичными работами

Вопросы тестирования конечных автоматов исследуются уже достаточно давно, они даже успели попасть в учебники [2]. Но применение для тестирования классов ООП различных обобщений графа состояния рассматриваются только в последнее время [1, 4, 5, 8]. В этих работах используется понятие "абстрактная модель конечного автомата". Гомоморфные графы (в частности, фактор-графы) являются особым случаем таких абстрактных моделей.

В работе C.D.Turner и D.J.Robson [8] подробно излагается шаг за шагом методология тестирования классов ООП при помощи модели класса как конечного автомата. После анализа традиционных методов тестирования вводятся понятия конечного автомата, его состояний и переходов. На примере показывается, как извлекаются из описания класса C++ состояния абстрактного автомата, описываются функции переходов как сценарии динамического изменения состояния. Затем рассматриваются шаги генерации тестовых последовательностей на основе полученных описаний. В работе очень кратко описываются прототипы инструментов для генерации тестовых последовательностей и пропуска тестов. Это скорее компиляторы описаний тестовых последовательностей (test script files), а не автоматические генераторы их. Эта работа может служить хорошим вводным пособием.

В работе D.Hoffman и P.Strooper [4] описывается инструмент (ClassBench) для автоматического построения обхода графа состояний конечного автомата. Для ClassBench необходимо создать эксплицитное описание автомата — всех вершин графа состояния и всех дуг-переходов. Все петли (дуги, начало и конец которых совпадают) описываются при вершине графа. Они проходятся при каждом попадании в вершину. Кроме этого описания создаются класс Driver и класс Oracle. Класс Driver обеспечивает выполнение собственно тестирования. В нем определены следующие методы: reset — приведение автомата в первоначальное состояние; transit — переход из текущего состояния в новое при проходе по указанной дуге, и node — выполняющий все предусмотренные действия при попадании в

текущую вершину (проходы по петлям). Класс Driver при выполнении переходов и проходов по петлям вызывает методы в объекте тестируемого класса и использует класс Oracle для проверки правильности работы тестируемого объекта. ClassBench обеспечивает тестирование при различных критериях тестового покрытия: покрытие всех состояний, всех дуг и всех путей. ClassBench снабжен вспомогательными средствами для создания описаний автомата, для разработки тел Driver'a и Oracle'a.

С этой работой тесно связана работа L.Murray *et al.* [5], в которой описывается методика построения абстрактного конечного автомата по формальным спецификациям класса, написанным на языке Object-Z. Описание автомата строится в соответствии с требованиями используемого инструмента — ClassBench.

В [1, 3] описывается набор инструментов, реализующих тестирование по алгоритму $\mathbb{A}^*(A,\varnothing)$, и автоматизирующих создание соответствующего генератора тестовых последовательностей. При этом ручным способом создаются только следующие тестовые компоненты: функция отображения реального состояния в абстрактное состояние, итератор обобщённых входных символов и функция вычисления символа. Всё остальное генерируется автоматически из формальных спецификаций на языке RAISE (RSL). Важно отметить, что работы [4, 5, 8] носят теоретический и экспериментальный характер, тогда как работы [1, 3] описывают практический результат – технологию, внедренную в процесс разработки и верификации крупного программного комплекса.

Данная статья развивает направление, предложенное в [1, 3], и представляет собой строгое истолкование эмпирических решений, найденных в ходе реальных программистских проектов. Подход, рассматриваемый здесь, отличается от упомянутых выше теоретических работ тремя основными позициями. Первое отличие заключается в подходе к решению упомянутой выше проблемы смежности дуг. Из-за этого проходу по одной дуге абстрактного графа в ClassBench соответствует, вообще говоря, путь в реальном графе, а в наших алгоритмах — проход ровно по одной дуге реального графа. Это различие ведёт, с одной стороны, к различным оценкам времени тестирования, а с другой стороны, к различным требованиям соответствия абстрактного и реального графов. В частности, абстрактный граф для ClassBench может быть не гомоморфен реальному.

Второе отличие нашей работы — возможность построения обхода графа и самого графа в процессе тестирования, то есть, не требуется эксплицитное описание графа. Все знания об автомате сосредоточены в оракуле, который генерируется автоматически по спецификациям. Соответствие абстрактного и реального графов задаётся минимальным образом: отображением реальных состояний в абстрактные состояния (или предикатом эквивалентности состояний), итератором обобщённых входных символов и функцией вычисления символа. Важно также отметить, что для наших алгоритмов не требуется выделенной операции reset, а только сильная связность графа.

Наконец, третье отличие: в работах [4,5] не описывается применение данной методики к недетерминированным автоматам.

Заключение

Естественным направлением дальнейших исследований является формализация процесса извлечения из формальных спецификаций информации, необходимой для генерации компонентов, которые сейчас создаются вручную. Эта формализация должна иметь своим

итогом, с одной стороны, формальные требования (методология) к форме спецификаций и, с другой стороны, набор инструментов (технология), автоматически генерирующих из спецификаций необходимые компоненты тестовых наборов. В частности, могут быть созданы инструменты, поддерживающие построение детерминированных и вполне определённых фактор-графов (алгоритмы 1 и 2).

Вторым направлением дальнейших исследований может быть использование других абстрактных моделей автоматов и их графов состояний, в частности тех, что исследуются в работах [4,5]. Здесь можно отметить, что на практике при ручном создании указанных выше тестовых компонентов часто применялись дополнительные приёмы, не нашедшие своего отражения в настоящей статье. В частности, иногда дуге абстрактного графа соответствовала не одна дуга, а путь в реальном графе состояний. В некоторых частных случаях тестирование по алгоритму $\mathbb{A}^*(\mathbf{A},\varnothing)$ удавалось выполнять по недетерминированным фактор-графам. Однако, то, что удаётся в отдельных случаях сделать вручную, далеко не всегда возможно реализовать в виде инструмента. Поэтому важным представляется анализ накопленного опыта спецификаций и тестирования с целью извлечения формализуемых приёмов и дальнейшей реализации их в виде инструментов.

Литература

1. A.Barantsev, I.Burdonov, A.Kossatchev, H.Wong. Report on Test Generation Methodology. Nortel.1997.

2. B.Beizer. Software Testing Techniques. 2-nd edition. Van Nostrand Reinhold. N-Y.1990.

3. I.Burdonov, A. Kossatchev, A. Petrenko, S. Cheng, H. Wong. Formal Specification and Verification of SOS Kernel. *BNR/NORTEL Design Forum*, June 1996.

4. D. Hoffman, P.Strooper. ClassBench: a Framework for Automated Class Testing. *Software Maintenance: Practice and Experience*, Vol. 27, No. 5, May 1997, pp. 573-579.

5. L. Murray, D. Carrington. I. MacColl, J. McDonald, P. Strooper. Formal Derivation of Finite State Machines for Class Testing. In J.P. Bowen, A.Fett, M.G.Hinchey, editors, *ZUM'98: The Z Formal Specification Notation.*, *11-th Int. Conf. of Z Users*, Vol. 1493 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998, pp. 42-59.

6. D.K.Peters, D.L.Parnas. Using Test Oracles Generated from Program Documentation. *IEEE Trans. on Software Engineering*, Vol. 24, No. 3, March 1998, pp. 161-173.

7. A.K.Petrenko, I.B.Burdonov, A.Yu. Drojjina, A.S. Kossatchev, A.V.Maximov, Yu.L.Sazanov.H.Sumar. Preliminary Test Methodology and Test System Report. Nortel. 24 May 1995.

8. C.D. Turner, D.J.Robson. The State-based Testing of Object-Oriented Programs. *Proc. IEEE Conf. Software Maintenance,* 1993, pp. 302-310.