Компьютерные инструменты в образовании, 2016
№ 5: 17–45
УДК: 519.682.1 + 681.142.2
http://ipo.spb.ru/journal

СИНТАКСИЧЕСКИ УПРАВЛЯЕМАЯ ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ПРОЦЕССОРОВ РЕГУЛЯРНЫХ ЯЗЫКОВ

Мартыненко Б.К.[1]

[1]СПбГУ, Санкт-Петербург, Россия

Аннотация

Описывается метод генерации тестов минимальной длины для синтаксически управляемых конечных процессоров, реализующих регулярные языки. Критерий выбора тестовых вариантов выражает заданную степень покрытия дуг графа, представляющего регулярное выражение, по которому строится процессор. Поскольку дуги графа помечены семантическими метками, то критерий выражает соответствующую степень взаимодействия между семантиками, и тест гарантирует выполнение этого критерия. Метод основан на алгоритме решения задачи Китайского почтальона на ориентированном графе, выводимом из регулярного выражения, определяющем конечно-автоматный язык, реализуемом процессором.

Ключевые слова: *задача Китайского почтальона, ориентированный граф, регулярное выражение, рёберный граф, синтаксическая диаграмма Н. Вирта, тест минимальной длины.*

Цитирование: Мартыненко Б.К. Синтаксически управляемая генерация тестов для процессоров регулярных языков // Компьютерные инструменты в образовании, 2016. № 5. С. 17–45 .

ВВЕДЕНИЕ

Детерминированные конечные автоматы применяются во многих областях: для лексического анализа в трансляторах, при сопоставлении образцов в анализе ДНК (поиск генетических маркеров), в области компьютерной безопасности (application firewall), в поисковых системах и т. д. В таких задачах языки задаются регулярными выражениями, которые компилируются в какой-либо алгоритм или программу, которые подвергаются тестированию. Как распознаватели они довольно полезны для многих проблем от проверки непротиворечивости (например, блоки проверки допустимости XML) до профилирования новых алгоритмов разбора.

В отличие от методов тестирования, рассматривающих тестируемую программу как «черный ящик», тестирование синтаксически управляемых программ проводится при полном учёте специфики их организации.

Большой класс технологических инструментов для производства программ обработки данных использует принцип синтаксического управления. В типичных приложениях, таких как трансляция языков программирования, этот принцип давно и успешно используется. Управление процессом трансляции определяется синтаксической структурой предложений входного языка.

Особенность таких программ состоит в том, что структура входных данных (входного языка) и система управления обработкой данных такой программы определяется одной и той же грамматикой. Главная часть логики такой программы сосредоточена в управляющей таблице, а сама программа обращается к ней на каждом шагу её работы. Управляющая таблица, в частности, определяет порядок вызова семантических процедур по ходу синтаксического анализа входного текста.

Предлагаемый подход к генерации тестов состоит в использовании граф-схем, аналога синтаксических диаграмм Н. Вирта [1, 3], выводимых из EBNF-грамматик. Граф-схемы содержат всю информацию о преобразованиях входного языка, определяемых через его синтаксическую структуру посредством разметки стрелок переходов в синтаксической диаграмме цепочками семантических символов, ассоциируемых с процедурами преобразования входного языка, определяемого той же диаграммой.

Наглядность синтаксических диаграмм Н. Вирта сама подсказывает естественный критерий полноты тестирования языкового процессора в терминах покрытия дуг синтаксической граф-схемы. Учёт диапазона взаимодействия частичных преобразований, выполняемых по ходу сканирования входного текста, естественно приводит к идее распределения тестов по степеням этого диапазона.

Тест степени $n \geqslant 0$ должен активизировать как минимум однократное исполнение каждой возможной из $(n+1)$ семантик в допустимой грамматикой последовательности вызовов при минимальной суммарной длине составляющих его тестовых вариантов.

Поскольку проблема касается регулярных языков, то уместно сослаться на следствие из теоремы Клини, согласно которому любой регулярный язык или язык типа 3 по Н. Хомскому представим регулярным выражением, то есть формулой с тремя операциями: объединения, конкатенации и замыкания Клини над конечными цепочками символов языка. Каждая такая формула представима в виде ориентированного графа [2, 3], в котором узлы помечены символами языка, а ориентированные дуги определяют порядок их обхода. Этот граф определяет регулярный язык как множество меток узлов при его обходе. Именно по нему генерируется тест минимальной длины для процессора регулярного языка, представляемого этим графом.

Минимальность теста достигается путём решения целочисленной задачи о потоке минимальной стоимости в ориентированной сети [4], представленной этим графом, в котором дуги имеют единичную длину.

1. СИНТАКСИЧЕСКИЕ ДИАГРАММЫ Н. ВИРТА И КС-ГРАММАТИКИ

КС-грамматики без леворекурсивных правил и синтаксические диаграммы Н. Вирта — это два эквивалентных способа описания КС-языков.

Определение 1. КС-грамматика есть четвёртка $G = (N, T, P, S)$, где N и T — непересекающиеся алфавиты, $S \in N$, и $P \subseteq N \times V^*$ для $V = N \cup T$. Элементы N называются *нетерминалами*, T — *терминалами*, а P — *правилами вывода* или *продукциями*. Каждое её правило имеет вид $A \to \beta \in P$, где $A \in N, \beta \in V^*$.

Для $x, y \in V^*$ пишут $x \Rightarrow y$, если и только если $x = x_1 A x_2$, $y = x_1 \alpha x_2$ для $x_1, x_2 \in V^*$, и $A \to \alpha \in P$, и говорят, что из x непосредственно выводится y в G. *Рефлексивно-транзитивное замыкание* отношения $\Rightarrow$ изображается значком $\overset{*}{\Rightarrow}$. *Язык, порождаемый* G, есть $L(G) = \{x \in T^* \mid S \overset{*}{\Rightarrow} x\}$.

КС-грамматика G называется *приведённой*, если для всех нетерминалов $A \in N$ существует вывод вида $S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} x$, где $\alpha, \beta \in (N \cup T)^*, x \in T^*$.

Определение 2. Нетерминал $A \in N$ в cfg $G = (N, T, P, S)$ называется *леворекурсивным*, если существует вывод $A \overset{+}{\Rightarrow} A\alpha$ для $\alpha \in V^+$. Здесь $V^+ = VV^* = V^*V$.

КС-грамматика называется *леворекурсивной*, если в ней существует, по крайней мере, один леворекурсивный нетерминал.

Любую КС-грамматику всегда можно эквивалентным образом преобразовать так, чтобы в ней не было ни одного леворекурсивного нетерминала. Именно, если для некоторого $A \in N$ существуют леворекурсивные правила вида

$$\{A \to A\alpha_i \mid \alpha_i \in V^*, i = 1, ..., m\},$$

а все другие правила имеют вид

$$\{A \to \beta_j \mid \beta_j \in V^*, \beta_j = B\gamma_j, B \neq A, j = 1, ..., n\},$$

то эквивалентные правила без левой рекурсии есть

$$\{A \to \beta_j Z,\ Z \to \alpha_i Z,\ Z \to \varepsilon\},\ Z \text{ — новый нетерминал.}$$

Определение 3. КС-грамматика $G = (N, T, P, S)$ называется грамматикой с *самовставлением* [5], если $A \overset{+}{\Rightarrow} \alpha_1 A \alpha_2$ для некоторых $\alpha_1, \alpha_2 \in V^+$ и $A \in N$.

Со времени публикации Сообщения об Алгоритмическом Языке АЛГОЛ 60 [6] правило КС-грамматики представляют в *форме Бэкуса-Наура* (BNF):

$$A ::= \beta_1 \mid \beta_2 \mid ... \mid \beta_m,\ m \geqslant 0,$$

где все альтернативы для нетерминала A объединяются в одном правиле. При этом вводятся параллельные термины:

1) *металингвистические переменные* — конечные последовательности знаков, заключённых в металингвистические скобки ⟨ и ⟩, аналог нетерминалов;
2) *основные символы языка* — все прочие символы, аналог терминалов грамматики.

Между символами ⟨ и ⟩ обычно пишут слова, представляющие название конструкции языка, например, ⟨арифметическое выражение⟩, ⟨переменная⟩ и т. д. Основные символы языка — это его алфавит, например **begin**, +, := и т. д. Из них состоит любое предложение языка.

С другой стороны, BNF-правило допускает другую эквивалентную интерпретацию. Каждая из альтернатив рассматривается как конкатенация символов, её составляющих, а знак | трактуется как *операция объединения* цепочек всех альтернатив.

Более того, правые части BNF-правил, рассматриваемые как формулы с двумя регулярными операциями *конкатенацией* (которая обычно никак не обозначается) и *объединением* (|) над символами V грамматики G, могут использовать ещё *операцию замыкания*, обозначаемую *звездой Клини* (*).

Грамматику, правые части правил которых интерпретируются как формулы с этими тремя регулярными операциями, называют расширенной BNF, а весь класс таких грамматик обозначается как EBNF.

Определение 4. Пусть $\mathscr{G} = (N, T, \mathscr{R}, S)$ — EBNF-грамматика, где N, T и S — то же самое, что в cfg G, $\mathscr{R} = \{A \to \rho_A \mid A \in N,\ \rho_A$ — регулярное выражение над $V = N \cup T,\ N \cap T = \varnothing\}$ [2, 3] — конечное множество EBNF-правил.

Определим множество терминальных цепочек $\lambda(\rho)$ в зависимости от вида регулярного выражения ρ следующим образом:

$$\lambda(\rho) = \begin{cases} \{a\}, & \text{если } \rho = a, a \in T \text{ или } a = \varepsilon. \\ \lambda(\rho_A) & \text{если } \rho = A, A \in N \text{ и существует правило } A : \rho_A. \\ \bigcup\limits_{k=0}^{\infty} \lambda((\rho_1))^k & \text{если } \rho = \rho_1^*. \\ \bigcup\limits_{k=1}^{\infty} \lambda((\rho_1))^k & \text{если } \rho = \rho_1^+. \\ \lambda(\rho_1)\lambda(\rho_2) & \text{если } \rho = \rho_1 \ , \ \rho_2. \\ \lambda(\rho_1) \cup \lambda(\rho_2) & \text{если } \rho = \rho_1 \ ; \ \rho_2. \\ \lambda(\rho_1) \cup \{\varepsilon\} & \text{если } \rho = [\rho_1]. \\ \lambda(\rho_1), & \text{если } \rho = (\rho_1). \end{cases}$$

Здесь ρ_1 и ρ_2 — некоторые регулярные выражения. Определение вида регулярного выражения производится с учётом старшинства операций и расстановки скобок. При этом предполагается, что наивысшее старшинство имеют унарные операции *-Клини и $^+$-Клини, за ними следует конкатенация (,) и, наконец, объединение (;).

Язык, порождаемый EBNF-грамматикой $\mathscr{G} = (N, T, \mathscr{R}, S)$, есть $L = L(\mathscr{G}) = \lambda(\rho_S)$.

Известно, что правила EBNF-грамматики без леворекурсивных правил могут быть представлены множеством синтаксических диаграмм Н. Вирта.

Определение 5. *Синтаксическая диаграмма Н. Вирта* — это направленный граф с узлами двух видов: (1) прямоугольники и (2) круги или овалы, и направленными дугами, изображёнными в виде стрелок, которые связывают узлы.

В прямоугольниках записываются имена металингвистических переменных (имена конструкций языка, нетерминалов), в кружках или овалах — основные символы языка (терминалы). Стрелки определяют возможные пути обхода графа. Путь, ведущий от стартовой дуги (идущей ниоткуда) к целевой дуге (ведущей никуда), фиксирует какой-то *путь порождения* конструкции языка, определяемой данной диаграммой. Последовательность меток узлов, встречаемых на этом пути, то есть цепочка символов грамматики, представляет сентенциальную форму, выводимую из нетерминала, представленного данной диаграммой.

Конечное множество диаграмм Н. Вирта определяет КС-язык посредством рекурсивного обхода многокомпонентного графа, начиная с обхода главной компоненты, то есть диаграммы, соответствующей стартовому нетерминалу грамматики S.

Определение 6. Пусть $D = (N, T, \mathrm{K}, S)$ — система синтаксических диаграмм Н. Вирта, где N — конечное множество конструкций языка (нетерминалов); T — конечное множество основных символов языка (терминалов); $\mathrm{K} = \{\mathrm{K}_A \mid A \in N\}$ — конечное множество компонент K_A, описывающих конструкцию $A \in N$; $S \in N$ — стартовый нетерминал.

Определим язык посредством обхода множества компонент системы K, начиная с главной компоненты K_S, следующим образом.

Пусть K_A — текущая диаграмма и n — текущий узел в ней. Если n — прямоугольный узел, помеченный именем конструкции B, то обходится диаграмма K_B, и затем продолжается обход диаграммы K_A с узла, следующего за n. Если n — круг или овал, то порождается основной символ, помечающий этот узел, и происходит переход по стрелке к следующему узлу в текущей диаграмме. Когда заканчивается обход главной диаграммы выхо-

дом по целевой стрелке, то полученная цепочка есть предложение языка, порождённого системой диаграмм Н. Вирта. Множество всех таких предложений есть язык $L = L(D)$.

Разумеется, система порождения языка диаграммами Н. Вирта должна быть эквивалентной приведённой КС-грамматике без леворекурсивных нетерминалов.

2. СПЕЦИФИКАЦИЯ ТРАНСЛЯЦИЙ В SYNTAX-ТЕХНОЛОГИИ

Методика построения синтаксически управляемых процессоров, используемых в SYNTAX-технологии, основывается на RBNF-грамматиках, то есть EBNF-грамматиках, дополненных ещё двумя алфавитами *контекстных* символов: символами *семантик* и *предикатными* символами, интерпретируемыми в *операционной среде*. Для описания операционной среды используется язык программирования Паскаль[1], и сама она создаётся соответствующим компилятором по выбору в виде модуля или присоединяемой библиотеки (DLL).

По конкретной RBNF-грамматике строятся таблицы, управляющие штатным языковым процессором. Во время его работы, вызываются процедуры, связанные с символами семантик, и логические функции, реализующие предикаты. Первые изменяют состояние операционной среды, а вторые реагируют на эти изменения. Ясно, что не для всяких RBNF-грамматик существует процессор, используемый в SYNTAX-технологии. Ограничения на допустимые грамматики проверяются самой SYNTAX-технологией. Все они проистекают из требования детерминизма процесса анализа как внутренней задачи процессора. Поскольку управляющие таблицы процессора строятся по управляющей граф-схеме, производной из грамматики, то упомянутые ограничения переносятся на управляющую граф-схему. В частности, в грамматике не допускается левосторонняя рекурсия (см. Опр. 2) и неоднозначность вывода (синтаксическая и семантическая).

Язык TSL (the **T**ranslation **S**pecification **L**anguage) является метаязыком для записи спецификаций трансляций и одновременно служит входным языком технологического комплекса SYNTAX.

Спецификация трансляции состоит из трёх разделов.

Описание микролексики (см. раздел **MICROLEXICS** в Примере 1 ниже) определяет классификацию литер по классам. Все члены одного класса воспринимаются правилами грамматики одинаково. На этапе синтаксического анализа микролексические классы используются для представления входного текста вместо самих литер. Описание микролексики одновременно можно рассматривать и как спецификацию транслитератора, для формирования микролексем. *Микролексема* есть запись, поля которой представляют (1) номер микролексического класса (LC), (2) номер члена этого класса (LN) и (3) строковое представление конкретного члена (LR).

Описание синтаксиса (см. раздел **SYNTAX** в Примере 1) задаёт обработку конструкций, распознаваемых процессором-анализатором. Имена конструкций разделяются на нетерминалы и *вспомогательные понятия* (AUXILIARY NOTIONS), которые воспринимаются по-разному во время генерации граф-схемы (см. раздел 3). Те и другие определяются правилами RBNF-грамматики.

Правило RBNF-грамматики состоит из левой и правой частей, которые разделены символом “:” Правило заканчивается символом точки. Левая часть правила представляется нетерминалом или вспомогательным понятием, а правая — регулярным выра-

[1] Выбор инструментального языка был продиктован академическими соображениями.

жением относительно символов всех алфавитов грамматики, включающая семантики и предикаты. Альтернативы отделяются друг от друга символом “;”, а конкатенирующие члены — символом “,”. Необязательные элементы правила включаются в квадратные скобки. Звезда и плюс Клини обозначаются символами “*” и “+”.

Роль терминалов в ней играют микролексические классы, а не отдельные члены класса, определённые в разделе описания микролексики данной спецификации. Именно поэтому имена именованных классов в разделе **SYNTAX** заключается в апострофы.

По описанию синтаксиса подсистема проектирования строит управляющие таблицы процессора, реализующего трансляцию, задаваемую данной спецификацией.

Описание операционной среды (см. раздел **ENVIRONMENT** в Примере 1) определяет семантику входного языка в терминах процедур и функций, ассоциированных с контекстными символами управляющей грамматики (см. раздел **IMPLEMENTATION** в Примере 1), и данных, над которыми они оперируют. По этому описанию Паскаль-компилятор создаёт модуль или DLL, к которому обращается штатный языковый процессор за вызовом семантической процедуры или логической функции во время обработки входного текста.

Пример 1. Спецификация трансляции чисел Алгола 68

```
ALGOL 68 real denotation (9.02.2016)
MICROLEXICS
LEXICAL CLASSES: d, '.', e, '+', '-', Escaped Symbols.
d: '0' .. '9'. { Описание именованного класса, содержащего  все цифры }
e: 'e', '\'. { Описание именованного класса, содержащего символы порядка }
Escaped Symbols: #32, #13, #10, #9. { Описание специального класса
                                      невидимых символов }
SYNTAX
NONTERMINALS: NUMBER.
TERMINALS : 'd', '.', 'e', '+', '-'.
AUXILIARY NOTIONS: INTEGRAL (integral_denotion),
                   FRACTIONAL (fractional_part), STAGNANT  (stagnant_part),
                   EXPONENT (exponent_part),  DIGIT (digit).
FORWARD PASS SEMANTICS: InitNum{ber}, SetNumber, InitInt{egral},
                        SetDig{it}, App{end}Dig{it}, SetInt{egral}Part,
                        InitFr{action}, Incr{ement}Fr{action},
                        InitExp{onent}Part, SetNeg, AddSign,
                        SetExp{onent}Part, ShowNum{ber}.
NUMBER: InitNum{ber}, STAGNANT, [EXPONENT], SetNumber, ShowNum{ber}.
STAGNANT: [INTEGRAL, SetInt{egral}Part], FRACTIONAL; INTEGRAL,
          SetInt{egral}Part.
INTEGRAL: InitInt{egral}, (SetDig{it}, DIGIT, App{end}Dig{it})+.
FRACTIONAL: InitFr{action}, '.', (SetDig{it}, DIGIT, IncrFr{action})+ .
EXPONENT: 'e' , InitExp{onent}Part, (['+']; SetNeg,'-'), INTEGRAL,
          AddSignSetExp{onent}Part.
DIGIT: 'd'.  {d представляет любую цифру}
ENVIRONMENT
var  Number, ExponentPart, Stagnant, k, Fraction : real;
     Integral, IntegralPart, DigitsNmb, Exponent, Digit, Sign : integer;
```

```
{ Auxiliary functions }
 function CurrentSymbol: char;
 var S : string;
 begin S := LR; CurrentSymbol:= S[1] end;
 function Power (A, B: integer): real; {To raise A to power B}
var C : real; i : integer;
begin C := 1; for I := 1 to B do C := C * A ; Power := C end;
IMPLEMENTATION
{ Semantics implementation }
procedure InitNum; { Number value initiation }
begin Stagnant := 0 ; ExponentPart := 1 end;
procedure SetNumber; { Set Number value }
begin Number := Stagnant * ExponentPart end;
procedure ShowNum; { Show Number value }
begin PrintString (' Number = '); PrintReal (Number); NewLine end;
{ --- Integral part --- }
procedure InitInt; { Integral part initiation }
begin Integral := 0; DigitsNmb := 0 end;
procedure SetDig; {Value of digit}
begin  Digit := LN - 1 end;
procedure AppDig; { Integral part incrementation }
begin Integral := Integral * 10 + Digit;  DigitsNmb := DigitsNmb +1 end;
procedure SetIntPart; { Set integral part value}
begin IntegralPart := Integral; Stagnant := IntegralPart end;
{ --- Fraction --- }
procedure InitFr; { Fraction value initiation }
begin Fraction := 0; k := 0.1 end;
procedure IncrFr; { Fraction incrementation }
begin Fraction := Fraction + Digit * k; k := k * 0.1;
      Stagnant :=  IntegralPart + Fraction
end;
{ --- Exponent part --- }
procedure InitExpPart; { Sign initiation }
begin Sign := 1 end;
procedure InitExp;
begin Exponent := Integral end;
procedure SetNeg; { Negate sign }
begin Sign := -1 end;
procedure AddSign;
begin Exponent := Integral * Sign end;
procedure SetExpPart; { Increment number value with exponent part }
begin
 if Exponent >= 0 then ExponentPart := Power ( 10 ,   Exponent )
 else begin ExponentPart := Power ( 10 ,   - Exponent );
            ExponentPart :=1 /  ExponentPart end
end;
```

3. ГЕНЕРАЦИЯ МНОЖЕСТВА КОМПОНЕНТ ГРАФ-СХЕМЫ

По TSL-спецификации в разделе SYNTAX строятся компоненты управляющей граф-схемы. По этому множеству компонент строятся управляющие таблицы штатного языкового процессора.

Определение 7. *Управляющая граф-схема* — это формальная система $\mathscr{G}_c = (N, T, \Re, \Sigma, K, S)$, где $N, T, \Re$ и Σ — конечные множества символов, *нетерминалов*, *терминалов*, *предикатов* и *семантик*, соответственно, $N \cap T \cap \Re \cap \Sigma = \varnothing$, $\mathrm{K} = \{\mathrm{K}_A \mid A \in N, \mathrm{K}_A$ — *компонента* графа, определяющая нетерминал $A\}$, $S \in N$ — начальный нетерминал.

Компоненту K_S назовем *главной компонентой* управляющей граф-схемы. Каждая компонента подобна синтаксической диаграмме Н. Вирта с теми лишь отличиями, что:

1. Каждая компонента управляющей граф-схемы имеет *начальный* (►) и *конечный* (◄) узлы. Путь прохода компоненты начинается с начального узла и заканчивается в конечном узле.
2. Дуги помечаются конечными цепочками контекстных символов из $\Re \cup \Sigma$. Если все дуги управляющей граф-схемы не помечены, то она называется *синтаксической граф-схемой*. Она определяет синтаксис входного языка.
3. В дополнение к узлам, ссылающимся на другие конструкции языка, как принято в диаграммах Н. Вирта, ещё используются прямоугольные элементы, выделенные пунктиром, называемые *вспомогательными понятиями*.

В TSL-спецификации они используются наряду с нетерминалами, но правила, их определяющие, используются иначе, чем правила для нетерминалов. Они играют роль макроопределений. Каждое вхождение вспомогательного понятия в правую часть правила автоматически *раскрывается* во время генерации компонент управляющей граф-схемы.

Пусть вспомогательное понятие B встречается в компоненте A в некотором узле n. Раскрытие B состоит в том, что вся компонента B без её начального и конечного узлов вписывается в упомянутую позицию n. Процесс преобразования компоненты A путём раскрытия вспомогательных понятий продолжается до тех пор, пока таких не остаётся в A.

В результате раскрытия всех применённых вхождений вспомогательных понятий, они вообще не участвуют в определении 7.

Разумеется, самовставления (см. Опр. 3) вспомогательных понятий не допускаются. Из всего множества имён, используемых в TSL-спецификации, вспомогательные понятия выделяются именно по признаку отсутствия самовставления. В предельном случае, когда все нетерминалы, кроме начального, объявляются вспомогательными понятиями, остаётся единственная регулярная компонента граф-схемы, определяющая начальный нетерминал.

На рис. 1 и 2 показаны компоненты управляющей граф-схемы чисел Алгола 68 до раскрытия вспомогательных понятий и после, соответственно.

Дуги управляющей граф-схемы помечены семантическими символами. Им соответствуют процедуры, описанные в разделе **IMPLEMENTATION** спецификации языка чисел Алгола 68.

На рис. 2 кружочком (○) отмечено соединение двух частей диаграммы. В верхней части показаны числа без порядка, а в нижней части — необязательный порядок числа. Как видно, диаграмма представляет регулярную грамматику.

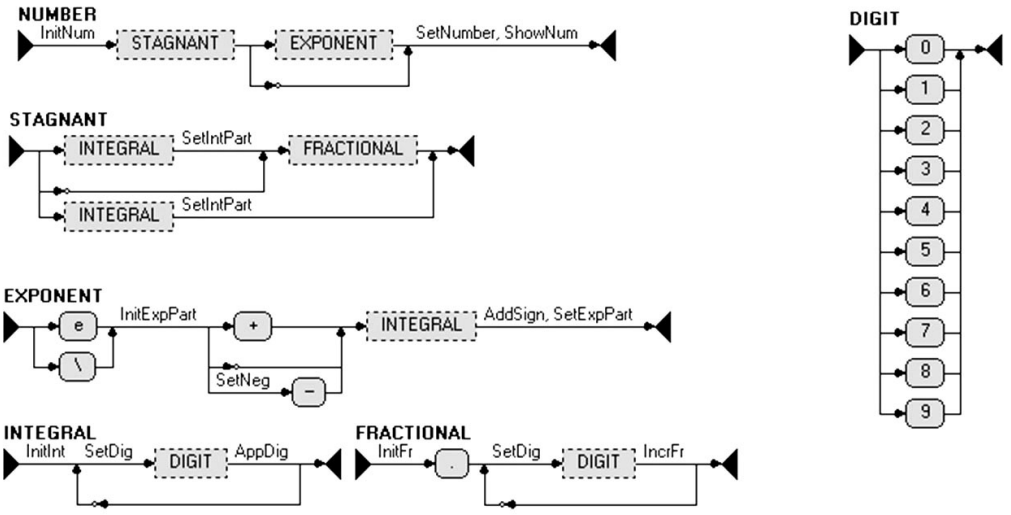


Рис. 1. Компоненты управляющей граф-схемы до раскрытия вспомогательных понятий

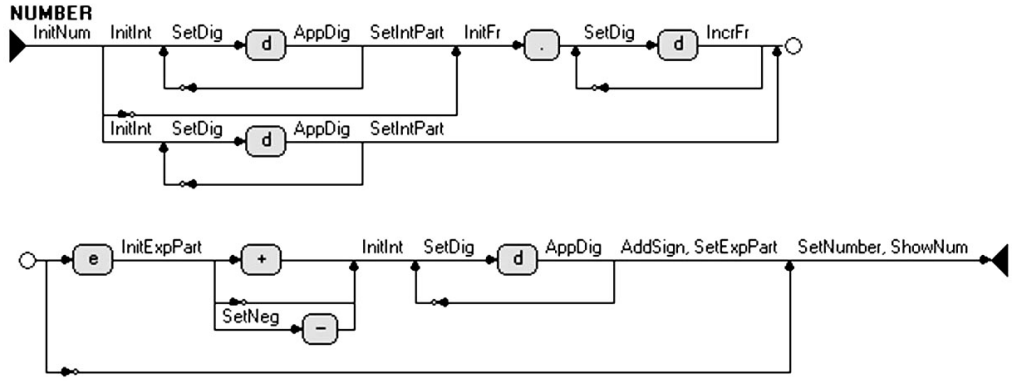


Рис. 2. Однокомпонентная управляющая граф-схема после раскрытия вспомогательных понятий

Управляющая граф-схема, дополненная описанием операционной среды, образует полное определение входного языка, его синтаксис и семантику.

Определение 8. *Трансляционная граф-схема* $\mathscr{G} = (\mathscr{G}_c, \mathscr{E})$ состоит из двух составляющих: управляющей граф-схемы ($\mathscr{G}_c$) и описания операционной среды ($\mathscr{E}$).

Описание операционной среды $\mathscr{E} = (\mathrm{E}, \mathrm{H}, \mathscr{I}_\Re, \mathscr{I}_\Sigma, e_0)$ включает *пространство состояний операционной среды* E (область определения предикатов и преобразований операционной среды), *объектное подпространство* H (часть операционной среды, состояние которой представляет для пользователя особый интерес), *предикаты* $\mathscr{I}_\Re = \{\, \iota_\rho : \mathrm{E} \to \{\, \mathbf{false}, \mathbf{true} \,\} \mid \rho \in \Re \,\}$, ассоциированные с предикатными символами, *преобразования операционной среды* $\mathscr{I}_\Sigma = \{\, \iota_\sigma : \mathrm{E} \to \mathrm{E} \mid \sigma \in \Sigma \,\}$ ассоциированные с символами семантик, *начальное состояние операционной среды* $e_0 \in \mathrm{E}$.

Задача языкового процессора — по входному предложению языка найти маршрут его порождения в граф-схеме и в ходе его нахождения проинтерпретировать контекстные символы в операционной среде. Заметим, что проход по дуге разрешается, если все предикаты, ассоциированные с предикатными символами, помечающими данную дугу, выполняются, и только в этом случае вызываются семантические процедуры, ассоции-

рованные с символами семантик, в порядке их следования в цепочке, помечающей данную дугу.

4. НЕКОТОРЫЕ СВЕДЕНИЯ ИЗ ТЕОРИИ ГРАФОВ И СЕТЕЙ

Переходя к главной теме, объявленной в названии статьи, напомним некоторые определения понятий, используемых в данной статье, следуя [4, 7].

Определение 9. *Граф* есть совокупность $G(X, E)$, где X — множество элементов, называемых *вершинами* (или *узлами*, или *точками*), а E — пары вершин, называемых *рёбрами* (дугами или линиями). Предполагается, что и множество X и множество E содержат только конечное число элементов.

Графически вершины представляются окружностями, а рёбра линиями.

Всякий раз, когда множество E состоит из *неупорядоченных* пар вершин, мы имеем в виду *неориентированный граф*. В неориентированном графе рёбра (x, y) и (y, x) не различаются, и нас интересуют лишь *конечные точки* ребра.

Во многих практических ситуациях требуется определять направление ребра. Направление ребра изображается в графе путём рисования стрелок вместо линий между вершинами. Ориентированные рёбра часто называют *дугами*, а граф называют *ориентированным графом*. Вершины в ориентированных графах принято называть узлами. Множество узлов ориентированного графа будем обозначать символом Q, а множество дуг — символом A.

Всюду здесь под термином граф будет подразумеваться понятие ориентированный граф.

Определение 10. Если (x, y) — направленная дуга, то x называют *начальным*, а y — *концевым* узлом дуги.

Дуга, начальный и концевой узел которой совпадают, называется *петлёй*.

Определение 11. Говорят, что узел и дуга *инцидентны* друг другу, если узел является для этой дуги концевым или начальным узлом.

Определение 12. Будем говорить, что две дуги являются *смежными* друг другу, если обе они инцидентны одному и тому же узлу.

Два узла называются *смежными*, если существует дуга, их соединяющая.

Определение 13. Для ориентированных графов определяется *полустепень входа* в узел как число дуг, направленных к этому узлу, и *полустепень выхода* из узла как число дуг, выходящих из него.

Определение 14. Ориентированный граф $G = (Q, A)$ называется *симметричным*, если в каждую её вершину входит такое же число дуг, как и выходит из неё. Иначе говоря, для всех узлов полустепени входа и полустепени выхода являются равными.

Определение 15. Рассмотрим произвольную последовательность узлов $x_1, x_2, ..., x_n$, $x_{n+1} \in Q$. *Путём* называется любая последовательность дуг $\alpha_1, \alpha_2, ..., \alpha_n \in A$, такая что концевыми точками дуги α_i являются узлы x_i и $x_i + 1$ для $i = 1, 2, ..., n$. Узел x_1 называется *начальным узлом* пути. Узел x_{n+1} называется *конечным узлом* пути. Будем говорить, что путь *соединяет* начальный узел с конечным узлом.

Длина пути совпадает с числом составляющих её дуг.

Определение 16. Путь, в котором $\alpha_i = (x_i, x_{i+1})$ при всех $i = 1, 2, ..., n$, представляет собой *ориентированный* путь. В ориентированном пути все дуги имеют одно и то же направление от начального узла до конечного узла.

Определение 17. *Цикл* — это путь, в котором начальная вершина и конечная вершина совпадают. *Ориентированный* цикл — это ориентированный путь, в котором начальный узел и конечный узел совпадают.

Определение 18. Замкнутый маршрут прохождения симметричного графа от исходной точки с возвратом в неё же называется *эйлеровым маршрутом*, или *циклом*.

Определение 19. Граф называют *ациклическим*, если он не содержит циклов. Ориентированный граф называют *ациклическим*, если он не содержит ориентированных циклов.

Определение 20. Путь или цикл (ориентированный или нет) называют *простым*, если никакой узел не инцидентен больше чем двум его дугам (то есть если путь или цикл, соответственно, не содержат циклов).

Определение 21. Граф называется *связным*, если в нём для каждой пары узлов найдётся соединяющий их путь.

Граф может состоять из нескольких связных графов. Каждый из этих связных графов называют его *компонентом*.

Определение 22. Пусть $G = (Q, A)$ — ориентированный граф. Если $u, v \in Q$, то (u, v) — дуга из узла u (начало дуги) в узел v (конец дуги).

Его *рёберный орграф* [7] определяется как орграф $G' = (Q', A')$, вершинами которого являются дуги G, причём две вершины $uv, wx \in Q'$, соответствующие дугам (u, v) и (w, x) графа G, связаны дугой из узла uv в узел wx в рёберном орграфе G', когда $v = w$. Таким образом, каждая дуга в рёберном орграфе соответствует пути длиной 2 в исходном графе.

Определим *итерированный реберный граф* $L^0(G) = G$, $L^1(G) = L(G)$, а в общем случае рекуррентным соотношением $L^n(G) = L(L^{n-1}(G))$, $n \geqslant 2$.

На рис. 3 показаны исходный граф G, G' — рёберный граф и G'' — итерированный один раз рёберный граф для регулярного выражения ba^*. Заметим, что однокомпонентный граф G в этом процессе его преобразования распадается на две компоненты в графе G''. *Тривиальная компонента* графа G'' состоит из одного изолированного узла β_1. Этот узел считается *начальным* и *конечным* узлом *тривиальной компоненты*. Прообраз тривиальной компоненты будем называть *свободной (free) дугой*. Прообраз узла β_1 есть дуга (α_1, α_2) графа G'. В то же время прообраз узла α_1 есть дуга $(\text{Н}, b)$, а прообраз узла α_2 есть дуга $(b, \text{К})$. Обе дуги являются элементами графа G.

В другой компоненте графа G'' узел β_2 является её *начальным узлом* (в него не входит ни одна дуга), а β_4 и β_6 являются *конечными узлами* (из них не выходит ни одна дуга).

Заметим, что если исходный граф не содержит циклов или петель, то число итераций имеет конечный предел.

Определение 23. Пусть $G = (Q, A)$ — однокомпонентный связанный граф, а $G' = (Q', A')$ — рёберный граф графа G, и пусть дуге $(\alpha_1, \alpha_2) \in A$ соответствует узел $\beta' \in Q'$. Запишем это в виде $L((\alpha_1, \alpha_2)) = \beta'$. То, что дуга (α_1, α_2) есть прообраз узла β', будем записывать как $\beta' = L^{-1}((\alpha_1, \alpha_2))$. Тем самым мы определили отображение L^{-1}, обратное к преобразованию L^1, которое можно относить к каждому узлу графа G'.

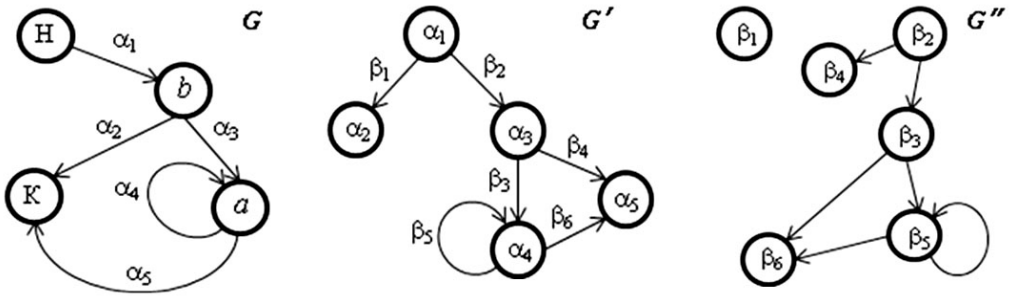


Рис. 3. Итерированные графы степени 0, 1 и 2 для регулярного выражения ba^*

Далее, можно ввести рекурсивное определение обратного отображения для любой степени $n \geqslant 1$, если положить для всех $1 \leqslant k \leqslant n : L^{-k}(G) = L^{-1}(L^{-(k-1)}(G))$, если $k \geqslant 1$.

Определение 24. *Сеть* — это граф, с каждым элементом (узлом или дугой) которого сопоставлено одно или более чисел.

Эти числа могли бы представлять расстояния, затраты или другие параметры.

Поток — это способ отправить объекты из одного места сети в другое. Объекты, которые перемещаются или текут через сеть, называют *единицами потока*. Узлы, из которых единицы начинают свой путь, называют *узлами-источниками*. Узлы, к которым единицы направляются, называются *узлами-стоками*. У узлов-источников обычно есть *поставка*, которая представляет число единиц, доступных в узле. У узлов-стоков обычно есть *требование*, представляющее число единиц, которые должны быть направлены им.

В задаче о потоке каждой дуге часто приписывается *пропускная способность*. *Пропускная способность дуги* — это максимальный допустимый поток, который может переместиться по ней. Пропускные способности дуг — это просто верхние границы, выраженные в единицах потока.

Фундаментальные уравнения, управляющие потоками в сетях, называются сохранением потока. То есть утверждается, что сохранение потока устанавливает в каждом узле, что

$$\text{Выходной поток} - \text{Входной поток} = 0$$

Поставку в узле-источнике рассматривают как «входной поток»; требование в узле-стоке рассматривают как «выходной поток». Поток называют *выполнимым*, если он удовлетворяет ограничениям сохранения потока и пропускным способностям дуг. Так как уравнения сохранения потока и ограничения пропускных способностей дуг линейны, то задачи о потоке включают линейное программирование.

5. ПОСТАНОВКА ЗАДАЧИ ТЕСТИРОВАНИЯ ПРОЦЕССОРОВ РЕГУЛЯРНЫХ ЯЗЫКОВ

В SYNTAX-технологии конечные и магазинные автоматы используются как управляющие устройства (engines) при реализации трансляции регулярных или КС-языков. Эти автоматы строятся по RBNF-грамматикам с ограничениями, гарантирующими детерминизм реализации трансляций. Сами эти преобразования распределяются по дугам компонент граф-схем, представляющих эти грамматики. Именно проход по дугам компонент предполагает изменение состояния окружения, в котором происходит процесс трансляции.

Рассмотрим задачу генерации регулярных тестов по однокомпонентной граф-схеме с семантиками, но без предикатов. Эти тесты используются для тестирования конечного процессора, управляющего трансляцией, который строится по этой же самой граф-схеме. Задача его тестирования как распознавателя не стоит, поскольку его корректность гарантируется алгоритмами его построения[2]. Остаётся только проверить правильность результата согласованного взаимодействия семантик между собой в последовательности, определяемой входной RBNF-грамматикой.

Минимальный по длине тест, гарантирующий покрытие всех дуг граф-схемы, назовем *оптимальным тестом степени* 0.

Для того чтобы тесты охватывали более широкий диапазон взаимодействия семантик, можно использовать тот же самый подход в применении к рёберным, точнее итерированным n раз графам $G_n = L^n(G), n \geqslant 1$, а затем отображать полученное решение на оригинальный прообраз исходного графа G. Исходя из этой идеи, мы естественным образом приходим к следующему критерию выбора тестовых вариантов, выражающему заданную степень взаимовлияния семантик через операционную среду.

Пусть задано некоторое целое $n \geqslant 0$. *Тест* степени n должен активизировать как минимум однократное исполнение каждой возможной $(n+1)$-и при минимальной суммарной длине составляющих его тестовых вариантов. Это значит, что при $n = 0$ тестом должна быть испытана каждая семантика в допустимой грамматикой последовательности вызовов по крайней мере по одному разу. При $n = 1$, кроме того, по крайней мере, по разу должно быть проверено взаимодействие между каждой возможной парой семантик и т. д.

Минимальный по длине тест, удовлетворяющий этому критерию при заданном значении n, назовем *оптимальным тестом порядка* n.

Итак, наша задача состоит в том, чтобы построить алгоритмы генерации оптимального теста степени $n \geqslant 0$ по заданному однокомпонентному орграфу[3]. В Приложении приведены примеры, иллюстрирующие метод, реализуемый алгоритмами, описываемыми в следующем разделе.

В дальнейшем всюду, где ради краткости будут употребляться термины *граф* или *орграф*, следует подразумевать ориентированный помеченный *псевдомультиграф*.

Тест состоит из множества тестовых вариантов. Тестовый вариант это последовательность меток узлов маршрута, проходящего от начального до конечного узла графа.

Если исходный граф не содержит циклов или петель, то тест имеет фиксированную длину, и для его получения достаточно получить след замкнутого маршрута в исходном графе[4]. В противном случае решается задача китайского почтальона[5] на итерированном графе заданной степени $n \geqslant 1$. Нетривиальные компоненты, оставшиеся после достижения заданной степени итерации, *связываются* посредством введения *главного начального* и *главного конечного узлов*. Из главного начального узла проводятся направленные дуги в начальные узлы этих компонент, а все их конечные узлы соединяются с главным конечным узлом.

[2] Однако для конечных распознавателей, полученных из других источников, тестирование актуально.

[3] Который, строго говоря, было бы уместнее назвать *ориентированным псевдомультиграфом*, поскольку в общем случае он может иметь петли и параллельные дуги.

[4] Ибо в этом случае итерирование вырождается в пустой граф, а тривиальные компоненты, появляющиеся в процессе итерирования в проекции на исходный граф, дают тот же самый результат, что исходный граф без итерирования.

[5] Первая публикация, посвящённая решению подобной задачи, появилась в одном китайском журнале, и потому её иногда называют задачей китайского почтальона.

Задача китайского почтальона заключается в том, чтобы, отправляясь из начальной точки, пройти все дуги маршрута и вернуться в начальную точку, минимизируя длину пройденного пути. В качестве начальной точки берётся главный начальный узел графа. Для возврата в начальную точку проводится дополнительная *возвратная дуга* из главного конечного узла графа в его начальный узел.

Будем считать, что все дуги имеют одинаковую длину или стоимость прохождения по ним, равную 1.

Если граф G — *симметричный*, то задача почтальона всегда имеет решение. Оно есть эйлеров цикл в G.

Когда граф G — *несимметричный*, то предлагается дополнить граф G минимальным множеством дуг, параллельных исходным, чтобы получить симметричный граф G^*. Это дополнение сводится к нахождению потока минимальной стоимости, то есть маршрута минимальной длины. Найденный поток определяет число дополнительных параллельных дуг, превращающих исходный граф в симметричный, в котором всегда существует эйлеров цикл.

Пусть $f(x,y)$ обозначает число повторных обходов дуги (x,y). Необходимо выбрать такие неотрицательные значения целых переменных $f(x,y)$, которые минимизируют общее число повторно проходимых дуг

$$\sum_{(x,y)\in A} f(x,y) \tag{1}$$

при условии, что в каждый узел почтальон входит столько раз, сколько выходит из него, то есть

$$d^-(x) + \sum_{y\in Q} f(x,y) = d^+(x) + \sum_{y\in Q} f(x,y), \tag{2}$$

где $d^-(x)$ обозначает число дуг, входящих в узел x, а $d^+(x)$ — число дуг, выходящих из узла x.

Уравнение 2 можно переписать следующим образом

$$\sum_{y\in Q} \left[f(y,x) - f(x,y) \right] = d^-(x) - d^+(x) = -D(x). \tag{3}$$

Таким образом, необходимо минимизировать выражение 1 при условии, что для всех узлов графа G удовлетворяется равенство 3. Эта задача представляет одну из модификаций задачи о потоке минимальной стоимости, относящейся к области линейного программирования.

Узлы, для которых $-D(x) < 0$, называются *стоками* с предельным значением суммарного выходящего потока, равным $-D(x)$. Узлы, где $-D(x) > 0$, называются *источниками* с предельным значением суммарного входящего потока, равным $D(x)$. Узлы, для которых $D(x) = 0$, являются *промежуточными* узлами. Значения пропускных способностей всех дуг графа G не ограничены.

Сформулированная задача о потоке минимальной стоимости может быть решена путем введения в граф G дополнительно *главного источника* и *главного стока*.

Так как правые части всех равенств 3 являются целыми числами, то с помощью алгоритма 1 (см. раздел 6 ниже) получаются неотрицательные целые значения $f(x,y)$, то есть дополнительные кратности дуг графа G. В графе G^* дуга (x,y), соединяющая вершины x и y для всех $(x,y) \in A$, повторяется $f(x,y)+1$ раз. В соответствии с равенством 3 такой граф G^* является симметричным. Эйлеров цикл в графе G^* соответствует оптимальному маршруту почтальона на графе G.

Построение эйлерова цикла в графе G^* реализуется Алгоритмом 3 из раздела 6. В нём используется Алгоритм 4 для нахождения контуров, которые сливаются в один общий цикл — цикл Эйлера.

Для построения теста степени $n = 0$ достаточно получить след узлов эйлерова цикла. Он состоит из участков, разделённых возвратными дугами. В то время как *след узлов* одного такого участка представляет тестовый вариант, *след* составляющих его *дуг* образует цепочку преобразований, активизируемых данным тестовым вариантом.

В случае $n > 0$ для получения теста необходимо отобразить узлы эйлерова цикла посредством обратного отображения L^{-n} и получить след прообраза маршрута степени 0. Кроме того, если в процессе итерирования образовались изолированные узлы, то есть тривиальные компоненты, то их тоже нужно отобразить посредством обратного отображения L^{-k}, где k — та степень итерации, на которой образовался изолированный узел. Мы исходим из того, что все дуги прообраза тривиальной компоненты всегда различны[6]. След каждого такого пути также составляет вариант теста степени n.

6. АЛГОРИТМЫ, КАСАЮЩИЕСЯ ГЕНЕРАЦИИ РЕГУЛЯРНЫХ ТЕСТОВ

Прежде чем описывать алгоритмы, реализующие генерацию оптимальных тестов, зафиксируем общий порядок генерации тестов минимальной длины относительно регулярной граф-схемы.

1. Построение исходного графа по заданной однокомпонентной граф-схеме.
2. Итерирование исходного графа до степени n, где n — степень теста[7].
3. *Нетривиальные* компоненты последней итерации *связываются* посредством введения *главных начального* и *конечного* узлов.
4. Дополнение графа, полученного в п. 3, до симметричного, за счёт определения *дополнительной кратности дуг* путём решения потоковой задачи на сети, полученной на шаге 3 (см. алгоритм 1). Величина потока на дуге принимается за её дополнительную кратность.
5. Нахождение *эйлерова цикла* в симметричном графе.
6. Построение *образа эйлерова цикла* при обратном отображении L^{-n} в исходном графе[8].
7. Построение *следа образа эйлерова цикла* в качестве теста порядка n.

В алгоритме 1 исходный граф, полученный в результате выполнения п.п. 1–3, называется *хорошо сформированным орграфом*.

Обоснование Алгоритма 1

Заметим, что исходный граф генерируется по регулярному выражению, имеющую «правильную» (well-formed) структуру:

1) он связный;
2) в начальный узел не входит ни одна дуга;
3) из конечного узла не выходит ни одна дуга;
4) все пути, начинающиеся в начальном узле, заканчиваются в конечном узле;
5) все петли и циклы продуктивны, то есть порождают непустые цепочки.

[6] Фактически используются свободные дуги, прообразы изолированных узлов.

[7] При $n = 0$ реберный граф не строится. В этом случае считается, что исходный граф и есть последняя итерация.

[8] С учётом свободных дуг, появляющихся при итерациях в случае, когда $n > 0$.

Algorithm 1: нахождение потока минимальной стоимости

Вход: $G = (Q, A)$ — хорошо сформированный орграф.

Выход: Потоки $\{f(x,y)\}_{(x,y)\in A}$, такие что $\sum_{(x,y)\in A} f(x,y) = \min$.

Метод: Алгоритм выполняется по следующим шагам:

1. Определить источники и стоки: $x \in Q$ — источник, если $D(x) > 0$; x — *сток*, если $D(x) < 0$. Если их не существует, то исходный граф — симметричный. Алгоритм завершён. Иначе

2. Провести дополнительную дугу из главного конечного узла в главный начальный узел, называемую *возвратной дугой*.

3. Удалить из исходного орграфа G все петли (поскольку каждая петля одновременно является входной и выходной дугой, и её удаление не изменяет отношения между числом входящих и числом выходящих дуг).

4. Построить главный *источник* S и соединить его ориентированными дугами со всеми имеющимися источниками; построить главный сток T и провести в него ориентированные дуги из всех имеющихся стоков.

5. Задать пропускные способности крайних дуг: $c(S,x) = D(x)$, если x — источник, $c(x,T) = -D(x)$, если x — сток. Пропускные способности всех других дуг не ограничены. Таким образом, теперь мы имеем дело с сетью.

6. Задать для начала на всех дугах сети нулевой поток: $f(x,y) = 0$.

7. «Окрасить» главный источник S.

8. Выполнить процедуру окрашивания вершин и дуг сети (см. алгоритм 4 далее).

9. Если главный сток T окрасился, то найти маршрут M из S в T, составленный из окрашенных дуг, и перейти к шагу 11. Иначе

10. Увеличить на единицу узловые числа (см. Обоснование Алгоритма 1 ниже) для всех неокрашенных вершин: $p(x) = p(x) + 1$, и перейти к шагу 8.

11. Определить величину изменения имеющегося потока вдоль маршрута M: $\Delta f = \min\{\min_{a\in I}(i(a)), \min_{a\in R}(r(a))\}$, где $i(a) = c(a) - f(a)$; $r(a) = f(a)$. При этом считается, что $a = (x,y) \in I$, если $(x,y) \in M$ и $f(x,y) < c(x,y)$; и $a = (x,y) \in R$, если $(y,x) \in M$ и $f(x,y) > 0$.

12. Изменить поток на каждой дуге маршрута M: $f(x,y) = f(x,y) + \Delta f$, если дуга $(x,y) \in I$; $f(x,y) = f(x,y) - \Delta f$, если дуга $(x,y) \in R$.

13. Проверить насыщенность крайних дуг. Дуга (x,y) считается *насыщенной*, если $f(x,y) = c(x,y)$. Если не все крайние дуги насыщены, сбросить окраску всех вершин сети, кроме вершины S, и перейти к шагу 8. В противном случае алгоритм завершен. Полученный поток f является *потоком* минимальной стоимости.

Рассмотрим состояние сети, после того как выполнено 6 первых подготовительных шагов алгоритма. В этот момент $f(x,y) = 0$ на всех дугах сети. Процесс окрашивания узлов и дуг сети, запускаемый на шаге 7 и выполняемый алгоритмом 2, начинается от главного источника и выполняется с целью идентификации самого короткого пути до главного стока.

Приращение узловых чисел на шаге 10 имитирует волну, распространяющуюся с равной скоростью от главного источника по всем направлениям от главного источника к главному стоку с учётом текущих значений потока и пропускных способностей дуг. За конечное число шагов волна достигает главного стока по самому короткому пути M.

Поскольку стоимость пропуска единицы потока по дуге равна 1, то стоимость прохождения единицы по всему пути минимальной длины тоже минимальна.

В момент достижения волной главного стока важно согласовать возможности конкретного поставщика (узла-источника, смежного с начальной дугой пути M) с потребностями потребителя (узла-стока, смежного с конечной дугой пути M). Это делается на шаге 11. Учитывается возможное существование прямых (в направлении стрелок) и реверсивных (против направления стрелок) потоков в найденном пути. Если поставщик способен послать равное или большее число единиц, чем потребитель может потребить, то можно пересылать то количество единиц, которое требуется потребителю (то есть разность между величиной потока и пропускной способностью дуги, смежной с узлом-стоком). Если же поставщик не способен удовлетворить потребителя, то можно увеличить поставку в той мере, которая требуется потребителю, за счёт сокращения ненулевых реверсивных потоков (потоков на дугах, окрашенных против стрелок), если они существуют в этом пути. Чтобы согласовать эти разные возможности увеличения потока на дугах пути M, берётся минимум Δf из двух величин. Первая величина вычисляется как минимум разностей между величиной потока на каждой прямой дуге пути M и пропускной способностью дуги, смежной с узлом-стоком. Вторая величина вычисляется как минимум реверсивных потоков на пути M.

На шаге 12 поток увеличивается на величину Δf на каждой дуге M, ориентированной в сторону главного стока, и уменьшается в противном случае. Для изменения потока на пути M он проходится от главного стока до главного источника в обратном направлении.

Напомним, что, хотя на всех дугах сети, кроме крайних, поток не ограничен, текущие значения потока на дугах сети на всё время исполнения алгоритма 1 не превосходят суммарного значения пропускных дуг, смежных с главным источником или с главным стоком (см. равенство 3 в Разделе 5).

В терминах модели почтовой службы узлы-источники x играют роль отправителей корреспонденций числом, равным $D(x) > 0$. Вся корреспонденция должна быть доставлена соответствующим адресатам, то есть узлам-стокам. А поскольку все отправители связаны с главным источником, а все получатели связаны с главным стоком, то вся корреспонденция, отправляемая из главного источника, должна прибыть в главный сток. Другими словами, все отправления из главного источника должны доходить до главного стока, и только они. В этом и состоит условие сохранения потока.

Если в какой-то момент дуга a, соединяющая главный источник S с каким-то источником, насыщается, то есть $f(a) = c(a)$, то это означает, что вся корреспонденция соответствующего отправителя уже доставлена его адресату. Если это случится со всеми источниками-отправителями, то это означает конец всей пересылки. В этот момент в силу сохранения потока все стоки тоже оказываются насыщенными, и алгоритм 1 совершается.

Принимая во внимание, что каждая простая цепь от главного источника к главному стоку является самой короткой по длине, причём стоимость перемещения одной единицы потока по ней равна её длине, а по ней перемещается $|\Delta f|$ единиц, мы заключаем, что полученный итоговый поток цепи имеет минимальную стоимость.

Заметим, что эти цепи разве лишь увеличиваются по длине в ходе выполнения Алгоритма 1.

Значения потока на всех не крайних дугах представляют минимальные дополнительные кратности дуг исходного графа G, превращающие его в *симметричный* граф G^*.

Можно показать, что этот алгоритм имеет сложность по времени $O(kn^2)$, где n — число дуг графа G. Постоянная k связана со структурой сети; для редких сетей коэффициент k мог быть намного меньшим числа узлов и дуг исходного графа.

Algorithm 2: окрашивание сети

Вход: Сеть с заданным на ней потоком f, пропускными способностями дуг c и узловыми числами p.

Выход: Множество окрашенных узлов C_Q и множество окрашенных дуг C_A.

Метод: Алгоритм состоит из следующих шагов:

1. Полагается для начала $C_Q = \{S\}, C_A = \varnothing$, то есть окрашивается главный источник.

2. Над каждой дугой сети $(x, y) \notin C_A$ производятся следующие действия:
если $x \in C_Q, y \notin C_Q, p(y)–p(x) = 1, f(x, y) < c(x, y)$,
то $C_Q = C_Q \cup \{y\}$ и $C_A = C_A \cup (x, y)$.

3. Шаг 2 повторяется до тех пор, пока не будут просмотрены все дуги $(x, y) \in C_A$.

Algorithm 3: нахождение эйлерова цикла

Вход: Симметричная сеть G с начальным узлом H и конечным узлом K.

Выход: C — искомый эйлеров цикл.

Метод: Предполагается, что на данной симметричной сети заданы кратности всех дуг $k(a) = f(a) + 1$, где $f(a)$ — поток, протекающий по дуге a.

Эйлеров цикл строится по следующим шагам:

1. Найти какой-нибудь цикл с началом в узле H: $C = \text{cycle(H)}$[9].

2. Если существуют узел $x \in C$ и дуга (x, y), такая, что $k(x, y) > 0$, то перейти к шагу 3. Иначе алгоритм завершён.

3. Найти какой-нибудь цикл C_x с началом в узле x: $C_x = \text{cycle}(x)$.

4. Присоединить цикл C_x к циклу C в узле x: $C = \text{merge}(C, C_x)$[10] и перейти к шагу 2.

Используемая в алгоритме 3 вспомогательная функция cycle, может быть реализована посредством алгоритма 4.

Остаётся пояснить выполнение пунктов 2, 3, 6 и 7 из раздела 6, собственно связанные с генерацией тестов.

Итерирование исходного графа до степени n (см. п. 2) состоит из n шагов, на каждом строится рёберный граф по отношению к предыдущему, начиная с исходного графа G. При этом каждая дуга графа G_i отображается в узел следующего графа G_{i+1}, $0 \leqslant i < n$. Узлы графа G_{i+1} помечаются, например, номерами дуг графа G_i. Узлы графа G_{i+1} соединятся в последовательности их прообразов в G_i. Вместо выпадающих изолированных узлов графа G_{i+1} лучше использовать их прообразы, то есть свободные дуги графа G_i (см. Пример 2, табл. 3 в Приложении).

Нетривиальные компоненты последней итерации связываются (см. п. 3) посредством введения главных начального и конечного узлов. Первый соединяется

[9] См. далее алгоритм 4.

[10] Очевидно, что функция merge не нуждается в дополнительных комментариях.

Algorithm 4: поиск цикла в графе

Вход: Симметричная сеть и некоторый её узел x.

Выход: C — некоторый цикл с началом в заданном узле x.

Метод: Предполагается, что заданы кратности всех дуг сети. Искомый цикл строится по следующим шагам:

1. Инициируется цепь C, содержащая единственную вершину x, которая запоминается как последняя вершина цепи: $l = x$.

2. Если в сети существует дуга (l, y) кратности $k(l, y) > 0$, то перейти к шагу 3. Иначе цикла, включающего вершину x, не существует — данная сеть не симметрична (но это не наш случай).

3. Присоединить вершину y к цепи C;
уменьшить кратность дуги (l, y): $k(l, y) = k(l, y)–1$;
запомнить её в качестве последней вершины цепи: $l = y$.

4. Если $l \neq x$, то перейти к шагу 2.
Иначе алгоритм завершается, и C — искомый цикл.

с начальными узлами компонент, а во второй проводятся дуги, исходящие из их конечных узлов.

Построение образа эйлерова цикла (см. п.п. 6, 7) и свободных дуг при их обратном отображении происходит подобным образом: метка узла заменяется на дугу, помечающую этот узел.

Последовательность образов следов дуг, получающихся при обратном отображении L^{-n} собственно эйлерова цикла, даёт часть вариантов теста степени n. Остальные получаются при обратном отображении свободных дуг (см. пример 3 в Приложении).

Замечание 1. Описанный алгоритм обеспечивает одно из возможных минимальных покрытий множества допустимых комбинаций из $m (1 \leqslant m \leqslant n)$ семантик.

Замечание 2. Если исходный граф G не имеет циклов и петель, то при всех $n \geqslant 0$ множества T_n одинаковы, то есть тесты всех степеней, построенные по нему, равны между собой.

Замечание 3. Очевидно, что, если тестирование степени n $(n > 0)$ проведено, нет смысла проводить тестирование степени k при $k < n$, поскольку тест степени n удовлетворяет критерию полноты степени k.

ЗАКЛЮЧЕНИЕ

Данная статья не ставит целью конкурировать с другими инструментами работы с грамматиками или языковыми средствами (grammarware tools and language workbenches).

Метод генерации минимальных тестов основан на триедином представлении регулярных множеств посредством *конечных автоматов, регулярных выражений и грамматик типа* 3. Конечный автомат M используется как прототип тех устройств, для которых ставится задача тестирования.

Пусть этот автомат распознаёт язык L, то есть $L = T(M)$. Регулярное выражение, представленное в форме синтаксической диаграммы Н. Вирта, понимается как вырожденный случай EBNF-грамматики G, определяющий язык $L = T(G)$. С одной стороны, по грамматике G строится конечный автомат M, распознающий язык $L = T(M)$, а с другой, — варианты теста для M генерируются по синтаксической диаграмме K_S, такой, что $L = L(K_S)$.

Казалось бы, для чего нужно тестирование, если теория регулярных языков и инструмент, правильно построенный по этой теории, гарантирует эквивалентность автомата M и грамматики G? Однако, эта эквивалентность — *синтаксическая*. Нужны *семантические* тесты, направленные на проверку правильного взаимодействия семантик под управлением синтаксической структуры языка, определяемой грамматикой. Данная статья касается именно проверки адекватной реализации *семантики* регулярных языков.

Наглядность синтаксических диаграмм Н. Вирта подсказала естественный критерий полноты тестирования в терминах покрытия дуг синтаксической граф-схемы. Учёт диапазона контекста взаимодействия частичных преобразований, выполняемых по ходу сканирования входного текста, естественно привёл к идее распределения тестов по степеням. Использование итерированных рёберных графов при применении того же самого алгоритма Китайского почтальона даёт желаемый результат.

Если цель — найти минимальный по длине тест для семантической проверки детерминированного конечного процессора, минимального по числу состояний, то необходимо применять метод генерации теста к графу переходов процессора, а не к графу, представляющему регулярное выражение, по которому строился этот процессор.

В Приложении приводятся примеры, иллюстрирующие вышеизложенную технику генерации тестов.

ПРИЛОЖЕНИЕ. Примеры генерации регулярных тестов

Пример 1: генерация регулярного теста для чисел без знака в языке Алгол 68

Проиллюстрируем результат генерации теста степени 0 по управляющей граф-схеме чисел Алгола 68, приведённой на рис. 2, и по диаграмме переходов минимального конечного процессора, полученного из этого же источника.

а) Генерация теста степени 0 по управляющей граф-схеме

На рис. 4 изображена сеть со уже вычисленными значениями потока на всех дугах. Если значение потока на дуге не указано, то имеется в виду нулевой поток. Показаны дополнительные главные начальный Start (–1) и конечный Finish (–2) узлы и возвратная дуга между ними. Остальные узлы и дуги принадлежат исходной граф-схеме. Узлы-источники (2, 6, 8) выделены окружностями средней толщины, узлы-стоки (0, 3, 4, 5) — пунктирными окружностями, остальные узлы промежуточные. Они изображены окружностями с тонкими линиями. Главные источник Source (–4) и сток Sink (–3) и проведены утолщёнными линиями, так же как и смежные с ними дуги.

Как обычно, дуги представлены парой узлов, с учётом её ориентации. Согласно алгоритму почтальона, для вычисления потока в сети, её дуги подлежат окраске, и только окрашенные дуги могут составлять простую цепь, увеличивающую поток. Для нахождения такой цепи используется дискретный волновой процесс, синхронизируемый пошагово увеличивающимися на одну единицу узловыми числами. Волны расходятся от

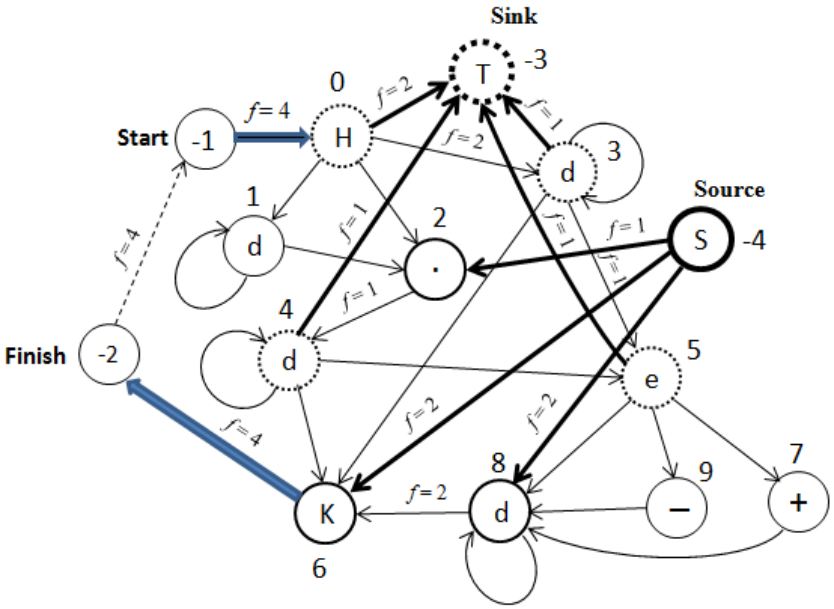


Рис. 4. Исходный граф компоненты NUMBER граф-схемы

главного источника к главному стоку. Когда волна доходит до главного стока, определяется простая цепь, увеличивающая поток.

Процесс нахождения таких цепей представлен в табл. 1. Таких простых цепей в рассматриваемом примере четыре. Для каждой цепи в табл. 1 представлен процесс окрашивания дуг (колонка Окр.) с указанием предельного значения узловых чисел и величины приращения потока (колонка Δf). Окрашенные дуги отмечены индикаторами: ❶ — дуга не входит в цепь, ❷ — дуга прямая ($\Delta f > 0$) и ❸ — дуга реверсивная ($\Delta f < 0$). В рассматриваемых здесь примерах этот последний случай не встречается.

Итоговое значение потока на дугах показано в колонке flow.

Все дуги имеют неограниченную пропускную способность (колонка c), за исключением крайних дуг, то есть дуг, смежных с главными источником (23, 27, 28) и со стоком (22, 24, 25, 26). Табл. 1 показывает, что процесс вычисления потока в сети (рис. 4) заканчивается по условию насыщения крайних дуг: c = flow.

Эйлеров цикл, построенный по элементам собственно исходной граф-схемы, включающей узлы –2, –1, 0–6, с учётом дополнительных кратностей дуг, представленных значениями потоков на дугах, имеет следующий вид:

$-1 \to 0 \to 3 \to 5 \to 9 \to 8 \to 8 \to 6 \to -2 \to$

$-1 \to 0 \to 3 \to 6 \to -2 \to$

$-1 \to 0 \to 3 \to 3 \to 5 \to 8 \to 6 \to -2 \to$

$-1 \to 0 \to 2 \to 4 \to 4 \to 6 \to -2 \to$

$-1 \to 0 \to 1 \to 1 \to 2 \to 4 \to 5 \to 7 \to 8 \to 6 \to -2$

$\to -1$

Таблица 1

			Цепь 1		Цепь 2		Цепь 3		Цепь 4		
№	Дуга	c	Окр. 3	$\Delta f = 1$	Окр. 5	$\Delta f = 2$	Окр. 7	$\Delta f = 1$	Окр. 8	$\Delta f = 1$	flow
0	(0,1)				❶		❶		❶		
1	(0,2)								❶		
2	(0,3)				❶		❷	1	❷	1	2
3	(1,2)										
4	(1,1)										
5	(2,4)		❷	1					❶		1
6	(4,5)		❶								
7	(4,6)										
8	(4,4)										
9	(3,5)						❶		❷	1	1
10	(3,6)										
11	(3,3)										
12	(5,7)								❶		
13	(5,8)										
14	(5,9)								❶		
15	(7,8)										
16	(9,8)										
17	(8,6)						❷	1	❷	1	2
18	(8,8)										
19	(–1,0)				❷	2	❷	1	❷	1	4
20	(6,–2)		❶		❷	2	❷	1	❷	1	4
21	(–2,–1)		❶		❷	2	❷	1	❷	1	4
22	(0,–3)	2			❷	2					2
23	(–4,2)	1	❷	1							1
24	(3,–3)	1					❷	1			1
25	(4,–3)	1	❷	1							1
26	(5,–3)	1							❷	1	1
27	(–4,6)	2	❶		❷	2					2
28	(–4,8)	2	❶		❶		❷	1	❷	1	2

След пути, то есть последовательности меток узлов эйлерова цикла, разделённых вхождениями экземпляров возвратной дуги (–2, –1), представляет 5 вариантов теста степени 0:

(1) d e – d d (2) d (3) d d e d (4) . d d (5) d d . d e + d

Длина этого теста 20 лексем. Табл. 2 показывает тест степени 0 в виде пяти управляющих цепочек, которые помимо символов языка, содержат ещё и семантические символы, активируемые этими вариантами.

Таблица 2. Тест уровня 0 для чисел Алгола 68

№	Управляющая цепочка
1	{ InitNum, InitInt, SetDig } **d** { AppDig, SetIntPart } **e** { InitExpPart, SetNeg } – { InitInt, SetDig } **d** { AppDig, SetDig } **d** { AppDig, AddSign, SetExpPart, SetNumber, ShowNum }
2	{ InitNum, InitInt, SetDig } **d** { AppDig, SetIntPart, SetNumber, ShowNum }
3	{ InitNum, InitInt, SetDig } **d** { AppDig, SetDig } **d** { AppDig, SetIntPart } **e** { InitExpPart,InitInt, SetDig} **d** { AppDig, AddSign, SetExpPart, SetNumber, ShowNum }
4	{ InitNum, InitFr } . { SetDig } **d** { IncrFr, SetDig } **d** { IncrFr, SetNumber, ShowNum }
5	{ InitNum, InitInt, SetDig } **d** { AppDig, SetDig } **d** { AppDig, SetIntPart, InitFr } . { SetDig } **d** { IncrFr } **e** { InitExpPart } + { InitInt, SetDig } **d** { AppDig, AddSign, SetExpPart, SetNumber, ShowNum }

Как можно убедиться, этот тест длиной в 20 символов является минимальным в том смысле, он покрывает все дуги граф-схемы, как минимум, по одному разу и не существует более короткого покрывающего теста для этой структуры.

Строго говоря, это прототест, ибо он использует микролексические классы **d** для цифр, а не сами цифры, и **e** для символа порядка (‘e’, ‘\’). Для получения реальных тестовых вариантов имена **d** и **e** заменяют на какой-нибудь элемент соответствующего класса.

На рис. 5 показан пропуск этого теста на конечном детерминированном процессоре с минимальным числом состояний, построенном на той же грамматике, по которой был построен сам тест.

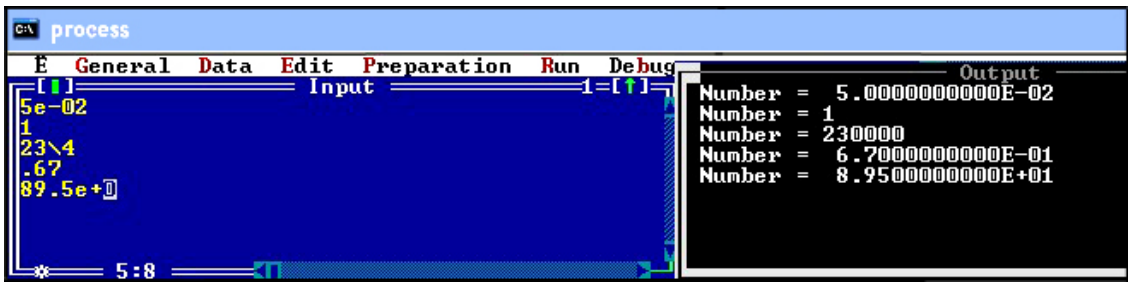


Рис. 5. Результат выполнения теста уровня 0 для чисел Алгола 68

Как видно из рис. 5, тест прошёл успешно.

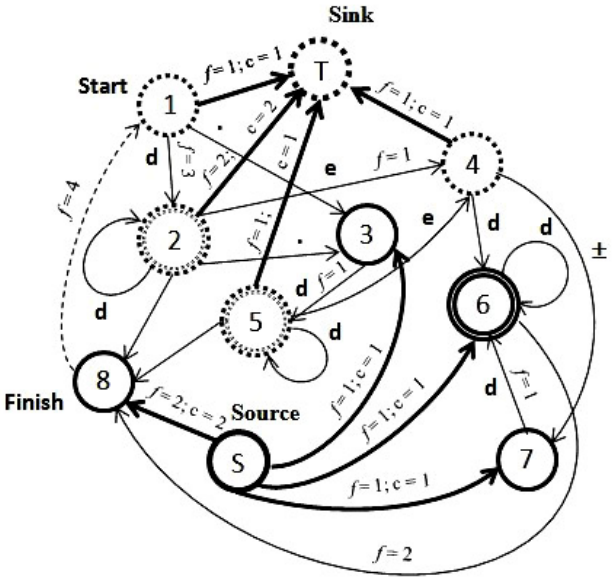
б) Генерация теста степени* 0 *по диаграмме переходов минимального конечного процессора

Матрица переходов, представляющая минимальный процессор, представлена на рис. 6а, а сеть, построенная по ней, на рис. 6б. Узлы 1–7 соответствуют состояниям процессора. Узел 1 считается начальным, а узел 8 — конечным. В него входят дуги, исходящие из конечных состояний 2, 5 и 6. В матрице они отмечены значком # в столбце ε.

В сети показаны источники 3, 6, 7 и 8, стоки 1, 2, 4, 5, пропускные способности крайних дуг c, и значения потока на всех дугах сети f. Простая цепь достигает главного стока

	d	.	e	+	–	ε
1	2	3				
2	2	3	4			#
3	5					
4	6			7	7	
5	5		4			#
6	6					#
7	6					

(а) Матрица переходов



(б) Сеть

Рис. 6

в каком-то из 5-х процессов распространения волны от главного источника. Все пути насыщают соответствующие крайние дуги.

1. Самый короткий путь $S \to 3 \to 5 \to$ длиной 3. По нему проходит 1 ед.
2. Следующий путь $S \to 6 \to 8 \to 1 \to T$ имеет длину 4. По нему проходит 1 ед.
3. Следующий путь $S \to 8 \to 1 \to 2 \to T$ имеет длину 4. По нему проходит 1 ед.
4. Следующий путь $S \to 8 \to 1 \to 2 \to 4 \to T$ имеет длину 5. По нему проходит 1 ед.
5. Следующий путь $S \to 7 \to 6 \to 8 \to 1 \to 2 \to T$ имеет длину 6. По нему проходит 1 ед.

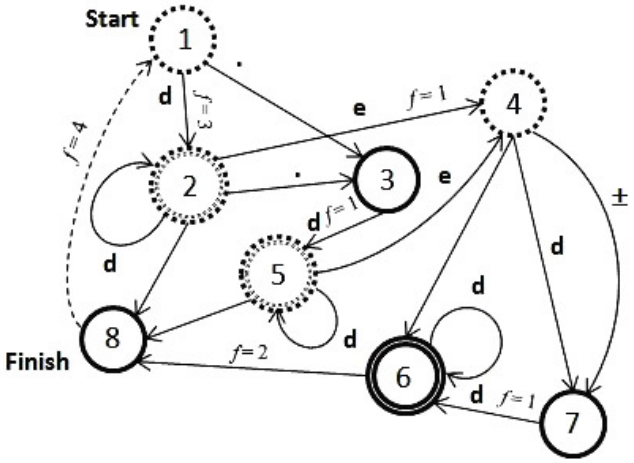


Рис. 7. Симметричный граф G^*

После удаления главных источника и стока эйлеров обход симметричного графа G^* (рис. 7):

$1 \to^{d} 2 \to 8 \to$
$1 \to^{\cdot} 3 \to^{d} 5 \to 8 \to$
$1 \to^{d} 2 \to^{e} 4 \to^{d} 6 \to 8 \to$
$1 \to^{d} 2 \to^{e} 4 \to^{+} 7 \to^{d} 6 \to 8 \to$
$1 \to^{d} 2 \to^{e} 4 \to^{-} 7 \to^{d} 6 \to 8 \to 1$

даёт 5 вариантов теста минимальной длины в 16 лексем:
(1) d (2) . d (3) d e d (4) d e + d (5) d . d e – d

Из сравнения примеров 1а и 1б следует вывод: метод генерации теста по синтаксической диаграмме Вирта, даёт минимальный по длине тест (20 лексем), который не является минимальным по длине (16 лексем) относительно конечного преобразователя с минимальным числом состояний, построенного по той же самой EBNF-грамматике.

Пример 2. Генерация теста степени 3

Рассмотрим случай генерации теста степени 3 для автоматного языка, который определяется EBNF-грамматикой с единственным правилом вида $S : 'b', 'a'^{*}$. Синтаксическая граф-схема, построенная по этой грамматике, представлена на рис. 8.

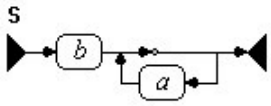
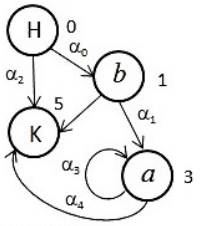
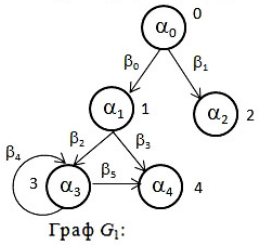
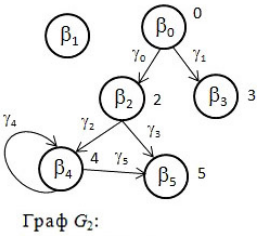
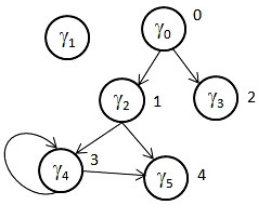


Рис. 8. Синтаксическая граф-схема для регулярного выражения ba^{*}

Исходный граф G, построенный по граф-схеме, представлен в табл. 3 (в верхнем левом углу). Чтобы получить тест степени 3, сначала нужно получить последовательность итерированных рёберных графов степеней 0–3: $G = L^0(G), G_1 = L^1(G), G_2 = L^2(G)$, $G_3 = L^3(G)$. При этом тривиальные компоненты в виде изолированных узлов в графах G_2 и G_3 фактически не используются в сети S_3 (рис. 9), а используются их прообразы — свободные дуги (free) в графах G_1 и G_2 (табл. 3).

Таблица 3. Построение итерированных рёберных графов

G $L^0(G)=G$	$G_1 = L^1(G) = L(L^0(G))$	$G_2 = L^2(G) = L(G_1)$	$G_3 = L^3(G) = L(G_2)$
			
Исходный граф компоненты S: Число узлов = 4, число дуг = 5. Список дуг G: **0.** (0, 1) : α_0 **1.** (0, 3) : α_1 **2.** (1, 5) : α_2 **3.** (3, 3) : α_3 **5.** (3, 5) : α_4	Граф G_1: Число узлов = 5, число дуг = 6. **Список дуг G_1:** **0.** (0, 1) : $\alpha_0 \to \alpha_1$ **1.** (0, 2) : $\alpha_0 \to \alpha_2$ free **2.** (1, 3) : $\alpha_1 \to \alpha_3$ **3.** (1, 4) : $\alpha_1 \to \alpha_4$ **4.** (3, 3) : $\alpha_3 \to \alpha_3$ **5.** (3, 4) : $\alpha_1 \to \alpha_4$	Граф G_2: Число узлов = 5, число дуг = 6. **Список дуг G_2:** **0.** (0, 1) : $\beta_0 \to \beta_2$ **1.** (0, 3) : $\beta_0 \to \beta_3$ free **2.** (1, 2) : $\beta_2 \to \beta_4$ **3.** (1, 4) : $\beta_2 \to \beta_5$ **4.** (2, 2) : $\beta_4 \to \beta_4$ **5.** (2, 4) : $\beta_4 \to \beta_5$	Граф G_3: Число узлов = 5, число дуг = 6. **Список дуг G_3:** **0.** (0, 1) : $\gamma_0 \to \gamma_2$ **1.** (0, 2) : $\gamma_0 \to \gamma_3$ **2.** (1, 3) : $\gamma_2 \to \gamma_4$ **3.** (1, 4) : $\gamma_2 \to \gamma_5$ **4.** (3, 3) : $\gamma_4 \to \gamma_4$ **5.** (3, 4) : $\gamma_4 \to \gamma_5$

Задача почтальона решается на нетривиальной компоненте графа G_3, пополненной главным начальным узлом (-1), главным конечным узлом (-2), дающим $G_3' = Connect(G_3)$, возвратной дугой $(-2,-1)$, главным источником (-4) и главным стоком (-3), а также соответствующими замыкающими $(-1,0)$, $(2,-2)$ и $(4,-2)$ и крайними дугами $(-4,-2)$, $(-4,4)$, $(1,-3)$ и $(0,-3)$ (см. рис. 9).

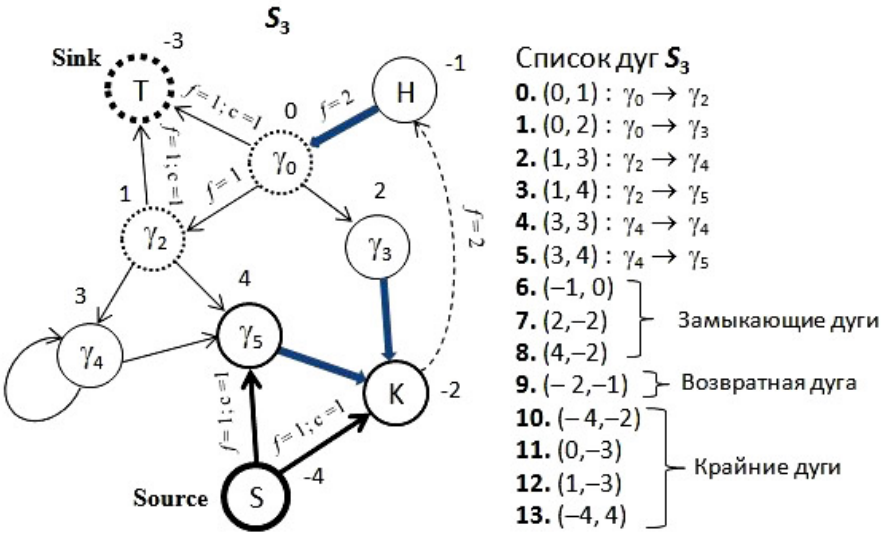


Рис. 9. Сеть степени 3

Процесс окрашивания дуг сети, построения простых цепей, увеличивающих поток и определения приращений потока на их дугах, представлен в табл. 4.

Эйлеров цикл, построенный по элементам сети S_3 (рис. 9) после удаления крайних дуг с учётом дополнительных кратностей дуг, представленных значениями потоков на дугах, имеет следующий вид:

$$-1 \to 0 \to 2 \to -2 \to -1 \to 0 \to 1 \to 4 \to -2 \to -1 \to 0 \to 1 \to 3 \to 3 \to 4 \to -2 \to -1$$

Далее обход эйлерова цикла и обратные отображения L^{-1}, L^{-2} и L^{-3}, показанные на рис. 10 и 11.

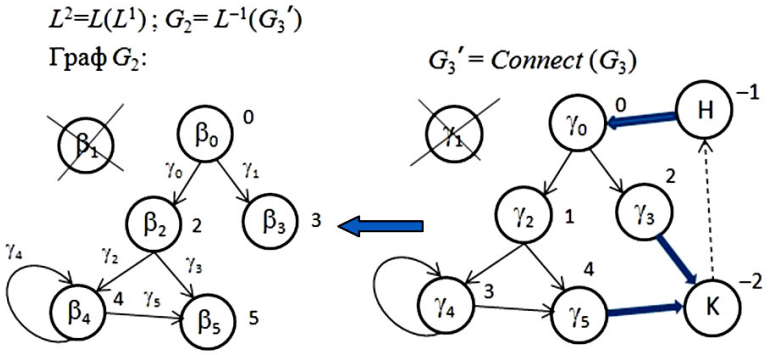


Рис. 10. Обратное отображение $L^{-1}(G_3')$

Таблица 4. Построение итерированных рёберных графов

№	Дуги	c	Окр. 4	$\Delta f_1=1$	Окр. 6	$\Delta f_2=1$	flow
0	(0, 1)		❶		❷	1	1
1	(0, 2)				❶		
2	(1, 3)				❶		
3	(1, 4)						
4	(3, 3)						
5	(3, 4)						
6	(−1, 0)		❷	1	❷	1	2
7	(2, −2)						
8	(4, −2)				❷	1	1
9	(−2, −1)		❷	1	❷	1	2
10	(−4, −2)	1	❷	1			1
11	(0, −3)	1	❷	1			1
12	(1, −3)	1			❷	1	1
13	(−4, 4)	1	❶		❷	1	1

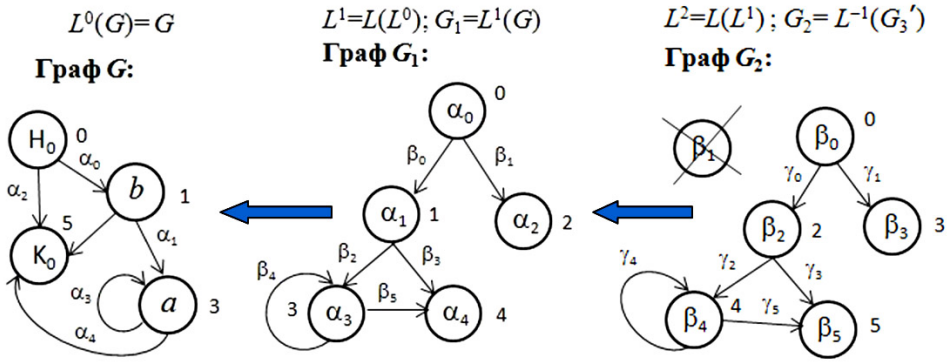


Рис. 11. Обратные отображения $L^{-2}(G_2)$ и $L^{-1}(G_1)$

Переход $G_3' \Rightarrow G_2$:

$0-\gamma_0 \to 2-\gamma_3 \Rightarrow 0-\beta_0 \to 2-\beta_2 \to 5-\beta_5$

$0-\gamma_0 \to 1-\gamma_2 \to 4-\gamma_5 \Rightarrow 0-\beta_0 \to 2-\beta_2 \to 4-\beta_4 \to 5-\beta_5$

$0-\gamma_0 \to 1-\gamma_2 \to 3-\gamma_4 \to 3-\gamma_4 \to 4-\gamma_5 \Rightarrow 0-\beta_0 \to 2-\beta_2 \to 4-\beta_4 \to 4-\beta_4 \to 4-\beta_4 \to 5-\beta_5$

Переходы $G_2 \Rightarrow G_1$**:**

$0 - \beta_0 \to 2 - \beta_2 \to 5 - \beta_5 \Rightarrow 0 - \alpha_0 \to 1 - \alpha_1 \to 3 - \alpha_3 \to 4 - \alpha_4$

$0 - \beta_0 \to 2 - \beta_2 \to 4 - \beta_4 \to 5 - \beta_5 \Rightarrow 0 - \alpha_0 \to 1 - \alpha_1 \to 3 - \alpha_3 \to 3 - \alpha_3 \to 4 - \alpha_4$

$0 - \beta_0 \to 2 - \beta_2 \to 4 - \beta_4 \to 4 - \beta_4 \to 4 - \beta_4 \to 5 - \beta_5 \Rightarrow 0 - \alpha_0 \to 1 - \alpha_1 \to 3 - \alpha_3 \to 3 - \alpha_3 \to 3 - \alpha_3 \to 3 - \alpha_3 \to 4 - \alpha_4$

След этих трёх путей в графе G при применении ещё одного шага обратного отображения даёт три варианта теста уровня 4:

(1) $0 - H_0 \to 1 -' b' \to 3 -' a' \to 3 -' a' \to 5 - K_0$;

(2) $0 - H_0 \to 1 -' b' \to 3 -' a' \to 3 -' a' \to 3 -' a' \to 5 - K_0$;

(3) $0 - H_0 \to 1 -' b' \to 3 -' a' \to 3 -' a' \to 3 -' a' \to 3 -' a' \to 3 -' a' \to 5 - K_0$.

Наконец, два шага обратного отображения свободной дуги графа G_2 и один шаг обратного отображения свободной дуги графа G_1 (табл. 3) даёт ещё два варианта теста:

(4) $0 - H_0 \to 1 -' b' \to 5 - K_0$; (5) $0 - H_0 \to 1 -' b' \to 3 -' a' \to 5 - K_0$.

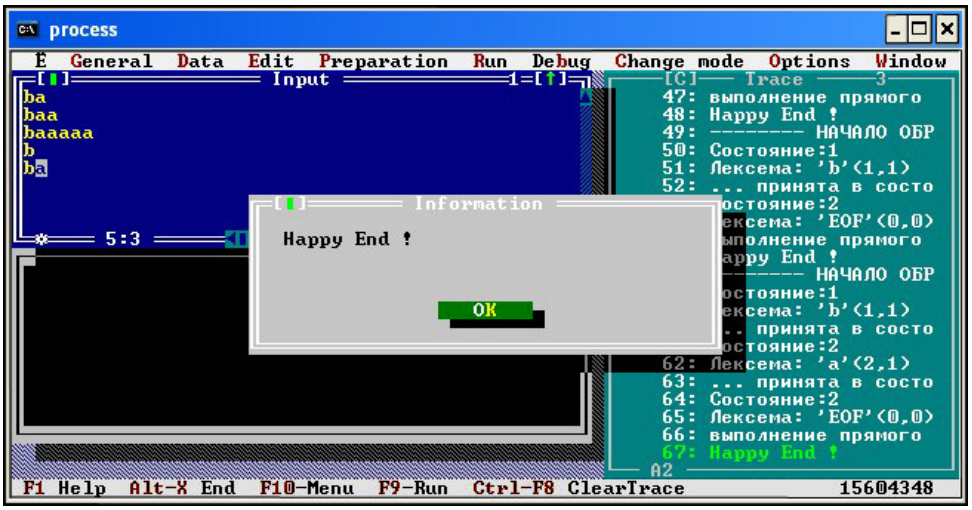


Рис. 12. Тест уровня 4 прошёл успешно

На конечном процессоре с минимальных числом состояний, построенном по той же грамматике, по которой генерировался тест, этот тест проходит успешно (рис. 12). На этот раз процессор используется как распознаватель языка ba^*.

Список литературы

1. *Wirth N.* The Programming Language Pascal. Acta Informatica, 1971. Vol. 1. P. 35–63.
2. *Martynenko B.* Towards the 80th Anniversary of N. Wirth: Wirth's Syntactic Charts in the SYNTAX-Technology. 3rd International Conference on Computer Technology in Russia and in the Former Soviet Union, SoRuCom 2014; Kazan; Russian Federation; 13 October 2014 through17 October 2014 ; Category Number E5460; Code 110870, 5 February 2015, Article number 7032988, P. 199–206.
3. *Мартыненко Б.К.* Синтаксически управляемая обработка данных. 2-е изд., исправленное и дополненное. СПб: СПбГУ, 2004. С. 315.
4. *Evanas J.R., Minieka E.* Optimization Algorithms for Networks and Graphs. Second Edition, Revised And Expanded — Marcel Dekker, inc. New York, New York, 1992.
5. *Hopcroft J.E., Ullman J.D.* Formal languages and their relation to automata. — Reading, MA: Addison-Wesley Pub. Co., Inc., 1969.
6. Алгоритмический язык Алгол-60. Пересмотренное сообщение. Перевод с англ. Под редакцией А.П. Ершова, С.С. Лаврова и М.Р. Шура-Бура. М.: «Мир», 1965.

7. *Harary F, Norman R.Z., Cartwright D.* Structural Models: An Introduction to the Theory of Directed Graphs. John Wiley & Sons, Inc. New York, London, Sydney. Second printing, 1966.

Поступила в редакцию 24.04.2016, окончательный вариант — 06.06.2016.

Computer tools in education, 2016

№ 5: 17–45

http://ipo.spb.ru/journal

SYNTAX DIRECTED GENERATION OF TESTS FOR PROCESSORS OF REGULAR LANGUAGES

Martynenko B.K.[1]

[1]SPBSU, Saint-Petersburg, Russia

Abstract

A method of generation of tests of minimum length for the purpose of testing of syntax directed finite processors recognizing regular languages is described. A criterion of the test cases selection is introduced that represents the requested level of coverage for the acres of the graph that corresponds to the regular expression for which the processor has been constructed. Since the graph acres are labeled with semantic labels, the criterion defines the degree of interaction between these semantics, so such a test must be built that the criterion conditions are met. The method is based on the algorithm of the solution of the Chinese Postman Problem on a directed graph, that is built for the input regular expression that defines the language recognized by the finite processor.

Keywords: *Chinese Postman Problem; Directed graph; Finite processor; Line graph; Regular expression; N. Wirth's syntactic diagrams, Test of minimal length.*

Citation: Martynenko, B., 2016. "Sintaksicheski upravlyaemaya generatsiya testov dlya protsessorov regulyarnykh yazykov" ["Syntax Directed Generation of Tests for Processors of Regular Languages"]. *Computertools in education*, no. 5, pp. 17–45.

Received 24.04.2016, the final version — 06.06.2016.

Boris K. Martynenko, doctor of Science, professor of Computer Science, Saint Petersburg state university, Mathematics and Mechanics Department, Universitetsky prospekt, 28, 198504, Saint Petersburg, Russia
mbk@ctinet.ru

Мартыненко Борис Константинович, доктор физико-математических наук, профессор кафедры информатики математико-механического факультета СПбГУ; 198504, Россия, Санкт-Петербург, Старый Петергоф, Университетский пр., дом 28, математико-механический факультет, кафедра информатики.
mbk@ctinet.ru