УДК 681.3.06

ДЕТЕРМИНИРОВАННАЯ СЕМАНТИКА РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

***Дмитрий Иванович Черемисинов**, к.т.н., ведущий научный сотрудник*
Тел.: 375 17 284 28 61, e-mail: Cher@newman.bas-net.by
Объединенный институт проблем информатики НАН Беларуси
http://uiip.bas-net.by

Проблема конвертации описания на формальном языке имеет большое практическое значение для системной интеграции программ и в области технологии программирования, называемой «реинжинирингом программного обеспечения» (software reengineering). Операция анализа строковых данных путём сопоставления с образцом, заданным регулярным выражением, образует основу для построения разнообразных конверторов. Предлагается изменить традиционную модель операции сопоставления с образцом на основе конечного автомата на модель в виде алгебры образцов.

Ключевые слова: Формальный язык, анализатор, регулярное выражение, алгебра Клини, конечный автомат, детерминированная семантика.

Регулярные выражения [1] часто используются на практике с целью построения программ, являющихся анализаторами текста. Для разработки анализатора языка с регулярной грамматикой имеются особые инструменты, например, Lex, ге2с и языки сценариев, такие как Sed, Awk и Perl. Все они строят анализаторы однотипной архитектуры, состоящие из двух уровней: процессор регулярных выражений и какой-то программный «клей», связывающий отдельные регулярные выражения вместе.

Д.И. Черемисинов

Процессор регулярных выражений является анализатором текста на основе детерминированного конечного автомата. Такой автомат может быть представлен как статические данные программы анализа в виде таблицы переходов и выходов. Таблично управляемые анализаторы используются в Perl, Python, Emacs, Tel и .Net. Поскольку признаком распознавания служит переход автомата в конечное состояние, процессор регулярных выражений не «ловит» подвыражений исходного регулярного выражения.

В отличие от традиционно рассматриваемой в теории формальных языков задачи распознавания принадлежности языку заданного текста, задачей анализатора языка является построение структуры анализируемого текста, выполняя его разбор. Алгоритм распознавания принадлежности имеет форму грамматики языка. Для построения анализатора в грамматику нужно добавить действия по формированию структуры данных, представляющей результат анализа. В рассматриваемом случае грамматика задана в виде регулярного выражения. В алгоритме анализатора действия по распознаванию участков текста и действия построения результатов разбора чередуются.

Анализаторы текста являются основой конверторов форматов данных, преобразующих описания данных. Однако в отличие от языков программирования форматы данных, используемые для системной интеграции программ – это языки регулярного типа. Программы конверторов, построенные с помощью инструментов построения компиляторов, неоправданно сложны, так как содержат поддержку рекурсии, необходимую для разбора контекстно-свободных языков. Предлагается изменить традиционную модель операции сопоставления с образцом на основе конечного автомата на модель в виде алгебры образцов. В интерпретации алгебры образцов язык регулярных выражений становится детерминированным, что обеспечивает включение действий не только при завершении выражения, но и после подвыражений.

Язык регулярных выражений

Регулярным выражением называется формула алгебры Клини K. Язык регулярных выражений составляет множество формул алгебры K. Элементами K являются строки символов фиксированного алфавита Σ, сигнатуру операции составляют константы 0 и 1 и операции объединения ($\vee$), конкатенации (.) и звезда Клини (*). В стандартной интерпретации регулярное выражение s задает язык $L(s)$ алфавита Σ; 0 интерпретируется как пустая строка, 1 – как язык, состоящий из пустой строки, операция $a \vee b$ – как язык $L(a \vee b) = L(a) \cup L(b)$, операция $a.b$ – как язык $L(a.b) = \{xy \mid x \in L(a), y \in L(b)\}$ и звезда Клини $a*$ – язык $L(a*)$ всех строк, полученных конкатенацией нуля или более строк из $L(a)$.

Пусть S – кортеж (линейно упорядоченное множество) строк. Система составляющих на S – это множество C отрезков S, которое содержит в качестве элементов как само S, так и каждое слово, входящее в S, и построено таким образом, что любые два отрезка, входящие в C, либо не пересекаются, либо один из них содержится в другом. Элементы такого множества C называются составляющими. Составляющая x доминирует над составляющей y (y вложено в x), если y является частью x и y отлично от x. Составляющими регулярного выражения являются символы алфавита Σ и выражения операций. Структуру регулярного выражения a задает дерево непосредственных составляющих (ДНС), отношение которого задается непосредственным доминированием составляющих. ДНС может быть преобразовано в множество правил переходов конечного автомата [2] (в общем случае недетерминированного). Этот автомат строится как композиция автоматов, изоморфная ДНС. Вершины ДНС однозначно соответствуют компонентным автоматам.

Конечный автомат – это пятерка $KA = (\Sigma, Q, q_0, T, P)$, где Σ – алфавит; Q – конечное множество состояний; q_0 – начальное состояние $(q_0 \in Q)$; T – множество терминальных состояний, $T \subset Q$; функция переходов P, заданная множеством правил $q_i a_k q_j$, где q_i и q_j – состояния, a_k – входной символ: $q_i, q_j \subset Q, a_k \subset \Sigma$ или является пустым символом. Автомат KA недерминирован, если среди его правил перехода есть правила с пустым входным символом [2]. Правила с пустым входным символом появляются в компонентных автоматах, соответствующим операциям объединения ($\vee$) и звезде Клини (*).

Состоянием разбора слова v называется упорядоченная пара (v,i), где $v \in \Sigma^*$, а i – целое число; $0 \le i < |v|, .|v|$ – длина слова v. Состояние разбора (v,i) задает представление слова v в виде конкатенации $br = v, |b| = i$. Правило перехода $q_i a_k q_j$ автомата KA применимо в состоянии разбора (v,l), если v представимо в виде конкатенации $ba_k r = v, |b| = l$ и текущим состоянием автомата является q_i. Для недерминированного автомата текущее состояние – это множество состояний, и q_i должно содержаться в этом множестве. Применение правила переводит автомат в состояние q_j,(для недерминированного автомата q_j становится элементом текущего состояния), а текущим состоянием разбора становится $(v, l+1)$. Автомат KA применим в состоянии разбора (v,l), если имеется правило для начального состояния, применимое в этом состоянии разбора, и после применения всех возможных правил автомат оказывается в терминальном состоянии. Исходное состояние разбора (v,l) и состояние разбора (v,m), когда автомат находится в терминальном состоянии, задают представление слова v в виде конкатена-

ции $bxt = v, |b| = l, |bx| = m$. Слово x распознается регулярным выражением s, по которому построен автомат KA. Множество всех слов, распознаваемых регулярным выражением – это язык $L(s)$, называемый регулярным.

Таким образом, регулярное выражение *s* можно интерпретировать как частичную функцию на множестве состояний разбора. Анализ текста с интерпретацией операций анализа как частичных функций на множестве состояний разбора называется анализом по образцу [3]. В случае регулярных выражений образцом для анализа является регулярное выражение. В большинстве языков программирования имеется реализация такой операции, в которой образцом является само распознаваемое слово (регулярное выражение, построенное операцией конкатенации). В операции поиска по образцу организуется лексикографический перебор состояний разбора с целью нахождения одного (нескольких) вхождений образца. Для языка C++ функция поиска в тексте слова, заданного регулярным выражением, реализована в известной библиотеке Boost [4].

Недетерминизм стандартной интерпретации языка регулярных выражений

Процесс применения правил перехода автомата может быть зафиксирован в виде дерева разбора аналогично тому, как это делается при синтаксическом разборе. В случае автомата роль нетерминальных символов играют символы состояний. В дереве разбора по грамматике обход листьев дерева в порядке применения правил дает анализируемую строку. Среди листьев дерева отсутствуют листья, помеченные нетерминалами. Это дает возможность связать с каждым нетерминалом определенное действие по построению дерева. В дереве разбора автоматом нелистовые вершины помечены текущими состояниями (в случае недетерминированного автомата – несколькими нетерминалами). Если эти вершины расщепить, то появляются листья, помеченные нетерминалами – тупики анализа. Для фиксации этого дерева нужно связывать действия не с отдельными терминалами (состояниями автомата), а с множествами состояний, которые зависят от анализируемой строки – в этом проявляется недетерминизм стандартной семантики регулярных выражений. Это означает, что дерево разбора не может быть построено путем связывания действий с правилами автомата (алгоритм разбора не «ловит» подвыражений).

Регулярное выражение может быть преобразовано в набор правил перехода, задающих детерминированный конечный автомат (текущее состояние всегда одноэлементное и не допускаются переходы по пустому входному символу) [3]. По регулярному выражению детерминированный автомат может быть построен непосредственно процедурой с экспоненциальной верхней оценкой временной сложности, или недерминированный автомат может быть преобразован процедурой детерминизации. Для детерминированного автомата дерево непосредственных составляющих, соответствующих состояниям автомата, при анализе заданной строки может быть построено. Однако этот набор правил перехода не имеет структурного подобия с ДНС регулярного выражения, и поэтому структуру регулярного выражения невозможно использовать в ходе анализа строки.

Для анализа подвыражений можно ограничиться подмножеством языка регулярных выражений, в котором композиция компонентных автоматов, соответствующих вершинам ДНС, задает детерминированный автомат. Некоторые правила переходов с пустым входным символом могут быть удалены с сохранением эквивалентности автоматов. Однако при таком походе существенно сокращаются изобразительные возможности языка регулярных выражений, и значительно возрастает риск ошибок программирования, так как процедура установления свойства детерминированности не тривиальна. Этот подход предлагается использовать в Ragel State Machine Compiler [5].

Алгебра образцов

Пусть f_i и p_i – символы некоторых функций и предикатов на состояниях разбора. Выражение $(p_1 \to f_1; p_2 \to f_2; ...; p_n \to f_n)$ называется условным выражением

Мак-Карти [6] и определяет частичную функцию h на состояниях разбора, совпадающую с одной из функций f_i, номер i которой удовлетворяет следующему условию

$\exists i(p_i(x) \vee \forall j((j < i \Rightarrow \neg p_j((x)))))$, здесь символ $\neg$ обозначает инверсию логического значения. Если такого i не существует, то функция h не определена.

Частичная функция преобразования состояний разбора строки f – образец, если для любых состояний $\alpha = (w, j), \beta = (v, i), \beta = f(\alpha)$ и $w = v, j \le i$. Пусть $p(f, \alpha)$ – предикат, обозначающий утверждение, что образец f определен на состоянии α. Если образцы $f_1 = f_2$, то и $p(f_1) = p(f_2)$.

Будем использовать инфиксную форму записи операции # применения образца f к некоторому состоянию строки: $f\#\beta = \alpha$. Введем следующие операции над образцами.

Катенацией fg образцов f и g называется операция, определяемая условным выражением $fg\#\alpha = p(f\alpha) \rightarrow g\#(f\#\alpha)$. Альтернация $f \vee g$ образцов f и g определяется условным выражением $f \vee g\#\alpha = p(f, \alpha) \rightarrow (f\#\alpha); p(g, \alpha) \rightarrow (g\#\alpha)$. Итерация образца f^* определяется через степень следующим выражением с бесконечным числом членов $f*\#\alpha = \neg p(f_1, \alpha) \rightarrow \alpha; P(f_2, \alpha) \rightarrow f_1\#\alpha; ...; P(f_i, \alpha) \rightarrow f_{i-1}\#\alpha; ....$ Степень определяется рекурсивно выражениями $f^1\#\alpha = f\#\alpha$ и $f^n\#\alpha = f\#(f^{(n-1)}\#\alpha)$.

Введенные операции образуют алгебру образцов, элементами которой в отличие от алгебры Клини являются состояния разбора – пары (v, i), где $v \in \Sigma^*$, а i – целое число. Нетривиальными константанами этой алгебры являются примитивные образцы, распознающие вхождение слов, состоящих из единственного символа алфавита. Константа 1 этой алгебры – тождественный образец – обладает следующими свойствами: если f образец, то $1* = 1$, $1f = f$, $f1 = f$, $1 \vee f = 1$, $f \vee 1 = g$ такой (всюду определенный), что на области определения $f.g = f$, в остальной области $g = 1$.

Язык регулярных выражений совпадает с множеством формул алгебры образцов. Операция применения образца играет ту же роль в процессе анализа строки, что операция применения автомата, построенного по регулярному выражению. Отличие состоит в том, что недерминированный выбор правил перехода в операции применения автомата заменен упорядоченной в порядке вхождения в выражение проверкой применимости альтернатив операции альтернации. Эта особенность делает алгебру образцов детерминированной семантикой регулярных выражений.

Пара состояний разбора α и $\beta = f(\alpha)$ задают представление слова v в виде axc, то есть образец f распознает слова x как часть слова v. Множество всех слов, распознаваемых образцом f, называется языком $L[f]$, распознаваемым образцом. Легко проверяется, что $L[f] \subset L(f)$.

Для удобства практического использования в сигнатуру алгебры Клини часто включают операцию разности. Разность регулярных выражений a и b $(a - b)$ есть язык $L(a - b)$, слова которого принадлежат $L(a)$, но не принадлежат $L(b)$. В сигнатуру алгебры образцов удобнее включить операцию отрицание f, задаваемую выражением $f\neg\#\alpha = \neg p(f, \alpha) \rightarrow \alpha$.

Подстановка языков

Формально языковую подстановку можно рассматривать как упорядоченную тройку (L_1, L_2, ϕ) состоящую из двух языков L_1, L_2 и отображения $\phi: L_1 \rightarrow L_2$ [7]. Операция применения_языковой подстановки_состоит в построении слова W из слова V.

Если существует $x \in L_1$ и x - вхождение в V, то w получается заменой в V вхождения X словом $y = \phi(x) \in L_2$. Если $X \in L_1$ не содержится в V, то подстановка (L_1, L_2, ϕ) к V не применима.

Сопоставим с образцом f, распознающим язык $L[f]$, однозначную функцию $\phi: L[f] \rightarrow L_2$. Такой образец называется образцом с побочным эффектом. В информатике функция или выражение программы имеет побочный эффект (side effect), если вызов конструкции изменяет состояние вызывающей программы [8]. Преобразование $\beta = f(\alpha)$ состояния разбора составляет основной эффект образца f.

Побочным эффектом образца может служить другой образец с побочным эффектом. Образцы с побочным эффектом являются классом рекурсивных функций. Так как рекурсивные функции являются одной из алгоритмических моделей, то вследствие тезиса Чёрча [9] образцы с побочным эффектом – это алгоритмически полная система. Следствием алгоритмической полноты языка регулярных выражений с побочным эффектом является возможность распознавания языка любого типа в классификации Хомского, а не только класса регулярных языков.

Алгебра образцов с побочным эффектом может быть построена модификацией интерпретации тождественного образца – константы 1 алгебры. Достаточно допустить, чтобы только тождественные образцы имели побочный эффект, то есть вести нужное количество тождественных образцов, различающихся побочным эффектом.

Можно считать, что образцы с побочным эффектом задают некоторую операцию, применимую к анализируемому слову только в том случае, если в этом слове имеется состояние разбора, в котором образец применим. Таким образом, образец с побочным эффектом задает алгоритм построения слова y на основе информации о системе составляющих анализируемого слова x. Образец с побочным эффектом представляет собой нормальный алгоритм Маркова [10]. Конструирование этого алгоритма заключается во «встраивании» операций построения слова y в процедуру анализа x. Для этого в образец, задающий процедуру анализа, включаются тождественные образцы с побочным эффектом. Операции, представляющие побочный эффект этих образцов, являются базисом алгоритмического разложения операции вычисления $(x) = y$.

Регулярное выражение с семантикой алгебры образцов можно интерпретировать как текст программы, реализующей алгоритм применения этого образца, и выраженной на особом языке программирования. Базовый набор операций этого языка составляют операции применения примитивных образцов, а функциональные формы служат средством задания последовательности применения этих базовых операций, так как условные выражения, являющиеся определениями функциональных форм, суть не что иное, как указания о порядке применения операндов. В тексте этой программы переменные, значениями которых являются состояния разбора, в явном виде не упоминаются, их существование предполагается для каждой такой программы.

Прототипом семантики алгебры регулярных выражений с побочным эффектом являются макроязыки. При макрообработке слово x обычно называется макровызовом, а y – макрорасширением. Подстановка языков описывается макроопределением, в котором слово x задается указанием его признаков, а y – алгоритмом формирования. В подстановке, заданной регулярным выражением, составляющие регулярного выражения – это формальные параметры макровызова; слова, распознаваемые составляющими, – это фактические параметры; побочный эффект – макропроцедура. По организации вычислений аналогом языка регулярных выражений является язык XSLT. В отличие от XSLT язык регулярных выражений лучше подходит для обработки неформатированных текстов.

Синтез программы по регулярному выражению с побочным эффектом

Пусть U – некоторое множество переменных, а V – множество их значений. Со-

стояние памяти машины Ω – это функция из U в V, а множество S состояний памяти – это множество всех функций $s:U\to V$. Состояние управления – это пара (v,i), в которой v – текст программы на языке $L(\Omega)$ машины Ω, $v\in\Sigma^*$, а i – целое число. Множество всех возможных состояний управления $\Psi(\Omega)$ совпадает с множеством состояний разбора слов для языка $L(\Omega)$. Командами машины Ω служат образцы с побочными эффектами, определенными на множестве $\Psi(\Omega)*L(\Omega)$. Если образец f представляет некоторую команду, то множество состояний управления, в которых он применим, задает множество допустимых состояний управления команды, а побочный эффект – функцию преобразования исходных состояний памяти и управления в конечные. Машина Ω – это образец $(f_1\vee f_2\vee\ldots\vee f_n)*$, где f_i – образец с побочным эффектом – команда машины Ω. Программы этой машины принадлежат языку $L(\Omega)$.

Рассмотрим программу p для машины Ω. Пусть P множество состояний управления, возникающих при интерпретации программы p машиной Ω. Для каждого состояния управления $\alpha\in P$ можно задать отображение $\pi_i:S\to P$, указывающее состояние управления, в которое перейдет машина Ω после выполнения команды, допустимой в состоянии. Объединение всех π_i дает отображение $\pi:P*S\to P$, называемое схемой программы p. Схема программы – это граф, вершинами которого являются состояния управления, а дуги помечены значениями переменных памяти машины Ω. Команда, изменяющая только состояние памяти, называется преобразователем, команда, не изменяющая состояние памяти, но изменяющая основной эффект образца, называется распознавателем. На команды машины Ω накладывается ограничение: все ее команды или преобразователи, или распознаватели. В графическом представлении схемы распознаватели обозначаются ромбами, преобразователи – прямоугольниками.

Память машины Ω состоит из переменных u,i, представляющих состояние разбора строки, являющейся значением u, стека для хранения числовых значений и рабочей переменной j числового типа. Преобразователи с пометками →, ← и ↙ соответствуют операции запоминания i *в* стеке, который в момент начала просмотра пуст, операции выборки из стека в переменную i и операции удаления верхнего элемента магазина соответственно. Элементы с пометкой ⇐ или ⇒ задают операцию запоминания верхнего элемента магазина в переменную j или операцию пересылки значения j в i.

Процесс выполнения программы p для машины Ω на состоянии памяти s_0 определяется следующим образом. Он состоит в путешествии по схеме, прокладывающем в ней ориентированный путь и сопровождающемся преобразованием состояний из S. Процесс начинается из состояния управления $(p,0)$ при состоянии памяти s_0. Переход через преобразователь с символом y сопровождается преобразованием текущего состояния памяти x в состояние $y(x)$. Переход через распознаватель не меняет текущего состояния памяти и состоит в выборе одной из исходящих из распознавателя дуг для продолжения путешествия по схеме; если u – сопоставленная распознавателю переменная, то выбирается дуга, помеченная значением $u(x)$. Выполнение схемы завершается, если путешествие по ней привело на выход схемы, и в этом случае текущее состояние воспринимается как результат выполнения программы p с исходными данными s_0. В ином случае результат не определен.

Для синтеза программы для машины Ω, реализующей алгоритм, заданный образцом a, используется обратная польская запись образца a. При обнаружении в поль-

ской записи обозначения примитивного образца в магазин заносится схема программы, реализующей образец (рис. 1 а). Обнаружение символов операций над образцами вызывает выборку из магазина одной или двух подсхем и формирование новой подсхемы комбинированием выбранных схем в соответствии с шаблоном схемы для соответствующей операции (рис. 1 б,в,г). Эта схема затем заносится в магазин. При синтаксически правильной записи обозначений исходного образца в конце просмотра магазин содержит единственную схему, которая и является схемой программы, реализующей данный образец.

Преобразование схемы программы, реализующей данный образец, в реальную программу можно реализовать путем построения транслятора с языка обозначений образцов в какой-нибудь язык программирования высокого уровня. Выбор объектного языка в основном определяется возможностями программирующей системы, в которой этот язык используется в качестве входного, так как требования к нему, предъявляемые таким транслятором, минимальны. Объектный язык должен допускать использование данных типа строки символов, и в нем должна иметься возможность доступа к отдельным символам строки.

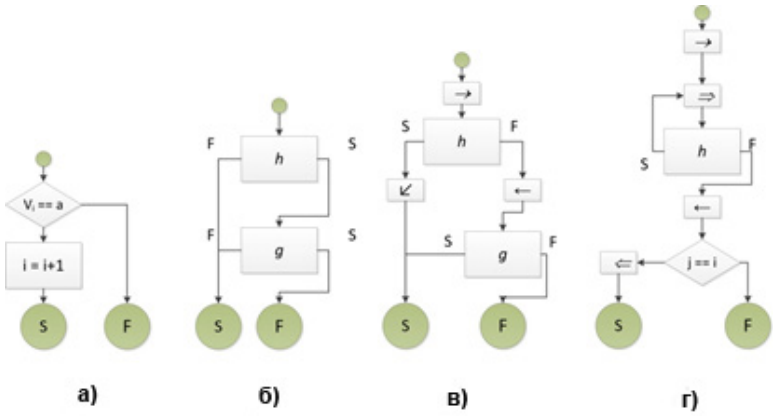

Рис 1. Шаблоны схем

Язык регулярных выражений и транслятор – это система программирования образцов. В настоящее время транслятор этой системы представляет собой препроцессор компилятора С, преобразующий обозначения образцов в программы на языке С. В программе, построенной этим препроцессором, для представления состояния строк используются указатель $char * A$. Шаблон примитивного образа (рис. 1 а) задается выражением int B = *(A + +)^ n; ; где n – распознаваемый символ. Катенация (рис. 1 б) задается шаблоном h if(!B){ g }.

Альтернация (рис. 1 в) задается шаблоном

P.push_back (A); h if (B){ A = P.pop_back (); g } else { P.pop_back (); B = 1;}.

Итерация (рис. 1 г) задается шаблоном

do { P.push_back (A); if (!*B){B =0;break;} h B =(!B); }while (!B); if (!B)B =-1;else {A =P.pop_back();B =0;}.

Анализаторы на основе детерминированного конечного автомата работают линейно по времени, поскольку не нуждаются в откатах (никогда не проверяют дважды один символ анализируемого текста). В анализаторе, построенном системой программирования образцов, откаты, как видно из шаблонов на рисунке 1 б-в, происходят.

Заключение

Образцы алгебры образцов одновременно являются функциональной моделью операции анализа системы составляющих и задают последовательность выполнения примитивных образцов в процессе анализа распознаваемого слова. Общепризнанным формализмом описания структуры системы составляющих текста являются граммати-

ки. Применение побочного эффекта в образцах делает регулярные выражения таким же универсальным формализмом, как и грамматики. Установление структуры составляющих в заданном слове происходит при интерпретации грамматики алгоритмом грамматического разбора. Однако сам алгоритм грамматического разбора частью формализма грамматик не является. Реализация операции анализа, основанная на применении образцов предлагаемого типа, потенциально более эффективна, так как в этом случае имеется возможность управлять последовательностью применения подстановок. Представление подстановки образцами более естественно, особенно в простых случаях, по сравнению с атрибутными грамматиками, т.к. указывая атрибуты для вычисления $f(x) = y$, приходится учитывать особенности алгоритма грамматического разбора. Однако забота о степени свободы в виде возможности управлять последовательностью подстановок в некоторых случаях по сравнению с грамматиками может увеличить трудоемкость разработки алгоритма анализа.

Система программирования образцов успешно использовалась для построения ряда анализаторов форматов [11].

Литература

1. *Фридл Дж*. Регулярные выражения / Дж. Фридл. СПб.: «Питер», 2001. 352 с.
2. *Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.
3. *Aho A.* Algorithms for finding patterns in strings // Handbook for theoretical computer science, MIT Press. Vol. A. 1990. P. 257-300.
4. Сик Д., Лай-Кван Ли, Ламсдэйн Э. C++ Boost Graph Library. – Питер, 2006. – 304 с.
5. *Thurston A.D.* Ragel State Machine Compiler User Guide [Электронный ресурс]. URL: http://www.colm.net/files/ragel/ragel-guide-6.9.pdf (дата обращения: 30.1.2016).
6. *McCarthy J.* Recursive function of symbolic expressions and their computation by machine, part 1. // Comm. ACM. 1960. V. 3. N. 4. P. 184-195.
7. *Черемисинов, Д. И.* Программирование подстановок языков / Д. И. Черемисинов // Программирование, 1981. № 5. С. 30-37.
8. *Себеста Р.У.* Основные концепции языков программирования = Concepts of programming languages. 5-е изд. М.: Вильямс, 2001. С. 282-284.
9. *Мальцев А.И.* Алгоритмы и рекурсивные функции. М.: Наука, 1986. 368 с.
10. *Марков А.А.* Теория алгорифмов // Тр. МИАН СССР, М.-Л.: Изд-во АН СССР, 1954. . Т. 42. С. 3-375.
11. *Черемисинов Д.И.* Перепроектирование ПЛИС на основе трансформации моделей // Проблемы разработки перспективных микро- и наноэлектронных систем – 2014. Сб. трудов / под общ. ред. акад. РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014, Ч. 1. С. 25-30.

The deterministic semantic of the regular expressions

***Cheremisinov Dmitry Ivanovich**, Ph.D.*

The problem of conversion of formal language texts is of considerable practical importance for software reengineering and for the software integration. Pattern matching is technique of searching a text string based on a specific search pattern. The pattern specified by regular expression forms the basis for building a variety of converters. It is proposed to change the traditional model of the language parser by pattern matching based on the finite state machine into the algebra of patterns with side effects. The proposed deterministic semantic of regular expression eliminates the need to switch from the regular expression engine and user code execution environment and back again.

Keywords: formal language parser, regular expression, Kleene algebra, finite automaton, deterministic semantic of regular expression.