УДК 32.973.26-018.2

ЭФФЕКТИВНАЯ ТРАНСЛЯЦИЯ ДЛЯ LL(1)-ГРАММАТИКИ НА ПРИМЕРЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ

Ю. Л. Костюк

Национальный исследовательский Томский государственный университет, г. Томск, Россия

Предлагаются правила построения и функционирования транслятора для LL(1)-грамматики, генерирующего объектный язык. Транслятор представляется в виде таблицы ссылок на набор простых семантических программ. Таблица строится на основе порождающих правил грамматики, преобразованных в нестрогую нормальную форму Грейбах. Такой способ существенно упрощает разработку транслятора. Приведён пример построения транслятора и семантических программ для простого языка программирования, представленного обратной польской строкой.

Ключевые слова: *трансляция, LL(1)-грамматика, обратная польская строка, язык программирования.*

DOI 10.17223/20710410/37/7

EFFECTIVE TRANSLATION FOR LL(1)-GRAMMAR IN THE EXAMPLE OF A PROGRAMMING LANGUAGE

Yu. L. Kostyuk

Tomsk State University, Tomsk, Russia

E-mail: kostyuk_y_l@sibmail.com

Top-down analysis algorithms are the most useful for implementing translators for programming languages. The analysis methods based on the LL(1)-grammar assume that the generating rules of the grammar are previously transformed into a non-strict normal Greibach form. The transformed generating rules are written in an analyzer table. To convert an input string of characters to an object language, a special program for generating individual elements of the object language should be created for each generating rule. The development of the set of such programs is rather a complex and voluminous task when a translator is created. In this article, we propose a method for constructing semantic programs implementing the generation of elements of the object language when each semantic program is associated with the separate symbols in the generating rule but not with the generating rule as a whole. As a result, the semantic programs are extremely simple and convenient to be implemented. The method is described for an exemplifying programming language which is translated into a reverse Polish notation.

Keywords: *translation, LL(1) grammar, reverse Polish notation, programming language.*

Введение

К настоящему времени разработано большое количество различных алгоритмов грамматического анализа контекстно-свободных грамматик [1, 2]. Как отмечают многие исследователи, например Н. Вирт [3], наиболее удобными при реализации трансляторов для языков программирования являются алгоритмы анализа «сверху вниз», в частности метод рекурсивного спуска, когда каждое порождающее правило грамматики программируется в виде рекурсивной функции. Другой метод анализа «сверху вниз» на основе LL(1)-грамматики предполагает, что порождающие правила грамматики предварительно преобразуются в нестрогую нормальную форму Грейбах [2]. В такой форме все правые части порождающих правил либо пусты, либо начинаются с терминальных символов. Преобразованные порождающие правила заносятся в таблицу анализатора. Если, кроме анализа, требуется также преобразовать входную цепочку символов в объектный язык, то для каждого порождающего правила создается специальная программа генерации отдельных элементов объектного языка. Разработка набора таких программ является довольно сложной и объёмной задачей при создании транслятора. В данной работе предлагается такой способ построения семантических программ, реализующих генерацию элементов объектного языка, когда каждая программа связывается не с полным порождающим правилом, а с отдельными символами в порождающем правиле. В результате семантические программы получаются предельно простыми и удобными для реализации. В кратком виде эта идея была предложена автором в [4]. В данной работе такой способ изложен на примере простого языка программирования, который транслируется в обратную польскую строку (ОПС).

1. LL(1)-анализатор и транслятор для ОПС

Контекстно-свободная грамматика, преобразованная к нестрогой нормальной форме Грейбах, допускает детерминированный LL(1)-анализ, если для каждой группы порождающих правил с одним и тем же нетерминальным символом в левой части разные правые части будут однозначно различимы по первому терминальному символу. Пусть входная цепочка символов всегда заканчивается символом-ограничителем «$\perp$». Для работы LL(1)-анализатора необходимо построить таблицу, в которой столбцы помечены терминальными символами (включая ограничитель), а строки — нетерминальными символами преобразованной грамматики. Для всех порождающих правил грамматики, имеющих вид

$$A \to a\gamma$$

(где A — нетерминальный символ; a — терминальный символ; γ — цепочка из терминальных и нетерминальных символов), правая часть правила $a\gamma$ заносится на пересечение строки, помеченной символом A, и столбца, помеченного символом a. Если порождающее правило имеет вид

$$A \to \lambda,$$

где λ — пустая цепочка, то во все клетки строки, помеченной нетерминальным символом A и не занятые другими правилами, записывается λ.

В качестве примера рассмотрим грамматику простых арифметических выражений [1], в которой символы S, T, F — нетерминальные, причём S — начальный. Символы «+», «*», круглые скобки и a — терминальные; вертикальная черта разделяет различные правые части при одном и том же нетерминальном символе в левой части правил:

$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$

После преобразования грамматики к нестрогой нормальной форме Грейбах получим порождающие правила с двумя дополнительными нетерминальными символами U и V:

$$S \to (S)VU \mid aVU$$
$$U \to +TU \mid \lambda$$
$$T \to (S)V \mid aV$$
$$V \to *FV \mid \lambda$$
$$F \to (S) \mid a$$

Функционирование анализатора. До начала работы в стек анализатора записывается ограничитель «$\perp$» (считающийся терминальным символом), а затем начальный нетерминальный символ. На каждом шаге анализатор проверяет, допустим ли очередной входной символ, и если да, то выполняет одно из двух действий:

1) если на вершине стека нетерминальный символ, то, в зависимости от того, каков очередной входной символ, этот нетерминальный символ заменяется в стеке символами правой части соответствующего правила, причём символы записываются в обратном порядке. Если для очередного входного символа в таблице записано λ, то нетерминальный символ просто удаляется из стека, а если в таблице пустая клетка, то анализатор фиксирует ошибку во входной цепочке;
2) если на вершине стека терминальный символ, то он сравнивается с очередным входным символом. При совпадении терминальный символ удаляется из стека и делается переход к следующему символу входной цепочки. При несовпадении анализатор фиксирует ошибку.

Работа анализатора прекращается, когда входная цепочка символов окажется полностью просмотренной. Если при этом стек пуст, то цепочка считается правильной, если не пуст — ошибочной.

Трансляция входной цепочки символов в ОПС выполняется параллельно с работой LL(1)-анализатора. Для транслятора необходим второй стек, действия с которым выполняются синхронно с действиями со стеком анализатора. Во второй стек записываются семантические символы, которые генерируют элементы ОПС. Генерация элементов ОПС выполняется при извлечении семантических символов из второго стека.

Для реализации транслятора наряду с основной таблицей LL(1)-анализатора необходимо задать семантическую таблицу. Её размеры в точности совпадают с таблицей анализатора, более того, для каждой непустой клетки таблицы анализатора, где находится правая часть какого-либо правила порождения, в семантическую таблицу помещаются семантические символы, количество которых в точности такое же, как в правой части этого правила порождения. При этом надо учесть, что терминальные символы в контекстно-свободной грамматике на самом деле являются лексемами, распознаваемыми лексическим анализатором. Некоторые из лексем (имена переменных и константы) содержат дополнительную семантическую информацию, т. е. одинаковые терминальные символы в контекстно-свободной грамматике могут различаться семантически.

В качестве примера рассмотрим семантические действия для грамматики простых арифметических выражений, обозначаемые следующими семантическими символами:

a — запись в ОПС операнда из входной цепочки (переменной или константы);

«+» — запись в ОПС операции сложения;

«∗» — запись в ОПС операции умножения;

«○» — пустое действие.

Действия выполняются в момент выталкивания соответствующих символов из второго стека. В табл. 1 представлена совмещённая таблица: анализатора и семантическая таблица транслятора.

Таблица 1

Нетерм. символ	$+$	$*$	$($	$)$	a	$\perp$
S			$(S)VU$ ○ ○ ○ ○ ○		aVU a ○ ○	
U	$+TU$ ○ ○ $+$	λ	λ	λ	λ	λ
T			$(S)V$ ○ ○ ○ ○		aV a ○	
V	λ	$*FV$ ○ ○ $*$	λ	λ	λ	λ
F			(S) ○ ○ ○		a a	

Так как одинаковые терминальные символы a в контекстно-свободной грамматике семантически могут различаться, во входной цепочке различные операнды будем обозначать разными символами: x, y и др., при этом все они соответствуют одному и тому же терминальному символу a. При этом операнды могут быть различных типов — именами переменных, константами, — а лексический анализатор, непосредственно просматривающий входную цепочку символов, на каждом шаге работы LL(1)-анализатора выдаёт ему тип терминального символа (лексемы).

До начала работы в стек анализатора записывается символ-ограничитель и начальный нетерминальный символ, а во второй стек — два символа, обозначающих пустое действие. На каждом шаге работы транслятора делается следующее:

— если верхний символ в стеке анализатора нетерминальный, то во втором стеке верхний символ (который при этом будет обязательно пустым действием) удаляется и во второй стек записывается последовательность символов из семантической таблицы (эта последовательность находится на пересечении строки, помеченной нетерминалом, и столбца, помеченного входным терминалом);

— если верхний символ в стеке анализатора — терминал, то его тип должен соответствовать типу терминала на входе (в противном случае — ошибка), и тогда из второго стека удаляется верхний символ и выполняется соответствующее действие; если это действие — операция, то она просто записывается в ОПС, а если это символ a и терминал на входе — константа, то значение этой константы записывается в ОПС; если действие — символ a и терминал на входе — имя, то генератор ищет это имя в таблице переменных, и если оно там найдено, то в ОПС записывается тип переменной и ссылка на неё в таблице.

В последнем случае, если имя в таблице переменных отсутствует, то действия зависят от входного языка. Если в языке требуется, чтобы все имена были предварительно

описаны, то фиксируется ошибка, а если имена считаются описанными по умолчанию, то в таблицу переменных заносится новое имя.

Пример работы LL(1)-анализатора и генератора ОПС, заданного в табл. 1, для входной цепочки «$x * (c + d)\ \perp$» представлен в табл. 2.

Таблица 2

№ шага	Входные символы	Содержимое стека анализатора	Содержимое второго стека	Порождающее правило	ОПС
1	$x * (c + d)\ \perp$	$S\ \perp$	$\circ\ \circ$	$S \to aVU$	
2	$x * (c + d)\ \perp$	$aVU\ \perp$	$a\ \circ\ \circ\ \circ$		x
3	$*(c + d)\ \perp$	$VU\ \perp$	$\circ\ \circ\ \circ$	$V \to *FV$	x
4	$*(c + d)\ \perp$	$*FVU\ \perp$	$\circ\ \circ * \circ\ \circ$		x
5	$(c + d)\ \perp$	$FVU\ \perp$	$\circ * \circ\ \circ$	$F \to (S)$	x
6	$(c + d)\ \perp$	$(S)VU\ \perp$	$\circ\ \circ\ \circ * \circ\ \circ$		x
7	$c + d)\ \perp$	$S)VU\ \perp$	$\circ\ \circ * \circ\ \circ$	$S \to aVU$	x
8	$c + d)\ \perp$	$aVU)VU\ \perp$	$a\ \circ\ \circ\ \circ * \circ\ \circ$		$x\ c$
9	$+d)\ \perp$	$VU)VU\ \perp$	$\circ\ \circ\ \circ * \circ\ \circ$	$V \to \lambda$	$x\ c$
10	$+d)\ \perp$	$U)VU\ \perp$	$\circ\ \circ + \circ * \circ\ \circ$	$U \to +TU$	$x\ c$
11	$+d)\ \perp$	$+TU)VU\ \perp$	$\circ\ \circ + \circ * \circ\ \circ$		$x\ c$
12	$d)\ \perp$	$TU)VU\ \perp$	$\circ + \circ * \circ\ \circ$	$T \to aV$	$x\ c$
13	$d)\ \perp$	$aVU)VU\ \perp$	$a\ \circ + \circ * \circ\ \circ$		$x\ c\ d$
14	$)\ \perp$	$VU)VU\ \perp$	$\circ + \circ * \circ\ \circ$	$V \to \lambda$	$x\ c\ d$
15	$)\ \perp$	$U)VU\ \perp$	$+\ \circ * \circ\ \circ$	$U \to \lambda$	$x\ c\ d+$
16	$)\ \perp$	$)VU\ \perp$	$\circ * \circ\ \circ$		$x\ c\ d+$
17	$\perp$	$VU\ \perp$	$*\ \circ\ \circ$	$V \to \lambda$	$x\ c\ d+*$
18	$\perp$	$U\ \perp$	$\circ\ \circ$	$U \to \lambda$	$x\ c\ d+*$
19	$\perp$	$\perp$	$\circ$		$x\ c\ d+*$

2. Простой язык програмирования и ОПС для него

Рассмотрим действия транслятора на примере простого языка программирования, в котором определены почти все типичные для таких языков элементы. Определим синтаксис языка следующим набором порождающих правил:

$P \to$ **begin** DAB **end**
$D \to$ **dim** $a[k]; D \mid \lambda$
$A \to aH = S \mid$ **if** C **then** $ABE \mid$ **while** C **do** AB **end** $\mid$ **in** $aH \mid$ **out** S
$B \to; AB \mid \lambda$
$E \to$ **else** AB **end** $\mid$ **end**
$C \to S = S \mid S < S \mid S > S \mid S \neq S$
$S \to S + T \mid S - T \mid T$
$T \to T * F \mid T/F \mid F$
$F \to (S) \mid aH \mid k$
$H \to [S] \mid \lambda$

Нетерминальные символы, обозначенные заглавными латинскими буквами, имеют следующий смысл:

— P — начальный нетерминальный символ, порождает всю программу;
— D — определяет описания массивов;
— A — определяет операторы (присваивание, условный, цикл, ввод, вывод);

— B — определяет последовательность операторов;
— E — определяет вторую часть условного оператора;
— C — определяет сравнение (равно, меньше, больше, не равно) для двух выражений;
— S, T, F — определяют выражение с операциями сложения, вычитания, умножения, деления, а также скобками;
— H — определяет индексирование массива.

Жирным шрифтом выделены зарезервированные служебные слова, они, наряду со знаками операций, скобками, именами (обозначаемыми буквой a) и числовыми константами (обозначаемыми буквой k), являются терминальными символами для контекстно-свободной грамматики и распознаются лексическим анализатором как лексемы. Имена простых переменных в языке описаны по умолчанию; в описании массива константа k задаёт количество элементов в нём. Типы переменных, массивов и констант в общем случае вещественные.

При трансляции генерируется не только ОПС, но также таблица массивов и таблица простых переменных. Каждый элемент таблицы массивов содержит имя массива, длину и ссылку на начало расположения его элементов в памяти. Каждый элемент таблицы простых переменных содержит имя и ссылку на расположение этой переменной в памяти. Эти таблицы используются в процессе работы транслятора.

Отдельный элемент ОПС может быть операндом или операцией. Интерпретация (исполнение) ОПС как программы производится с использованием стека для операндов и результатов операций. Операнды последовательно заносятся в стек. Если встретилась операция, то она исполняется с использованием операндов в верхней части стека, их количество определяется операцией. Результат операции либо заносится в стек, либо запоминается другим способом.

Операнд в ОПС состоит из двух частей: типа, а также значения или ссылки. Типы операндов:

— переменная, ссылка указывает расположение переменной в памяти;
— массив, ссылка указывает расположение начального элемента массива в памяти;
— константа, вторая часть — её значение;
— метка, ссылка указывает порядковый номер того элемента ОПС, на который требуется передать управление.

Операция в ОПС задаётся её номером или обозначением. Для рассматриваемого языка определены следующие операции:

— арифметические (сложение, вычитание, умножение, деление), обозначаемые знаками «+», «−», «*», «/»; они требуют по два операнда, результат заносится в стек;
— сравнения (равно, меньше, больше, не равно), обозначаемые знаками «=», «<», «>», «≠»; они требуют по два операнда, результат может быть 1 (истина) или 0 (ложь) и заносится в стек;
— присваивание, обозначается знаком «←» (чтобы различать с операцией равно); требует два операнда, результат (значение второго операнда) запоминается в памяти по ссылке (из первого операнда), в стек ничего не заносится;
— индексация, обозначается **ind**, требует два операнда, ссылка на массив (первый операнд) складывается со значением второго операнда (индексом), результат (ссылка на индексируемый элемент массива) заносится в стек; предполагается, что элементы в массиве пронумерованы числами 0, 1 и т. д.;

— ввод, обозначается **in**, требует один операнд — ссылку, вводит числовое значение со стандартного устройства ввода и запоминает в памяти по ссылке, в стек ничего не заносится;
— вывод, обозначается **out**, требует один операнд — значение, выводит это значение на стандартное устройство вывода в виде изображения числа, в стек ничего не заносится;
— выделение памяти для массивов и простых переменных, обозначается **mem**, требует один операнд, выделяет память указанного размера, в стек ничего не заносится;
— освобождение выделенной памяти, обозначается **del**, не требует операндов, в стек ничего не заносится;
— безусловный переход по метке, обозначается **j**, требует один операнд — метку (номер элемента ОПС), производит переход к указанному элементу ОПС, в стек ничего не заносится;
— условный переход по метке, обозначается **jf**, требует два операнда; если значение первого операнда равно 0 (ложь), то производит переход по второму операнду — метке — к указанному элементу ОПС; в противном случае ничего не делается; в обоих случаях в стек ничего не заносится.

3. Трансляция языка програмирования

Для построения транслятора необходимо сначала преобразовать порождающие правила языка в нестрогую нормальную форму Грейбах. Если после этого для некоторого нетерминального символа различные правые части начинаются с одного и того же терминального символа, то к ним можно применить преобразование, называемое факторизацией. При этом в грамматику добавится ещё один нетерминальный символ.

Затем для каждой правой части порождающего правила (исключая пустое порождение λ) необходимо сконструировать последовательность семантических действий, генерирующих элементы ОПС, и записать их в виде семантических символов. Их должно быть ровно столько, сколько символов (терминальных и нетерминальных) имеется в соответствующей правой части порождающего правила. Семантические символы обозначают следующие действия:

— a — имя, распознанное лексическим анализатором из входной цепочки символов, ищется в таблице переменных и в таблице массивов; если отсутствует в обеих таблицах, то добавляется в таблицу переменных, при этом счетчик количества занятой памяти увеличивается на 1; затем ссылка на расположение в памяти этой переменной или массива записывается в ОПС как операнд;
— k — значение константы, распознанное лексическим анализатором из входной цепочки символов, записывается в ОПС как операнд;
— операция, обозначенная знаком («+», «−» и др.) или словом (**in**, **out** и др.) — записывается в ОПС;
— «○» — пустое действие;
— число (от 1 до 9) — задаёт выполнение соответствующей семантической программы, алгоритмы этих программ приведены ниже.

Для синхронизации работы транслятора с первым и вторым стеком некоторые порождающие правила в конце правой части необходимо дополнить нетерминальным символом Z, который всегда порождает пустую цепочку. Например, правило $A \to a = S$ расширено до $A \to a = SZ$. Это сделано для того, чтобы операция присваивания записывалась в ОПС после того, как будут сгенерированы все операнды и операции в правой части присваивания.

Далее приведены преобразованные порождающие правила и взятые в фигурные скобки соответствующие им последовательности семантических символов:

$P \to$ **begin** DAB **end** $\{1 \circ \circ \circ 9\}$
$D \to$ **dim** $a[k]; D$ $\{\circ\, 2 \circ 3 \circ \circ \circ\} \mid \lambda$
$A \to aH = S\ \{a \circ \circ \circ \leftarrow\} \mid$ **if** C **then** $ABE\ \{\circ \circ 4 \circ \circ \circ\}$ $\mid$
while C **do** AB **end** $\{5 \circ 4 \circ \circ 6\} \mid$ **in** $aHZ\{\circ \circ \circ$ **in**$\} \mid$ **out** $SZ\{\circ \circ$ **out**$\}$
$B \to ; AB\ \{\circ \circ \circ\} \mid \lambda$
$E \to$ **else** AB **end** $\{7 \circ \circ 8\} \mid$ **end** $\{8\}$
$C \to (S)VUQ\ \{\circ \circ \circ \circ \circ \circ \circ\} \mid aVUQ\ \{\circ \circ \circ \circ\} \mid kVUQ\ \{\circ \circ \circ \circ\}$
$Q \to = SZ\ \{\circ \circ =\} \mid < SZ\ \{\circ \circ <\} \mid > SZ\ \{\circ \circ =\} \mid \neq SZ\ \{\circ \circ \neq\}$
$S \to (S)VU\ \{\circ \circ \circ \circ \circ \circ\} \mid aVU\ \{\circ \circ \circ\} \mid kVU\ \{\circ \circ \circ\}$
$U \to +TU\ \{\circ \circ +\} \mid -TU\ \{\circ \circ -\} \mid \lambda$
$T \to (S)V\ \{\circ \circ \circ \circ \circ\} \mid aV\ \{\circ \circ\} \mid kV\ \{\circ \circ\}$
$V \to *FV\ \{\circ \circ *\} \mid /FV\ \{\circ \circ /\} \mid \lambda$
$F \to (S)\ \{\circ \circ \circ\} \mid aH\ \{a \circ\} \mid k\ \{k\}$
$H \to [S]\ \{\circ \circ$ **ind**$\} \mid \lambda$
$Z \to \lambda$

Рассмотрим алгоритмы семантических программ. При их выполнении используется ещё один, третий, стек, а также две переменные — i и m. Переменная i — счётчик (количество) сгенерированных элементов ОПС, m — размер выделенной памяти для массивов и переменных. Счётчик i увеличивается на 1 всякий раз, как только генерируется очередной элемент ОПС.

Программа 1.

1. $i := 1$, $m := 0$.
2. В ОПС записывается 0 — место для записи в последующем размера выделенной памяти.
3. В третий стек записывается 1 (ссылка на зарезервированный элемент ОПС).

Программа 2.

1. Имя, распознанное лексическим анализатором из входной цепочки символов, ищется в таблице массивов.
2. Если имя найдено, то фиксируется ошибка.
3. Если имя в таблице массивов отсутствует, то добавляется в таблицу массивов.
4. В ту же строку таблицы массивов записывается значение переменной m (ссылка на начало расположения массива в памяти).

Программа 3.

1. Константа, распознанная лексическим анализатором из входной цепочки символов, добавляется к переменной m.

Программа 4.

1. В третий стек записывается i.
2. В ОПС записывается 0 — место для будущей метки.
3. В ОПС записывается операция **jf** — переход при условии `false`.

Программа 5.

1. В третий стек записывается i.

Программа 6.

1. Через верхний элемент третьего стека, как ссылку на ранее заготовленное место для метки, записывается $i + 2$.

2. В ОПС записывается метка, значение для которой читается из третьего стека.
3. В ОПС записывается операция **j** — безусловный переход.

Программа 7.

1. Через верхний элемент третьего стека, как ссылку на ранее заготовленное место для метки, записывается $i + 2$.
2. В третий стек записывается i.
3. В ОПС записывается 0 — место для будущей метки.
4. В ОПС записывается операция **j** — безусловный переход.

Программа 8.

1. Через верхний элемент третьего стека, как ссылку на ранее заготовленное место для метки, записывается i.

Программа 9.

1. Через верхний элемент третьего стека, как ссылку на ранее заготовленное место в ОПС, записывается значение переменной m (размер выделенной памяти для массивов и переменных).

Благодаря использованию третьего стека генерируется корректная ОПС для вложенных условных операторов и циклов. Например, по следующей входной цепочке символов (фрагменту программы, которая упорядочивает массив `A`):

```
i=1;
while i<n do
  if A[i-1]>A[i] then
    x=A[i-1]; A[i-1]=A[i]; A[i]=x;
    if i>1 then i=i-1 end
  else i=i+1 end
end;
```

будет сгенерирована следующая ОПС, в которой метки как операнды обозначены именами `m1, m2, m3,` а места ОПС, куда они ссылаются, вынесены влево и обозначены именами меток с двоеточием:

```
    i 1 ←
m1: i n < m3 jf
    A i 1 - ind A i ind > m2 jf
    x A i 1 - ind ← A i 1 - ind A i ind ← A i ind x ←
    i 1 > m2 jf i i 1 - ←
m2: i i 1 + ←
    m1 j
m3:
```

Заключение

Рассмотренный пример грамматики простейшего языка программирования является демонстрационным, но его нетрудно расширить, добавив и другие операторы, операции, типы и структуры переменных и констант, описания и т. п., получив в результате универсальный язык программирования. Использование предложенных принципов построения транслятора позволяет большую часть описания анализируемого языка разместить в таблицах, за исключением некоторых действий, программируемых дополнительно в виде небольших семантических программ. В результате трудозатраты при разработке транслятора существенно сокращаются. Использование ОПС не является существенным ограничением, так как это удобная форма внутреннего представ-

ления транслируемой программы. При необходимости, скорректировав правила функционирования транслятора, можно генерировать иные структуры (тройки, четвёрки и т. п.). Наконец, на основе полученной ОПС можно генерировать последовательность машинных команд.

ЛИТЕРАТУРА

1. *Ахо А. В., Лам М. С., Сети Р., Ульман Дж. Д.* Компиляторы: принципы, технологии и инструментарий. 2-е изд. М.: Вильямс, 2008. 1184 с.
2. *Mogensen T. A.* Introduction to Compiler Design. Springer, 2011. 225 p.
3. *Вирт Н.* Построение компиляторов. М.: ДМК Пресс, 2010. 192 с.
4. *Костюк Ю.Л̇.* Табличный способ генерации обратной польской строки для LL(1)-грамматики // Информационные технологии и математическое моделирование (ИТММ-2015). Материалы XIV Междунар. конф. им. А. Ф. Терпугова, 18–22 ноября 2015. Ч. 1. Томск: Изд-во Том. ун-та, 2015. С. 190–195.

REFERENCES

1. *Aho V. A., Lam M. S., Sethi R., and Ullman J. D.* Compilers: Principles, Techniques, and Tools. Boston, Pearson Education, 2007.
2. *Mogensen T. A.* Introduction to Compiler Design. Springer, 2011. 225 p.
3. *Virt N.* Theory and Techniques of Compiler Construction. Addison-Wesley, Reading, 1996.
4. *Kostyuk Yu. L.* Tablichnyj sposob generacii obratnoj pol'skoj stroki dlya LL(1)-grammatiki [The tabular method of generating reverse Polish notation for LL(1) grammars]. Proc. ITMM-2015, vol. 1. Tomsk, TSU Publ., 2015, pp. 190–195. (in Russian)