О LL(1)-грамматиках, алгоритмах на них и методах их анализа в программировании

С. В. Козлов, А. В. Светлаков

Аннотация **– В статье рассматриваются теоретические основы синтаксического анализа, а именно определяются LL(1)-грамматики и доказываются ключевые для практики положения, связанные с ними. Авторами на примерах демонстрируется непрактичность доказательства по определению, что контекстно-свободная грамматика является LL(1)-грамматикой. Ввиду этого формулируется и доказывается теорема, которая задает критерий LL(1)-грамматики. Производится построение множеств *first* и *follow*, которые определяют необходимые и достаточные условия существования LL(1)-грамматики. На примерах показывается применение сформулированного критерия. Центральное место статьи занимает вопрос поиска грамматик эквивалентных LL(1)-грамматике. Рассматривается ряд утверждений, позволяющих выявить, что анализируемая грамматика не является LL(1)-грамматикой. В частности анализируются два необходимых условия существования LL(1)-грамматики. А именно, теорем, что если грамматика содержит левую рекурсию или правое ветвление, то она не является LL(1)-грамматикой. При этом на примере демонстрируется факт того, что эти условия не являются достаточными. Обсуждаются и сравниваются два метода анализа LL(1)-грамматик, применяемых на практике: метод рекурсивного спуска и метод выбросов-переносов. Для каждого из методов приводится его содержательное описание и реализация на псевдокоде. Все положения статьи сопровождаются необходимыми примерами. Актуальность статьи связана с поиском и изучением алгоритмов синтаксического анализа грамматик естественных и искусственных языков, которые успешно применяются как инструменты для написания систем распознавания образов в области искусственного интеллекта.**

Ключевые слова **– синтаксический анализ, LL(1)-грамматика, левая рекурсия, правое ветвление, множества *first* и *follow*, таблица прогнозов, метод рекурсивного спуска, метод выбросов-переносов.**

Статья получена 28 декабря 2021.

Козлов Сергей Валерьевич, Смоленский государственный университет, доцент кафедры прикладной математики и информатики, кандидат педагогических наук, доцент (email: svkozlov1981@yandex.ru)

Светлаков Алексей Владимирович, Смоленский государственный университет, студент физико-математического факультета (email: seferlian@mail.ru)

I. ВВЕДЕНИЕ

В настоящее время развитие IT-технологий и их внедрение в большинство сфер жизнедеятельности человека служит катализатором поиска все новых более эффективных решений задач практики [1, 2, 3]. Так теория формальных грамматик выступает средством моделирования работы многих инструментальных сред. Она определяет математический аппарат для формализации различных языков программирования, функциональные методы которых образуют библиотеки ресурсов для выполнения различных сценариев работы [4, 5]. Так, например, одной из актуальных областей приложения формальных грамматик является написание частотных словарей [6, 7, 8], трансляторов [9, 10, 11] и систем распознавания в алгоритмах искусственного интеллекта [12, 13]. Теория формальных грамматик составляет основу для систем распознавания образов и речи, автоматического перевода текста и генерации программного кода, криптографии и интеллектуального анализа данных [14, 15]. При этом в теории формальных грамматик существует несколько направлений методологий от контекстно-зависимых и контекстно-свободных, до автоматных грамматик и языков [16, 17, 18].

LL-анализ является одним из способов синтаксического анализа цепочек контекстно-свободных языков (КС-языков), время работы которого линейно зависит от длины цепочки [19, 20]. Более того, его частный случай – LL(1)-анализ является наиболее простым и интуитивно понятным [21]. Разумеется, за подобные достоинства приходится платить тем, что LL(1)-анализ применим лишь к достаточно ограниченному классу грамматик. В частности, он практически бесполезен при анализе естественных языков, а грамматики искусственных языков приходится приводить к определенному виду, а иногда менять и сам язык. В этих случаях целесообразно применять иные методологии анализа данных и связей между ними [22, 23]. Несмотря на это, на практике LL(1)-анализа часто бывает достаточно для проведения синтаксического

анализа. Например, язык арифметических выражений может быть описан подходящей грамматикой, как и многие языки исполнителей. Известен не один язык программирования, интерпретатор которых основан на LL-анализе.

Две буквы «L» в названии говорят о том, что, во-первых, цепочка считывается слева направо (Left to right), а во-вторых, имитируется ее левосторонний вывод (Leftmost derivation). Единица говорит о том, что одномоментно считывается один символ. Обратим внимание, что «символ» здесь и далее употребляется в широком смысле – это может быть как буква или знак, так и лексема (токен) языка, а то и цепочка, удовлетворяющая регулярному выражению. LL(1)-анализ проводится над так называемыми LL(1)-грамматиками. Дадим необходимые определения.

II. ОПРЕДЕЛЕНИЕ LL(1)-ГРАММАТИКИ

Определение 1. Пусть G – КС-грамматика. Рассмотрим два произвольных левосторонних вывода цепочки ω в этой грамматике: $S \Rightarrow pA\beta \to p\alpha\beta \Rightarrow py\eta$ и $S \Rightarrow pA\beta \to p\alpha'\beta \Rightarrow py\xi$, где $p, y \in T^*$ – цепочки из терминалов: разобранная часть цепочки ω, $A \in N$, причем существуют правила $A \to \alpha$ и $A \to \alpha'$, а также $\alpha, \alpha', \beta, \eta, \xi \in (T \cup N)^*$. Если из выполнения условий: $|y| = 1$ или $|y| = 0, \eta = \xi = \varepsilon$ следует равенство $\alpha = \alpha'$, то G – LL(1)-грамматика.

Говоря простыми словами, LL(1)-грамматика – это такая грамматика, что посмотрев на очередной символ (или на конец) выведенной части цепочки, можно однозначно сказать, какое правило было использовано.

Пример 1. Грамматика задана правилами $S \to SS|aSb|\varepsilon$. Эта грамматика порождает язык с правильно сбалансированными символами a и b, то есть каждый символ «a» закрывается справа символом b, причем никакой символ не может быть закрыт раньше, чем закроются вложенные (символы a и b можно заменить на открывающуюся и закрывающуюся скобки соответственно). Требуется определить, является ли грамматика LL(1)-грамматикой.

Рассмотрим следующую терминальную цепочку, выводимую из аксиомы: «ab». Она выводима из сентенциальной формы вида $a\eta$. Так как $|a| = 1$, то должно выполняться $\alpha = \alpha'$, но в данной грамматике $S \to SS \to aSbS = a\eta$ и $S \to aSb = a\xi$ (оба левосторонние выводы), где $\alpha = SS$ и $\alpha' = aSb$. Следовательно, так как $\alpha \neq \alpha'$, данная грамматика – не LL(1)-грамматика. Заметим, что из $a\eta$ и $a\xi$ может выводиться одна и та же цепочка «ab», однако в общем случае это требование необязательно. Это иллюстрирует следующий пример.

Пример 2. Грамматика имеет правила $S \to aSa|bSb|\varepsilon$. Эта грамматика порождает язык палиндромов четной длины над алфавитом $\{a, b\}$. Проверим, является ли она LL(1)-грамматикой. Будем анализировать цепочку $aa\eta$. Построим возможные левосторонние выводы этой цепочки: $S \to aSa \to aa = aa\eta$ и $S \to aSa \to aaSaa = aa\xi$. Так как $|a| = 1$ (второй символ), то должно выполняться $\alpha = \alpha'$, но получаем $\alpha \neq \alpha'$, так как $\alpha = \varepsilon$ и $\alpha' = aSa$. Неформально говоря, получаем, что при встрече второго символа a мы не знаем, каким правилом воспользоваться: $S \to \varepsilon$ или $S \to aSa$. Следовательно, это не LL(1)-грамматика.

Пример 3. Докажем, что никакой неоднозначный язык не может быть порожден LL(1)-грамматикой. Действительно, по определению, неоднозначный язык содержит хотя бы одну цепочку, вывести которую можно как минимум двумя различными способами. Рассмотрим эту цепочку. В таком случае на некотором этапе вывода окажется, что $pA\beta \to p\alpha\beta \Rightarrow \omega$ и $pA\beta \to p\alpha'\beta \Rightarrow \omega$, причем $\alpha \neq \alpha'$. Здесь возможны два случая: 1) в цепочке ω непосредственно справа от подцепочки p может стоять символ y, но тогда $|y| = 1$ и $\alpha \neq \alpha'$; 2) $\omega = p$, то есть непосредственно справа от p ничего не стоит, но в этом случае $|y| = 0$, а значит $p\alpha\beta \Rightarrow py\eta = \omega$ и $p\alpha'\beta \Rightarrow py\xi = \omega$, или $\eta = \xi = \varepsilon$. Из этого условия следует $\alpha \neq \alpha'$. Таким образом, любая грамматика неоднозначного языка не будет LL(1)-грамматикой.

Очевидно, доказывать по определению данный факт достаточно проблематично, а порой и невозможно, потому на практике оно не используется.

III. КРИТЕРИЙ LL(1)-ГРАММАТИКИ

Введем в рассмотрение следующие множества.

Определение 2. Множеством $first(\alpha)$ назовем множество всех терминалов (а также ε), с которых могут начинаться любые выводы из β, то есть $first(\alpha) \stackrel{\text{def}}{=} \{a | \alpha \Rightarrow a\beta\} \cup \{\varepsilon \text{ если } \alpha \Rightarrow \varepsilon\}$.

Определение 3. Множеством $follow(A)$ назовем множество всех терминалов и символа $ (конца цепочки), которые могут появляться в сентенциальных формах непосредственно справа от нетерминала A : $follow(A) \stackrel{\text{def}}{=} \{a | S \Rightarrow \alpha A c \beta\} \cup \{\$ \text{ если } S \Rightarrow \alpha A\}$.

Сформулируем и докажем теорему о связи LL(1)-грамматик со множествами $first(\alpha)$ и $follow(A)$.

Теорема 1 (критерий LL(1)-грамматики). G является LL(1)-грамматикой, тогда и только тогда, когда выполняются условия:

1) $A \to \alpha|\beta \Rightarrow first(\alpha) \cap first(\beta) = \emptyset$;

2) $A \to \alpha|\beta, \varepsilon \in first(\alpha) \Rightarrow follow(A) \cap first(\beta) = \emptyset$.

Доказательство. Будем рассматривать сентенциальную форму $pc\gamma$, где $p \in T^*$ - выведенная подцепочка терминалов, $c \in T$ – текущий терминал, $\gamma \in (N \cup T)^*$ - оставшаяся еще невыведенная часть цепочки.

Необходимость. G – LL(1)-грамматика.

Случай 1: допустим, что для $A \to \alpha|\beta$ при $\alpha \neq \beta$ выполняется $first(\alpha) \cap first(\beta) \neq \emptyset$, то есть существует символ c, который находится на пересечении этих множеств. Рассмотрим два левосторонних вывода цепочки $pc\gamma$: $S \Rightarrow pA\delta \to p\alpha\delta \Rightarrow pc\alpha'\delta$ и $S \Rightarrow pA\delta \to p\beta\delta \Rightarrow pc\beta'\delta$ (последние выводы возможны, так как $c \in first(\alpha) \cap first(\beta)$). Так как $|c| = 1$, то по определению LL(1)-грамматики $\alpha = \beta$, но это противоречит тому, что $A \to \alpha$ и $A \to \beta$ различные правила.

Случай 2: допустим, что для $A \to \alpha|\beta$ при $\alpha \neq \beta$ и $\varepsilon \in first(\alpha)$ выполняется $c \in follow(A) \cap first(\beta)$. Рассмотрим два левосторонних вывода какой-либо цепочки $pc\gamma$, где непосредственно справа от нетерминала A встречается символ c: $S \Rightarrow pAc\delta \to p\alpha c\delta \Rightarrow pc\delta$ и $S \Rightarrow pAc\delta \to p\beta c\delta \Rightarrow pc\beta' c\delta$. Так как $|c| = 1$, то по определению LL(1)-грамматики $\alpha = \beta$, получено противоречие.

Достаточность. Известно, что выполняются условия (1) и (2). Пусть G – не LL(1)-грамматика.

Случай 1: нетерминал не обращается в ε. Тогда у какой-то цепочки $pc\gamma$ возможны два различных вывода вида $S \Rightarrow pA\delta \to p\alpha\delta \Rightarrow pc\alpha'\delta$ и $S \Rightarrow pA\delta \to p\beta\delta \Rightarrow pc\beta'\delta$ такие, что $\alpha \neq \beta$, но это противоречит условию (1), так как $follow(A) \cap first(\beta) = \{c\}$.

Случай 2: если $A \Rightarrow \varepsilon$, то у какой-либо цепочки $pc\gamma$ возможны различные выводы вида $S \Rightarrow pAc\delta \to p\alpha c\delta \Rightarrow pc\delta$ и $S \Rightarrow pAc\delta \to p\beta c\delta \Rightarrow pc\beta' c\delta$. Но в этом случае $follow(A) \cap first(\beta) = \{c\}$, что противоречит условию (2). ■

Следует заметить, что в доказательстве рассматривались только левосторонние выводы. Действительно, любой вывод можно сделать левосторонним (об этом говорит еще одна теорема, не рассматриваемая в статье).

Рассмотрим примеры. По определению грамматика из примера 1 не является LL(1)-грамматикой. Покажем, что грамматика не удовлетворяет доказанному критерию. Действительно, $first(aSb) = \{a\}$, так как все выводы этой цепочки будут начинаться с символа a . $first(SS) = \{a\}$, так как $S \to aSb$,таким образом, нарушается условие (1): $first(aSb) \cap first(SS) = \{a\}$.

Пример 4. Грамматика имеет правила $S \to xSyS|ySxS|\varepsilon$. Эта грамматика порождает язык с равным количеством символов x и y, расположенных в произвольном порядке. Проверить по критерию, является ли она LL(1)-грамматикой. Очевидно, что условие (1) критерия выполняется: $first(xSyS) = \{x\}$, $first(ySxS) = \{y\}$, $first(\varepsilon) = \{\varepsilon\}$. Так как $\varepsilon \in first(\varepsilon)$, проверим условие (2): $follow(S) = \{x, y, \$\}$. Таким образом, данная грамматика – не LL(1)-грамматика, так как $follow(S) \cap first(xSyS) = \{x\}$.

Пример 5. Рассмотрим язык Дика. Его грамматика выглядит так: $S \to a_1Sb_1S|a_2Sb_2S| \ldots |a_nSb_nS|\varepsilon$. Проверим грамматику, согласно критерию. Условие (1) выполняется, так как $first(a_1Sb_1S) = \{a_1\}$, $first(a_2Sb_2S) = \{a_2\}$, …, $first(a_nSb_nS) = \{a_n\}$, $first(\varepsilon) = \{\varepsilon\}$ – все множества попарно не пересекаются. Проверим условие (2): $follow(S) = \{b_1, b_2, \ldots, b_n, \$\}$ – множество не пересекается ни с одним из $first(a_iSb_iS)$. Таким образом, указанная грамматика является LL(1)-грамматикой. Это важный пример, так как язык – это последовательность правильно вложенных скобок n типов, что существенно с точки зрения практики, поскольку вложенные скобки присутствуют в записях математических функций и в языках программирования, как в явном виде, так и в виде зарезервированных слов.

IV. АЛГОРИТМЫ ПРИВЕДЕНИЯ К LL(1)-ГРАММАТИКЕ

Заметим, что множества $first$ и $follow$ отличаются даже для одного языка, если он задан разными правилами грамматики. В частности, грамматика $S \to aSbS|\varepsilon$ – частный случай языка Дика – является LL(1)-грамматикой и порождает тот же язык, что и грамматика, указанная в примере 1. В связи с этим возникает вопрос: если грамматика не является LL(1)-грамматикой, можно ли для нее найти эквивалентную грамматику [24], которая является LL(1)-грамматикой?

Теорема 2. Не существует алгоритма, определяющего для произвольной КС-грамматики, существует ли для нее эквивалентная LL(1)-грамматика.

Эта теорема доказывается в рамках теории алгоритмов.

Несмотря на то, что в общем виде эта проблема неразрешима, в некоторых случаях это можно выяснить. Сформулируем некоторые утверждения.

Утверждение 1. Если грамматика содержит левую рекурсию, то она не является LL(1)-грамматикой.

По определению левой рекурсии, такая грамматика содержит правила вида $A \to A\alpha|\beta$,

где $\beta[1] \neq A$ (наличие только $A \to A\alpha$ возможно, если A бесплодный символ, однако правила с бесплодными символами можно удалить, не меняя порождаемого языка). Но в этом случае $first(A\alpha) \cap first(\beta) \neq \emptyset$, что противоречит условию (1) критерия. ■

Приведем алгоритм устранения левой рекурсии.

1) Для фиксированного нетерминала запишем все правила в виде $A \to A\alpha_1|A\alpha_2| \dots |A\alpha_n|\beta_1|\beta_1| \dots |\beta_n$, где $\beta_i[1] \neq A$ и $\alpha_i \neq \varepsilon$.

2) Заменим все правила на следующие: $A \to \beta_1 B|\beta_2 B| \dots |\beta_n B|\beta_1|\beta_2| \dots |\beta_n$.

3) Добавим новые правила $B \to \alpha_1 B|\alpha_2 B| \dots |\alpha_n B|\alpha_1|\alpha_2| \dots |\alpha_n$.

Замечание 1. Покажем факт того, что алгоритм не меняет порождаемого языка. До алгоритма нетерминал A порождал цепочки $\beta_k\alpha_{i1}\alpha_{i2} \dots \alpha_{in}$. После применения алгоритма A порождает цепочку $\beta_k B$, а B порождает цепочку $\alpha_{i1}\alpha_{i2} \dots \alpha_{in}$, что совпадает с цепочкой до алгоритма.

Замечание 2. Выше описан алгоритм устранения непосредственной левой рекурсии, однако встречается более сложная косвенная рекурсия через промежуточные нетерминалы. Существует алгоритм и для ее устранения.

Заметим, что в примере 1 грамматика является леворекурсивной.

Утверждение 2. Если грамматика содержит правое ветвление, то она не является LL(1)-грамматикой.

Правое ветвление в грамматике означает, что в ней присутствуют правила вида $A \to \alpha\beta|\alpha\gamma$, где $\alpha \neq \varepsilon$, но тогда $first(\alpha\beta) \cap first(\alpha\gamma) \neq \emptyset$. ■

Приведем алгоритм устранения правого ветвления.

1) Зафиксируем нетерминал A и запишем его правила в виде $A \to \alpha\beta_1| \dots |\alpha\beta_n|\gamma_1| \dots |\gamma_k$, где $\alpha \neq \varepsilon$ наибольший общий префикс $\alpha\beta_i$ и γ_j не содержат этот префикс.

2) Заменим все правила на $A \to \alpha B|\gamma_1| \dots |\gamma_k$.

3) Введем дополнительные правила $B \to \beta_1| \dots |\beta_n$.

Очевидно, что данный алгоритм не меняет порождаемого языка, так как вводит промежуточный нетерминал, из которого выводятся различные подцепочки находящиеся после общего префикса.

Замечание 3. Отметим, что утверждения являются необходимыми, но не достаточными, то есть, если грамматика не содержит левую рекурсию и правого ветвления, она еще может оставаться не LL(1)-грамматикой, что иллюстрируют примеры 2, 4.

Пример 6. Составим упрощенную грамматику арифметических выражений: $S \to n|SBS|(S), B \to +|*$, где n - лексема в виде числа. Данная грамматика является неоднозначной, а значит, ее невозможно привести к LL(1)-виду. Однако можно придумать более удачную грамматику: $S \to A + S|A$, $A \to B * A|B$, $B \to n|(S)$. Эта грамматика содержит правое ветвление, от которой можно избавиться вышеприведенным алгоритмом. Получится следующая грамматика: $S \to AS'$, $S' \to +S|\varepsilon, A \to BA', A' \to *A|\varepsilon, B \to n|(S)$.

Полученная грамматика не содержит ни левой рекурсии, ни правого ветвления. Проверим ее по критерию LL(1)-грамматики.

1) $first(AS') = \{n, (\}$;

2) $first(+S) = \{+\}$, $first(\varepsilon) = \{\varepsilon\}$, $follow(S') = \{), \$\}$;

3) $first(BA') = \{n, (\}$;

4) $first(*A) = \{*\}$, $first(\varepsilon) = \{\varepsilon\}$, $follow(A') = \{), +, \$\}$;

5) $first(n) = \{n\}, first((S)) = \{(\}$.

Грамматика удовлетворяет теореме 1.

V. МЕТОДЫ АНАЛИЗА LL(1)-ГРАММАТИК

Рассмотрим методы анализа LL(1)-грамматик. LL(1)-анализ представляет собой нисходящий разбор: построение синтаксического дерева начинается сверху вниз – с аксиомы грамматики.

Различают два широко распространенных способа анализа: метод рекурсивного спуска и метод выбросов-переносов. Вне зависимости от выбранного метода по нему сначала строится таблица прогнозов рассматриваемой грамматики.

В первой строке таблицы прогнозов перечисляются терминалы грамматики, включая символ конца цепочки $, в первом столбце – нетерминалы. В ячейках пишутся правила грамматики, которые из данного нетерминала «приближают» нас к указанному терминалу. Алгоритм построения таблицы прогнозов следующий:

1) Для каждого правила $A \to \alpha$ (где α – непустая цепочка) и для каждого терминала $a \in first(\alpha)$ помещаем правило $A \to \alpha$ в ячейку [A,a];

2) Для каждого правила $A \to \alpha$ такого, что $a \in follow(A)$ и $\varepsilon \in first(\alpha)$ помещаем правило $A \to \alpha$ в ячейку [A,a].

3)

Пример 7. Построим таблицу прогнозов для грамматики из примера 6.

Таблица 1
Таблица прогнозов грамматики арифметических выражений

	+	*	n	(	)	$
S			$S \to AS'$	$S \to AS'$		
S'	$S' \to +S$				$S' \to \varepsilon$	$S' \to \varepsilon$
A			$A \to BA'$	$A \to BA'$		
A'	$A' \to \varepsilon$	$A' \to * A$			$A' \to \varepsilon$	$A' \to \varepsilon$
B			$B \to n$	$B \to (S)$		

В первую очередь рассмотрим метод рекурсивного спуска для синтаксического разбора LL(1)-грамматик. Опишем этот метод словесно, а затем на псевдокоде напишем его для примера 7.

С помощью отдельной функции *gc*() цепочка посимвольно считывается слева направо, начиная с первого символа, который считается текущим: *cur*. В дальнейшем будем иметь в виду, что *cur* можем быть как символом в классическом понимании, так и лексемой (ее тоже будем называть символом).

Для каждого нетерминала создается функция, носящая его имя, и задача которой найти подцепочку, которая выводится из этого нетерминала, из которой в дальнейшем будет выводиться текущий терминал cur.

Тело каждой функции пишется непосредственно по таблице прогнозов.

В условии оператора *if* (*elseif*) текущий символ cur сравнивается с каждым терминалом из таблицы прогнозов, ячейки которой непусты на пересечении имени текущей функции и проверяемого терминала. Если не найдено ни одного совпадения, то работа метода останавливается с ошибкой.

Тело условного оператора пишется по правым частям правил: если в правой части правила стоит нетерминал, то рекурсивно вызывается функция, соответствующая этому нетерминалу, если же терминал, то производится проверка соответствия текущему токену *cur* (когда он не на первом месте, ведь в противном случае проверка уже произведена). При совпадении вызывается функция *gc*(), считывающая следующий символ. При несовпадении работа метода рекурсивного спуска останавливается с ошибкой. Если в правой части ε, то ничего делать не надо. Когда правая часть правила закончилась, то автоматически осуществляется возврат в точку вызова.

Работа метода начинается с вызова функции *Scan*(), вызывающей *gc*() и *S*(), а также содержащей перехватчик ошибок (для мгновенной остановки программы). После реализации всех правил, эта функция проверяет факт того, что *gc*() остановилась в конце считываемой цепочки – если еще остались символы, то вызывается ошибка, в противном случае работа метода завершается успешно. Заметим, что функции могут рекурсивно вызывать друг друга, отсюда и название данного метода.

Пример 8. Псевдокод метода рекурсивного спуска по таблице прогнозов из примера 7 (по аналогии можно делать и для других грамматик):

```
var cur: char;
Scan();

function Scan(){
try{ gc(); S(); if(cur!=$){ERROR;}
write("успешно"); return; }
catch(ERROR){
write("ошибка в символе",cur);
return;
}

function gc(){ cur:=nextSymbol();}//считывается следующий символ
function S(){
if(cur=n|| cur='(' ){ A(); S'();} else { ERROR; }
}
function S'(){
if(cur='+'){ gc(); S(); } elseif( cur=')'|| cur='$' ) { } else { ERROR; }
}
function A(){
if(cur=n|| cur='(' ){ B(); A'();} else { ERROR; }
}
function A'(){
if(cur='*'){ gc(); A();} elseif( cur=')'|| cur='$'|| cur='+') { } else { ERROR; }
}
function B(){
if(cur=n){ gc();}
elseif(cur="("){gc(); S();  if(cur=")"){gc();} else { ERROR;} }
else { ERROR;}
}
```

Преимущество метода рекурсивного спуска состоит в том, что это самый простой и интуитивно понятный метод синтаксического анализа КС-языков. Между тем следует отметить, что метод рекурсивного спуска неявно использует стек, так как программе необходимо запоминать, из какого места была вызвана внутренняя функция, чтобы затем вернуться к точке вызова. Зачастую встроенный стек внутренних вызовов имеет скромные размеры, и при большом числе рекурсивных вызовов (в примере – большого числа скобок) существует опасность переполнения стека, а это существенный недостаток. Эта проблема решается с помощью нерекурсивного анализатора, явно использующего стек – это метод выбросов-переносов.

Рассмотрим работу метода выбросов переносов. Опишем его словесно, а затем реализуем программу на псевдокоде.

Стек анализатора содержит в себе все символы грамматики, а также маркер дна «⊥». В начале работы он содержит только аксиому S и маркер дна непосредственно под ним.

Анализатор считывает текущий терминал cur входной цепочки и смотрит на символ A на вершине стека, а затем принимает решение в зависимости от ситуации.

1) Если A = ⊥ и *cur* = $, то анализатор успешно заканчивает работу.

2) Если A = *cur*, то анализатор снимает со стека верхний символ, а в цепочке переходит к следующему терминалу (вызывается функция *nextSymbol*();). Таким образом, происходит **выброс** символа *cur* со стека.

3) Если A нетерминал, то анализатор должен просмотреть заранее записанную (например, в массив) таблицу прогнозов, а именно ячейку [A, *cur*]. Если ячейка пуста, то работа анализатора останавливается с ошибкой. Если ячейка непуста, то анализатор снимает нетерминал со стека и **переносит** правую часть правила в стек, записывая символы по одному в обратном порядке. Заметим, что если правая часть содержит пустую цепочку, то нетерминал просто снимается со стека. В массив следует записывать только правые части правил.

4) Во всех остальных случаях выводится сообщение об ошибке.

Реализация указанного метода на псевдокоде:

```
var s: stack;
var cur:char;
s.push(⊥);
s.push(S);
NextSymbol();
function parser(){
while (s.top!= ⊥){
A:=s.top();
if(A=⊥ && cur=$){write(“успешно”); break;}//Анализ завершен
elseif(A=cur){NextSymbol(); s.pop();} //Выброс
elseif(A=нетерминал){
if(M[A,cur]=X1X2…Xn){//M – массив, построенный на основе таблице прогнозов, X1X2…Xn – правая часть правила A -> X1X2…Xn
        s.pop();
        for i=n downto 1{ s.push(Xi)}//перенос в обратном порядке
} else {write(“ошибка в символе”,cur); ERROR;} //пустая ячейка
} else {write(“ошибка в символе”,cur); ERROR;} //прочие случаи
}
}
```

VI. ЗАКЛЮЧЕНИЕ

В заключение рассмотрим еще один важный аспект. Можно обратить внимание на то, что метод выбросов-переносов реализуется в общем виде, меняя для конкретной грамматики лишь массив, тогда как код метода рекурсивного спуска сильно зависит от грамматики. Однако при семантической интерпретации общий код придется подстраивать под конкретные моменты: необходимо выделять разные случаи в зависимости от того, какой терминал или нетерминал считывается.

Пример 9. Рассмотрим несколько состояний стека при разборе цепочки (*n*+*n*)***n* грамматики из примера 6.

0) Стек {⊥,S}

1) Указатель перемещается и считывает символ «(», это соответствует учейке [S,(]. Стек: {⊥,S',A} – нетерминал снимается, а правая часть переносится в обратном порядке.

2) Стек: {⊥,S',A',B}

3) {⊥,S',A',B}

4) {⊥,S',A',),S,(}

5) Так как символ на вершине стека совпадает со считываемым символом, то происходит выброс открывающей скобки, а указатель перемешается к числу n: {⊥,S',A',),S}

6) И так далее. В конце в стеке останется только маркер дна, а указатель прочитает конец цепочки. Тогда метод успешно завершится.

Подводя итог, можно отметить, что LL(1)-анализатор самый простой для построения интерпретаторов, но применим к ограниченному числу языков (однако для практики, как правило, этого хватает, если не брать в расчет естественные языки). Он реализуется двумя методами, которые обладают своими недостатками и достоинствами: с явным и неявным использованием стека. Таким образом, рассмотренные в статье аспекты являются необходимыми и достаточными условиями для реализации LL(1)-анализа этими методами.

БИБЛИОГРАФИЯ

[1] Баженов Р. И., Лопатин Д. К. О применении современных технологий в разработке интеллектуальных систем // Журнал научных публикаций аспирантов и докторантов. – 2014. – № 3 (93). – С. 263-264.

[2] Козлов С. В. Концептуальные возможности использования цифровых технологий в сфере образования // Цифровой регион: опыт, компетенции, проекты: сборник статей III Международной научно-практической конференции, посвященной 90-летию Брянского государственного инженерно-технологического университета. – Брянск, 2020. – С. 396-402.

[3] Мунерман В. И. Реализация параллельной обработки данных в облачных системах // Современные информационные технологии и ИТ-образование. – 2017. Т. 13. № 2. – С. 57-63.
[4] Козлов С. В., Светлаков А.В. Теория формальных грамматик и ее применение // Системы компьютерной математики и их приложения. – 2021. № 22. – С. 358-364.
[5] Муха В. С. Математические модели многомерных данных // Доклады Белорусского государственного университета информатики и радиоэлектроники. – 2014. – № 2 (80). – С. 143-158.
[6] Киселева О.М., Тимофеева Н.М. Построение концептуальной модели учебных словарей по педагогическим дисциплинам // Научно-методический электронный журнал «Концепт». – 2013. – Т. 3. – С. 3216-3220.
[7] Втюрин М. В. Применение формальных грамматик для сокращения объема текстовой информации // Инновационное развитие: технический и технологический аспекты. Сборник статей международной научно-практической конференции. – 2019. – С. 22-25.
[8] Кагиров И. А., Леонтьева А. Б. Автоматический синтаксический анализ русских текстов на основе грамматики составляющих // Известия высших учебных заведений. Приборостроение. – 2008. – Т. 51. № 11. – С. 47-51.
[9] Волкова И. А., Вылиток А. А., Руденко Т. В. Формальные грамматики и языки. Элементы теории трансляции: учебное пособие для студентов II курса. – М., 2009 – 115 с.
[10] Компиляторы. Принципы, технологии, инструментарий / А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман. – М., 2008. – 1184 с.
[11] Вирт Н. Построение компиляторов. – М., 2013. – 192 с.
[12] Борисенкова А. В., Козлов С. В. Использование метода каскадов Хаара при распознавании образов на изображениях // Развитие научно-технического творчества детей и молодежи: Сборник материалов III Всероссийской научно-практической конференции с международным участием. – 2019. – С. 28-33.
[13] Фаворская М. Н. К вопросу об использовании формальных грамматик при распознавании объектов в сложных сценах // Решетневские чтения. – 2009. – Т. 2. – С. 540-541.
[14] Янченко Е. В. Использование формальных грамматик в криптографии // Современные проблемы телекоммуникаций: материалы международной научно-технической конференции. – Новосибирск, 2021. – С. 155-158.
[15] Козлов С. В. Интеллектуальная система поддержки принятия решений «Advanced Tester» // Компьютерная интеграция производства и ИПИ-технологии: сборник материалов X Всероссийской конференции. – Оренбург, 2021. – С. 127-131.
[16] Малявко А.А. Формальные языки и компиляторы: учебное пособие для вузов. – М.: Издательство Юрайт, 2020. – 429 с.
[17] Пентус А. Е., Пентус М. Р. Теория формальных языков: учебное пособие. – М: Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004. — 80 с.
[18] Хопкрофт Дж. Э., Мотвани Р., Ульман Дж. Д.. Введение в теорию автоматов, языков и вычислений. – М., 2008. – 528 с.
[19] Скрипов А. В. Описание контекстных условий формальных языков грамматики с контекстуальными аргументами // Вестник Уральского института экономики, управления и права. – 2013. № 1 (22). – С. 111-116.
[20] Мартыненко Б. К. Регулярные языки и КС-грамматики // Компьютерные инструменты в образовании. – 2012. № 1. – С. 14-20.
[21] Козлов С.В., Светлаков А.В. Применение теории формальных грамматик в информатике. Дистанционные образовательные технологии. Сборник трудов VI Международной научно-практической конференции. – Симферополь, 2021. – С. 255-259.
[22] Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.
[23] Козлов С. В. Цифровое моделирование процессов управления социально-экономическими системами с применением методов функционального анализа // Вызовы цифровой экономики: итоги и новые тренды: сборник статей II Всероссийской научно-практической конференции (г. Брянск, 07 июня 2019 г.) [Электронный ресурс]. – Брянск: Брян. гос. инженерно-технол. ун-т., 2019. – С. 233-239.
[24] Байдарманова Б. Н. Некоторые способы нахождения эквивалентных преобразований в контексте свободных грамматик // Theoretical & Applied Science. – 2013. – № 5 (1). – С. 5-11.

About LL(1)-grammars, algorithms on them and methods of their analysis in programming

Kozlov S.V., Svetlakov A. V.

***Abstract* – The article discusses the theoretical foundations of parsing, namely, the definition of LL(1)-grammars and proves the key positions associated with them for practice. The authors demonstrate the impracticality of the proof by definition that context-free grammar is an LL(1)-grammar. In view of this, a theorem is formulated and proved that sets the criterion of LL(1)-grammar. First and follow sets are constructed, which determine the necessary and sufficient conditions for the existence of LL(1)-grammar. Examples show the application of the formulated criterion. The central point of the article is the question of finding grammars equivalent to LL(1)-grammar. A number of statements are considered to reveal that the grammar being analyzed is not an LL(1)-grammar. In particular, two necessary conditions for the existence of LL(1)-grammar are analyzed. Namely, theorems that if a grammar contains a left recursion or a right branching, then it is not an LL(1)-grammar. The example shows that these conditions are not sufficient. Two methods for analyzing LL(1)-grammars used in practice are discussed and compared: the recursive descent method and the emission-transfer method. For each of the methods, its meaningful description and implementation in pseudocode is given. All the provisions of the article are accompanied by the necessary examples. The relevance of the article is associated with the search and study of parsing algorithms for grammars of natural and artificial languages, which are successfully used as tools for writing pattern recognition systems in the field of artificial intelligence.**

***Keywords* – parsing, LL(1)-grammar, left recursion, right branching, first and follow sets, prediction table, recursive descent method, emission-carry method.**

REFERENCES

[1] Bazhenov R. I., Lopatin D. K. O primenenii sovremennyh tehnologij v razrabotke intellektual'nyh sistem // Zhurnal nauchnyh publikacij aspirantov i doktorantov. – 2014. – # 3 (93). – S. 263-264.
[2] Kozlov S. V. Konceptual'nye vozmozhnosti ispol'zovanija cifrovyh tehnologij v sfere obrazovanija // Cifrovoj region: opyt, kompetencii, proekty: sbornik statej III Mezhdunarodnoj nauchno-prakticheskoj konferencii, posvjashhennoj 90-letiju Brjanskogo gosudarstvennogo inzhenerno-tehnologicheskogo universiteta. – Brjansk, 2020. – S. 396-402.
[3] Munerman V. I. Realizacija parallel'noj obrabotki dannyh v oblachnyh sistemah // Sovremennye informacionnye tehnologii i IT-obrazovanie. – 2017. T. 13. # 2. – S. 57-63.
[4] Kozlov S. V., Svetlakov A.V. Teorija formal'nyh grammatik i ee primenenie // Sistemy komp'juternoj matematiki i ih prilozhenija. – 2021. # 22. – S. 358-364.
[5] Muha V. S. Matematicheskie modeli mnogomernyh dannyh // Doklady Belorusskogo gosudarstvennogo universiteta informatiki i radiojelektroniki. – 2014. – # 2 (80). – S. 143-158.
[6] Kiseleva O.M., Timofeeva N.M. Postroenie konceptual'noj modeli uchebnyh slovarej po pedagogicheskim disciplinam // Nauchno-metodicheskij jelektronnyj zhurnal «Koncept». – 2013. – T. 3. – S. 3216-3220.
[7] Vtjurin M. V. Primenenie formal'nyh grammatik dlja sokrashhenija ob"ema tekstovoj informacii // Innovacionnoe razvitie: tehnicheskij i tehnologicheskij aspekty. Sbornik statej mezhdunarodnoj nauchno-prakticheskoj konferencii. – 2019. – S. 22-25.
[8] Kagirov I. A., Leont'eva A. B. Avtomaticheskij sintaksicheskij analiz russkih tekstov na osnove grammatiki sostavljajushhih // Izvestija vysshih uchebnyh zavedenij. Priborostroenie. – 2008. – T. 51. # 11. – S. 47-51.
[9] Volkova I. A., Vylitok A. A., Rudenko T. V. Formal'nye grammatiki i jazyki. Jelementy teorii transljacii: uchebnoe posobie dlja studentov II kursa. – M., 2009 – 115 s.
[10] Kompiljatory. Principy, tehnologii, instrumentarij / A. V. Aho, M. S. Lam, R. Seti, D. D. Ul'man. – M., 2008. – 1184 s.
[11] Virt N. Postroenie kompiljatorov. – M., 2013. – 192 c.
[12] Borisenkova A. V., Kozlov S. V. Ispol'zovanie metoda kaskadov Haara pri raspoznavanii obrazov na izobrazhenijah // Razvitie nauchno-tehnicheskogo tvorchestva detej i molodezhi: Sbornik materialov III Vserossijskoj nauchno-prakticheskoj konferencii s mezhdunarodnym uchastiem. – 2019. – S. 28-33.

[13] Favorskaja M. N. K voprosu ob ispol'zovanii formal'nyh grammatik pri raspoznavanii ob"ektov v slozhnyh scenah // Reshetnevskie chtenija. – 2009. – T. 2. – S. 540-541.
[14] Janchenko E. V. Ispol'zovanie formal'nyh grammatik v kriptografii // Sovremennye problemy telekommunikacij: materialy mezhdunarodnoj nauchno-tehnicheskoj konferencii. – Novosibirsk, 2021. – S. 155-158.
[15] Kozlov S. V. Intellektual'naja sistema podderzhki prinjatija reshenij «Advanced Tester» // Komp'juternaja integracija proizvodstva i IPI-tehnologii: sbornik materialov X Vserossijskoj konferencii. – Orenburg, 2021. – S. 127-131.
[16] Maljavko A.A. Formal'nye jazyki i kompiljatory: uchebnoe posobie dlja vuzov. – M.: Izdatel'stvo Jurajt, 2020. – 429 s.
[17] Pentus A. E., Pentus M. R. Teorija formal'nyh jazykov: uchebnoe posobie. – M: Izd-vo CPI pri mehaniko-matematicheskom f-te MGU, 2004. — 80 s.
[18] Hopkroft Dzh. Je., Motvani R., Ul'man Dzh. D.. Vvedenie v teoriju avtomatov, jazykov i vychislenij. – M., 2008. – 528 s.
[19] Skripov A. V. Opisanie kontekstnyh uslovij formal'nyh jazykov grammatiki s kontekstual'nymi argumentami // Vestnik Ural'skogo instituta jekonomiki, upravlenija i prava. – 2013. # 1 (22). – S. 111-116.
[20] Martynenko B. K. Reguljarnye jazyki i KS-grammatiki // Komp'juternye instrumenty v obrazovanii. – 2012. # 1. – S. 14-20.
[21] Kozlov S.V., Svetlakov A.V. Primenenie teorii formal'nyh grammatik v informatike. Distancionnye obrazovatel'nye tehnologii. Sbornik trudov VI Mezhdunarodnoj nauchno-prakticheskoj konferencii. – Simferopol', 2021. – S. 255-259.
[22] Kristofides N. Teorija grafov. Algoritmicheskij podhod. – M.: Mir, 1978. – 432 s.
[23] Kozlov S. V. Cifrovoe modelirovanie processov upravlenija social'no-jekonomicheskimi sistemami s primeneniem metodov funkcional'nogo analiza // Vyzovy cifrovoj jekonomiki: itogi i novye trendy: sbornik statej II Vserossijskoj nauchno-prakticheskoj konferencii (g. Brjansk, 07 ijunja 2019 g.) [Jelektronnyj resurs]. – Brjansk: Brjan. gos. inzhenerno-tehnol. un-t., 2019. – S. 233-239.
[24] Bajdarmanova B. N. Nekotorye sposoby nahozhdenija jekvivalentnyh preobrazovanij v kontekste svobodnyh grammatik // Theoretical & Applied Science. – 2013. – # 5 (1). – S. 5-11.