Лекция 10. Синтаксический анализ. Часть 2

Главные вопросы, которые мы обсуждаем, представлены на СЛАЙДЕ 1. При рассмотрении переборного алгоритма НСА с возвратами мы давали рекомендации по его ускорению путем предпросмотра k входных символов. С их помощью можно определить, надо ли использовать заданную альтернативу. Выделим подкласс КСГ, который позволяет реализовать эту рекомендацию с $k = 1$.

10.1 s-грамматики

Рассмотрим процесс НСА строк языка на следующем примере.

Пример 65.

Пусть задана КСГ, показанная на СЛАЙДЕ 2. Возьмем строку *accedd*, которая принадлежит языку, определяемому заданной грамматикой. Начинаем с аксиомы и пытаемся найти левые порождения рассматриваемой строки, попутно сравнивая начальные терминалы СФ с символами строки.

На первом шаге можно использовать первую или вторую продукции. Учитывая, что наша строка начинается с символа *a*, мы используем первую продукцию и получаем СФ *aA*.

Для замены нетерминала *A* на втором шаге порождения можно использовать третью или четвертую продукции. Для выбора мы анализируем второй символ входной строки (первый – использован на предыдущем шаге). Поскольку рассматривается символ *c*, то для порождения выбирается третья продукция, которая также начинается с символа *c*.

На СЛАЙДЕ 3 показано дерево разбора нашей строки.

Особенностью этой КСГ является то, что в каждой продукции RHS начинается с терминального символа, причем альтернативы начинаются с различных символов. Это позволяет достаточно легко построить **детерминированный синтаксический анализатор**.

Данный анализатор можно создать на основе МПА. Мы будем считать **конфигурацией** анализатора тройку (ax, $\alpha\perp$, π), где ax – это необработанная часть входной строки, a – текущий входной символ, $\alpha\perp$ – содержимое магазина, π – разбор, под которым мы понимаем цепочку номеров применяемых на шаге порождения продукций. СЛАЙД 4.

Анализатор начинает работу в конфигурации (*accedd*, $S\perp$, ε). Показанное содержимое магазина утверждает, что входная строка должна порождаться из аксиомы грамматики.

Далее выполняется шаг порождения из символа S строки, начинающейся с a. Для этого анализатор может использовать первую либо вторую продукцию. Учитывая, что вторая продукция порождает строку, начинающуюся с b, анализатор выбирает первую продукцию и переходит в конфигурацию (*accedd*, $aA\perp$, 1).

В вершине магазина находится терминальный символ a. Поскольку этот символ **совпадает** с первым символом строки, его можно вытолкнуть из магазина и продолжить анализ входной строки. Конфигурация – (*ccedd*, $A\perp$, 1).

Теперь текущим символом является символ c, а верхним элементом магазина является нетерминал A. Для продолжения анализатор может выбрать третью или

четвертую продукции. Текущий символ требует, чтобы строка, порождаемая нетерминалом *A*, начиналась с символа *c*, поэтому выбирается третья продукция. Анализатор переходит в конфигурацию (*ccedd*, *cAd*⊥, 1 3).

Продолжая аналогичные действия, получим последовательность шагов, показанную в следующей таблице. Входная строка делится на две части: слева обработанная часть, справа – необработанная, текущий символ выделен (СЛАЙД 5).

	Входная строка		**Магазин**	**Действия анализатора**
1		***a****ccedd*	*S*⊥	Применить первую продукцию
2		***a****ccedd*	*aA*⊥	Вытолкнуть
3	*a*	***c****cedd*	*A*⊥	Применить третью продукцию
4	*a*	***c****cedd*	*cAd*⊥	Вытолкнуть
5	*ac*	***c****edd*	*Ad*⊥	Применить третью продукцию
6	*ac*	***c****edd*	*cAdd*⊥	Вытолкнуть
7	*acc*	***e****dd*	*Add*⊥	Применить четвертую продукцию
8	*acc*	***e****dd*	*edd*⊥	Вытолкнуть
9	*acce*	***d****d*	*dd*⊥	Вытолкнуть
10	*acced*	***d***	*d*⊥	Вытолкнуть
11	*accedd*	*ε*	⊥	Допустить

Рассмотренная грамматика относится к классу простых LL(1)-грамматик, или *s*-грамматик.

КСГ $G = (V_N, V_T, P, S)$ без *ε*-продукций называется **простой** LL(1)-грамматикой (**разделенной** грамматикой, ***s*-грамматикой**), если для каждого нетерминала *A* все альтернативные RHS начинаются с разных терминальных символов. СЛАЙД 6.

Для *s*-грамматик можно сформулировать очевидные правила построения ДМПА, выполняющего НСА. Из нашего примера видно, что для МПА достаточно определить две операции: 1) замена верхнего элемента магазина на строку, которая является RHS некоторой продукции; 2) выталкивание элемента из магазина.

Существенный недостаток *s*-грамматик заключается в том, что большинство конструкций ЯП не могут быть описаны с их помощью. Это иллюстрируется следующим примером.

Пример 66.

Пусть задана КСГ, показанная на СЛАЙДЕ 7. Четвертая продукция позволяет порождать пустую строку. Значит, КСГ не является *s*-грамматикой. Тем не менее, попробуем использовать ее для порождения строки *acbb*. Начальная конфигурация (*acbb*, *S*⊥, *ε*). СЛАЙД 8.

Шаг 1. Анализатор выбирает первую продукцию для порождения из аксиомы строки, начинающейся с *a*, и переходит в конфигурацию (*acbb*, *aAS*⊥, 1).

Шаг 2. Анализатор выталкивает символ из магазина и читает следующий входной символ, переходя в конфигурацию (*cbb*, *AS*⊥, 1).

Шаг 3. Для порождения строки, начинающейся с *c*, из символа на вершине магазина анализатор использует третью продукцию, а затем переходит в конфигурацию (*cbb*, *cASS*⊥, 1 3).

Шаг 4. Аналогично шагу 2, анализатор переходит в конфигурацию (*bb*, *ASS*⊥, 1 3).

До этого момента анализатор работал так же, как в примере 65.

Шаг 5. Продолжая анализ, необходимо выбрать продукцию, позволяющую из *A* породить строку, начинающуюся с *b*. Применить третью продукцию нельзя, т.к. она начинается с символа *c*. Делаем предположение, что строка порождается из *A* при помощи четвертой продукции, и она пустая. В этом случае символ *b* – это начальный символ строки, выводимой из символа, который в магазине находится под *A*. В нашем случае – это символ *S*.

Шаг 6. Применение четвертой продукции приводит к тому, что анализатор выталкивает верхний символ из магазина, но не меняет текущий входной символ. Из конфигурации (bb, $ASS\perp$, 1 3) он попадает в конфигурацию (bb, $SS\perp$, 1 3 4).

Остальные действия анализатора похожи на предыдущие:

(bb, $SS\perp$, 1 3 4) |— (bb, $bS\perp$, 1 3 4 2)

|— (b, $S\perp$, 1 3 4 2) |— (b, $b\perp$, 1 3 4 2 2) |— (ε, $\perp$, 1 3 4 2 2).

Анализ действий анализатора показывает, что успешное применение ε-продукций для некоторого нетерминала возможно в том случае, если строка, порождаемая символом, лежащим ниже этого нетерминала, начинается с символа, совпадающего с текущим входным символом. Это было показано, например, на шаге 6.

Перейдем к процессу анализа КСГ, определив два множества символов.

Множество **первых порождаемых символов** *FIRST* – это множество терминальных символов, которыми начинаются строки, выводимые из нетерминала A:

$FIRST(A) = \{a$ из $V_T \mid A =>^+ \alpha\beta$, где β принадлежит $(V_N \cup V_T)^*\}$.

Множество **символов-последователей** *FOLLOW* – это множество терминальных символов, которые могут встретиться непосредственно справа от нетерминала A в некоторой СФ:

$FOLLOW(A) = \{a$ из $V_T \mid S =>^* \alpha A\gamma$, и a принадлежит $FIRST(\gamma)\}$.

Пример 67.

Построим эти множества для КСГ из примера 66. $FIRST(S) = \{a, b\}$. $FOLLOW(S) = \{\perp\}$, т.к. S – аксиома. $FIRST(A) = \{c\}$. $FOLLOW(A) = FIRST(S) = \{a, b\}$, т.к. непосредственно справа от A в любой СФ может находиться символ S, множество символов-последователей, которого нами уже получено. СЛАЙД 9.

При моделировании работы анализатора можно установить, что при анализе применяется некоторая продукция, если анализатор находится в конфигурации (ax, $A\alpha\perp$, π) и выполняется одно из условий (СЛАЙД 10):

- продукция имеет вид $A \to a\beta$;
- продукция имеет вид $A \to \varepsilon$ и символ a принадлежит $FOLLOW(A)$.

Для рассмотрения этих условий удобно ввести множество выбора продукций (множество-**селектор**) *SELECT*, которое определяется для продукции следующим образом:

- если продукция имеет вид $A \to a\beta$, то $SELECT(A \to a\alpha) = FIRST(a\alpha) = \{a\}$;
- если продукция имеет вид $A \to \varepsilon$, то $SELECT(A \to a\alpha) = FOLLOW(A)$.

Можно определить более широкий, нежели *s*-грамматики, класс КСГ.

КСГ $G = (V_N, V_T, P, S)$ называется ***q*-грамматикой**, если RHS начинается с терминала или пуста, и множества-селекторы для каждого нетерминала A **не пересекаются**.

Построение МПА по *q*-грамматике может быть выполнено так же, как и по *s*-грамматике. СЛАЙД 10.

10.2 Определение LL(1)-грамматики

Сначала мы дадим более общее определение множества первых порождаемых символов так, чтобы его можно было применить к продукциям любого вида.

$FIRST(\alpha) = \{a$ из $V_T \mid S =>^* \alpha =>^* a\beta$, где α принадлежит $(V_N \cup V_T)^+$, β принадлежит $(V_N \cup V_T)^*\}$.

Данное определение означает, что множество первых порождаемых символов состоит из множества терминальных символов, которыми начинаются строки, выводимые

из строки α.

КСГ $G = (V_N, V_T, P, S)$ называется ***LL*(1)-грамматикой**, если из существования двух левых порождений: 1) $S =>^* wA\alpha =>^* w\beta\alpha =>^* wx$ и 2) $S =>^* wA\alpha =>^* w\gamma\alpha =>^* wy$, для которых из $FIRST(x) = FIRST(y)$, следует, что $\beta = \gamma$. СЛАЙД 12.

В данном определении утверждается, что выбор продукции для замены нетерминала A в строке $wA\alpha$ однозначно определяется цепочкой w и следующим за ней входным символом. В действительности это не так.

Примем без доказательства утверждение о том, что КСГ $G = (V_N, V_T, P, S)$ является $LL(1)$-грамматикой тогда и только тогда, когда для двух различных продукций $A \to \alpha$ и $A \to \beta$ из P пересечение $FIRST(\beta\alpha) \cap FIRST(\gamma\alpha) = \varnothing$ при всех таких $wA\alpha$, что $S =>^* wA\alpha$.

Из этого утверждения следует, что для LL(1)-грамматики продукция для замены нетерминала однозначно определяется текущим входным символом. СЛАЙД 13.

На СЛАЙДЕ 14 показано дерево разбора строки wxy. Если в некоторый момент времени построено дерево с кроной $wA\alpha$, то для определения продукции, которая используется для замены нетерминала A, достаточно рассмотреть текущий входной символ – первый символ строки – x.

Также примем без доказательства утверждение, что КСГ G является LL(1)-грамматикой тогда и только тогда, когда для двух различных продукций $A \to \beta$ и A $\to \gamma$ из P справедливо $FIRST(\beta\ FOLLOW(A)) \cap FIRST(\gamma\ FOLLOW(A)) = \varnothing$ для всех нетерминалов A.

На основании последних утверждений можно дать конструктивное определение LL(1)-грамматики.

КСГ $G = (V_N, V_T, P, S)$ называется ***LL*(1)-грамматикой** тогда и только тогда, когда для каждой A-продукции грамматики ($A \to \alpha_1 |\ldots| \alpha_n$, $n > 0$) выполняются следующие условия:

1. Множества $FIRST(\alpha_1), \ldots, FIRST(\alpha_n)$ попарно не пересекаются.
2. Если $\alpha_i =>^* \varepsilon$, то $FIRST(\alpha_j) \cap FOLLOW(A) = \varnothing$ для $1 \le j \le n$, $i \ne j$.

Важным следствием этого определения является то, что леворекурсивная грамматика не может быть LL(1)-грамматикой. СЛАЙД 15.

10.3 Алгоритм синтаксического анализа для LL(1)-грамматик

Разбор для LL(1)-грамматики можно осуществить с помощью так называемого 1-предиктивного (т.е. безвозвратного) алгоритма, использующего ТСА (СЛАЙД 16).

ТСА $M[A, a]$ строится из множеств первых порождаемых символов и символов-последователей и представляет собой двумерный массив, где A – нетерминал, а a – терминал или символ \$ (маркер конца входного потока). Алгоритм конструирования ТСА основан на следующей идее: если очередной входной символ a находится во множестве $FIRST(a)$, выбирается продукция $A \to a$. Единственная сложность возникает при $a = \varepsilon$ или, в общем случае, когда $a =>^* \varepsilon$. В этом случае мы снова должны выбрать $A \to a$, если текущий входной символ имеется в $FOLLOW(A)$ или если из входного потока получен \$, который при этом входит в $FOLLOW(A)$. Формально, алгоритм выполняется так (СЛАЙД 17).

ВХОД: грамматика G и множества первых порождаемых символов и символов-последователей.

ВЫХОД: ТСА $M[A, a]$.

Для каждой A-продукции ($A \to \alpha$) в G выполняем следующие действия.

1. Для каждого терминала *a* из *FIRST*(*α*) добавляем *A* -> *α* в ячейку *M*[*A*, *a*].

2. Если *ε* принадлежит *FIRST*(*α*), то для каждого терминала *b* из *FOLLOW*(*A*) добавляем *A* -> *α* в *M*[*A*, *b*]. Если *ε* принадлежит *FIRST*(*α*) и $ принадлежит *FOLLOW*(*A*), то добавляем *A* -> *α* также и в *M*[*A*, $].

Если после выполнения этих действий ячейка *M*[*A*, *a*] осталась без продукции, устанавливаем ее значение равным *error*.

Пример 68.

Для КСГ, показанной на СЛАЙДЕ 18, описанный выше алгоритм дает приведунную там же ТСА. Пустые места в ТСА означают записи ошибок; непустые указывают RHS продукций, при помощи которых выполняется «разворачивание» нетерминала.

Рассмотрим продукцию *E* -> *TA*. Поскольку *FIRST* (*TA*) = FIRST(*T*) = {(, *i*}, то эта продукция добавляется к *M*[*E*, (] и *M*[*E*, *i*]. Продукция *A* -> *+TA* добавляется к *M*[*A*, +], т.к. *FIRST*(*+TA*) = {+}. Поскольку *FOLLOW*(*A*)={),$}, то продукция *A* -> *ε* добавляется к *M*[*A*,)] и *M*[*A*, $].

Данный алгоритм можно применить для получения ТСА *M* любой КСГ *G*. Для любой LL(1)-грамматики каждая запись ТСА единственным образом определяет продукцию или сообщает об ошибке. Однако для некоторых грамматик таблица *M* может иметь записи с несколькими продукциями. Такое будет происходить, если *G* – леворекурсивная или неоднозначная грамматика. Хотя такие виды преобразований грамматик как устранение левой рекурсии и левая факторизация – легко выполняемые задачи, существуют такие грамматики, никакие изменения которых не приведут к LL(1)-грамматике.

Язык в следующем примере не является LL(1)-грамматикой.

Пример 69.

КСГ, показанная на СЛАЙДЕ 19, иллюстрирует проблему «кочующего» *else*. приведенный выше алгоритм дает приведенную там же ТСА. Пустые места в ТСА означают записи ошибок; непустые указывают RHS продукций, при помощи которых выполняется «разворачивание» нетерминала. Видно, что запись *M*[*O*, *e*] относится к двум продукциям *O* -> *eS* и *O* -> *ε*.

Грамматика неоднозначна, и эта неоднозначность проявляется в выборе используемой продукции при встрече во входном потоке символ *e* (наше обозначение для *else*). Ее можно разрешить путем выбора продукции *O* -> *eS*. Данный выбор соответствует связыванию *else* с ближайшим предыдущим *then*. Заметим, что выбор *O* -> *ε* приводит к тому, что *e* не помещается в стек и не удаляется из входного потока, что, очевидно, неверно.

LL(1)-грамматики не дают ТСА с множественными записями. Первое *L* означает просмотр входной строки **слева** направо. Второе *L* означает использование **левых** порождений. Единица означает использование **одного** входного символа на каждом шаге порождения для принятия решений.

Предиктивный синтаксический анализ, управляемый ТСА

Нерекурсивный предиктивный синтаксический анализатор можно построить с помощью явного использования стека. Синтаксический анализатор имитирует левое порождение. Если *w* – входная строка, соответствие которой проверено до текущего момента, то в стеке хранится последовательность символов *α*, такая, что имеется левое порождение $S \Rightarrow^* w\alpha$.

Синтаксический анализатор на СЛАЙДЕ 20, управляемый ТСА, имеет входной буфер, стек, содержащий последовательность символов, таблицу синтаксического анализа, построенную при помощи описанного выше алгоритма, и выходной поток.

Входной буфер содержит анализируемую строку, за которой следует маркер конца строки $. Мы также используем символ $ для указания дна стека, который изначально содержит над символом $ стартовый символ грамматики.

Синтаксический анализатор управляется программой, которая рассматривает символ на вершине стека X и текущий входной символ a. Если X является нетерминалом, синтаксический анализатор выбирает X-продукцию в соответствии с записью $M[X, a]$ ТСА M (здесь обычно выполняется дополнительный код, например, для построения узла дерева разбора). В противном случае проверяется соответствие между терминалом X и текущим входным символом a.

Как и ранее, поведение синтаксического анализатора может быть описано в терминах его **конфигураций**, которые дают содержимое стека и оставшийся входной поток. Приведенный далее алгоритм описывает работу с конфигурациями.

ВХОД: строка w и таблица M для грамматики G.

ВЫХОД: если w принадлежит $L(G)$ – левое порождение w; в противном случае – сообщение об ошибке.

Изначально синтаксический анализатор находится в конфигурации с w\$ во входном буфере и аксиомой S грамматики G на вершине стека, над $\$$. СЛАЙД 21.

Псевдокод программы приведен на СЛАЙДЕ 22 использует таблицу предиктивного синтаксического анализа M для анализа входной строки.

Пример 70.

В системе JFLAP рассматривается пошаговое выполнение анализа строки $i+i*i$.

10.4 Метод рекурсивного спуска

Вместо явного использования стека при НСА мы можем положиться на реализации ЯП. В большинстве из них имеется возможность рекурсивного вызова процедуры (или функции), т.е. стек используется неявно. Каждому нетерминалу грамматики в соответствие ставится процедура, которая распознает строку, порождаемую этим нетерминалом. Работа начинается с вызова процедуры для аксиомы и успешно заканчивается, если полностью просмотрена вся входная строка. Псевдокод для распознавания типичного нетерминала показан на СЛАЙДЕ 23.

Для распознавания терминальных символов так же должна быть предусмотрена соответствующая процедура на ЯП. К этой процедуре обращаются за очередным символом входной строки (листом дерева разбора).

Такой синтаксический анализатор работает **методом рекурсивного спуска**. Он применим к любой LL(1)-грамматике.

Пример такой грамматики, описывающей очень простой язык программирования приведен далее (СЛАЙД 24).

```
PROGRAM -> begin DECLIST comma STMTLIST end
DECLIST -> d X
X ->semi DECLIST | ε
STMTLIST -> s Y
Y -> semi STMTLIST | ε
```

Заготовка кода для синтаксического анализа данного языка приведена на СЛАЙДАХ 25-30.

Продемонстрированы все три возможные ситуации. Если текущим символом RHS грамматики является нетерминал, то ему соответствует вызов процедуры распознавания цепочки, порождаемой этим нетерминалом.

Если текущим символом RHS является терминала, то ему соответствует условный

оператор, осуществляющий его проверку на равенство с текущим входным символом. Если они совпадают, то осуществляется обращение за следующим символом путем вызова процедуры *get_token*(). Во всех остальных случаях выполняется вызов процедуры *error*().

Пустой RHS соответствует переход на конец функции.

10.5 LL(*k*)-грамматики с *k* > 1

LL(*k*)-грамматики при $k > 1$ на практике используются редко.

Для построения распознавателей *LL*(*k*)-грамматик используются два важных множества, определяемые следующим образом (СЛАЙДЫ 31-32):

FIRST(*k*, α) – множество терминальных цепочек, выводимых из $\alpha \in (V_T \cup V_N)^*$, укороченных до *k* символов;

FOLLOW(*k*, *A*) – множество укороченных до *k* символов терминальных цепочек, которые могут следовать непосредственно за $A \in V_N$ в цепочках вывода

Формально эти два множества могут быть определены следующим образом:

$FIRST(k, \alpha) = \{\omega \in V_T^* \mid$ либо $|\omega| \le k$ и $\alpha \Rightarrow^* \omega$, либо $|\omega| > k$ и $\alpha \Rightarrow^* \omega x$, $x \in (V_T \cup V_N)^*\}$, $\alpha \in (V_T \cup V_N)^*$, $k > 0$.

$FOLLOW(k,A) = \{\omega \in V_T^* \mid S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \gamma)$, $\alpha \in V_T^*\}$, $A \in V_N$, $k > 0$.

Очевидно, что если имеется цепочка терминальных символов $\alpha \in V_T^*$, то $FIRST(k,\alpha)$ – это первые *k* символов цепочки α.

Доказано, что КСГ *G* является *LL*(*k*)-грамматикой тогда и только тогда, когда выполняется следующее условие:

$\forall A \to \beta \in$ P и $\forall A \to \gamma \in P$ $(\beta \neq \gamma)$: $FIRST(k,\beta\omega)) \cap FIRST(k,\gamma\omega)) = \varnothing$ для всех цепочек ω таких, что $S \Rightarrow^* \alpha A \omega$.

Иначе говоря, если существуют две цепочки вывода:

$S \Rightarrow^* \alpha A \gamma \Rightarrow \alpha z \gamma \Rightarrow^* \alpha\omega$

$S \Rightarrow^* \alpha A \gamma \Rightarrow \alpha t \gamma \Rightarrow^* \alpha\upsilon$

то из условия $FIRST(k, \omega) = FIRST(k, \upsilon)$ следует, что $z = t$.

За подробностями желающие могут обратиться к дополнительной литературе.

Иногда при СА требуется переменное число символов предпросмотра. Такие грамматики и анализаторы относятся к классу *LL*(*). Подробнее, см. например, -http://www.antlr.org/papers/LL-star-PLDI11.pdf. СЛАЙД 32.

Литература к лекции 10

1. LL(1) - http://ru.wikipedia.org/wiki/LL(1)
2. Ахо, А. Компиляторы: принципы, технологии и инструментарий, 2 издание / А. Ахо, М.Лам, Р. Сети, Дж. Ульман. – М.: Издательский дом «Вильямс», 2008. – 1184 с.
3. Метод рекурсивного спуска - http://ru.wikipedia.org/wiki/Метод_рекурсивного_спуска
4. LL-анализатор - http://ru.wikipedia.org/wiki/LL
5. The Compiler Generator Coco/R - http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/