

Задание 10

LL-анализ

Ключевые слова ¹: язык, контекстно-свободный язык, магазинный автомат, грамматика, LL(k)-грамматика, LL(1)-анализатор, функции FIRST, FOLLOW.

1 Нисходящий и восходящий разбор

Напомним определение вывода КС-грамматики.

Выводом цепочки α называется такая последовательность применений правил с указанием раскрываемого нетерминала, что применяя правила из неё начиная с аксиомы получается цепочка α . Если цепочка α не содержит нетерминалов, то α принадлежит языку, порождаемому КС-грамматикой. Нам будет удобно пользоваться такими понятиями как левый вывод (правый вывод). *Левым выводом* называют такой вывод, что на каждом его шаге раскрывается самый левый нетерминал в промежуточной цепочке. Правый вывод определяется аналогично.

Также напомним что мы называем деревом вывода или деревом разбора. С формальным определением дерева разбора вы можете познакомиться, например, в книге Хопкрофта, Мотвани и Ульмана, а мы воспользуемся неформальным описанием этого понятия. *Деревом разбора* для грамматики G называется упорядоченное дерево, в корне которого находится аксиома S , каждая вершина помечена нетерминалом, терминалом или пустым словом, если вершина помечена терминалом или ε , то эта вершина является листом, если же вершина помечена нетерминалом A , то существует такое правило $A \rightarrow X_1 X_2 \dots X_n \in P$ ($X_i \in N \cup T$), что вершины-потомки A помечены символами $X_1, X_2 \dots X_n$ слева направо.

Будем говорить, что для КС-грамматики G слово w разобрано, если известно хотя бы одно из её деревьев вывод.

Левым разбором цепочки $\alpha \in (N \cup T)^*$ будем называть последовательность правил, применённых при левом выводе цепочки α . *Правым разбором* цепочки α назовём обратную последовательность правил, применённых при правом выводе цепочки α .

¹минимальный необходимый объем понятий и навыков по этому разделу)

Пример 1. Грамматики $G = (N, T, P, E)$, и $G_\pi = (N, T', P', E)$, $T = \{a, +, *\}$ заданы правилами:

$$E \rightarrow E + T \quad (1)$$

$$E \rightarrow T \quad (2)$$

$$T \rightarrow T * F \quad (3)$$

$$T \rightarrow F \quad (4)$$

$$F \rightarrow (E) \quad (5)$$

$$F \rightarrow a \quad (6)$$

$$E \rightarrow 1ET$$

$$E \rightarrow 2T$$

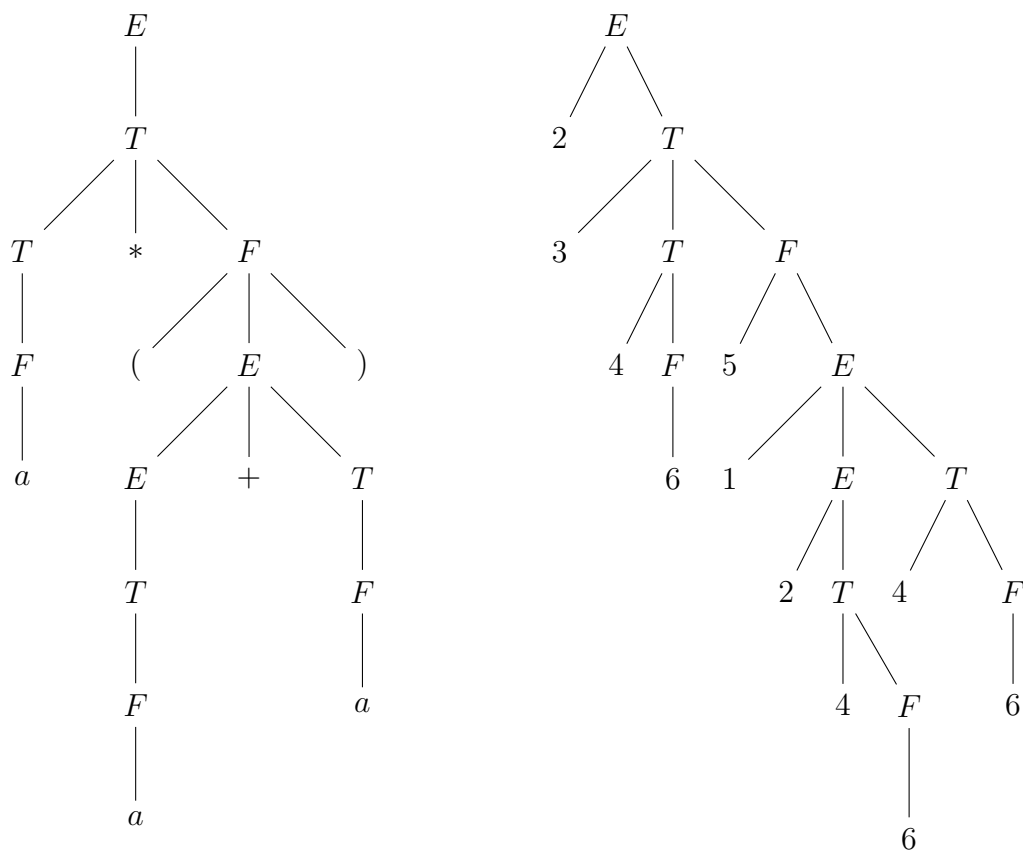
$$T \rightarrow 3TF$$

$$T \rightarrow 4F$$

$$F \rightarrow 5E$$

$$F \rightarrow 6$$

Построим дерево разбора для слова $w = a + (a * a)$ и дерево вывода в грамматики G' , соответствующее выводу w :



Если грамматика G выводит слово w , то применяя соответствующие правила в G' выводим из неё слово $\pi_l(w)$, соответствующее левому выводу слова w .

Назовём *переводом* бинарное отношение T , действующее из языка L_1 в язык L_2 . Если пара слов (u, v) удовлетворяет отношению T , то будем говорить, что слово u транслируется переводом T в слово v , а слово v будем называть *выходом* для u . Мы будем рассматривать синтаксически управляемые переводы. Неформально, перевод является синтаксически управляемым, если существует пара грамматик, правила которых занумерованы и если на шаге вывода из одной грамматики получена цепочка α , а для соответствующего шага вывода из другой грамматики получена цепочка β , то любой нетерминал входящий в цепочку α , входит и в цепочку β с одинаковой кратностью. Например, в лингвистике нетерминалы могут соответствовать частям речи. Тогда при переводе с одного языка

на другой подлежащее перейдёт в подлежащее, а сказуемое в сказуемое, таким образом, синтаксис определяет некоторые особенности семантики языка. Эта особенность также весьма полезна и при построении компиляторов. Формально, перевод называется синтаксически управляемым, если есть синтаксически управляемая схема (СУ-схема), его реализующая. Определим формально СУ-схему.

Определение 1. Синтаксически управляемой схемой назовём пятёрку $T = (N, \Sigma, \Delta, R, S)$, где N множество нетерминалов, Σ и Δ алфавиты входа и выхода схемы, R – множество правил вида $A \rightarrow \alpha, \beta$, причём нетерминалы входящие в цепочку α входят также и в цепочку β , причём с той же кратностью.

Как легко видеть, из языка $L(G)$ существует СУ-перевод в язык $L(G')$, схема которого строится по грамматикам. А именно множество R строится по соответствующим парам правил, описанных выше.

Упражнение 1. Предъявить алгоритм построения по грамматике G синтаксический перевод T_l , переводящий слово w из $L(G)$ в левый вывод данного слова $\pi_l(w)$.

Восходящий разбор строится аналогично по правому выводу.

Упражнение 2. Построить правый вывод $w = a + (a * a)$ по дереву разбора. По правому выводу построить разбор $\pi_r(w)$.

Упражнение 3. Построить по описанной выше грамматике G схему СУ-перевода, реализующую перевод $w \rightarrow \pi_r(w)$. Предъявить алгоритм построения схемы данного СУ-перевода по грамматике.

2 Функция FIRST

При построении (детерминированных) анализаторов по грамматике, нам потребуется определять множество первых k символов слов, выводимых из цепочки $\alpha \in (N \cup T)^*$. Для этого мы будем использовать функцию FIRST_k , которая определена через функцию FIRST_1 или просто FIRST . Таким образом умение вычислять функцию FIRST является ключевым при построении анализаторов.

Формально,

$$\text{FIRST}_k(\alpha) = \{w[1, k] \mid \alpha \Rightarrow w, |w| > k\} \cup \{w \mid \alpha \Rightarrow w, |w| < k\}$$

Если $\alpha \Rightarrow \varepsilon$, то пустое слово лежит в $\text{FIRST}_k(\alpha)$.

Приведём процедуру вычисления функции $\text{FIRST}(\alpha)$.

Идея алгоритма: Если $\alpha = X_1X_2 \dots X_n$ начинается с терминала σ , то первым символом может быть только этот терминал, таким образом, мы сразу получаем ответ σ . Если же α начинается с нетерминала, то $\text{FIRST}(\alpha) = \text{FIRST}(X_1)$, если из нетерминала X_1 не выводится пустое слово, и $\text{FIRST}(\alpha) = \text{FIRST}(X_1) \cup \text{FIRST}(X_2X_3 \dots X_n)$, если $X_1 \Rightarrow \varepsilon$.

Таким образом, мы описали вычисление функции FIRST на множестве терминалов и цепочек, начинающихся с терминалов, осталось описать вычисление функции на множестве нетерминалов, как видно вычисление функции FIRST на множестве синтаксических форм сводится к вычислению функции на отдельных нетерминалах.

Пусть мы вычисляем функцию $\text{FIRST}(X)$ для нетерминала X . Рассмотрим все правила вида $X \rightarrow \beta$. Очевидно, что $\text{FIRST}(\beta)$ является подмножеством $\text{FIRST}(X)$, но просто добавляя множество $\text{FIRST}(\beta)$ к $\text{FIRST}(X)$ мы получим порочный круг, в случае правил вида $X \rightarrow Xa$. Как нам избежать порочного круга при вычислении множества $\text{FIRST}(X)$?

Определим множества $F_i(Y)$, $Y \in N$. При $i = 0$ для любого нетерминала Y , множество $F_i(Y) = \emptyset$, или $\{\varepsilon\}$, если есть правило $Y \rightarrow \varepsilon$. На i -ом шаге алгоритма будем вычислять множества $F_i(X)$ следующим образом. В начале шага $F_i(X)$ включает себя множество $F_{i-1}(X)$. Если есть правило $X \rightarrow \beta = Y_1Y_2 \dots Y_n$ и Y_1 – терминал или β – пустое слово, то добавим к множеству $F_i(X)$ элемент Y_1 (быть может пустое слово). Если же Y_1 – нетерминал, и при этом пустое слово не лежит в $F_{i-1}(Y_1)$, то добавим к множеству $F_i(X)$ множество $F_{i-1}(Y_1)$ и вычислим множество $F_i(Y_1)$. Если же $\varepsilon \in F_{i-1}(Y_1)$, то добавим к $F_i(X)$ множество $F_i(Y_1) \setminus \{\varepsilon\}$ и повторим описанную операцию для $\beta = Y_2 \dots Y_n$.

Алгоритм останавливается, как только для каждого нетерминала Y , множества F_i и F_{i-1} совпадают.

Алгоритм:

Шаг 0. Для каждого терминала σ положим $F_i(\sigma) = \sigma$ для любого i . Для каждого нетерминала Y , если есть правило $Y \rightarrow \varepsilon$, положим $F_0(Y) = \{\varepsilon\}$, иначе положим $F_0(Y) = \emptyset$.

Шаг i . Добавить к множеству $F_i(X)$ множество $F_{i-1}(X)$. Для каждого правила $X \rightarrow \beta = Y_1 \dots Y_n$ выполнить:

$$j = 1$$

Пока $\varepsilon \in F_{i-1}(Y_j)$
добавить $F_{i-1}(Y_j) \setminus \{\varepsilon\}$ к $F_i(X)$,
вычислить $F_i(Y_j)$,
увеличить j .
Добавить $F_{i-1}(Y_j)$ к $F_i(X)$,
вычислить $F_i(Y_j)$.
Остановка. $F_i(Y) = F_{i-1}(Y)$ для любого Y из N . Положить $\text{FIRST}(X) = F_i(X)$.

Упражнение 4. Доказать корректность данного алгоритма.

3 Функция FOLLOW

Помимо префикса порождаемого цепочкой β нас будет интересовать также и множество слов, которые могут следовать после слова, выведенного из цепочки β . Запишем сначала формальное определение функции FOLLOW_k .

$$\text{FOLLOW}_k(\beta) = \{w \mid S \Rightarrow \alpha\beta\gamma, w \in \text{FIRST}_k(\gamma)\}.$$

Неформально, в множестве $\text{FOLLOW}_k(\beta)$ содержатся те слова, которые могут следовать за словом, выведенным из β , в цепочке $\alpha\beta\gamma$, выводимой из аксиомы. Длина этих слов ограничена k , что означает, что если $\gamma \Rightarrow w$ и длина слова w меньше k , то w лежит в множестве $\text{FOLLOW}_k(\beta)$, а если же длина слова w больше k , то в множестве $\text{FOLLOW}_k(\beta)$ лежит префикс w длины k .

Аналогично функции FIRST , мы будем обозначать FOLLOW_1 как FOLLOW .

Мы будем часто пользоваться функцией FOLLOW_k в теоретических целях и для обозначения объектов, однако на практике мы будем вычислять функцию FOLLOW только на множестве нетерминалов.

Приведём алгоритм для вычисления функции FOLLOW .

Идея алгоритма: Если в грамматике есть правило $A \rightarrow \alpha X \beta$, то за словом, выведенным из нетерминала X следует слово выведенное из β , таким образом множество $\text{FOLLOW}(X)$ включает в себя множество $\text{FIRST}(\beta)$. Если, при этом из цепочки β выводимо пустое слово, то

за словом, выводимым из нетерминала X следует слово из множества $\text{FOLLOW}(A)$, поскольку из вывода

$$S \Rightarrow^* \gamma Aw \Rightarrow \gamma \alpha X \beta w \Rightarrow \gamma \underbrace{\alpha X}_A w$$

следует, что если элемент $\text{FIRST}(w)$ лежит в множестве $\text{FOLLOW}(A)$, то элемент $\text{FIRST}(w)$ лежит так же в множестве $\text{FOLLOW}(X)$. Таким образом, по определению функции FOLLOW , если в грамматике есть правило $A \rightarrow \alpha X \beta$ и при этом из цепочки β выводимо пустое слово, то множество $\text{FOLLOW}(X)$ включает в себя множество $\text{FOLLOW}(A)$. В частности, возможно что $\beta = \varepsilon$, поэтому при наличии в грамматике правила $A \rightarrow \alpha X$, справедливо $\text{FOLLOW}(X) \supseteq \text{FOLLOW}(A)$.

В итоге, мета-алгоритм сводится к следующим шагам:

- Вычислить множества FIRST для грамматики G ;
- Для правил $A \rightarrow \alpha X \beta$ добавить $\text{FIRST}(\beta) \setminus \{\varepsilon\}$ к $\text{FOLLOW}(X)$;
- Для правил $A \rightarrow \alpha X \beta$, таких что, $\varepsilon \in \text{FIRST}(\beta)$ добавить $\text{FOLLOW}(A)$ к $\text{FOLLOW}(X)$.

Упражнение 5. Доказать корректность данного мета-алгоритма. То есть, что все элементы множеств FOLLOW будут найдены и ничего лишнего найдено не будет.

Замечание 1. По хорошему, возникает проблема с тем, лежит ли пустое слово в $\text{FOLLOW}(X)$. Эта проблема решается следующим образом: ко всем словам, порождаемым G добавляется маркер конца слова, и если этот маркер оказывается в $\text{FOLLOW}(X)$, то пустое слово принадлежит $\text{FOLLOW}(X)$. Для этого по грамматике G строится пополненная грамматика G' , которая содержит правило $S' \rightarrow S\$$, где $\$$ – маркер конца слова. Все остальные правила грамматики G' взяты из грамматики G . На практике, функция FOLLOW используется в анализаторах, на вход которым и так подаётся пополненная грамматика, поэтому решать проблему наличия пустого слова в множестве $\text{FOLLOW}(X)$ не надо.

Теперь опишем сам алгоритм. Идея алгоритма схожа с индуктивным вычислением множеств FIRST .

Алгоритм:

Шаг 0. Для каждого нетерминала Y положим $F_0(Y) = \emptyset$. Вычислим значение функции FIRST для грамматики G .

Шаг i . Положить множество $F_i(X)$ равным множеству $F_{i-1}(X)$. Для каждого правила $A \rightarrow \alpha X \beta$ добавить $\text{FIRST}(\beta) \setminus \{\varepsilon\}$ к $F_i(X)$; Если $\varepsilon \in \text{FIRST}(\beta)$ добавить $F_{i-1}(A)$ к $F_i(X)$.

Остановка. Как только $F_i(Y) = F_{i-1}(Y)$ для любого Y из N . Положить $\text{FIRST}(X) = F_i(X)$.

4 От FIRST к FIRST_k

Сначала введём вспомогательную операцию на множествах. Пусть L_1 и L_2 некоторые языки. Тогда язык $L_1 \oplus_k L_2$ состоит из всех таких слов w , что либо в языке L_1 есть слово w_1 длины не меньшей k и $w = w_1[1, k]$, либо слово x длины не большей k лежит в L_1 , слово y лежит в L_2 , слово u есть их конкатенация xy и, наконец, $w = u[1, k]$, если $|u| > k$ или просто $w = u$, если $|u| < k$. Формально

$$L_1 \oplus_k L_2 = \{w \mid \exists x \in L_1, \exists y \in L_2, u = xy, |u| \leq k \Rightarrow w = u, |u| > k \Rightarrow w = u[1, k]\}$$

Другой вариант формального определения, чтобы окончательно запутать читателя:

$$L_1 \oplus_k L_2 = \{xy \mid x \in L_1, y \in L_2, |xy| \leq k\} \cup \{u[1, k] \mid \exists x \in L_1, \exists y \in L_2, u = xy, |xy| > k\}$$

Из определения операции \oplus_k следует, что для $X_1, X_2, \dots, X_n \in N \cup T$ справедливо

$$\text{FIRST}_k(X_1 X_2 \dots X_n) = \text{FIRST}_k(X_1) \oplus_k \text{FIRST}_k(X_2) \oplus_k \dots \oplus_k \text{FIRST}_k(X_n).$$

Фактически, когда мы вычисляли функцию FIRST, мы вычисляли её используя оператор \oplus_1 . Перепишем алгоритм вычисления функции FIRST для вычисления функции FIRST_k .

Алгоритм:

Шаг 0. Для каждого терминала σ положим $F_i(\sigma) = \sigma$ для любого i . Для каждого нетерминала Y , рассмотрим все правила вида $Y \rightarrow x\alpha$, где x – слово (быть может пустое!), а цепочка α либо начинается с нетерминала, либо пуста. Если $|x| \geq k$, добавим к множеству $F_0(Y)$ слово $x[1, k]$. Иначе, если $\alpha = \varepsilon$, добавим к множеству $F_0(Y)$ слово x .

Шаг i . Добавить к множеству $F_i(X)$ множество $F_{i-1}(X)$. Для каждого правила $X \rightarrow \beta = Y_1 \dots Y_n$

добавить к $F_i(X)$ множество $F_{i-1}(Y_1) \oplus_k \dots \oplus_k F_{i-1}(Y_n)$,

вычислить $F_i(Y_j)$, для $j = 1..n$

Остановка. $F_i(Y) = F_{i-1}(Y)$ для любого Y из N . Положить $\text{FIRST}_k(X) = F_i(X)$.

Упражнение 6. Доказать корректность работы данного алгоритма.

На практике удобно вычислять функцию FIRST_k для всех нетерминалов сразу.

5 LL(k)-грамматики

Мы не будем строить анализаторы для LL(k)-грамматик, где $k > 1$ в силу нехватки времени. Тем не менее, мы будем работать с определением LL(k)-грамматики и её свойствами.

Вспомним, что грамматика является LL(k)-грамматикой тогда и только тогда, когда она левоанализируема, т.е. существует детерминированный анализатор (ДМП-автомат с выходом), который реализует СУ перевод $w \rightarrow \pi_l(w)$.

С таким определением не очень удобно работать с точки зрения анализа грамматики, поэтому мы будем также использовать эквивалентные ему определения.

Теорема 1. Грамматика является LL(k)-грамматикой тогда и только тогда, когда для любых двух правил $A \rightarrow \beta, A \rightarrow \gamma$, $\text{FIRST}_k(\gamma\alpha) \cap \text{FIRST}_k(\beta\alpha) = \emptyset$ для таких α , что $S \Rightarrow_l^* wA\alpha$.

Не все грамматики, задающие LL-языки являются LL-грамматиками. Но некоторые из них можно преобразовать к LL(k)-грамматике используя приёмов левой факторизации и удаления левой рекурсии. Изучите эти приёмы по книжке Серебрякова или по Ахо и Ульману.

6 Задачи

В первых двух задачах под грамматикой G понимается грамматика, порождающая арифметические выражения.

Задача 1. Построить дерево вывода, левые и правые разборы для слова $((a))$ в грамматике G , определённой выше.

Задача 2. Построить детерминированный левый анализатор для грамматики

$$S \rightarrow 0S \quad (1)$$

$$S \rightarrow 1S \quad (2)$$

$$S \rightarrow \varepsilon \quad (3)$$

З* Добавим в грамматику G правило $E \rightarrow \varepsilon$. Вычислите значение функции $FIRST(E)$.

Задача 4. Докажите, что грамматика не является LL(1)-грамматикой, но является LL(2)-грамматикой. Вычислите функции $FIRST_2$ и $FOLLOW_2$ для всех нетерминалов.

$$S \rightarrow aAaa|bAba$$

$$A \rightarrow b|\varepsilon$$

Задача 5. Для грамматики написать эквивалентную LL(1)-грамматику и вычислить функции $FIRST$ и $FOLLOW$ для каждого нетерминала. Постройте по получившейся грамматике LL(1)-анализатор.

$$S \rightarrow ba|A$$

$$A \rightarrow a|Aab|Ab$$

Задача 6*: Докажите, что язык $a^* \cup a^n b^n$ не является LL-языком.