Титульный лист материалов по дисциплине

ДИСЦИПЛИНА *Теория формальных языков*

ИНСТИТУТ *Информационных технологий*

КАФЕДРА *Вычислительной техники*

ВИД УЧЕБНОГО МАТЕРИАЛА *Лекция*

ПРЕПОДАВАТЕЛЬ *Унгер Антон Юрьевич*

СЕМЕСТР *3 семестр*

3. ФОРМАЛЬНЫЕ ЯЗЫКИ И ГРАММАТИКИ

Современная теория формальных языков строится на базе математики и логики. Для изложения дальнейшего материала нам потребуется ряд определений. Предполагается, что читатель имеет представление об основных понятиях теории множеств.

Определение 1. *Алфавитом V* называется конечное множество символов. Здесь под *символом* подразумевается некий атомарный, грамматически неделимый элемент множества.

Определение 2. Любая последовательность символом конечной длины в алфавите *V* называется *цепочкой символов*.

Определение 3. *Пустая цепочка* – цепочка, не содержащая ни одного символа. Пустая цепочка обозначается греческой буквой ε.

Здесь необходимо отметить, что сама по себе пустая цепочка и буква, которой она обозначается ε, не входят в алфавит *V*.

Определение 4. *Конкатенацией* называется бинарная операция, операндами, которой служат две цепочки α и β.

Пусть, например, $\alpha = ab,\ \beta = cd,$ тогда $\alpha\beta = abcd$ есть конкатенация этих двух цепочек.

Для любой цепочки справедливы следующие равенства:

1. $\alpha\varepsilon = \varepsilon\alpha = \alpha.$

2. Для любых цепочек $\alpha,\ \beta,\ \gamma$ действует свойство ассоциативности, т.е. $(\alpha\beta)\gamma = \alpha(\beta\gamma) = \alpha\beta\gamma.$

Определение 5. *Реверсом* цепочки называется операция обращения цепочки, в результате которой получается цепочка, все символы которой записаны в противоположном порядке. Реверс обозначается так $R(\alpha)$.

Очевидны следующие свойства реверса:

1. Реверс пустой строки приводит к пустой строке, т.е. $R(\varepsilon) = \varepsilon.$

2. Реверс строки, состоящей из одного символа приводит к той же самой строке.

Определение 6. α^n – возведение в степень n цепочки α – называется n-кратная конкатенация цепочки с самой собой, т.е.

$$\alpha^n = \alpha\alpha \ldots \alpha\ (n \text{ раз}).$$

Свойствами возведения в степень являются:

1. $\alpha^0 = \varepsilon.$

2. $\alpha^n = \alpha^{n-1}\alpha = \alpha\alpha^{n-1}.$

Определение 7. Количество символов в цепочке определяет *длину цепочки*. Длина цепочки обозначается так $|\alpha|$.

Очевидно, длина пустой цепочки $|\varepsilon| = 0$.

Определение 8. Множество V^* над алфавитом V называется множество всех цепочек, включая пустую цепочку.

Например, пусть алфавит состоит из символов 0 и 1, $V = \{0,1\}$. Тогда, $V^* = \{\varepsilon, 0,1,00,01,10,11,000,001 \dots\}$.

Определение 9. Множество V^+ над алфавитом V называется множество всех цепочек, не включая пустую цепочку. Аналогично рассмотренному выше примеру $V^+ = \{0,1,00,01,10,11,000,001, \dots\}$.

Приведенные определения дают возможность ввести понятие языка.

Определение 10. *Языком* L в алфавите V называется подмножество всех цепочек, определенных для данного алфавита, т.е. $L \subseteq V$.

Согласно данному определению в язык могут быть включены не все возможные перестановки символов данного алфавита. Если множество всех цепочек, составляющих язык конечно, то язык можно определить простым перечислением всех возможных цепочек. Однако, все действующие языки программирования включают бесконечное множество цепочек, которые составляются по определенным правилам. Поэтому существует два подходя для определения бесконечных языков.

Первый подход основывается на процедуре *распознавания*. Данная процедура отвечает на вопрос о принадлежности цепочки языку. Говорят, что цепочка принадлежит языку или допускается языком, если распознаватель признает ее, т.е. процедура распознавания завершается с ответом ***да***. Иначе, распознаватель завершается с ответом ***нет***. Таким образом, определение языка таково: язык – это множество всех цепочек, которые он допускает.

Распознаватель можно схематично представить в виде совокупности входной ленты, читающей головки, которая указывает на очередной символ на ленте, устройства управления (УУ) и дополнительной памяти (стек).

Конфигурацией распознавателя является:

- содержимое входной ленты;
- положение читающей головки;
- состояние УУ;
- содержимое дополнительной памяти.

Существует множество примеров распознавателей. Вот некоторые из них:

1. Машина Тьюринга [7], допускает все рекурсивно-перечислимые языки. Это наиболее общий вид распознавателей.

2. Линейно-ограниченный автомат, допускает контекстно-зависимые языки. Автомат отличается от машины Тьюринга тем, что его лента не бесконечна.

3. Автомат со стековой памятью, допускает все контекстно-свободные языки. Данный автомат в отличие от линейно-ограниченного автомата, не позволяет читающей головке перемещаться по ленте влево и не позволяет изменять входное слово.

4. Конечный автомат допускает регулярные языки и является наиболее ограниченным типом распознавателей формальных языков.

Данные примеры следуют классификации Хомского [8], о которой будет идти речь ниже.

Вторым способом описания бесконечных языков является механизм порождения, основанный на порождающих грамматиках. Использование порождающих грамматик является самым распространенным средством описания формальных языков в целом и языков программирования в частности.

3.1. Порождающие грамматики

Термин грамматика чаще всего встречается в контексте естественного языка, на котором общаются люди. Не стоит удивляются тому, что в теории трансляции данному термину дается совершенно четкий смысл. Формы Бэкуса-Наура, о которых уже говорилось выше, являются попыткой описать свод грамматических правил с помощью формальной нотации, однако для строго математического определения того, что же такое грамматика, требуется нечто большее.

Определение 11. Декартово произведение двух множеств, обозначаемое $A \times B$, есть множество, элементами которого являются все пары (a, b), такие что $a \in A$ и $b \in B$.

Определение 12. Порождающая грамматика есть четверка элементов $G =< T, N, P, S >$, где

T – множество терминальных символов (иначе, терминалов);

N – множество нетерминальных символов (иначе, нетерминалов);

P – множество правил вывода, вида $\alpha \to \beta$. Здесь α называют левой частью правила, а β – правой. Символ $\to$ читается, как «может иметь вид».

S – символ предложения, начальный символ или цель грамматики.

Рассмотрим эти множества подробнее. *Терминалами* называют конечные неделимые элементы языка, которые нельзя вывести из других элементов. Другими словами, терминалы составляют алфавит, из которого строятся все цепочки, принадлежащие языку, определяемому данной порождающей грамматикой.

Нетерминалы служат для описания того, как можно из одних цепочек по определенным правилам получить другие цепочки, принадлежащие данному языку. Множества терминальных и нетерминальных символов не пересекаются, т.е. $T \cap N = \emptyset$.

Правила вывода, как раз и являются этими правилами перевода одних допустимых цепочек в другие. Множество P представляет собой декартово произведение множеств $(T \cup N)^+ \times (T \cup N)^*$. Здесь левая часть каждого правила должна включать хотя бы один нетерминал.

Правила вывода по-другому называют *продукциями*. Продукции используются для генерации возможных строк языка по следующему алгоритму:

1. Начать с символа *S*, заменить его строкой справа от →.

2. Продолжать замену до тех пор, пока в строке есть хотя бы один символ нетерминальный символ.

Грамматика используется для генерации последовательностей символов, которые составляют цепочки языка. Процесс, в ходе которого к каждому нетерминалу, начиная с *S*, последовательно применяется грамматическое правило из числа продукций *P*, продолжается до тех пор, пока в строке не останутся одни терминалы.

Из множества всех нетерминалов выделяют один $S \in N$ – *цель грамматики*, с которого начинается вывод (порождение) всех цепочек данного языка.

Введем следующие удобные сокращения. Одной и той же левой части некоторого правила вывода может соответствовать несколько альтернативных правых частей. Вместо записи

$$\alpha \to \beta_1;$$
$$\alpha \to \beta_2;$$
$$\dots;$$
$$\alpha \to \beta_n,$$

мы будем использовать сокращенную запись

$$\alpha \to \beta_1 | \beta_2 | \dots | \beta_n. \qquad (2.1)$$

Пример 1. Простейшая грамматика, состоящая из трех правил вывода, имеет вид: $G_1 =< \{0,1\}, \{A, S\}, P, S >$, где правила вывода таковы:

$$S \to 0A1;$$
$$0A \to 00A1;$$
$$A \to \varepsilon.$$

Забегая вперед, скажем, что данная грамматика является неукорачивающей контекстно-зависимой грамматикой.

Определение 13. Если при последовательном применении правил вывода можно из цепочки α за конечное число шагов получить цепочку β, т.е. $\alpha \to \gamma_1 \to \gamma_2 \to \cdots \to \gamma_n \to \beta$, то говорят, что цепочка β *выводима* из цепочки α. Последовательность правил $\gamma_1, \gamma_2, \ldots, \gamma_n$ называется выводом длины n.

На примере грамматики G_1 цепочку $000A111$ можно получить из цели грамматики за 3 шага: $S \to 0A1 \to 00A11 \to 000A111$.

Определение 14: если в грамматике существует правило $\gamma \to \delta$ и $\alpha \to f(\gamma)$ и $\beta \to g(\delta)$, то говорят, что цепочка β *непосредственно выводится* из α.

Например, в приведенной выше грамматике цепочка $00A11$ непосредственно выводима из цепочки $0A1$. В самом деле, если к левой и правой частям второго правила прибавить символ 1, то из цепочки $0A1$ непосредственно получаем $00A11$.

Пришло время связать формально эти два понятия – язык и грамматику.

Определение 15. язык L, порождаемый заданной грамматикой G, есть множество цепочек, которые можно вывести из стартового символа S. Это значит, что с помощью правил вывода P можно получить все строки этого языка. И обратно, нельзя породить ни одну строку, не принадлежащую данному языку.

Пример 2. Пусть язык формально определен так $L = \{x^n | n > 0\}$. Данный язык допускает все строки вида: x, xx, xxx и т.д., т.е. строки, состоящие из произвольного количества символов x. Данному языку соответствует грамматика $G =< \{x\}, \{S\}, P, S >$, где правило вывода P таково:

$$S \to xS | x.$$

Пример 3. Пусть язык определен так $L = \{x^n y^n | n > 0\}$. В данном случае язык определяет строки, состоящие из произвольного количества символов x, за которым следует точно такое же количество символов y. Данному языку соответствует грамматика $G =< \{x, y\}, \{S\}, P, S >$ с одним правилом вывода:

$$S \to xSy | xy.$$

Пример 4. Пусть язык определен так $L = \{x^n y^m | n, m \geq 0\}$. Данные язык отличается от предыдущего, так как определяет строки, состоящие из произвольного количества символов x, за которым следует произвольное количество символов y, причем n не зависит от m. Кроме того, данный язык допускает пустую строку ε. Грамматика, порождающая данный язык $G = < \{x, y\}, \{S, B\}, P, S >$. Здесь правила вывода таковы:

$$S \to xS;$$
$$S \to yB;$$
$$S \to x;$$
$$S \to y;$$
$$S \to \varepsilon;$$
$$B \to yB;$$
$$B \to y.$$

Определение 16. *Сентенциальная форма* представляет собой некоторую цепочку символов, состоящую из терминалов и нетерминалов $\alpha \in (T \cup N)^*$, такую что ее можно получить, последовательно применяя правила вывода. Другими словами $S \Rightarrow \alpha$. Итоговая форма, состоящая из одних терминалов, называется *сентенцией* (от англ. *sentence* – предложение).

Множество сентенциальных форм, в которых присутствуют только терминальные символы, представляет собой все строки данного языка.

Возвращаясь к предыдущей главе, можно увидеть в структуре примера с описанием арифметических выражений с помощью нотации Бэкуса-Наура грамматику формального языка арифметических выражений. В самом деле, нетерминалы `<выражение>`, `<операнд>`, `<множитель>`, `<число>` и `<цифра>` соответствуют второму элементу грамматики. т.е. N, множество десятичный цифр, а также символы «+», «−», «*», «/», «(», «)» составляют алфавит языка, т.е. T, множество P правил вывода – это просто последовательность правил БНФ, в которых символ `::=` заменен на символ →. Наконец, стартовым символом грамматики является первое грамматическое правило БНФ, т.е. `<выражение>`.

Здесь необходимо сделать важное замечание. Правила в БНФ всегда содержат один единственный нетерминал в левой части, но порождающие грамматики в общем случае могут в левой части содержать любую строку, состоящую из терминалов и нетерминалов. Такие грамматики относятся к классу контекстно-зависимых грамматик, для которых замена нетерминала на правую часть возможна лишь в определенном *контексте*. Формы Бэкуса-

Наура, таким образом, позволяют описывать только грамматики свободные от контекста, т.е. контекстно-свободные (КС) грамматики.

Определение 17. Грамматики G_1 и G_2 называются *эквивалентными*, если они порождают один и тот же язык (одно и то же множество строк).

Пример 5. Рассмотренная выше грамматик G_1 эквивалентна следующей грамматике

$$S \to 0S1|01.$$

В самом деле, и та и другая грамматика порождают язык $L = \{0^n1^n|n > 0\}$.

Определение 18. Если две грамматики G_1 и G_2 порождают языки L_1 и L_2, которые отличаются не более чем на пустую строку, то такие грамматики называют *почти эквивалентными*.

Пример 6. Рассмотренная выше грамматика G_1 почти эквивалентна следующей грамматике

$$S \to 0S1|\varepsilon.$$

В самом деле, в данном случае получается язык $L = \{0^n1^n|n \geq 0\}$, который допускает пустую строку. Грамматика G_1 пустую строку не допускает.

3.2. Классификация грамматик по Хомскому

Сделаем одно замечание. Буквы в нижнем регистре мы будем использовать для обозначения терминалов, а заглавные буквы, записанные в верхнем регистре, (как в примерах выше) для обозначения нетерминалов.

Рассмотрим более сложный пример чем те, что мы рассматривали ранее.

Пример 7. Пусть дана грамматика $G_3 = (\{a\}, \{S, N, Q, R\}, P, S)$, для которой правила вывода следующие:

$$S \to QNQ;$$
$$QN \to QR;$$
$$RN \to NNR;$$
$$RQ \to NNQ;$$
$$N \to a;$$
$$Q \to \varepsilon.$$

Согласно этим правилам можно подменить нетерминал *N* на *R*, но только в случае, если перед *N* стоит *Q*. Далее, *R* можно заменить на *NN*, только если после *R* следует *Q*.

Данная грамматика относится к классу *контекстно-зависимых* грамматик. Для данного примера возможны следующие порождения:

$S \Rightarrow QNQ \Rightarrow QRQ \Rightarrow QNNQ \Rightarrow aa;$

$S \Rightarrow QNQ \Rightarrow QRQ \Rightarrow QNNQ \Rightarrow QRNQ \Rightarrow QNNRQ \Rightarrow QNNNNQ \Rightarrow aaaa.$

Можно показать [5], что число терминалов (a) в получающихся строках составляет степень двойки. Другими словами, данная грамматика генерирует следующий язык:

$$\{a^m | m = 2,4, \dots, 2^n; n > 0\}.$$

Согласно общепринятой классификации (*иерархии Хомского*) все грамматики подразделяются на 4 типа в зависимости от ограничений, накладываемых на правила вывода.

К грамматике **0-го типа** относятся все грамматики без каких бы то ни было ограничений. Данный тип грамматик эквивалентен множеству рекурсивно-перечислимых языков.

3.2.1. Грамматики с ограничениями на правила вывода

К грамматике **1-го типа** относятся грамматики, у которых для всех продукций $\alpha \to \beta$ длина строки α не больше длины строки β. Математически это можно выразить так:

$$|\alpha| \leq |\beta|. \qquad (2.2)$$

У данного правила есть одно исключение: допускается продукция вида $S \to \varepsilon$, хотя формально здесь длина строки *S* больше длины строки ε.

Подобные грамматики называются *контекстно-зависимыми*. Языки, порождаемые контекстно-зависимыми грамматиками, называют контекстно-зависимыми языками.

В литературе контекстно-зависимые грамматики еще называют неукорачивающими, благодаря правилу (2.2). Можно показать, что класс контекстно-зависимых грамматик (и языков) совпадает с классом неукорачивающих грамматик (языков) [9].

К грамматикам **2-го типа** относятся все грамматики 1-го типа, у которых в левых частях продукций может находиться только один нетерминал. Такие грамматики называются *контекстно-свободными* (КС). Согласно определению, правила вывода для КС-грамматик имеют вид:

$$A \to \beta.$$

Здесь $A \in N, \beta \in (T \cup N)^*$.

Так же как и контекстно-зависимые грамматики КС-грамматики допускают правило вывода вида $S \to \varepsilon$.

Язык, порождаемый КС-грамматикой, называется *контекстно-свободным* языком.

К грамматике **3-го типа** относят все КС-грамматики, для которых все продукции могут быть вида:

$$A \to a|bC, \tag{2.3}$$

или

$$A \to a|Cb. \tag{2.4}$$

В первом случае, грамматика называется *праволинейной*, а во втором *леволинейной*. Праволинейные и леволинейные грамматики образуют класс *регулярных* грамматик. Важно отметить, что, если грамматика включает в себя и праволинейные и леволинейные продукции, она уже не будет являться регулярной.

Утверждение 1. Если для языка L существует леволинейная порождающая грамматика $G_l(L)$, то для него существует эквивалентная праволинейная грамматика $G_r(L)$.

Таким образом, праволиненые и леволинейные грамматики порождают один и тот же класс *регулярных* языков.

По-другому регулярные грамматики еще называют *автоматными*.

3.2.2. Иерархия Хомского

Следующие соотношения справедливы для любой формальной грамматики.

1. Любая регулярная грамматика является одновременно КС-грамматикой.

2. Любая КС-грамматик является одновременно контекстно-зависимой грамматикой.

3. Любая контекстно-зависимая грамматика является одновременно грамматикой типа 0.

Данные соотношения мы приводим без доказательств, однако они напрямую вытекают из определений типов грамматик, представленных выше. Приведем несколько примеров.

Пример 8. Язык

$$L = \{a^n b^n | n > 0\}$$

является КС-языком, однако не является регулярным. Доказательство этому будет дано после того, как будут введены такие понятия, как регулярное множество и регулярное выражение.

Пример 9. Язык

$$L = \{a^n b^n c^n | n > 0\}$$

является контекстно-зависимым языком, однако не является КС-языком. Доказательство этому можно найти, например, в [7].

Язык, генерируемый регулярной грамматикой, называется регулярным. Между регулярным языком, регулярными выражениями и конечными автоматами существует тесная связь. С регулярными выражениями мы познакомимся ниже. Таким образом, любой регулярный язык можно описать регулярным выражением, а любое регулярное выражение можно описать *конечным автоматом*. Схематически это представлено на рис. 1.

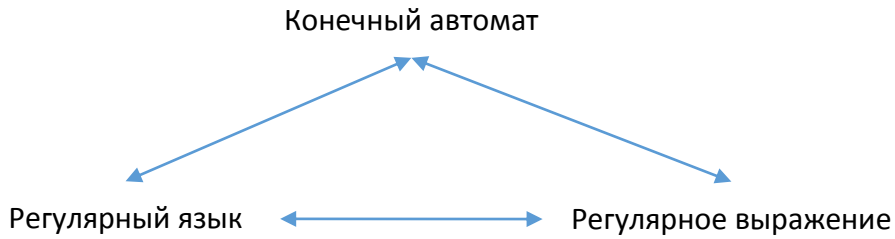

Рисунок 1. Связь между регулярным языком и конечным автоматом

Аналогично, грамматики 2-го типа допускаются автоматами с магазинной (стековой) памятью, грамматики 1-го типа – линейной ограниченными автоматами, а грамматики 0-го типа – полной машиной Тьюринга.

Для справки, **машина Тьюринга** является обобщением конечного автомата, содержащего бесконечную ленту, в которой хранятся данные (символы) необходимые для выполнения некоторого алгоритма. Можно показать, что на машине Тьюринга можно вычислить любой алгоритм, который можно разложить на последовательность элементарных действий (полнота по Тьюрингу).

Иерархия типов грамматик, которую впервые ввел американский лингвист Ноам Хомским, схематически показана на рис. 2.

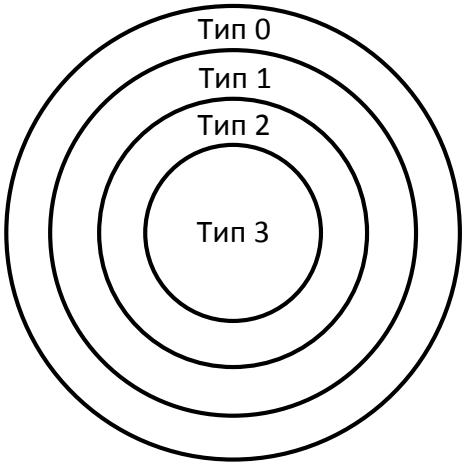

Рисунок 2. Иерархия грамматик по Хомскому

Для нас наибольший интерес представляют уровни 2 и 3 на данной схеме, так как большинство языков программирования относятся именно к типу 2, т.е. являются контекстно-свободными (КС). На практике, даже если в языке присутствуют зависящие от контекста структуры, для него строят КС-грамматику, а эти особенности учитывают на этапе семантического анализа.

Отметим, что язык данного типа можно описать грамматикой данного типа или грамматикой более общего типа.

Пример 10. Выше мы определили язык $L = \{x^n y^m | x, y \geq 0\}$, как регулярный, т.е. 3-го типа. Соответствующая регулярная грамматика такова:

$$S \to xS;$$
$$S \to yB;$$
$$S \to x;$$
$$S \to y;$$
$$S \to \varepsilon;$$
$$B \to yB;$$
$$B \to y.$$

В то же время данный язык можно описать и контекстно-свободной грамматикой, т.е. грамматикой 2-го типа:

$$S \to XY;$$
$$X \to xX;$$
$$X \to \varepsilon;$$
$$Y \to yY;$$
$$Y \to \varepsilon.$$

Утверждение 2. Если для заданного языка существует грамматика типа k, то не существует метода, позволяющего сказать, существует ли для данного языка эквивалентной грамматики типа k+1. Другими словами, если язык является языком типа k (k=1,2,3), то он автоматически является языком типа $k-1$, однако обратное утверждение неверно. Это утверждение является прямым следствием диаграммы на рис. 2.

Вообще, чем выше тип языка, тем легче построить для него распознаватель. Наиболее простыми для анализа являются регулярные языки. Поэтому, естественно попытаться ответить на вопрос, можно ли описать заданный язык возможно более высоким типом грамматики.

Пример 11. Пусть дана грамматика G_1 со следующими правилами вывода:

$$S \to 0A1;$$
$$A \to 0A0;$$
$$A \to \varepsilon.$$

Необходимо отнести данную грамматику к некоторому типу. Очевидно, что данная грамматика удовлетворяет ограничениям, накладываемым на грамматики типа 2, т.е. КС-грамматики. Таким образом, грамматика G_1 является грамматикой типа 2. Однако, она не является регулярной.

С другой стороны язык, описываемый КС-грамматикой G_1 является регулярным. В самом деле, язык включает все строки, начинающие с нуля, за которым следует число нулей, кратное 2, и оканчивающиеся на 1. Данное описание позволяет задать язык следующей грамматикой:

$$S \to 0A;$$
$$A \to 0B|1;$$
$$B \to 0A.$$

Данная грамматика является регулярной. Следовательно, язык, который мы хотим описать, является языком типа 3, т.е. регулярным языком.

Пример 12. Пуст дана грамматика G_2 со следующей продукцией:

$$S \to aSb|\varepsilon.$$

Данная грамматика, очевидно, является грамматикой типа 2. Язык, который она порождает таков $L = \{a^n b^n | n \geq 0\}$. Данный язык, как будет показано ниже, невозможно описать регулярной грамматикой, следовательно, язык является языком типа 2.

3.2.3. Примеры формальных языков и грамматик

Рассмотрим несколько примером.

Пример 13. Язык, описываемый грамматикой

$$S \to aS|a, \tag{2.5}$$

включает все строки вида $L = \{a^n | n > 0\}$. Данный язык является регулярным (автоматным) языком, и может быть порожден как праволинейной грамматикой (2.5), так и леволинейной

$$S \to Sa|a. \tag{2.6}$$

Пример 14. Грамматика из предыдущего примера, допускающая пустые строки, может быть переписана следующим образом

$$S \to aS|\varepsilon. \tag{2.7}$$

Данная грамматика является праволинейной регулярной грамматикой, и язык, который ей соответствует, может быть описан так $L = \{a^n | n \geq 0\}$.

Пример 15. Рассмотрим следующую грамматику

$$\begin{gathered} S \to A \perp | B \perp; \\ A \to a|Ba; \\ B \to b|Bb|Ab. \end{gathered} \tag{2.8}$$

Данная грамматика леволинейная регулярная. Здесь символ $\perp$ является символом конца цепочки. Таким образом, все строки языка, порождаемого данной грамматикой, заканчиваются на символ конца. Эти строки включают все цепочки, состоящие и символов a и b, которые не содержат двух, идущих подряд символов aa.

Пример 16. Рассмотрим грамматику со следующими правилами вывода

$$\begin{gathered} S \to aQb|accb; \\ Q \to cSc. \end{gathered}$$

Данная грамматика является контекстно-свободной, т.е. относится к типу 2.

Язык, который она порождает может быть описан так

$$L = \{(ac)^n (cb)^n | n > 0\}.$$

Для данного языка не существует порождающей его регулярной грамматики.

Пример 17. Рассмотрим грамматику

$$S \to aSBC|abC; \tag{2.9}$$

$$CB \to BC; \tag{2.10}$$

$$bB \to bb; \tag{2.11}$$

$$bC \to bc; \quad (2.12)$$
$$cC \to cc. \quad (2.13)$$

Данная грамматика является примером контекстно-зависимой грамматики, о чем свидетельствуют правила (2.10) – (2.13). Данная грамматика порождает следующий язык

$$L = \{a^n b^n c^n | n > 0\}. \quad (2.14)$$

Данный язык уже встречался нам выше. В [10] доказывается, что для данного языка не существует КС-грамматики, поэтому он является языком типа 1.

Пример 18. Рассмотрим грамматику

$$S \to SS;$$
$$SS \to \varepsilon.$$

Эта, несложная не первый взгляд грамматика относится к грамматикам типа 0, равно как и язык, ею порождаемый. Интересно отметить, что несмотря на то, что данная грамматика не имеет никаких ограничений на правила вывода, порождаемый ею язык состоит из одной единственной цепочки $L = \{\varepsilon\}$.