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*

4. СВЯЗЬ МЕЖДУ ЯЗЫКОМ И ГРАММАТИКОЙ

Несмотря на множество приведенных примеров языков и грамматик, для доказательства того, что язык может быть порожден данной грамматикой необходимо ответить на следующие вопросы:

1. Правила вывода позволяют получить любую цепочку, принадлежащую данному языку.

2. Правила вывода не позволяют получить ни одну цепочку, которая не принадлежит данному языку.

Пример 19. Покажем, что следующая грамматика с правилами вывода (2.9) – (2.13) порождает язык (2.14).

Все цепочки, принадлежащие языку, содержат равное количество символов *a, b* и *c*, следующих друг за другом. Докажем методом математической индукции, что любая такая цепочка может быть получена применением правил (2.9) – (2.13).

1) цепочка *abc* получается применением правила (2.9) и (2.12). В самом деле, $S \to abC \to abc$. Этот случай соответствует $n = 1$.

2) для случая $n > 1$: $S \to aSBC \to aaSBCBC \to \cdots \to a^{n-1}S(BC)^{n-1} \to a^n bC(BC)^{n-1} \to \cdots \to a^n bB^{n-1}C^n \to a^n bbB^{n-2}C^n \to \cdots \to a^n b^n C^n \to a^n b^n cC^{n-1} \to a^n b^n ccC^{n-2} \to \cdots \to a^n b^n c^n$.

Мы показали, что любая цепочка вида $a^n b^n c^n$ может быть получена путем применения правил грамматики, но мы не показали, что какая-нибудь другая цепочка, не принадлежащая языку, не может быть получена, путем применения данных правил. Покажем это.

1. Новый символ *a, b* или *B,* а также *c* или *C*, могут появиться в сентенциальной форме только в равном количестве.

2. Нетерминал *B* можно заменить только на терминал *b*.

3. Нетерминал *C* можно заменить только на терминал *c*.

4. В любой сентенциальной форме терминал *a* всегда расположен левее терминала *b*, который, в свою очередь, расположен левее терминала *c*.

5. Терминал *b* может появиться только в результате применения правила (2.9).

6. Согласно правилу (2.11) нетерминал *B* заменяется на терминал *b,* тогда и только тогда, когда слева от него уже стоит терминал *b*. Другими словами, при применении правил грамматики терминал *b* появляется только справа от уже прочитанной строки.

7. Правило (2.12) применяется только после того, как в сентенциальной форме все нетерминалы B уже заменены на терминал b. Из этого следует, что символы c могут появляться только справа от символов b.

Таким образом, доказано, любая цепочка, выводимая из данных грамматических правил, содержит равное количество символов a, b и c, причем символы b следуют за a, символы c следуют за b, т.е. язык имеет вид (2.14). Что и требовалось доказать.

4.3. Разбор цепочек

Основной задачей анализа формального языка является построение вывода некоторой цепочки путем последовательного применения грамматических правил вывода. Это процесс называется *разбором*. Разбор строки можно осуществлять двумя способами. Первый способ состоит в том, что в зависимости от текущего символа строки применяется то или иное правило, начиная с правила для стартового символа S.

Второй способ состоит в том, что в исходной строке ищется подстрока, соответствующая правой части некоторого грамматического правила, и эта подстрока заменяется (*сворачивается*) в левую часть данного правила. Этот процесс повторяется до тех пор, пока сентенциальная форма не свернется к цели грамматики S.

Грамматики 0-го и 1-го типов практически не используются для анализа языков программирования, поскольку для них не существует стандартных алгоритмов разбора. Это не значит, что язык 0-го и 1-го типов невозможно проанализировать. Это означает только, что процесс анализа требует индивидуального «ручного» подхода.

Для целей трансляции языков программирования ключевой интерес представляют контекстно-свободные и регулярные грамматики, так как для них разработаны эффективные алгоритмы разбора. КС-грамматик как правило достаточно для того, чтобы описать все основные особенности современных языков программирования.

Рассмотрим подробнее процесс разбора цепочки по КС-грамматике. Напомним, что в КС-грамматике в левой части каждого правила вывода стоит один нетерминал.

Определение 19. Если каждая следующая сентенциальная форма отличается от предыдущей тем, что в нем заменен самый левый нетерминал, то такой вывод называется *левосторонним*.

Определение 20. Если каждая следующая сентенциальная форма отличается от предыдущей тем, что в нем заменен самый правый нетерминал, то такой вывод называется *правосторонним*.

Важно отметить, что при заданной грамматики вывод данной цепочки можно получить по-разному, применяя продукции в разном порядке. Например, пусть дана цепочка $a + b + a$. Получить вывод данной цепочки в грамматике G_4 с правилами вывода: $S \to T|T + S; T \to a|b$, можно так

1) $S \to T + S \to T + T + S \to T + T + T \to a + T + T \to a + b + T \to a + b + a.$

2) $S \to T + S \to a + S \to a + T + S \to a + b + S \to a + b + T \to a + b + a.$

3) $S \to T + S \to T + T + S \to T + T + T \to T + T + a \to T + b + a \to a + b + a.$

Здесь вывод (2) – левосторонний, вывод (3) – правосторонний, а вывод (1) – не является ни правосторонним, ни левосторонним.

Можно видеть, что каким бы ни был вывод, на каждом шаге согласном правилам грамматики заменяется один единственный нетерминал. Введем важное понятие.

Определение 21. *Деревом вывода* для заданной грамматики $G =< T, N, P, S >$ называется древовидная структура данных, которая удовлетворяет набору условий:

1. Каждая вершина дерева – это либо терминал T, либо нетерминал N, либо, если грамматика допускает пустую строку, ε.

2. Корень дерева есть стартовый символ – цель грамматики S.

3. Листьями дерева могут быть только терминальные символы.

4. Потомки $a_1, a_2, \dots, a_i$ каждого узла A дерева соответствуют правилу вывода $A \to a_1 a_2 \dots a_i$.

5. Если грамматика содержит правило вывода $A \to \varepsilon$, то этому правилу соответствует узел дерева, единственным потомком которого является лист ε.

Пример дерева вывода для цепочки $a + b + a$ в грамматике G_4 приведен на рис. 3.

Определение 22. *Неоднозначная* грамматика есть грамматика, в которой для одной и той же цепочки существует несколько различных деревьев вывода.

Определение 23. Язык, порожденный неоднозначной грамматикой, называется *неоднозначным*.

Рассмотрим пример неоднозначной грамматики, которая встречается едва ли не в каждом языке программирования.

Пример 20. Дана грамматика

$$S \to \textbf{if}\ b\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{if}\ b\ \textbf{then}\ S \mid a. \qquad (2.15)$$

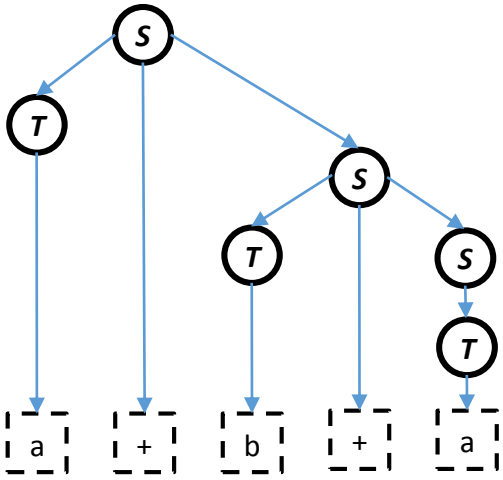

Рисунок 3. Дерево разбора для цепочки a+b+a

Данная грамматика представляет собой условный оператор. Здесь терминалами являются ключевые слова ***if***, ***then***, ***else***, а также два предиката *a* и *b*.

Покажем, что данная грамматика неоднозначна. Для этого необходимо для одной и той же цепочки построить два различных дерева вывода. Два эквивалентных дерева вывода для цепочки ***if*** *b* ***then if*** *b* ***then*** *a* ***else*** *a* показаны на рис. 4 (а) и (б).

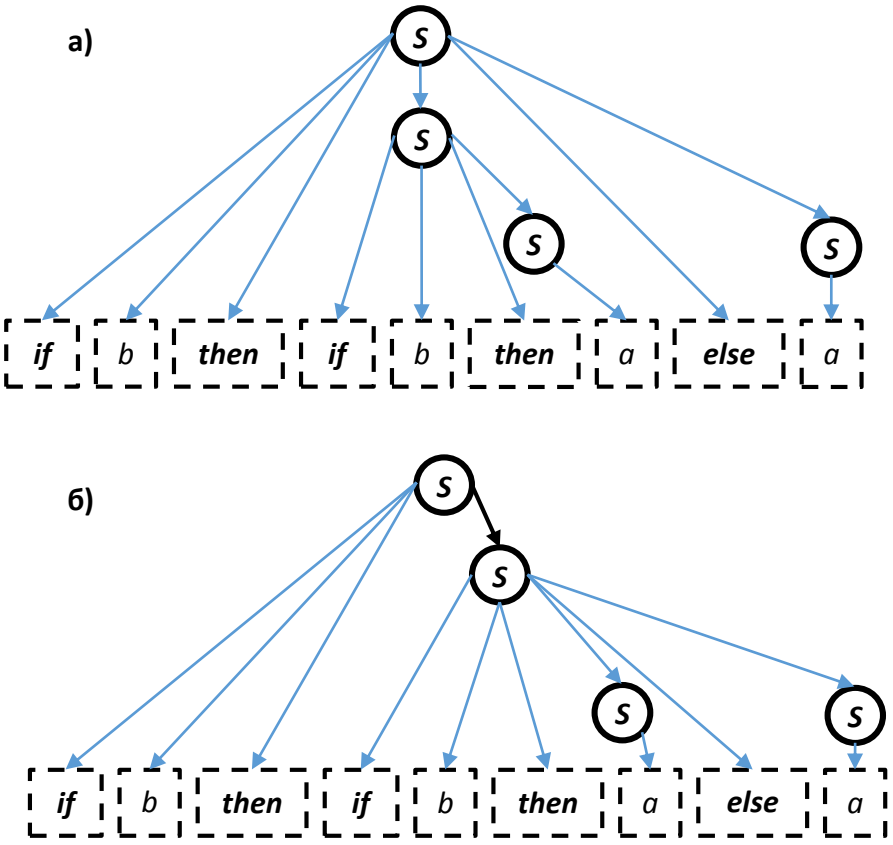

Рисунок 4. Два различных дерева вывода для строки ***if*** *b* ***then if*** *b* ***then*** *a* ***else*** *a*

Важно отметить, что в данном случае неоднозначность грамматики не говорит о неоднозначности языка. Другими словами, можно найти эквивалентную данному языку грамматику, которая будет однозначной. Естественно, что грамматики, применяемые для языков программирования, должны быть однозначными.

Для грамматики из примера 20 эквивалентной однозначной грамматикой будет следующая

$$S \to \boldsymbol{if}\ b\ \boldsymbol{then}\ S \mid \boldsymbol{if}\ b\ \boldsymbol{then}\ S'\boldsymbol{else}\ S \mid a;$$
$$S' \to \boldsymbol{if}\ b\ \boldsymbol{then}\ S'\boldsymbol{else}\ S' \mid a.$$

Заметим, что в этом, как и в большинстве прочих случаев, неоднозначность устраняется за счет введения в грамматику дополнительных нетерминалов.

Утверждение 3. Не существует алгоритма, который по заданным грамматическим правилам даст ответ на вопрос «однозначна грамматика или нет».

Для доказательства неоднозначности грамматики достаточно найти цепочку с двумя различными деревьями вывода. С другой стороны, для доказательства однозначности грамматики, необходимо доказать, что таких цепочек не существует.

Дерево вывода можно строить двумя различными способами. Первый способ отличается тем, что строит дерево, начиная с корня и следует к листьям. Данный способ построения называется *нисходящим* методом разбора. Суть метода заключается в том, чтобы для каждого нетерминала (каждой вершины дерева) найти подходящее грамматические правило вывода, которое бы проецировалось на входную цепочку.

Другой способ заключается в том, что разбор начинается с листьев и продвигается вверх, к корню дерева (цели грамматики *S*). Данный метод построения называется *восходящим* разбором и подразумевает использование *сверток*, заменяющих символы исходной цепочки на терминалы в соответствии с правилами вывода.

Каким бы ни был способ разбора, если грамматика однозначна, должно получиться единственное дерево разбора.