МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА

Факультет информатики и систем управления

Кафедра теоретической информатики и компьютерных технологий

**Лабораторная работа № 3.**

**по курсу «Теория формальных языков»**

Выполнил: Иванов Г.М.

ИУ9-52

Проверил: Магазов С.C.

«17» декабря 2016 г.

**Вариант 6.**

**Задание 1**

Задать предложенный фрагмент языка в виде грамматики не леворекурсивной нормальной грамматики:

* Определение тега XML и описание

**Задание 2**

* Если в построенной грамматике есть левая рекурсия устранить ее.
* Оптимизировать грамматику .
* Привести грамматику к форме Грейбах.

**Задание 3**

Построить вывод сверху вниз и снизу вверх.

**Задание 1**

<TAG> ::= <EMPTY\_TAG> | <START\_TAG><CONTENT><END\_TAG>

<EMPTY\_TAG> ::= **<** <NAME> **/>** | **<**<NAME> <ATTRIBUTES> **/>**

<START\_TAG> ::= **<** <NAME> **>** | **<** <NAME> <ATTRIBUTES> **>**

<END\_TAG> ::= **</** <NAME> **>**

<ATTRIBUTES> ::= <ATTR> | <ATTR><ATTRIBUTES>

<ATTR> ::= <NAME>**="**<VALUE>**"**

<CHAR> ::= a | … | z | A | … | Z | 0 | … | 9

<NAME> ::= <CHAR> | <CHAR><NAME>

<VALUE> ::= <CHAR> | <CHAR><VALUE>

<CONTENT> ::= <TAG> | <VALUE> | <TAG><CONTENT> |

<VALUE><CONTENT>

Пример правильного выражения:

<a>

b

<e f="g">

<h/>

</e>

</a>

**Задание 2.** Если в построенной грамматике есть левая рекурсия устранить ее.

Оптимизировать грамматику .

Привести грамматику к форме Грейбах

**- Левая рекурсия в правиле**

Правила, вывод которых начинается с того же нетерминала. Таких правил нет.

**- Левая рекурсия в грамматике**

Возможность получения из нетерминала цепочки, начинающейся с него же. Пример:

S → Ac|b

A → ca|Sa

В итоге можем из S получить Sac. Для удаления используем следующий алгоритм:

Пронумеруем нетерминалы: S0, A1. Проходим по всем, проверяя, что из нетерминала с большим номером не следует правило, начинающееся с нетерминала меньшего номера. Если встречаем такое, заменяем младший нетерминал его непосредственным выводом:

S → Ac|b

A → ca| Aca| ba

И удаляем полученную левую рекурсию в правиле.

Таких правил нет.

**- Недостижимые символы**

Символ, до которого не существует цепочки из начального состояния (S).

Таких символов нет.

**- Цепное правило**

Нетерминал, из которого выводится другой нетерминал. Заменяем его на вывод этого нетерминала.

Перепишем правила более подробно

<TAG> ::= <EMPTY\_TAG> //Цепное

<TAG> ::= <START\_TAG><CONTENT><END\_TAG>

<EMPTY\_TAG> ::= **<** <NAME> **/>**

<EMPTY\_TAG> ::= **<** <NAME> <ATTRIBUTES> **/>**

<START\_TAG> ::= **<** <NAME> **>**

<START\_TAG> ::= **<** <NAME> <ATTRIBUTES> **>**

<END\_TAG> ::= **</** <NAME> **>**

<ATTRIBUTES> ::= <ATTR> //Цепное

<ATTRIBUTES> ::= <ATTR><ATTRIBUTES>

<ATTR> ::= <NAME>**="**<VALUE>**"**

<CHAR> ::= a | … | z | A | … | Z | 0 | … | 9

<NAME> ::= <CHAR> //Цепное

<NAME> ::= <CHAR><NAME>

<VALUE> ::= <CHAR> //Цепное

<VALUE> ::= <CHAR><VALUE>

<CONTENT> ::= <TAG> //Цепное

<CONTENT> ::= <VALUE> //Цепное

<CONTENT> ::= <TAG><CONTENT>

<CONTENT> ::= <VALUE><CONTENT>

Заменим цепные правила на их вывод

~~<TAG> ::= <EMPTY\_TAG>~~ //Цепное

<TAG> ::= **<** <NAME> **/> | <** <NAME> <ATTRIBUTES> **/>**

<TAG> ::= <START\_TAG><CONTENT><END\_TAG>

~~<EMPTY\_TAG> ::=~~ **~~<~~** ~~<NAME>~~ **~~/>~~ //**Недостижимое правило

~~<EMPTY\_TAG> ::=~~ **~~<~~** ~~<NAME> <ATTRIBUTES>~~ **~~/>~~**

<START\_TAG> ::= **<** <NAME> **>**

<START\_TAG> ::= **<** <NAME> <ATTRIBUTES> **>**

<END\_TAG> ::= **</** <NAME> **>**

~~<ATTRIBUTES> ::= <ATTR> //Цепное~~

<ATTRIBUTES> ::= <NAME>**="**<VALUE>**"**

<ATTRIBUTES> ::= <ATTR><ATTRIBUTES>

<ATTR> ::= <NAME>**="**<VALUE>**"**

<CHAR> ::= a | … | z | A | … | Z | 0 | … | 9

~~<NAME> ::= <CHAR> //Цепное~~

<NAME> ::= a | … | z | A | … | Z | 0 | … | 9

<NAME> ::= <CHAR><NAME>

~~<VALUE> ::= <CHAR> //Цепное~~

<VALUE> ::= a | … | z | A | … | Z | 0 | … | 9

<VALUE> ::= <CHAR><VALUE>

~~<CONTENT> ::= <TAG> //Цепное~~

<CONTENT> ::= **<** <NAME> **/>** | **<** <NAME> <ATTRIBUTES> **/>** |

<START\_TAG><CONTENT><END\_TAG>

~~<CONTENT> ::= <VALUE> //Цепное~~

<CONTENT> ::= a | … | z | A | … | Z | 0 | … | 9

<CONTENT> ::= <TAG><CONTENT> | <CHAR><VALUE>

<CONTENT> ::= <VALUE><CONTENT>

**Привести грамматику к форме Грейбах.**

Грамматика в форме Грейбах:

1) Нет *е*  правил отличных S->*е*

2) *A -> a α,*  где *a —* терминал и *α —* последовательность нетерминалов

Все правила необходимо перенумеровать в порядке, где нетерминал А < B, если А=>B. Таким образом в конце списка оказываются все правила, имеющие вид *A -> a,* где  *a —* терминал.

Начиная с предпоследнего правила: если вывод начинается с нетерминала (по построению он имеют больший номер => по индукции начинается с терминала), то он заменяется на свой вывод.

Таким образом получается грамматика вида:

<TAG> ::= < <NAME> <SLASH> <BIGGER> |

< <NAME> <ATTRIBUTES> <SLASH> <BIGGER> |

< <NAME> <BIGGER> <CONTENT><END\_TAG> |

< <NAME> <ATTRIBUTES> <BIGGER> <CONTENT><END\_TAG>

<CONTENT> ::= < <NAME> <SLASH> <BIGGER> |

< <NAME> <BIGGER> <CONTENT><END\_TAG> |

< <NAME> <ATTRIBUTES> <BIGGER> <CONTENT><END\_TAG> |

a | … | z | a<VALUE> | … | z<VALUE> |

//<TAG><CONTENT>

< <NAME> <SLASH> <BIGGER><CONTENT> |

< <NAME> <ATTRIBUTES> <SLASH> <BIGGER><CONTENT> |

< <NAME> <BIGGER> <CONTENT><END\_TAG> <CONTENT> |

< <NAME> <ATTRIBUTES> <BIGGER> <CONTENT><END\_TAG> <CONTENT> |

//<VALUE> <CONTENT>

a<CONTENT>| … | z<CONTENT> |

a <VALUE><CONTENT>| … | z <VALUE><CONTENT>

// <START\_TAG> стал бесполезным

<END\_TAG> ::= < <SLASH> <NAME> <BIGGER>

<ATTRIBUTES> ::= a <EQ><QUOTE><VALUE><QUOTE> | … |

z <EQ><QUOTE><VALUE><QUOTE> |

a <NAME> <EQ><QUOTE><VALUE><QUOTE>| … |

z <NAME><EQ><QUOTE><VALUE><QUOTE> |

a <EQ><QUOTE><VALUE><QUOTE><ATTRIBUTES>| … |

z <EQ><QUOTE><VALUE><QUOTE><ATTRIBUTES>|

a <NAME><EQ><QUOTE><VALUE><QUOTE><ATTRIBUTES>| … |

z <NAME><EQ><QUOTE><VALUE><QUOTE><ATTRIBUTES>

// <ATTR> стал бесполезным

<NAME> ::= a | … | z | a <NAME>| … | z <NAME>

<VALUE> ::= a | … | z | a <VALUE>| … | z <VALUE>

<SLASH> ::= /

<BIGGER> ::= >

<CHAR> ::= a | … | z

<EQ> ::= =

<QUOTE> ::= "

# Построение вывода

Алгоритм вывода сверху вниз:

**Идея**: идём из корня S, строим дерево, выбирая каждый раз первое правило грамматики, если приходим к терминалу, который будет выведен первым, сравниваем его с первым терминалом выводимого слова. Если совпадает, то продолжаем то же самое. Если нет, то откатываемся на шаг назад и используем альтернативные правила (второе, третье...), если все правила для этого нетерминала исчерпаны — еще на шаг назад и так далее, пока не выведем слово, или не попробуем все возможные альтернативы, чтобы сказать, что вывод невозможен.

**Алгоритм**: на каждом шаге алгоритм имеет состояние (Q, n, L1, L2), где

1) Q — состояние

а) q — вывод

б) b — возврат

в) t — конец

2) n — номер символа из слова, с которым сравниваем текущий символ

3) L1 — стек с применяемыми правилами

4) L2 — стек с выведенной частью слова

1) Если первым в стеке L2 идёт нетерминал — мы раскрываем его по первому правилу, пишем результат раскрытия в L2, нетерминал переносим в L1, добавив номер правила, по которому раскрыли (в данный момент это номер один).

2) Если первым в стеке L2 идёт терминальный символ, мы сравниваем его с символом распознаваемого слова под номером n.

а) Если сравнение успешно — мы переносим терминал в L1, инкрементируем n, начинаем рассматривать следующий элемент из L2, используя предыдущие пункты 1 и 2.

б) Если сравнение неудачно — мы переходим в состояние обратного хода (меняем состояние q на b), идём с конца по L1, выкидывая в L2 терминалы (при этом декрементируя n), пока не дойдём до нетерминала.

x) Если возможно — мы применяем следующее правило для этого нетерминала (применяем альтернативу), вписывая её в L2 и меняя номер правила в L1.

A) Если после этого у нас L2 начинается с нетерминала, мы переходим в состояние прямого хода (меняем b на q) и применяем первое правило для него.

B) Если же L2 начинается с терминала, мы сравниваем его с символом распознаваемого слова под номером n.

В этом случае состояние обратного хода не переключается, пока мы не получим верное сравнение.

Если получаем — действуем по пункту 2.a

y) Если перебрали все альтернативы правила, не получив верного сравнения — откатываемся на шаг назад, выкидывая из L1 последний нетерминал и все терминалы, пока не дойдём до следующего нетерминала, и применяем для него следующее правило.

Если в конце у нас остался пустой стек L2 — слово выведено, переходим в состояние t (конец).

Если мы перебрали все возможные альтернативы из самого первого правила (S), и не смогли получить результат — слово невыводимо.

<a>

b

<e f="g">

<h/>

</e>

</a>

## Алгоритм вывода снизу вверх:

**Идея**: идём по слову, если буква или набор букв выводится из какого то правила без нетерминалов, предполагаем это правило, переходим к следующей букве, пытаемся соединить с уже имеющимся выводом, если такого правила не существует, пытаемся вывести из другого правила и потом объединить два правила и так далее, пока не приведём всё к корню S. Если буква не выводится в одиночку (или с другими терминалами) из правила, то пропускаем её, и позже пытаемся объединить с правилами, полученными из следующих символов.

**Алгоритм:** на каждом шаге (Q, n, L1, L2), где

1) Q — состояние

а) q — вывод

б) b — возврат

в) t — конец

2) n — номер рассматриваемого символа из слова

3) L1 — стек с выведенной частью слова

4) L2 — стек с применяемыми правилами

(q, 1, $, e) — начальное состояние

1) Пишем в L1 следующую букву распознаваемого слова. Инкрементируем n, пишем в L2 «-»

2) Ищем правило, из которого выводится то, что находится в L1, или часть L1, обязательно содержащая его конец (т.е. можно пропустить часть начальных символов)

a) Если такое правило нашлось, в L1 пишем соответствующий нетерминал, удаляя его непосредственный вывод, в L2 пишем номер использованного правила, возвращаемся к пункту 2.

б) Если не нашлось, то переходим к пункту 1.

в) Если буквы в слове кончились, и перейти к пункту 1 мы не можем, то включаем состояние отката (меняем q на b).

X) Ищем другой вариант свёртки — отменяем последнее применённое правило, удаляем его номер из L2 и возвращаем вместо него непосредственный вывод в L1, после чего переходим к пункту 2.

Y) Удаление буквы — если все варианты свёртки для L1 исчерпаны — откидываем последнюю добавленную букву из L1, удаляем «-» из L2, декрементируем n, переходим к пункту X.

Если в конце у нас в L1 остаётся $S и n соответствует длине слова + 1 - слово выведено, удаляем из L2 все «-», переходим в состояние t (конец).

Грамматика должна быть приведенной, то есть не содержать циклов, цепных правил, бесполезных (в том числе недостижимых) символов (получена в результате выполнения задания 2)

1) <TAG> ::= < <NAME> />

2) <TAG> ::= < <NAME> <ATTRIBUTES> />

3) <TAG> ::= <START\_TAG><CONTENT><END\_TAG>

4) <START\_TAG> ::= < <NAME> >

5) <START\_TAG> ::= < <NAME> <ATTRIBUTES> >

6) <END\_TAG> ::= </ <NAME> >

7) <ATTRIBUTES> ::= <NAME>="<VALUE>"

9) <ATTRIBUTES> ::= <ATTR><ATTRIBUTES>

10) <ATTR> ::= <NAME>="<VALUE>"

11) <NAME> ::= a | … | z | A | … | Z | 0 | … | 9

12) <NAME> ::= <CHAR><NAME>

13) <VALUE> ::= a | … | z | A | … | Z | 0 | … | 9

14) <VALUE> ::= <CHAR><VALUE>

15) <CONTENT> ::= < <NAME> />

16) <CONTENT> ::= <START\_TAG><CONTENT><END\_TAG>

17) <CONTENT> ::= a | … | z | A | … | Z | 0 | … | 9

18) <CONTENT> ::= <CHAR><VALUE>

19) <CONTENT> ::= <TAG><CONTENT>

20) <CONTENT> ::= <VALUE><CONTENT>

21) <CHAR> ::= a | … | z | A | … | Z | 0 | … | 9