Московский авиационный институт
(национальный исследовательский университет)

Факультет «Информационные технологии и прикладная математика»

**Курсовой проект**
**по курсу**

**«Системное программное обеспечение»**

**IV семестр**

# «Анализ леммы о Накачке Контекстно-свободного языка и леммы о Накачке Регулярного языка »

*Студент:* Дюсекеев А.Е.
*Группа:* М8О-204Б-17

*Руководитель:* Семёнов А. С.

*Оценка:*
*Дата:*

**Москва, 2019**

**Лемма накачки для контекстно-свободных языков**

Теорема. Для каждого КС языка L, размер которого не является конечным, найдется такое натуральное число p, что для каждого z ∈ L такое, что |z| ≥ p, существуют такие u, v, w, x, y ∈ ∑ *, что

z = uvwxy с |vwx| ≤ p и |vx| ≥ 1 такой, что

для всех i ≥ 0, uviwxiy ∈ L.

Эта лемма может быть переписана, как показано ниже, чтобы показать логический контекст. Обратите внимание, что в этой лемме p играет ту же роль, что и n в лемме накачки для обычных языков, а v и x, «место накачки», могут быть где угодно в z. Напомним, что «место накачки» v в лемме о накачке на обычном языке может находиться только в префиксе z длины n.

(1) Для всех контекстно-свободных языков L,

(2) существует постоянная p такая, что

(3) для всех z ∈ L таких, что | z |≥ p,

(4) существуют такие u, v, w, x, y и z, что

(i) z = uvwxy,

(ii) |uwx| ≤ p,

(iii) |ux| ≥ 1 и

(iv) для всех i ≥ 0, uviwxiy ∈ L

**Лемма накачки для регулярных языков**

В теории формальных языков, **лемма о накачке для регулярных языков** описывает существенное свойство всех регулярных языков. Неформально она утверждает, что все достаточно длинные слова регулярного языка можно *накачать*, то есть повторить внутреннюю часть слова сколько угодно раз, производя новое слово, также принадлежащее языку.

**Доказательство леммы накачки для контекстно-свободных языков**

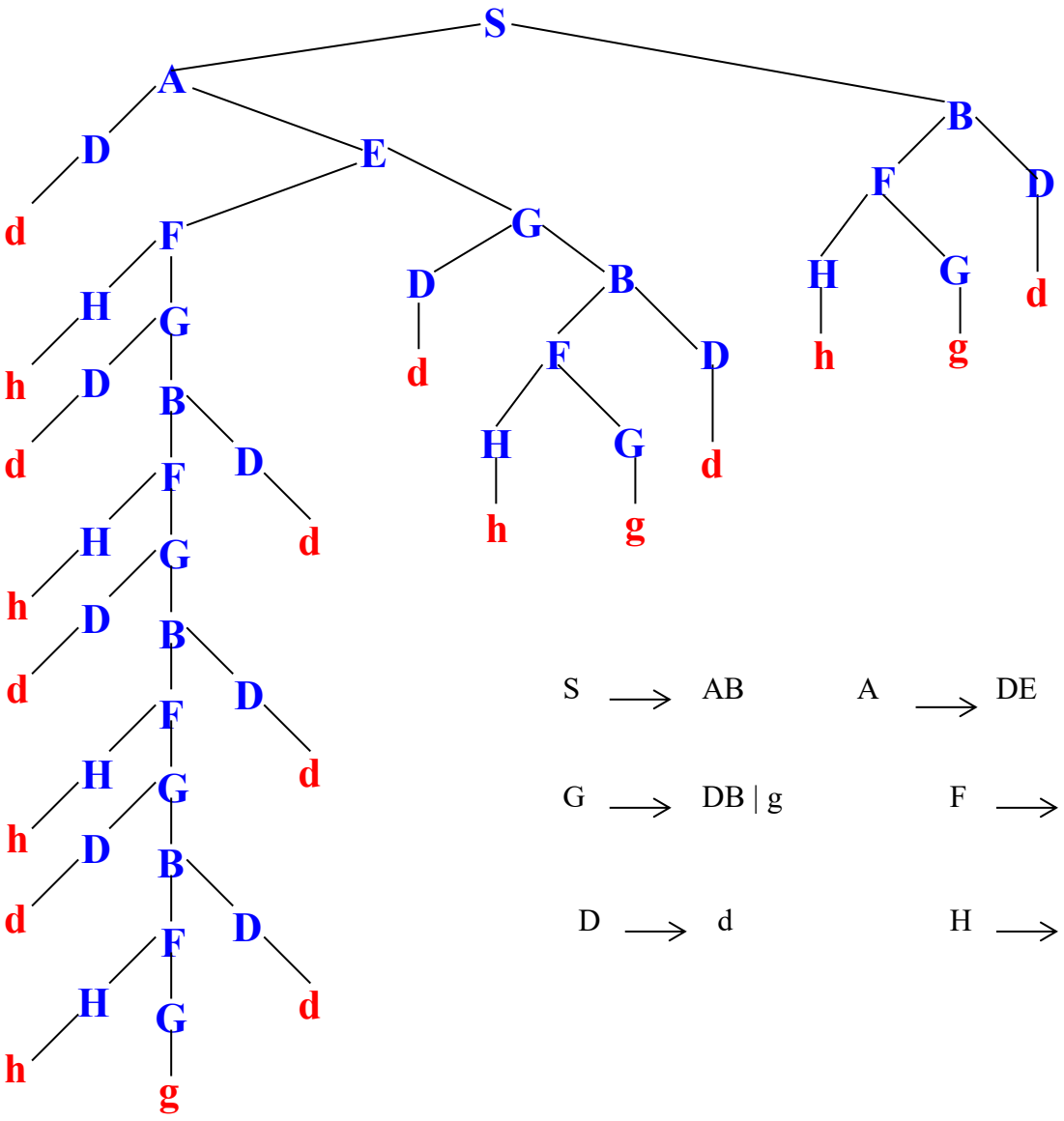
Доказательство.

Рассмотрим следующий КС языка в нормальной форме Хомского.

$S \to AB$ $A \to DE$ $B \to FD$ $E \to FG$

$G \to DB \mid g$ $F \to HG$ $D \to d$ $H \to h$

Дерево вывода для строки $z = d\ (h\ d)^3\ h\ g\ (d)^3\ d\ g\ h\ d\ g\ h\ d$ приведено на следующей странице. Обратите внимание на паттерн нетерминальной последовательности FGB-FGB-FGB, которая встречается на самом длинном стволе дерева синтаксического анализа. Он имеет повторяющийся шаблон FGB, который имеет листья “hd” слева и “d” справа. Легко видеть, что мы можем “вырастить” стебель, добавив столько повторяющихся шаблонов F-G-B, сколько мы хотим, или сократить его, удалив шаблон. Полученное дерево по-прежнему является деревом синтаксического анализа грамматики. Из этого следует, что грамматика имеет дерево разбора для $z' = d\ (h\ d)^i\ h\ g\ (d)^i\ d\ g\ h\ d\ g\ h\ d$, для всех $i \geq 0$, что означает, что z' также находится на языке грамматики.




S → AB  A → DE  E → FG

G → DB | g  F → HG

D → d  H → h

$Z = d\,(h\,d)^3\,h\,g\,(d)^3\,d\,h\,g\,d\,h\,g\,d$

u v w x y

uvwxy

uvvwxx

uvvvwxxxy

# Реализация

Класс **UViWXiY** содержит поля **U,V,i,W,X,Y** соответствующие представлению строки $Z = UV^iWX^iY$

Класс **PumpingLema** следующие поля и методы:

- **private string Z** - цепочка символов, над которой будет производиться разбор
- **private List< UViWXiY > ResultZ** - результат разбора Z, список структур, в которых содержится результат разбиения **Z** = **UViWXiY** при максимальном возможном значении i
- **private List<string> WereChecked** - список подцепочек **Z**, для которых лема была проверена
- **private void FindV()** - для цепочки Z перебираются все возможные подцепочки, которые принимаются за V
- **private void FindVi(string V)** - для заданной подцепочки проверяется максимальное количество её непрерывных повторений. Если получается превысить полученное до этого число повторений i, результаты разбиения строки **UViWXiY**, результат заменяет предидущий.
- **private void FindX()** - для цепочки Z перебираются все возможные подцепочки, которые принимаются за X
- **private void FindXi(string X)** - для заданной подцепочки проверяется максимальное количество её непрерывных повторений. Если получается превысить полученное до этого число повторений i, результаты разбиения строки **UViWXiY**, результат заменяет предидущий.
- **public void PrintResult()** - выводит на консоль результаты проверки

Цепочки, которые принимаются за  V and X выбираются таким образом: во внешнем цикле изменяется длинна, от 1 до половины длинны Z (если длинна V and X превысит половину длины  Z, V and X не встретится более 1 раза), во внутреннем цикле выбирается p из Z, в котором будет начинаться получения подцепочка V and X длинны Length V and LengthX соответственно.

```
for (int LengthV = 1; LengthV <= Z.Length / 2; LengthV++)
{
     for (int p = 0; p < Z.Length - LengthV; ++p)
     {
          this.FindVi(Z.Substring(p, LengthV));
```

```
    }
}
for (int LengthX = 1; LengthX <= Z.Length / 2; LengthX++)
{
    for (int p = 0; p < Z.Length - LengthX; ++p)
    {
        this.FindXi(Z.Substring(p, LengthX));
    }
}
```

Для каждой выбранной подцепочки проверяется максимальное количество её непрерывных повторений.
В начале проверки проверяется, была ли просмотрена подпоследовательность, выбранная в качестве V. Если не была, то заносится в список **WereChecked**, алгоритм продолжается. Цепочку Z можно покрыть V .Lenght способами. Например Для V Длинной 3 Z можно покрыть 3 способами:
Для Каждого из тих способов не сложно подсчитать самое большое количество непрерывных повторений, при этом необходимо запоминать позицию начала соответствующей последовательности повторений. После прохода получившееся максимальное количество повторений сравнивается с предыдущим полученым, и если оно превосходит, предидущие результаты стираются, в стек помещаются новые. Если одному покрытию соответствует несколько одинаковых по количеству повторений последовательностей, в стек добавляются все эти варианты. После завершения проверки для каждого покрытия, в стеке будет содержаться информация разбиении $Z = UV^iWX^iY$, для заданой подцепочки V и максимально возможной i, если таких разбиений несколько - тоже. Результат из стека переносится в **ResultZ** только если значение i больше 1 и не меньше найденого ранее при другом выборе V. Аналогично приводится пример с подцепочкой X

# Тестирование

Я подготовил тесты с неоднозначным разбиением, программа корректно выявила разбиения разбиения строки $Z = UV^iWX^iY$ с наибольшим возможным значением i.

```
abcdefedcba

aaaaaaa
 a^7
ddddddd
 d^7
aaabaaadaaa
aaabaaad a^3
```

```
aaab a^3 daaa
 a^3 baaadaaa
d^3

abbbbc
a b^4 c

abbbcccddde
a b^3 cccddde
abbb c^3 ddde
abbbccc d^3 e

abcdabcdabcdabcd
 abcd^4

aaabcdabcdabcdabcddaab
aa abcd^4 daab

aaabcdbcdbcdaaabcdbcdbcdaaabcdbcdbcd
aaabcdbcdbcdaaabcdbcdbcd a^3 bcdbcdbcd
aaabcdbcdbcd a^3 bcdbcdbcdaaabcdbcdbcd
 a^3 bcdbcdbcdaaabcdbc
d^3 dbcdaaabcdbcdbcd
aaabcdbc d^3
dbcdaaabcdbcdbcdaaa bcd^3
aaabcdbcdbcdaaa bcd^3 aaabcdbcdbcd
aaa bcd^3 aaabcdbcd d^3 dcdaaabcdbcdbcd
 aaabcdbcdbcd^3

abbbbabbbbabbbbabbbb
abbbbabb d^3 bbabbbba b^4
abbbbabbbba b^4 abbbb
abbbba b^4 abbbbabbbb
a b^4 abbbbab d^3 bbbabbbb
 abbbb^4


abcdbcdbcdbcdbcd
a bcd^5

abcbcbcbd
a bc^3 bd
ab cb^3 d d^3 cd
```