

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

# ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника Кафедра Кибербезопасность информационных систем

# Практическое занятие № 1

на тему «Распознавание типов формальных языков и грамматик»

Выполнил обучающийся гр. ВКБ42

Приходько А. Р.

(Фамилия, Имя, Отчество)

Проверил:

доц. Пиневич Е. В.

(должность, Фамилия, Имя, Отчество)

Ростов-на-Дону

2023

**Цель:**

– закрепить понятия «алфавит», «цепочка», «формальная грамматика» и «формальный язык», «выводимость цепочек», «эквивалентная грамматика»;

– сформировать умения и навыки распознавания типов формальных языков и грамматик по классификации Хомского, построения эквивалентных грамматик.

**Задание 1**

1) Язык

**Задание 2**

Является грамматикой типа 0 (неограниченной грамматикой)

Является неукорачивающей грамматикой

Является контекстно-зависимой грамматикой

Является контекстно-свободной грамматикой

Является укорачивающей контекстно-свободной грамматикой

Не является регулярной грамматикой

**Задание 3**

def is\_unlimited\_grammar(grammar):

    for rule in grammar:

        if '->' not in rule:

            return 'Не является грамматикой типа 0 (неограниченной грамматикой)'

    return 'Является грамматикой типа 0 (неограниченной грамматикой)'

def is\_non\_shortening\_grammar(grammar):

    for rule in grammar:

        rule = rule.replace(' ', '')

        left\_part = rule.split('->')[0]

        right\_part = rule.split('->')[1]

        if ('ε' in list(left\_part) or 'ε' in list(right\_part)):

            return 'Не является неукорачивающей грамматикой'

        if (len(left\_part) <= len(right\_part) and len(left\_part) >= 1):

            pass

        else:

            return 'Не является неукорачивающей грамматикой'

    return 'Является неукорачивающей грамматикой'

def is\_context\_sensitive\_grammar(grammar):

    for rule in grammar:

        rule\_left\_part = rule.split('->')[0]

        if not any(char.isupper() for char in rule\_left\_part):

            return 'Не является контекстно-зависимой грамматикой'

        rule\_right\_part = rule.split('->')[1]

        index\_of\_upper\_char\_left = 0

        for i in range(len(rule\_left\_part)):

            if (rule\_left\_part[i].isupper()):

                index\_of\_upper\_char\_left = i

                break

        xi\_left = rule\_left\_part[:index\_of\_upper\_char\_left]

        index\_of\_upper\_char\_right = 0

        for i in range(len(rule\_left\_part)-1, 0, -1):

            if (rule\_left\_part[i].isupper()):

                index\_of\_upper\_char\_right = i

                break

        xi\_right = rule\_left\_part[index\_of\_upper\_char\_right+1:]

        # если в α есть терминал

        if any(char.islower() for char in rule\_left\_part[index\_of\_upper\_char\_left:index\_of\_upper\_char\_right+1]):

            return 'Не является контекстно-зависимой грамматикой'

        equality\_xi\_left = True

        if not (len(xi\_left) == 0):

            # сравниваем ξ1 и ξ1 в левой и правой частях правила

            equality\_xi\_left = rule\_right\_part[:len(xi\_left)] == xi\_left

        equality\_xi\_right = True

        if not (len(xi\_right) == 0):

            # сравниваем ξ2 и ξ2 в левой и правой частях правила

            equality\_xi\_right = rule\_right\_part[-len(xi\_right):] == xi\_right

        # проверяем равна ли γ (гамма) пустому значению

        len\_gamma\_more\_than\_one = len(xi\_left) + len(xi\_right) < len(rule\_right\_part)

        if ('ε' in list(rule\_left\_part) or 'ε' in list(rule\_right\_part)):

            return 'Не является контекстно-зависимой грамматикой'

        if not (equality\_xi\_left and equality\_xi\_right and len\_gamma\_more\_than\_one):

            return 'Не является контекстно-зависимой грамматикой'

    return 'Является контекстно-зависимой грамматикой'

def is\_context\_free\_grammar(grammar):

    for rule in grammar:

        rule\_left\_part = rule.split('->')[0]

        if any(char.islower() for char in rule\_left\_part):

            return 'Не является контекстно-свободной грамматикой'

        rule\_right\_part = rule.split('->')[1]

        if (rule\_right\_part == 'ε'):

            return 'Не является контекстно-свободной грамматикой'

    return 'Является контекстно-свободной грамматикой'

def is\_shortening\_context\_free\_grammar(grammar):

    for rule in grammar:

        rule\_left\_part = rule.split('->')[0]

        if any(char.islower() for char in rule\_left\_part):

            return 'Не является укорачивающей контекстно-свободной грамматикой'

    return 'Является укорачивающей контекстно-свободной грамматикой'

def is\_reg\_grammar(grammar):

    left\_flag = False

    for rule in grammar:

        rule\_left\_part = rule.split('->')[0]

        rule\_right\_part = rule.split('->')[1]

        if (rule\_right\_part == 'ε'):

            return 'Не является регулярной грамматикой'

        if any(char.islower() for char in rule\_left\_part):

            return 'Не является регулярной грамматикой'

        if any(char.isupper() for char in rule\_right\_part):

            index\_of\_upper\_char\_right = 0

            for i in range(len(rule\_right\_part)-1, 0, -1):

                if (rule\_right\_part[i].isupper()):

                    index\_of\_upper\_char\_right = i

                    break

            index\_of\_upper\_char\_left = 0

            for i in range(len(rule\_right\_part)):

                if (rule\_right\_part[i].isupper()):

                    index\_of\_upper\_char\_left = i

                    break

            if any(char.islower() for char in rule\_right\_part[index\_of\_upper\_char\_left:index\_of\_upper\_char\_right+1]):

                return 'Не является регулярной грамматикой'

            left\_part = rule\_right\_part[:index\_of\_upper\_char\_left]

            right\_part = rule\_right\_part[index\_of\_upper\_char\_right+1:]

            if (len(left\_part) > 0 and len(right\_part) > 0):

                return 'Не является регулярной грамматикой'

            if (len(left\_part) > 0):

                left\_flag = True

    if left\_flag:

        return 'Является леволинейной грамматикой'

    else:

        return 'Является праволинейной грамматикой'

grammar = ['S->aaAab', 'A->aAbb', 'A->aaab']

print(is\_unlimited\_grammar(grammar))

print(is\_non\_shortening\_grammar(grammar))

print(is\_context\_sensitive\_grammar(grammar))

print(is\_context\_free\_grammar(grammar))

print(is\_shortening\_context\_free\_grammar(grammar))

print(is\_reg\_grammar(grammar))

# Является грамматикой типа 0 (неограниченной грамматикой)

# Является неукорачивающей грамматикой

# Является контекстно-зависимой грамматикой

# Является контекстно-свободной грамматикой

# Является укорачивающей контекстно-свободной грамматикой

# Не является грамматикой типа 3

**Вывод:**

– закрепили понятия «алфавит», «цепочка», «формальная грамматика» и «формальный язык», «выводимость цепочек», «эквивалентная грамматика»;

– сформировали умения и навыки распознавания типов формальных языков и грамматик по классификации Хомского, построения эквивалентных грамматик.