Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №3

Регулярные выражения, грамматики и языки
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 _____ Е.В. Железкин
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

1 Цель работы

Исследование автоматов с магазинной памятью, контекстно-свободных грамматик и свойств контекстно-свободных языков, а также доказательство принадлежности языков к классу контекстно-свободных.

2 Задача работы

Часть 1. Необходимо с использованием системы JFLAP, построить МПА, предназначенный для распознавания заданного языка, либо формально доказать невозможность этого. Если не оговорено особо, то алфавитом является набор {a, b, c}. Запись n_s(w) означает количество символов s в цепочке w. Предложить программную реализацию МПА.

Часть 2. Необходимо с использованием системы JFLAP, построить контекстно-свободную грамматику, описывающую заданный язык, который может быть распознан алгоритмом перебора или управляемым пользователем, или формально доказать невозможность этого.

Часть 3. Необходимо доказать контекстно-свободность либо ее отсутствие для предложенных системой JFLAP языков с применением леммы о разрастании контекстно-свободных языков. Привести пошаговое выполнение доказательства.

Часть 4. Доказать формально контекстно-свободность либо ее отсутствие заданных языков. Для доказательства рекомендуется использовать лемму о разрастании контекстно-свободных языков.

Вариант (16, 16, 2, 16)

1) Язык L_{16} = $\{a^n b^m c^m : m, n \geq 0\}$

2) Язык L_{32} = $\{uvwv^r$: u, v, w принадлежат $\{a, b\}^+$, |u| = |v| = 2$\}$.

3) $L = \{ww : w \in \{a, b\}^*\}$

4) Язык $L_{48} = \{a^n b^j a^k b^l : n \leq k, j \leq l\}$ на алфавите {a, b}

2.1 Инструкция по запуску

Необходимо установить *python*, желательно версии 3 и выше (выполнено на версии 3.8.2):

- Страница загрузки для Windows: https://www.python.org/downloads/
- Для Linux есть несколько способов, один из них инструмент apt-get:

  $ sudo apt-get update

  $ sudo apt-get install python3.8
- Или загрузить, распаковать и установить образ:

  $ wget https://www.python.org/ftp/python/3.8.2/Python-3.8.2.tgz

  $ tar -xvf Python-3.8.2.tgz

  Для следующего шага понадобится компилятор gcc, но, думаю, это не проблема. Переходим в распакованную папку и собираем+устанавливаем:

  $ cd Python-3.7.1

  $ ./configure

  $ make

  $ sudo make install

(способы не проверялись на практике)

Далее на любой из двух систем перейти в каталог с распакованным архивом Lab_3 и выполнить:

$ python PyPDA/main.py

($ python main.py; если из папки *PyPDA)*

Ввести тестовую цепочку, нажать «ввод»

3 Ход работы

Часть 1 – МПА

Реализованный МПА допускает входную строку по заключительному состоянию q_3. К слову, стек в данном состоянии всегда равняется начальному магазинному символу Z (если входная строка пустая) или остаётся пустым (во всех прочих случаях).

На рисунке 2 приведён более совершенный МПА, магазин которого в заключительном состоянии q_3 всегда пуст. Он является допускающим и по состоянию, и по пустому магазину. (проверено на тех же тестовых цепочках)

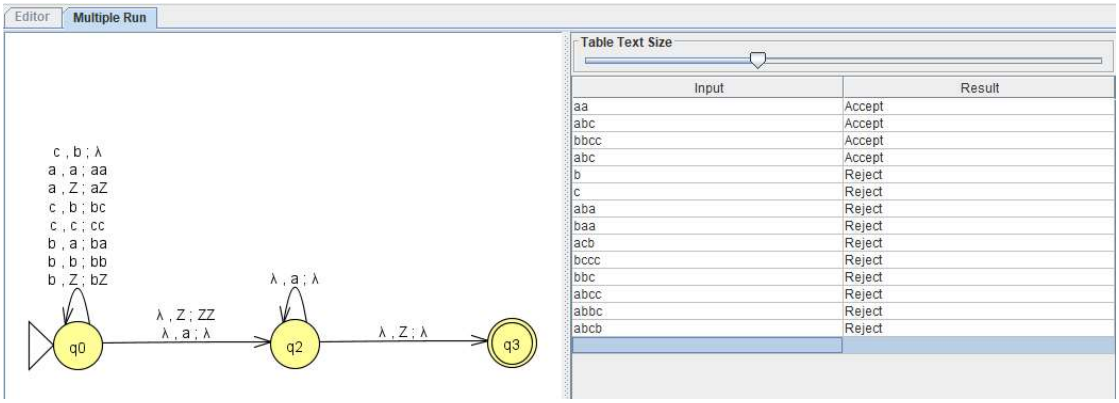

Рисунок 1 – Полученный МПА

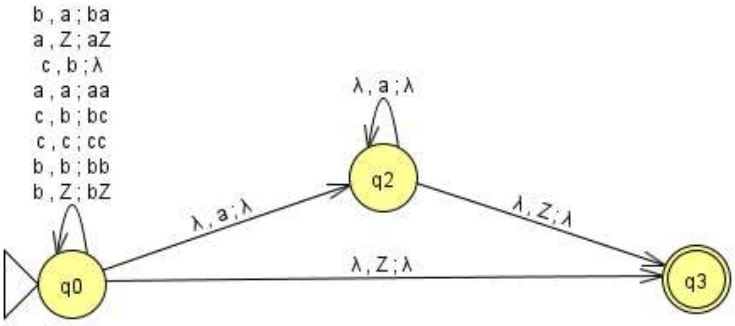

Рисунок 2 – Оптимизированный МПА

```
    try:
        if (state, current_symbol, top) in self._sigma:
            transitions = self._sigma[(state, current_symbol, top)]
            for temp in transitions:
                print('δ(' + str(cur_state.state) +
                      str(', ' + current_symbol + ', ').replace(', ,', ', λ,') +
                      top + str(cur_state.stack) + ') -> (' + str(temp[0]) +
                      str(', ' + str(temp[1]) + str(cur_state.stack) + ')').replace(', )', ', λ)'))

        # if (state, '', top) in self._sigma:
        #     empty_transitions = self._sigma[(state, '', top)]
        #     for temp in empty_transitions:
        #         print('δ(' + str(cur_state.state) + ', ' + 'e' + ', ' + top + str(cur_state.stack) +
        #               ') -> (' + str(temp[0]) + ', ' + str(temp[1]) + str(cur_state.stack) + ')')

        for transition in transitions:
            # δ(state, input, pop_stack) -> (new_state, push_stack)
            # tuple( , ); transition(state, stack); transition(0, 1)
            new_step.append(NewState(transition[0], transition[1] + cur_state.stack))

        # for transition in empty_transitions:
        #     new_step.append(NewState(transition[0], transition[1] + cur_state.stack))
    except Exception as e:
        # Чтобы IDE не ругалась
        print(e)

print()
```

Рисунок 3 – Отключён вывод для бесполезных λ переходов

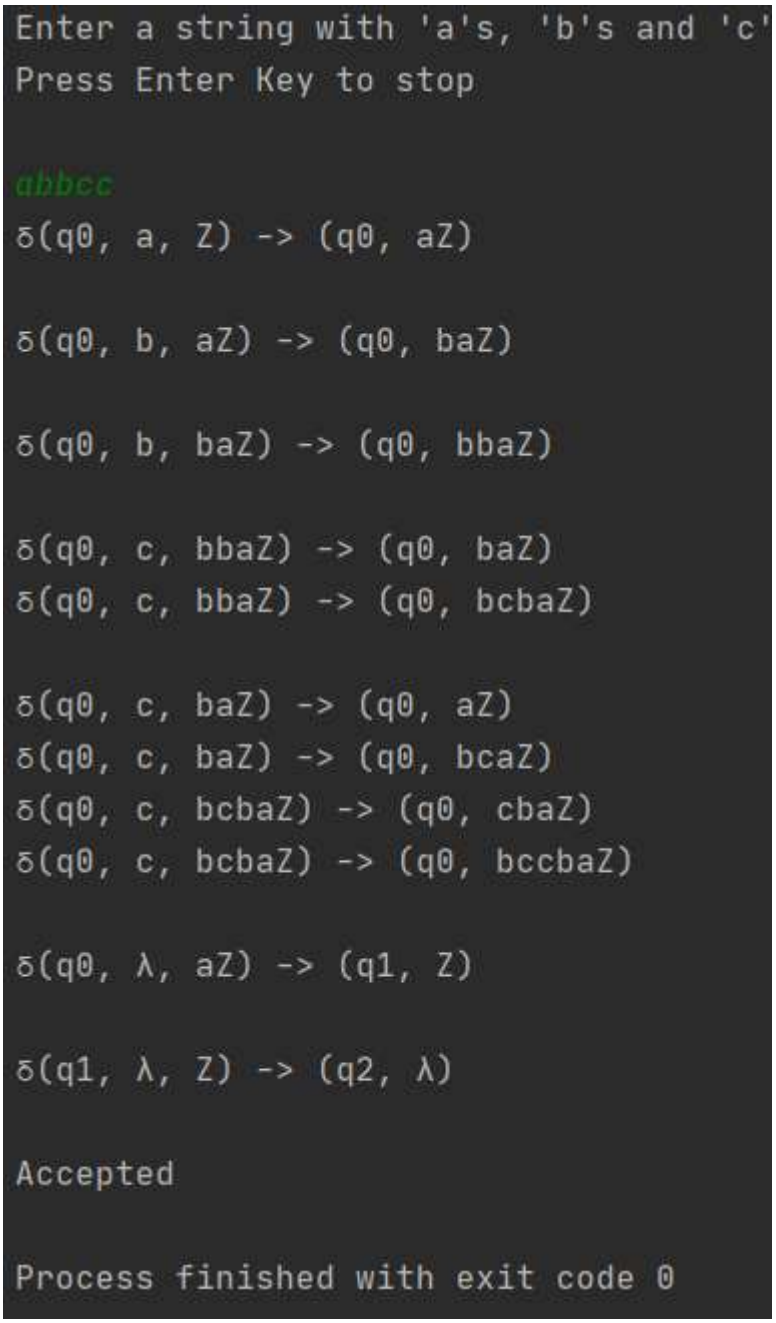

Рисунок 4 – Результат выполнения 1

```
Enter a string with 'a's, 'b's and 'c's:
Press Enter Key to stop


δ(q0, λ, Z) -> (q1, ZZ)

δ(q1, λ, ZZ) -> (q2, Z)

Accepted

Process finished with exit code 0
```

Рисунок 5 – Результат выполнения 2

```
Enter a string with 'a's, 'b's and 'c's:
Press Enter Key to stop

aaa
δ(q0, a, Z) -> (q0, aZ)

δ(q0, a, aZ) -> (q0, aaZ)

δ(q0, a, aaZ) -> (q0, aaaZ)

δ(q0, λ, aaaZ) -> (q1, aaZ)

δ(q1, λ, aaZ) -> (q1, aZ)

δ(q1, λ, aZ) -> (q1, Z)

δ(q1, λ, Z) -> (q2, λ)

Accepted

Process finished with exit code 0
```

Рисунок 6 – Результат выполнения 3

```
Enter a string with 'a's, 'b's and 'c's:
Press Enter Key to stop

abcc
δ(q0, a, Z) -> (q0, aZ)

δ(q0, b, aZ) -> (q0, baZ)

δ(q0, c, baZ) -> (q0, aZ)
δ(q0, c, baZ) -> (q0, bcaZ)

δ(q0, c, bcaZ) -> (q0, caZ)
δ(q0, c, bcaZ) -> (q0, bccaZ)


Rejected

Process finished with exit code 0
```

Рисунок 7 – Результат выполнения 4

Часть 2 - КСГ

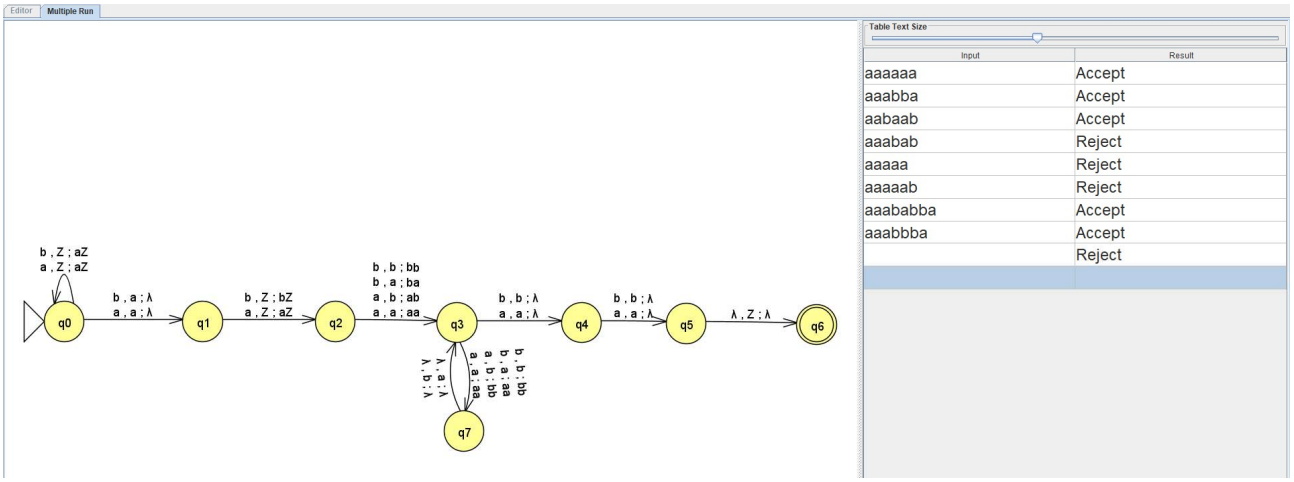

Рисунок 8 – Полученный МПА

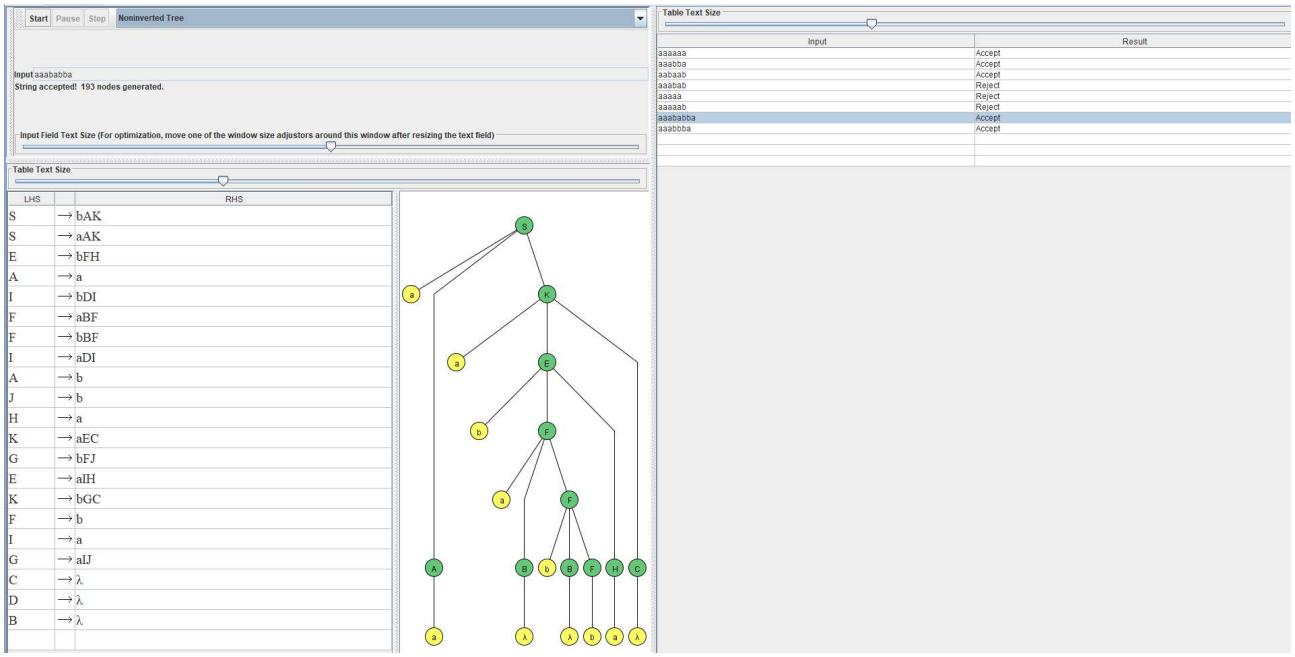

Рисунок 9 – КСГ, полученная по определению (в результате конвертации МПА)

Часть 3 – применение леммы о возрастании КСЯ

$$L = \{ww : w \in \{a, b\}^*\}$$

Лемма: пусть L — контекстно-свободный язык над алфавитом Σ, тогда существует такое n, что для любого слова ω ∈ L длины не меньше n найдутся слова u, v, x, y, z ∈ Σ^*, для которых верно:

1) uvxyz = ω
2) vy ≠ ε
3) |vxy| ⩽ n
4) ∀k ⩾ 0 uv^kxy^kz ∈ L

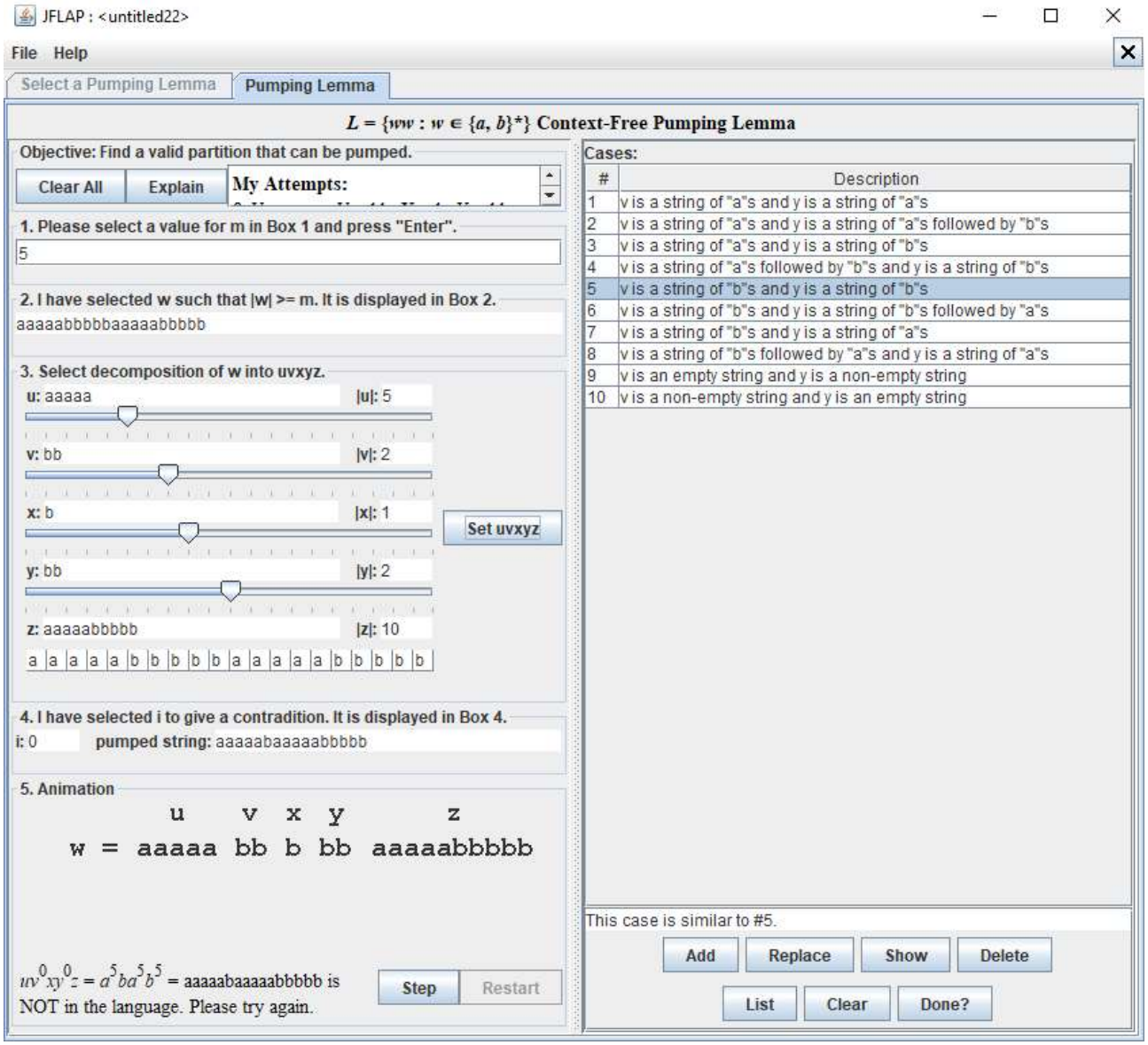

Рисунок 10 – Невыполнение леммы о разрастании для $3 \leq m \leq 6$ => язык, не является контекстно-свободным

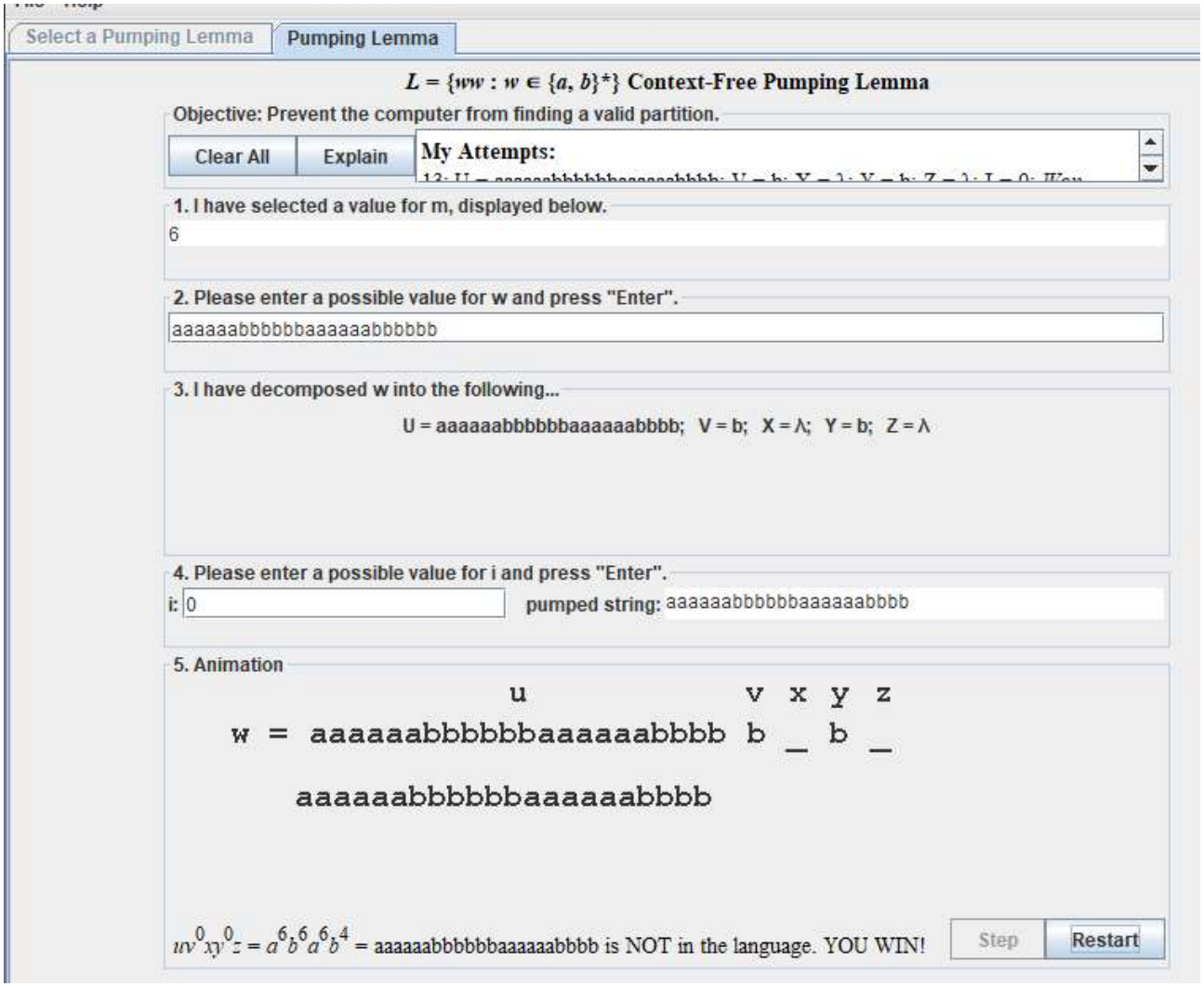

Рисунок 11 – Обратная «партия»

Unfortunately no valid partition of *w* exists.
For any *m* value, a possible value for *w* is "$a^m b^m a^m b^m$". To be in the language with this example, *v* & *y* together cannot possess identical letters that are from separate blocks of alike letters (ex: *v* has "b"s from the first set of "b"s, while *y* has "b"s from the second set of "b"s. Because of this, any increase or decrease in "a"s or "b"s will not be matched by any corresponding change in the other blocks of similar letters, resulting in an inequality that prevents the decomposition from working. Thus, this language is not context-free.

Часть 4 – Формальное доказательство нерегулярности

$L_{48} = \{a^n b^j a^k b^l : n \leq k, j \leq l\}$ на алфавите {a, b}

Вновь обратимся к лемме о разрастании КСЯ: пусть L — контекстно-свободный язык над алфавитом Σ, тогда существует такое n, что для любого

слова ω ∈ L длины не меньше n найдутся слова u, v, x, y, z ∈ Σ^*, для которых верно:

1) uvxyz = ω
2) vy ≠ ε
3) |vxy| ⩽ n
4) ∀k ⩾ 0 $uv^kxy^kz \in$ L

Если L_{48} – КСЯ, то для него выполняется описанная выше лемма. Рассмотрим частный случай:

n = 8, ω = aa**a**bb**b**aaaabbbb

u = aa; v = a; x = bb; y = b; z = aaaabbbb

Проверим условия:

- vy = ab ≠ ε
- |vxy| = 4 ⩽ n = 8
- k = 3 (∀k ⩾ 0) => ω = uv^3xy^3z = aaaaabbbbbaaaabbbb – не выполняется (слово не принадлежит L_{48}) => L_{48} не является КСЯ.

4 Вывод

В ходе данной лабораторной работы были исследованы автоматы с магазинной памятью, контекстно-свободные грамматики и свойства контекстно-свободных языков, а также приведено доказательство принадлежности языков к классу контекстно-свободных.

ПРИЛОЖЕНИЕ А

Листинг 1 – файл main.py

```
UNKNOWN_SYMBOL_ERR = 0
NOT_REACHED_FINAL_STATE = 1
REACHED_FINAL_STATE = 2


class NewState:
    state = ''
    stack = ''

    def __init__(self, cur_state, cur_stack):
        self.state = cur_state
        self.stack = cur_stack


class PDA:
    _states = []
    _inputs = []
    _pd_inputs = []
    _sigma = None
    _start_state = ''
    _start_pd_character = ''
    _final_states = []
    _progress = []

    def __init__(self, states: list, inputs: list, pd_inputs:
list,
                 transitions: dict, start_state: str,
start_pd_character: str, final_states: list):
        self._states = states
        self.total_states = len(states)

        self._inputs = inputs
        self._alphabet_characters = len(inputs)

        self._pd_inputs = pd_inputs
        self._pd_alphabet_characters = len(pd_inputs)

        self._sigma = transitions

        self._start_state = start_state

        self._start_pd_character = start_pd_character

        self._final_states = final_states
        self._total_final_states = len(final_states)
```

```
        self._progress.append(NewState(start_state,
start_pd_character))

    def makeStep(self, current_symbol: chr) -> int:
        if current_symbol not in self._inputs and current_symbol
!= '':
            return UNKNOWN_SYMBOL_ERR

        new_step = []
        for cur_state in self._progress:
            print(str(cur_state.state) + ' ' +
str(cur_state.stack))
        print()

        for cur_state in self._progress:
            state = cur_state.state
            top = cur_state.stack[0]
            cur_state.stack = cur_state.stack[1:]

            try:
                transitions = self._sigma[(state, current_symbol,
top)]

                for transition in transitions:
                    # δ(state, input, pop_stack) -> (new_state,
push_stack)
                    # tuple( , ); transition(state, stack);
transition(0, 1)
                    new_step.append(NewState(transition[0],
transition[1] + cur_state.stack))
            except Exception as e:
                # Чтобы IDE не ругалась
                temp = e

        self._progress = new_step

        for variable in self._progress:
            if (variable.state in self._final_states and
                    (variable.stack == '' or variable.stack ==
self._start_pd_character)):
                return REACHED_FINAL_STATE

        return NOT_REACHED_FINAL_STATE

    def count_of_transitions_left(self):
        return len(self._progress)


if __name__ == "__main__":
    result = NOT_REACHED_FINAL_STATE
```

```
    # PDA description
    cur_pda = PDA(states=['q0', 'q1', 'q2'],
                  inputs=['a', 'b', 'c'],
                  pd_inputs=['a', 'b', 'c', 'Z'],

                  # δ(q, a, X) -> (p, γ)
                  # δ(state, input, pop_stack) -> (new_state,
push_stack)
                  transitions={('q0', 'a', 'a'): [('q0', 'aa'),
('q1', '')],
                               ('q0', 'a', 'Z'): [('q0', 'aZ'),
('q1', 'ZZ')],
                               ('q0', 'b', 'a'): [('q0', 'ba')],
                               ('q0', 'b', 'b'): [('q0', 'bb')],
                               ('q0', 'b', 'Z'): [('q0', 'bZ')],
                               ('q0', 'c', 'b'): [('q0', ''),
('q0', 'bc')],
                               ('q0', 'c', 'c'): [('q0', 'cc')],
                               ('q0', '', 'Z'): [('q1', 'ZZ')],
                               ('q0', '', 'a'): [('q1', '')],
                               ('q1', '', 'a'): [('q1', '')],
                               ('q1', '', 'Z'): [('q2', '')],
                              },
                  start_state='q0',
                  start_pd_character='Z',
                  final_states=['q2']
                  )

    print("Enter a string with 'a's, 'b's and 'c's:\nPress Enter
Key to stop\n")

    input_str = input()
    i = 0

    while cur_pda.count_of_transitions_left() > 0 and result !=
REACHED_FINAL_STATE:
        if i < len(input_str):
            symbol = input_str[i]
        else:
            symbol = ''
        i += 1

        result = cur_pda.makeStep(symbol)

        if UNKNOWN_SYMBOL_ERR == result:
            print('Unknown symbol error')
            break

    if REACHED_FINAL_STATE == result:
        print('Accepted')
```

```
if NOT_REACHED_FINAL_STATE == result:
    print('Rejected')
```