Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Курсовая работа по дисциплине
Теория языков программирования и методы трансляции

Вариант 11

Выполнил:

Студент гр. ИП-811 _____/Мироненко К.А./
ФИО студента

«__»_____2021 г.

Проверил:

Ассистент кафедры ПМиК _____/Павлова У. В./
ФИО преподавателя

«__»_____2021 г. Оценка_____

Новосибирск, 2021г

Оглавление

Постановка задачи

Тема задания №2: преобразование конструкций, задающих язык.

Вариант №11: (11) Написать программу, которая по заданному детерминированному конечному автомату построит эквивалентную регулярную грамматику (ЛЛ или ПЛ по желанию пользователя). Функцию переходов ДКА задавать в виде таблицы, но предусмотреть возможность автоматического представления её в графическом виде. Программа должна сгенерировать по построенной грамматике несколько цепочек в указанном диапазоне длин и проверить их допустимость заданным автоматом. Процессы построения цепочек и проверки их выводимости отображать на

экране (по требованию). Предусмотреть возможность проверки автоматом цепочки, введённой пользователем.

Описание алгоритма

Необходимо программно построить по заданному детерминированному конечному автомату эквивалентную регулярную грамматику (ЛЛ или ПЛ по желанию пользователя).

Предусмотрена возможность сохранения и загрузки исходных данных с помощь кнопок «Сохранить данные» и «Загрузить данные» соответственно.

В первой форме расположены пункты «Программа», «Тема», «Автор», «Выход», в которых представлена информация о задании курсовой работы, также сведения об авторе работы.

Во второй форме («Программмма») расположена основная логика программы..

Для того, чтобы проверить принадлежит ли цепочка языку, программа ищет переход из текущего состояния по символу в таблице переходов. Если цепочка выводима – ДКА находится в конечном состоянии и все символы в цепочку обработаны, иначе цепочка не выводима.

Процесс построения ДКА:

- определяется количество нужных программе состояний (максимум 26 – кол-во символов английского алфавита);
- происходит инициализация переходов первой группы состояний ДКА;
- при переходе из состояния по следующему символу необходимой подцепочки, происходит движение на уровень ниже – на одно из следующих состояний графа, и так далее;
- если в процессе ввода подцепочки один из символов оказался не принадлежащим ей, то ДКА возвращается к первой группе состояний;

- если все символы цепочки были введены, то происходит переход к самой нижней группе состояний – нет перехода по символу подцепочки, а также одно из состояний является конечным.

Описание основных блоков программы

Основные блоки программы:

1. __main__.py – файл с основной логикой программы
2. mainwindow.py – файл, преобразованный в python из ui файла основной формы
3. startwindow.py – файл, преобразованный в python из ui файла меню формы

Основные функции программы:

1. def generate_regular_grammar_btn_clicked(self) – функция генерирующая ЛЛ и ли ПЛ регулярную грамматику на основе ДКА;
2. def clear_log_bnt_clicked(self) – функция очищающая поле log
3. def save_log_bnt_clicked(self) – функция сохраняющая поле log в файл
4. def check_chain_btn_clicked(self) – функция осуществляющая проверку цепочки ДКА
5. def save_dka_btn_clicked(self) – функция сохраняющая ДКА в файл
6. def load_dka_btn_clicked(self) – функция загружающая ДКА из файла
7. def lock_widget(self) – функция блокирующая элементы (виджеты) формы при некорректности данных
8. def draw_table(self) – Функция отрисовывающая таблицу переходов ДКА

Результаты работы программы

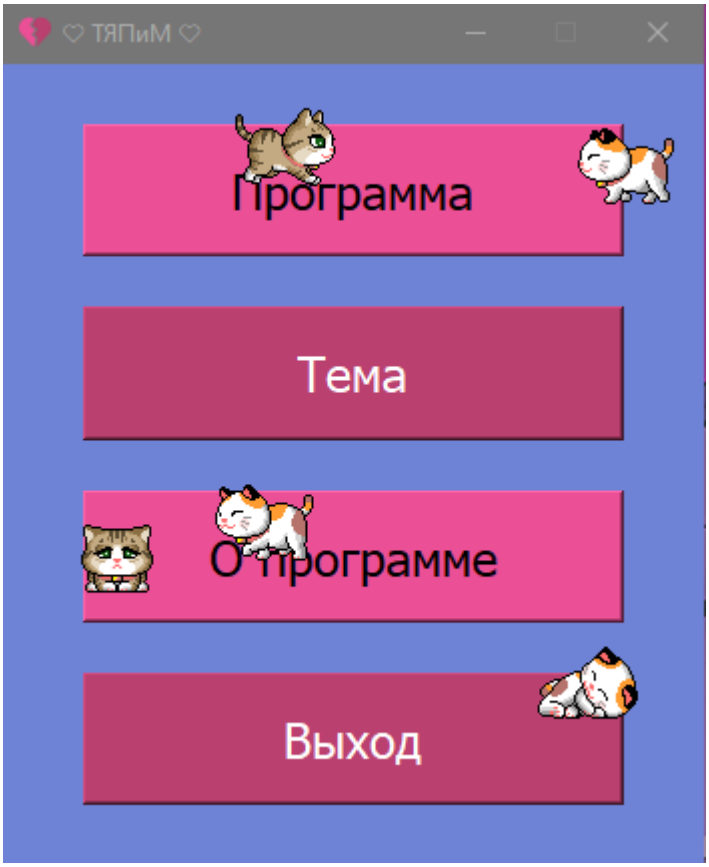


Рисунок 1. Интерфейс меню программы.

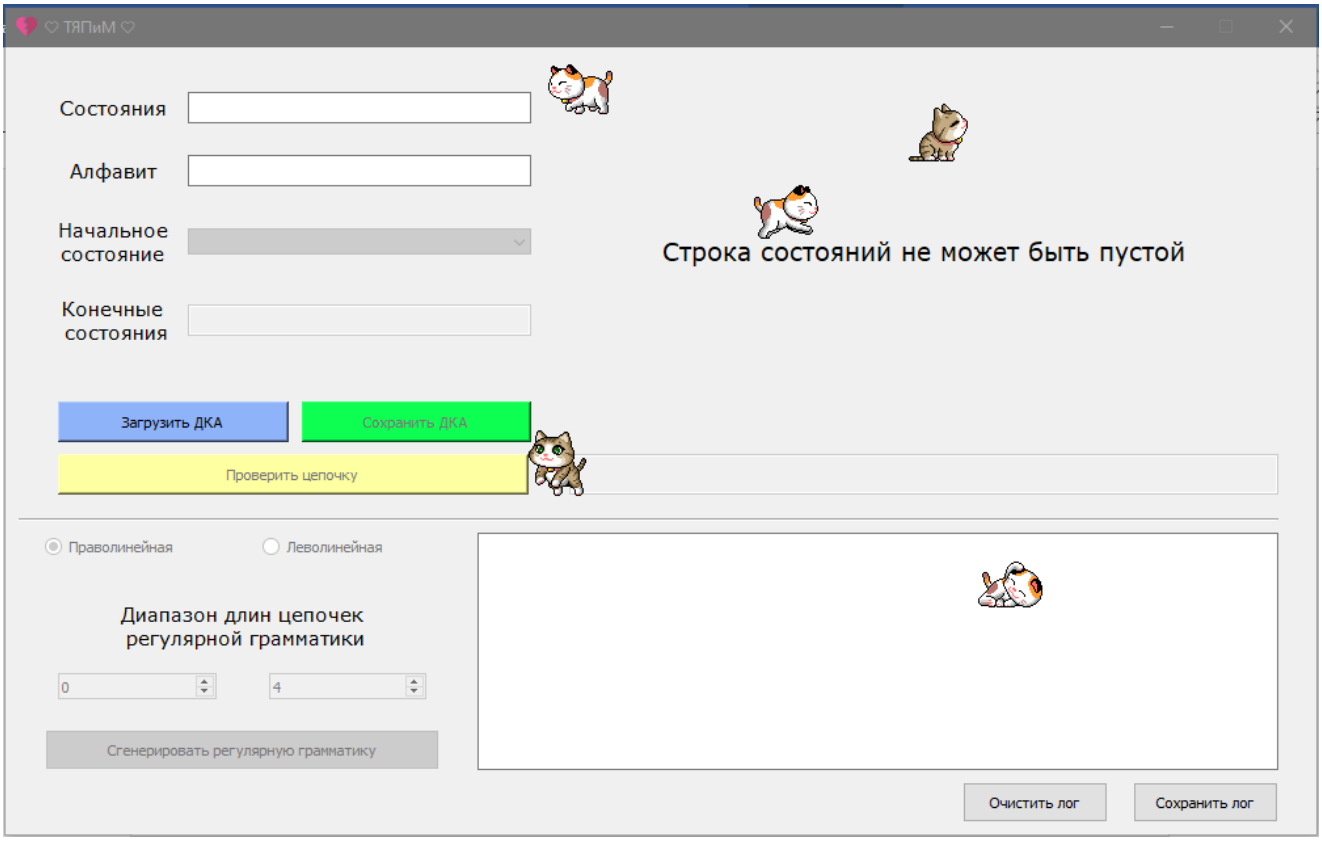


Рисунок 2. Интерфейс программы.

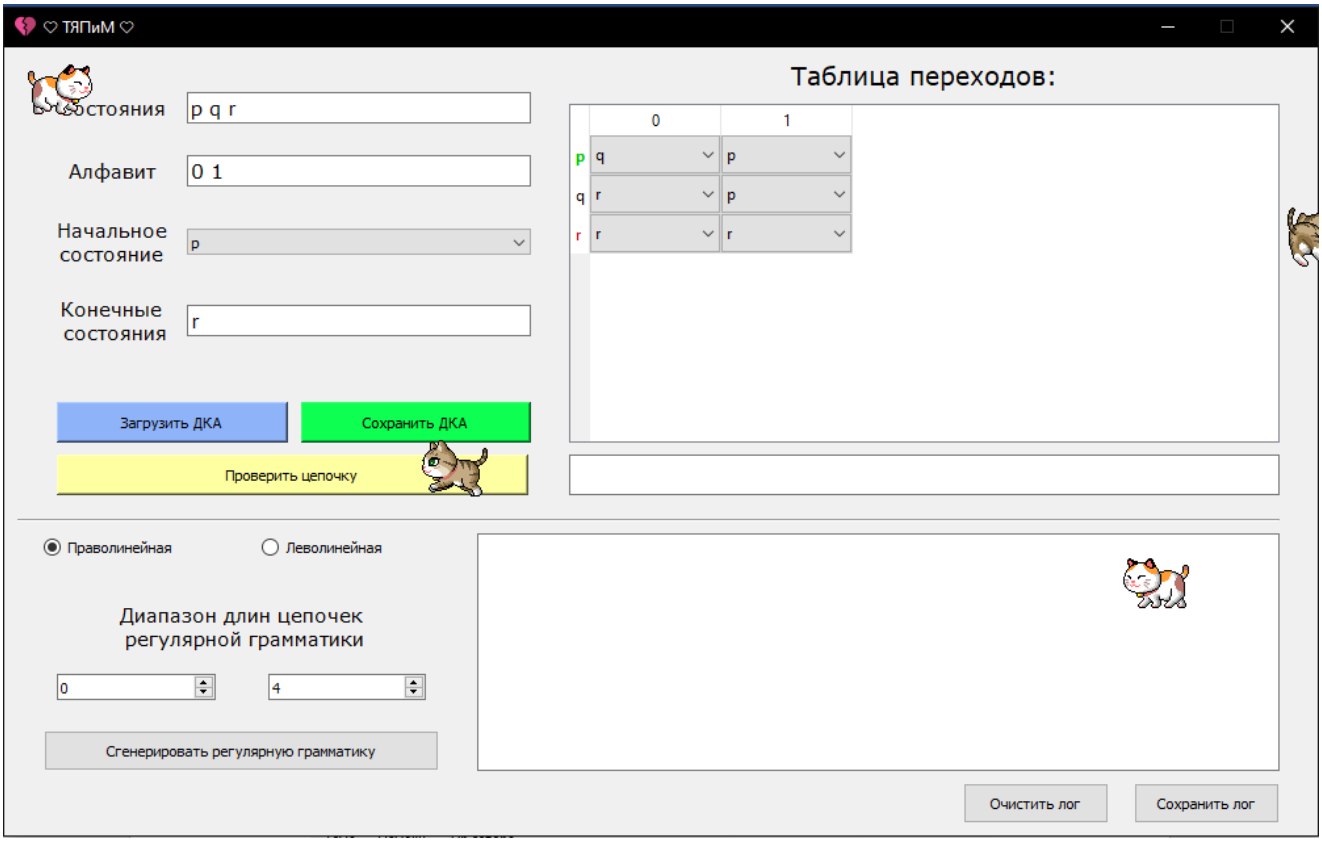


Рисунок 3. Интерфейс программы.

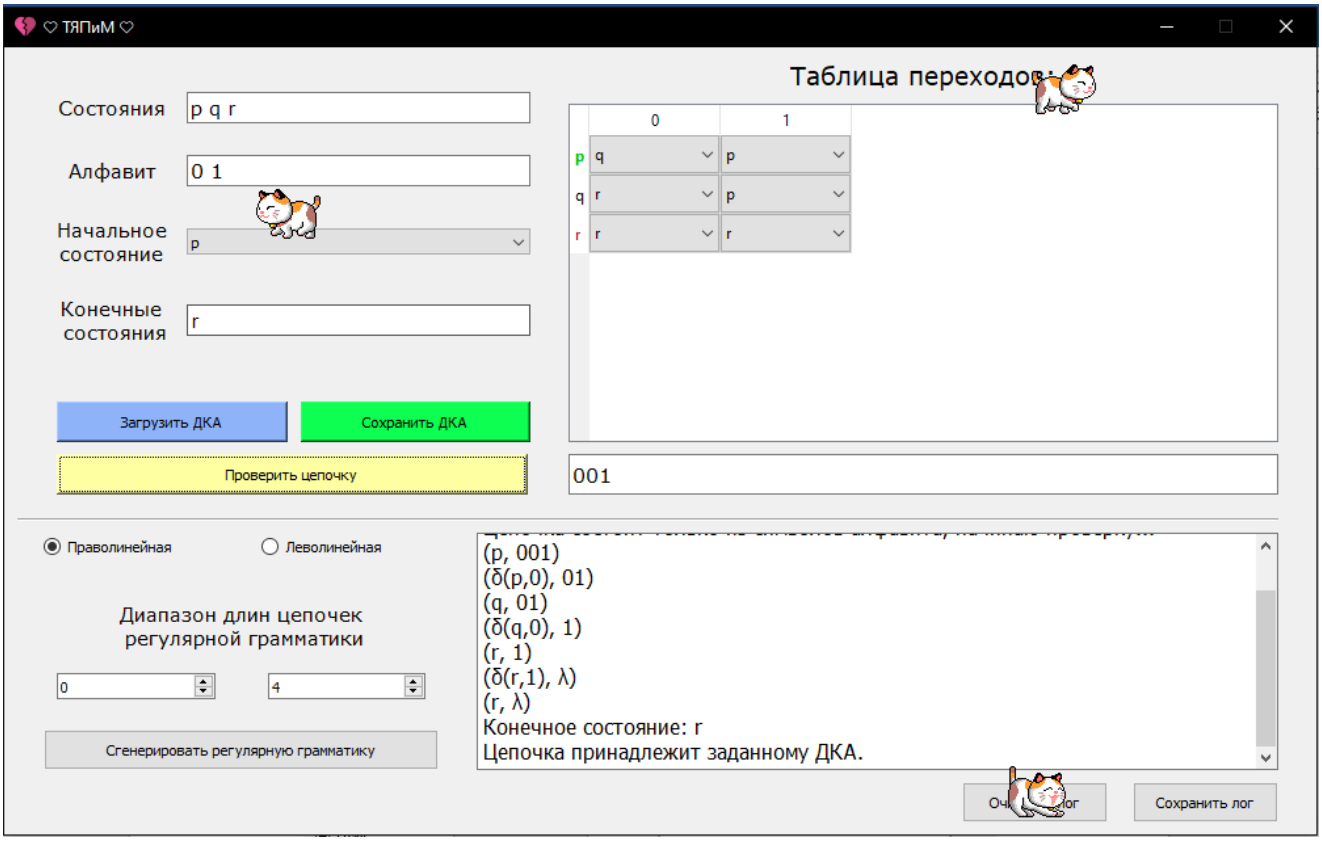


Рисунок 4. Проверка цепочки (принадлежащей языку).

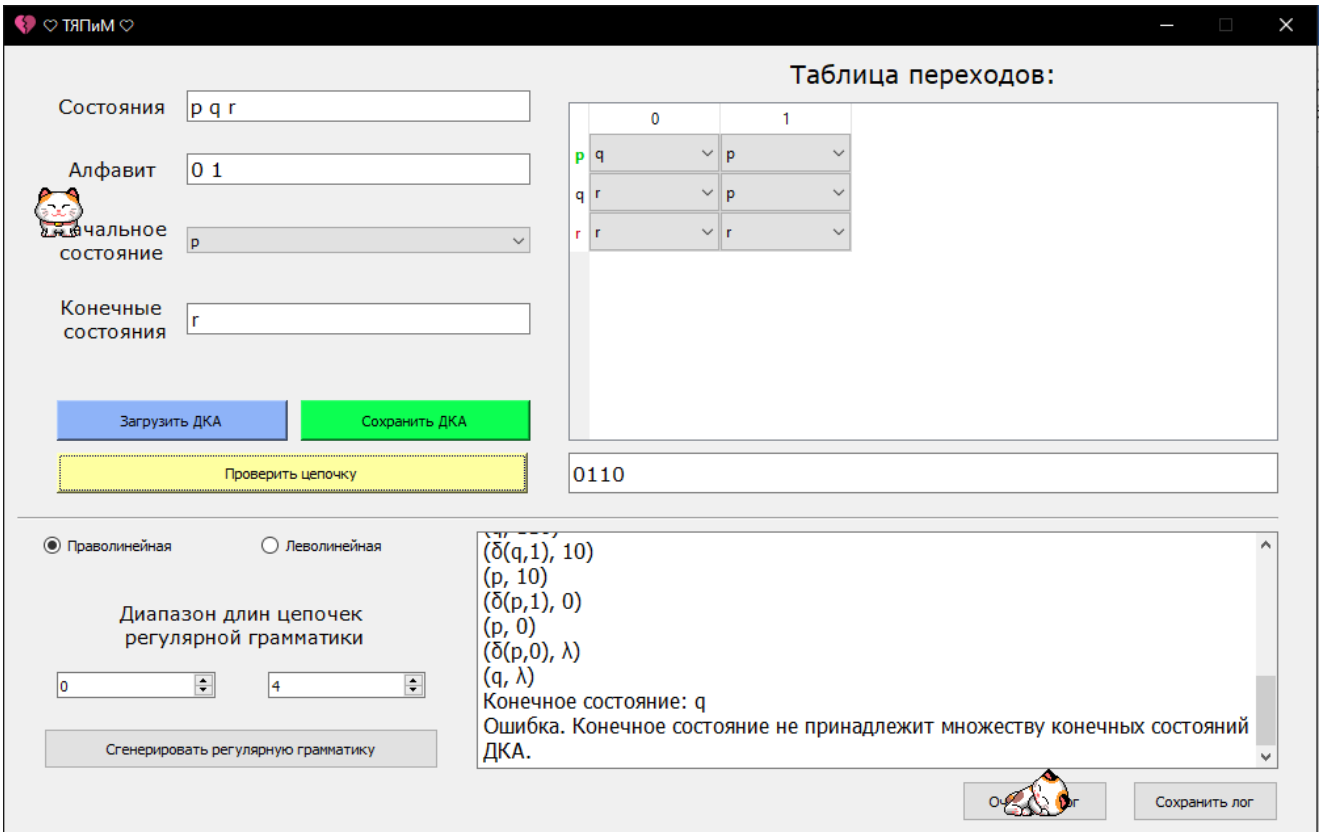
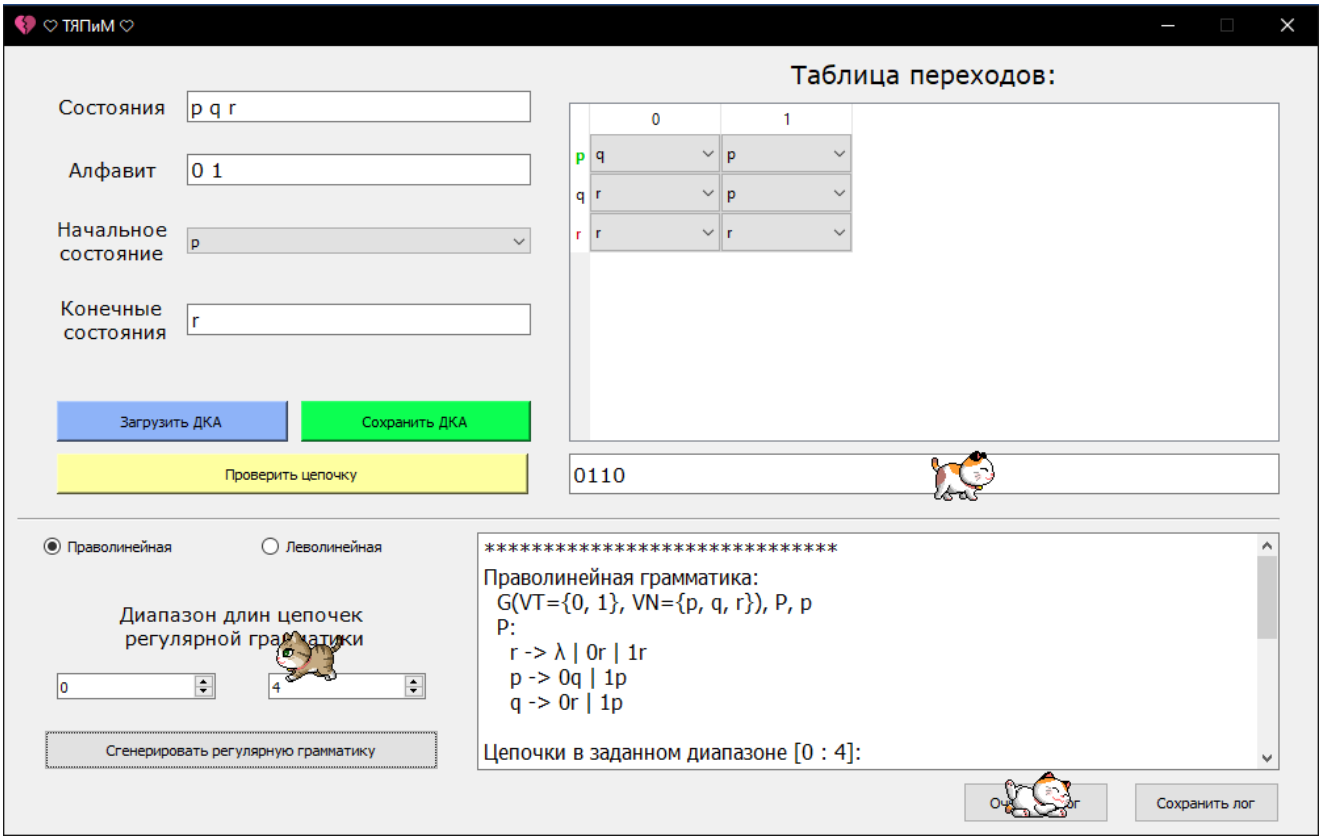


Рисунок 5. Проверка цепочки (не принадлежащей языку).



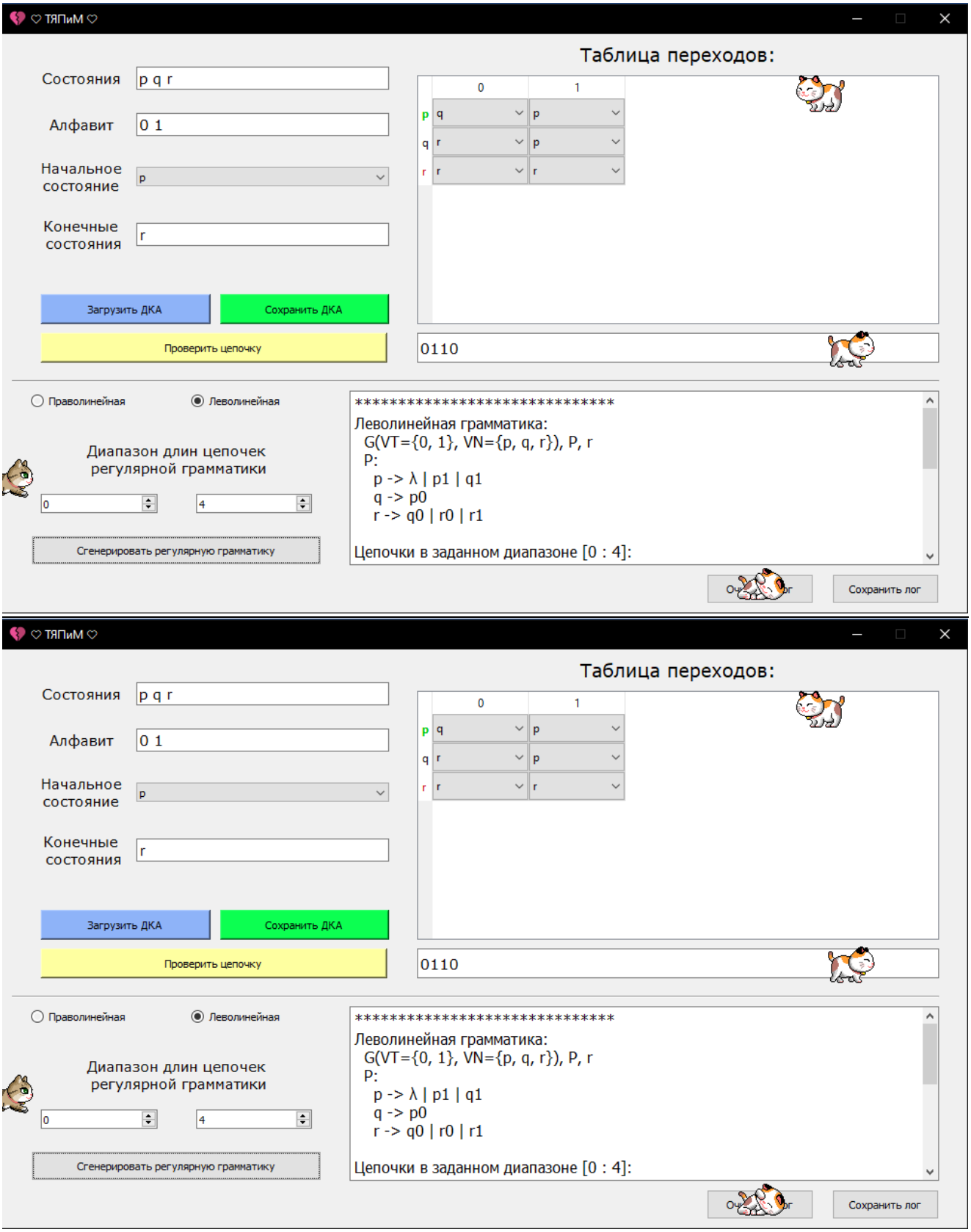


Рисунок 6-7. Геерация ЛЛ и ПЛ цепочек

Текст программы

__main__.py

```
from PyQt5.QtWidgets import QApplication
from PyQt5 import QtCore, QtGui, QtWidgets
from dataclasses import dataclass
from os import path
import sys
import json
from mainwindow import *
from startwindow import *

@dataclass
class Machine:
    states: list[str]
    alphabet: list[str]
    func: dict[str, dict[str, str]]
    start: str
    ends: list[str]

@dataclass
class Grammar:
    VT: list[str]
    VN: list[str]
    P: dict[str, list[str]]
    S: str


class StartWindow(QtWidgets.QWidget, Ui_StartWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.icon = QtGui.QIcon("resources\images\logo\logo6.png")
        self.setupUi(self)

        self.program_btn.clicked.connect(self.program_btn_click)
        self.theme_btn.clicked.connect(self.theme_btn_click)
        self.about_btn.clicked.connect(self.about_btn_click)
        self.exit_btn.clicked.connect(self.exit_btn_click)

    @QtCore.pyqtSlot()
    def program_btn_click(self):
        self.win = MainWindow()
        self.win.show()
        self.hide()



    @QtCore.pyqtSlot()
    def theme_btn_click(self):
        msgBox = MessageBox()
        msgBox.setWindowIcon(self.icon)
        # msgBox.setIcon(QtWidgets.QMessageBox.Information)
        msgBox.setWindowTitle("Тема")
        msgBox.setText("(11) Написать программу, которая по заданному
детерминированному конечному автомату построит эквивалентную регулярную грамматику
(ЛЛ или ПЛ по желанию пользователя). Функцию переходов ДКА задавать в виде таблицы,
но предусмотреть возможность автоматического представления её в графическом виде.
Программа должна сгенерировать по построенной грамматике несколько цепочек в
указанном диапазоне длин и проверить их допустимость заданным автоматом. Процессы
построения цепочек и проверки их выводимости отображать на экране (по требованию).
Предусмотреть возможность проверки автоматом цепочки, введённой пользователем.")
        msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
```

```
        x = msgBox.exec_()


    @QtCore.pyqtSlot()
    def about_btn_click(self):
        msgBox = QtWidgets.QMessageBox()
        msgBox.setWindowIcon(self.icon)
        msgBox.setIcon(QtWidgets.QMessageBox.Information)
        msgBox.setWindowTitle("О программе")
        msgBox.setText("Версия: 0.0.1 Alpha\nРазработчик: Мироненко Кирилл, ИП-
811\n\n        © 2021-2022 уч.год, СибГУТИ")
        msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
        x = msgBox.exec_()

    @QtCore.pyqtSlot()
    def exit_btn_click(self):
        self.close()

class MainWindow(QtWidgets.QWidget, Ui_MainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.machine = Machine(list(), list(), dict(), None, list())

        self.icon = QtGui.QIcon("resources\images\logo\logo6.png")
        MainWindow.setWindowIcon(self, self.icon)
        self.setupUi(self)

        self.states.setFocus()

        self.chain_lenght_from.setMaximum(self.chain_lenght_to.value())
        self.chain_lenght_to.setMinimum(self.chain_lenght_from.value())

        self.start_state.setEnabled(False)
        self.end_states.setEnabled(False)

        self.lock_widget()

        self.states_editingFinished()
        self.connect_signals()

    def connect_signals(self):
        self.states.editingFinished.connect(self.states_editingFinished)
        self.alphabet.editingFinished.connect(self.alphabet_editingFinished)
        self.start_state.currentIndexChanged.connect(
            self.start_state_currentIndexChanged)
        self.end_states.editingFinished.connect(
            self.end_states_editingFinished)

        self.save_dka_btn.clicked.connect(self.save_dka_btn_clicked)
        self.load_dka_btn.clicked.connect(self.load_dka_btn_clicked)

        self.check_chain_btn.clicked.connect(self.check_chain_btn_clicked)

        self.chain_lenght_from.valueChanged.connect(self.spinBox_valueChanged)
        self.chain_lenght_to.valueChanged.connect(self.spinBox_valueChanged)

        self.generate_regular_grammar_btn.clicked.connect(
            self.generate_regular_grammar_btn_clicked)

        self.clear_log_bnt.clicked.connect(self.clear_log_bnt_clicked)
        self.save_log_bnt.clicked.connect(self.save_log_bnt_clicked)

    @QtCore.pyqtSlot()
```

```
    def generate_regular_grammar_btn_clicked(self):

        states = self.machine.states
        alphabet = self.machine.alphabet
        start = self.machine.start
        ends = self.machine.ends
        func = self.machine.func

        # VT, VN, P, S
        vt = alphabet
        vn = states

        if self.radioButton_LL_regular_grammar.isChecked():
            p = dict()
            if len(ends) > 1:
                msgBox = QtWidgets.QMessageBox()
                msgBox.setWindowIcon(self.icon)
                msgBox.setIcon(QtWidgets.QMessageBox.Warning)
                msgBox.setWindowTitle("Ошибка")
                msgBox.setText(
                    "Построение леволинейной грамматики возможно только при одном
конечном состоянии")
                msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
                x = msgBox.exec_()
                return

            s = ends[0]
            p[start] = [""]

            for k_i, v_i in func.items():
                for k_j, v_j in v_i.items():
                    if (tmp := p.get(v_j)):
                        p[v_j].append(k_i + k_j)
                    else:
                        p[v_j] = [k_i + k_j]

        elif self.radioButton_RL_regular_grammar.isChecked():
            p = dict()
            s = start

            for end in ends:
                p[end] = [""]

            for k_i, v_i in func.items():
                for k_j, v_j in v_i.items():
                    if (tmp := p.get(k_i)):
                        p[k_i].append(k_j + v_j)
                    else:
                        p[k_i] = [k_j + v_j]

        grammar = Grammar(vt, vn, p, s)

        self.log.append("*" * 30)
        if self.radioButton_LL_regular_grammar.isChecked():
            self.log.append(f"Леволинейная грамматика:")
        else:
            self.log.append(f"Праволинейная грамматика:")

        self.log.append(
            f"  G(VT={{{', '.join(grammar.VT)}}}, VN={{{', '.join(grammar.VN)}}}), P,
{grammar.S}")
        self.log.append(f"  P:")

        for key, value in grammar.P.items():
```

```
            self.log.append(
                f"    {key} -> {' | '.join(list((x if x else 'λ') for x in value))}")

        def count_non_term_sym(grammar, sequence):
            length = 0
            for sym in sequence:
                if sym in grammar.VT:
                    length += 1
            return length

        self.log.append(
            f"\nЦепочки в заданном диапазоне [{self.chain_lenght_from.value()} :
{self.chain_lenght_to.value()}]:")

        def rec(s: str, line: str, rec_c: int):
            if rec_c > 50:
                return

            no_VN = True
            for i, symbol in enumerate(s):
                if symbol in grammar.VN:
                    no_VN = False
                    if not grammar.P.get(symbol):
                        print("Отсутствует переход по символу" + symbol)
                        return
                    for elem in grammar.P[symbol]:
                        _tmp = s[:i] + elem + s[i + 1:]
                        if count_non_term_sym(grammar, _tmp) <=
self.chain_lenght_to.value():
                            rec(_tmp, line + "->" + _tmp, rec_c + 1)

            if no_VN and self.chain_lenght_from.value() <= len(s) <=
self.chain_lenght_to.value():
                self.log.append(line)

        rec(grammar.S, grammar.S, 0)
        # stack = list(grammar.S)
        # inp = list()
        # used_sequence = set()
        # while stack:
        #     print(stack)
        #     sequence = stack.pop()
        #     # print("seq: " + sequence)
        #     if sequence in used_sequence:
        #         inp.pop()
        #         continue
        #     used_sequence.add(sequence)
        #     inp.append(sequence)

        #     no_VN = True
        #     for i, symbol in enumerate(sequence):
        #         print(i)
        #         if symbol in grammar.VN:
        #             no_VN = False
        #             if not grammar.P.get(symbol):
        #                 print("Отсутствует переход по символу" + symbol)
        #             for elem in grammar.P[symbol]:
        #                 _tmp = sequence[:i] + elem + sequence[i + 1:]
        #                 if count_non_term_sym(grammar, _tmp) <=
self.chain_lenght_to.value() and _tmp not in stack:
        #                     stack.append(_tmp)

        #     if no_VN and self.chain_lenght_from.value() <= len(sequence) <=
self.chain_lenght_to.value():
```

```
        #         print(used_sequence)
        #         print(inp)
        #         print(sequence if sequence else "λ")

    @QtCore.pyqtSlot()
    def clear_log_bnt_clicked(self):
        self.log.clear()

    @QtCore.pyqtSlot()
    def save_log_bnt_clicked(self):
        options = QtWidgets.QFileDialog.Options()
        options |= QtWidgets.QFileDialog.DontUseNativeDialog
        file, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Сохранить файл",
path.dirname(
            __file__), "log Files (*.log);;All Files (*)", options=options)
        if not file:
            return
        if not QtCore.QFileInfo(file).suffix():
            file += ".log"
        with open(file, 'w',  encoding="utf-8") as f:
            f.write(str(self.log.toPlainText()))
        msgBox = QtWidgets.QMessageBox()
        msgBox.setWindowIcon(self.icon)
        msgBox.setIcon(QtWidgets.QMessageBox.Information)
        msgBox.setWindowTitle("Уведомление")
        msgBox.setText("Файл успешно сохранен")
        msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
        x = msgBox.exec_()

    @QtCore.pyqtSlot()
    def spinBox_valueChanged(self):
        self.chain_lenght_from.setMaximum(self.chain_lenght_to.value())
        self.chain_lenght_to.setMinimum(self.chain_lenght_from.value())

    @QtCore.pyqtSlot()
    def check_chain_btn_clicked(self):
        self.update_table()

        chain = c if (c := self.check_chain.text().strip()) else "λ"
        alphabet = self.machine.alphabet + ["λ"]
        state = self.machine.start
        ends = self.machine.ends
        func = self.machine.func

        if all([c in alphabet for c in chain]):
            self.log.append("#" * 40)
            self.log.append(
                "Цепочка состоит только из символов алфавита, начинаю проверку...")
            while True:
                if chain == "λ":
                    self.log.append(f"({state}, {chain})")
                    self.log.append(f"Конечное состояние: {state}")
                    if state in ends:
                        self.log.append("Цепочка принадлежит заданному ДКА.")
                    else:
                        self.log.append(
                            "Ошибка. Конечное состояние не принадлежит множеству
конечных состояний ДКА.")
                    return

                self.log.append(f"({state}, {chain})")
                if len(chain) > 1:
                    self.log.append(f"(δ({state},{chain[0]}), {chain[1:]})")
                    try:
```

```
                        state = func[state][chain[0]]
                    except KeyError:
                        self.log.append(
                            "Ошибка. Отсутсвует переход для данного состояния.")
                        return
                    chain = chain[1:]
                else:
                    self.log.append(f"(δ({state},{chain[0]}), λ)")
                    try:
                        state = func[state][chain[0]]
                    except KeyError:
                        self.log.append(
                            "Ошибка. Отсутсвует переход для данного состояния.")
                        return
                    chain = "λ"

        else:
            msgBox = QtWidgets.QMessageBox()
            msgBox.setWindowIcon(self.icon)
            msgBox.setIcon(QtWidgets.QMessageBox.Information)
            msgBox.setWindowTitle("Ошибка")
            msgBox.setText(
                "Цепочка состоит из символов, которых нет в алфавите")
            msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
            x = msgBox.exec_()

    @QtCore.pyqtSlot()
    def save_dka_btn_clicked(self):
        options = QtWidgets.QFileDialog.Options()
        options |= QtWidgets.QFileDialog.DontUseNativeDialog
        file, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Сохранить файл",
path.dirname(
            __file__), "JSON Files (*.json);;All Files (*)", options=options)

        if not file:
            return
        if not QtCore.QFileInfo(file).suffix():
            file += ".json"

        states = self.machine.states
        alphabet = self.machine.alphabet
        start = self.machine.start
        ends = self.machine.ends
        func = self.machine.func
        res = {"states": states, "alphabet": alphabet,
               "func": func, "start": start, "ends": ends}
        with open(file, "w") as f:
            data = json.dump(res, f)

        msgBox = QtWidgets.QMessageBox()
        msgBox.setWindowIcon(self.icon)
        msgBox.setIcon(QtWidgets.QMessageBox.Information)
        msgBox.setWindowTitle("Уведомление")
        msgBox.setText("Файл успешно сохранен")
        msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
        x = msgBox.exec_()

    @QtCore.pyqtSlot()
    def load_dka_btn_clicked(self):
        msgBox = QtWidgets.QMessageBox()
        msgBox.setWindowIcon(self.icon)
        msgBox.setIcon(QtWidgets.QMessageBox.Warning)
        msgBox.setWindowTitle("Ошибка")
        options = QtWidgets.QFileDialog.Options()
```

```
        options |= QtWidgets.QFileDialog.DontUseNativeDialog
        file, _ = QtWidgets.QFileDialog.getOpenFileName(self, "Открыть файл",
path.dirname(
            __file__), "JSON Files (*.json);;All Files (*)", options=options)
        if not file:
            return

        with open(file, "r") as f:
            try:
                data = json.load(f)
                machine = Machine(*data.values())
            except json.JSONDecodeError as e:
                msgBox.setText("Некоректный файл формата JSON")
                msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
                x = msgBox.exec_()
                return
            except TypeError as e:
                msgBox.setText("Грамматика некоректна")
                msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok)
                x = msgBox.exec_()
                return

        states = machine.states
        alphabet = machine.alphabet
        start = machine.start
        ends = machine.ends
        func = machine.func

        self.states.setText(" ".join(states))
        self.states_editingFinished()

        self.alphabet.setText(" ".join(alphabet))
        self.alphabet_editingFinished()

        self.start_state.setCurrentText(start)
        self.start_state_currentIndexChanged()

        self.end_states.setText(" ".join(ends))
        self.end_states_editingFinished()

        if func:
            for i in range(self.transition_table.rowCount()):
                for j in range(self.transition_table.columnCount()):
                    self.transition_table.cellWidget(i,
j).setCurrentText(func.get(self.transition_table.verticalHeaderItem(
                        i).text()).get(self.transition_table.horizontalHeaderItem(j).
text()))

    @QtCore.pyqtSlot()
    def states_editingFinished(self):
        states = list(dict.fromkeys(self.states.text().strip().split()).keys())
        self.machine.states = states
        if not states:
            self.start_state.clear()
            self.machine.start_state = None
            self.start_state.setEnabled(False)
            self.machine.end_states = None
            self.end_states.setText(None)
            self.end_states.setEnabled(False)
        else:
            self.start_state.clear()
            self.start_state.setEnabled(True)
            self.start_state.addItems(states)
            self.start_state.currentIndex = -1
```

```
            self.end_states.setEnabled(True)
        self.draw_table()

    @QtCore.pyqtSlot()
    def alphabet_editingFinished(self):
        alphabet = list(dict.fromkeys(
            self.alphabet.text().strip().split()).keys())
        self.machine.alphabet = alphabet
        self.draw_table()

    @QtCore.pyqtSlot()
    def start_state_currentIndexChanged(self):
        self.machine.start = self.start_state.currentText()
        self.draw_table()

    @QtCore.pyqtSlot()
    def end_states_editingFinished(self):
        end_states = list(dict.fromkeys(
            self.end_states.text().strip().split()).keys())
        self.machine.ends = end_states
        self.draw_table()

    def lock_widget(self):
        self.save_dka_btn.setEnabled(False)
        self.check_chain_btn.setEnabled(False)
        self.check_chain.setEnabled(False)
        self.chain_lenght_from.setEnabled(False)
        self.chain_lenght_to.setEnabled(False)
        self.generate_regular_grammar_btn.setEnabled(False)
        self.radioButton_LL_regular_grammar.setEnabled(False)
        self.radioButton_RL_regular_grammar.setEnabled(False)
        self.transition_table.reset()
        self.transition_table.hide()

    def update_table(self):
        data = dict()
        for i in range(self.transition_table.rowCount()):
            tmp = dict()
            for j in range(self.transition_table.columnCount()):
                value = self.transition_table.cellWidget(i, j).currentText()
                if value:
                    tmp[self.transition_table.horizontalHeaderItem(
                        j).text()] = value
            data[self.transition_table.verticalHeaderItem(i).text()] = tmp

        self.machine.func = data

    def draw_table(self):
        states = self.machine.states
        alphabet = self.machine.alphabet
        start = self.machine.start
        ends = self.machine.ends
        func = self.machine.func

        if not states:
            self.lock_widget()
            self.machine.func = dict()
            self.label_table.setText("Строка состояний не может быть пустой")
            return

        if not all(len(x) == 1 for x in states):
            self.lock_widget()
            self.machine.func = dict()
            self.label_table.setText("Состояния должны быть односимвольными")
```

```
    return

if not alphabet:
    self.lock_widget()
    self.machine.func = dict()
    self.label_table.setText("Алфавит не может быть пустым")
    return

if not all(len(x) == 1 for x in alphabet):
    self.lock_widget()
    self.machine.func = dict()
    self.label_table.setText("Алфавит должен состоять из символов")
    return

for state in states:
    if state in alphabet:
        self.lock_widget()
        self.label_table.setText(
            "Состояния и алфавит не могут пересекаться")
        return

if not ends:
    self.lock_widget()
    self.label_table.setText(
        "Должно быть как минимум одно конечное состояние")
    return

for state in ends:
    if state not in states:
        self.lock_widget()
        self.label_table.setText("Некорректные конечные состояния")
        return

self.label_table.setText("Таблица переходов:")
self.save_dka_btn.setEnabled(True)
self.check_chain_btn.setEnabled(True)
self.check_chain.setEnabled(True)

self.radioButton_LL_regular_grammar.setEnabled(True)
self.radioButton_RL_regular_grammar.setEnabled(True)
self.chain_lenght_from.setEnabled(True)
self.chain_lenght_to.setEnabled(True)
self.generate_regular_grammar_btn.setEnabled(True)

self.transition_table.show()

self.transition_table.setRowCount(len(self.machine.states))
self.transition_table.setVerticalHeaderLabels(self.machine.states)

self.transition_table.setColumnCount(len(self.machine.alphabet))
self.transition_table.setHorizontalHeaderLabels(self.machine.alphabet)

font_bold = QtGui.QFont()
font_bold.setBold(True)

for i, item in enumerate(self.machine.states):
    header = QtWidgets.QTableWidgetItem(item)
    if item == start:
        header.setForeground(QtGui.QColor(0, 200, 0))
        if item in ends:
            header.setForeground(QtGui.QColor(0, 0, 200))
        header.setFont(font_bold)
    elif item in ends:
        header.setForeground(QtGui.QColor(200, 0, 0))
```

```
            self.transition_table.setVerticalHeaderItem(i, header)

        for i, item in enumerate(self.machine.alphabet):
            header = QtWidgets.QTableWidgetItem(item)
            self.transition_table.setHorizontalHeaderItem(i, header)

        for i, var_i in enumerate(self.machine.states):
            for j, var_j in enumerate(self.machine.alphabet):
                item = QtWidgets.QComboBox()
                item.addItems([None] + self.machine.states)
                if (g := func.get(var_i)):
                    item.setCurrentText(g.get(var_j))

                item.currentIndexChanged.connect(self.update_table)
                self.transition_table.setCellWidget(i, j, item)

        self.update_table()

if __name__ == "__main__":
    # TODO: при ЛЛ должно быть одно конечное?
    app = QApplication(sys.argv)
    win = StartWindow()
    # win = MainWindow()
    win.show()
    sys.exit(app.exec_())

# pyuic5 tmp.ui -o tmp.py

# Референс: https://c-stud.ru/work_html/look_full.html?id=176483&razdel=6977
# ЛЛ и ПЛ
http://cmcstuff.esyr.org/n10/2%20%D0%BA%D1%83%D1%80%D1%81/%D0%A1%D0%9F/SP_gdrive/new%
20version/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D0%B8/%D0%9E%20%D1%80%D0%B5%D0%B3%D1%83%D0%B
B%D1%8F%D1%80%D0%BD%D1%8B%D1%85%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0%D1%85.pdf
# Что-то: https://www.cyberforum.ru/cpp-beginners/thread2396459.html

# может нада:
#
https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0
%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80
%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B
2_%D0%94%D0%9A%D0%90#.D0.90.D0.BB.D0.B3.D0.B5.D0.B1.D1.80.D0.B0.D0.B8.D1.87.D0.B5.D1.
81.D0.BA.D0.B8.D0.B9_.D0.BC.D0.B5.D1.82.D0.BE.D0.B4_.D0.91.D0.B6.D0.BE.D0.B7.D0.BE.D0
.B2.D1.81.D0.BA.D0.BE.D0.B3.D0.BE
# https://masters.donntu.org/2017/fknt/gerbutova/library/article10_2.0.htm
# https://qastack.ru/cs/2016/how-to-convert-finite-automata-to-regular-expressions
# https://masters.donntu.org/2017/fknt/gerbutova/library/article10_2.0.htm

# если кто-то решит порисовать:
# https://stackoverflow.com/questions/60661557/join-two-circles-using-a-join-
function-in-pyqt5
# https://coderoad.wiki/41732808/%D0%9A%D0%B0%D0%BA-
%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D1%81%D1%82%D0%B8%D1%82%D1%8C-
%D1%82%D0%BE%D1%87%D0%BA%D1%83-%D0%BF%D0%BE-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D1%83-
%D0%B2-PyQt5
# https://question-it.com/questions/3071558/kak-peremestit-figuru-sozdannuju-s-
pomoschju-paintevent-prosto-peretaschiv-ee-v-pyqt5
```

startwindow.py

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_StartWindow(object):
```

```
    def setupUi(self, StartWindow):
        StartWindow.setObjectName("StartWindow")
        StartWindow.setEnabled(True)
        StartWindow.resize(350, 400)
        StartWindow.setMinimumSize(QtCore.QSize(350, 400))
        StartWindow.setMaximumSize(QtCore.QSize(350, 400))
        StartWindow.setWindowIcon(self.icon)
        StartWindow.setStyleSheet("background-color: #6F83D6;")

        self.verticalLayoutWidget = QtWidgets.QWidget(self)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(0, 0, 351, 401))
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")

        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setSizeConstraint(QtWidgets.QLayout.SetMaximumSize)
        self.verticalLayout.setContentsMargins(40, 30, 40, 30)
        self.verticalLayout.setSpacing(25)
        self.verticalLayout.setObjectName("verticalLayout")

        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

        font = QtGui.QFont()
        font.setPointSize(18)
        font.setStrikeOut(False)
        font.setStyleStrategy(QtGui.QFont.PreferDefault)

        self.program_btn = QtWidgets.QPushButton(self.verticalLayoutWidget)

sizePolicy.setHeightForWidth(self.program_btn.sizePolicy().hasHeightForWidth())
        self.program_btn.setSizePolicy(sizePolicy)
        self.program_btn.setFont(font)
        self.program_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.program_btn.setObjectName("program_btn")
        self.program_btn.setStyleSheet("background-color: #eb4f96;")
        self.verticalLayout.addWidget(self.program_btn)

        self.theme_btn = QtWidgets.QPushButton(self.verticalLayoutWidget)
        sizePolicy.setHeightForWidth(self.theme_btn.sizePolicy().hasHeightForWidth())
        self.theme_btn.setSizePolicy(sizePolicy)
        self.theme_btn.setFont(font)
        self.theme_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.theme_btn.setObjectName("theme_btn")
        self.theme_btn.setStyleSheet("background-color: #ba406f; color: #ffffff;")
        self.verticalLayout.addWidget(self.theme_btn)

        self.about_btn = QtWidgets.QPushButton(self.verticalLayoutWidget)
        sizePolicy.setHeightForWidth(self.about_btn.sizePolicy().hasHeightForWidth())
        self.about_btn.setSizePolicy(sizePolicy)
        self.about_btn.setFont(font)
        self.about_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.about_btn.setObjectName("about_btn")
        self.about_btn.setStyleSheet("background-color: #eb4f96;")
        self.verticalLayout.addWidget(self.about_btn)

        self.exit_btn = QtWidgets.QPushButton(self.verticalLayoutWidget)
        sizePolicy.setHeightForWidth(self.exit_btn.sizePolicy().hasHeightForWidth())
        self.exit_btn.setSizePolicy(sizePolicy)
        self.exit_btn.setFont(font)
        self.exit_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exit_btn.setObjectName("exit_btn")
        self.exit_btn.setStyleSheet("background-color: #ba406f; color: #ffffff;")
```

```
        self.verticalLayout.addWidget(self.exit_btn)

        StartWindow.setLayout(self.verticalLayout)

        self.retranslateUi(StartWindow)
        QtCore.QMetaObject.connectSlotsByName(StartWindow)

    def retranslateUi(self, StartWindow):
        _translate = QtCore.QCoreApplication.translate
        StartWindow.setWindowTitle(_translate("StartWindow", "❤️ ТЯПиМ ❤️"))
        self.program_btn.setText(_translate("StartWindow", "Программа"))
        self.theme_btn.setText(_translate("StartWindow", "Тема"))
        self.about_btn.setText(_translate("StartWindow", "О программе"))
        self.exit_btn.setText(_translate("StartWindow", "Выход"))



class MessageBox(QtWidgets.QMessageBox):
    def __init__(self, parent=None):
        super().__init__(parent)
        grid_layout = self.layout()

        qt_msgboxex_icon_label = self.findChild(QtWidgets.QLabel,
"qt_msgboxex_icon_label")
        qt_msgboxex_icon_label.deleteLater()

        qt_msgbox_label = self.findChild(QtWidgets.QLabel, "qt_msgbox_label")
        qt_msgbox_label.setAlignment(QtCore.Qt.AlignLeft)
        grid_layout.removeWidget(qt_msgbox_label)

        qt_msgbox_buttonbox = self.findChild(QtWidgets.QDialogButtonBox,
"qt_msgbox_buttonbox")
        grid_layout.removeWidget(qt_msgbox_buttonbox)

        grid_layout.addWidget(qt_msgbox_label, 0, 0, alignment=QtCore.Qt.AlignLeft)
        grid_layout.addWidget(qt_msgbox_buttonbox, 1, 0,
alignment=QtCore.Qt.AlignCenter)
```

mainwindow.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'test.ui'
#
# Created by: PyQt5 UI code generator 5.15.6
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1000, 600)
        MainWindow.setMinimumSize(QtCore.QSize(1000, 600))
        MainWindow.setMaximumSize(QtCore.QSize(1000, 600))
        self.layoutWidget = QtWidgets.QWidget(MainWindow)
        self.layoutWidget.setGeometry(QtCore.QRect(430, 10, 541, 291))
        self.layoutWidget.setObjectName("layoutWidget")
        self.verticalLayoutTable = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.verticalLayoutTable.setContentsMargins(0, 0, 0, 0)
        self.verticalLayoutTable.setSpacing(10)
        self.verticalLayoutTable.setObjectName("verticalLayoutTable")
```

```
        self.label_table = QtWidgets.QLabel(self.layoutWidget)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(14)
        self.label_table.setFont(font)
        self.label_table.setAlignment(QtCore.Qt.AlignCenter)
        self.label_table.setObjectName("label_table")
        self.verticalLayoutTable.addWidget(self.label_table)
        self.transition_table = QtWidgets.QTableWidget(self.layoutWidget)
        self.transition_table.setObjectName("transition_table")
        self.transition_table.setColumnCount(0)
        self.transition_table.setRowCount(0)
        self.verticalLayoutTable.addWidget(self.transition_table)
        self.line = QtWidgets.QFrame(MainWindow)
        self.line.setGeometry(QtCore.QRect(10, 350, 961, 21))
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.layoutWidget_2 = QtWidgets.QWidget(MainWindow)
        self.layoutWidget_2.setGeometry(QtCore.QRect(40, 10, 361, 241))
        self.layoutWidget_2.setObjectName("layoutWidget_2")
        self.gridLayout = QtWidgets.QGridLayout(self.layoutWidget_2)
        self.gridLayout.setContentsMargins(0, 0, 0, 0)
        self.gridLayout.setHorizontalSpacing(15)
        self.gridLayout.setVerticalSpacing(0)
        self.gridLayout.setObjectName("gridLayout")
        self._label_alphabet = QtWidgets.QLabel(self.layoutWidget_2)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self._label_alphabet.setFont(font)
        self._label_alphabet.setAlignment(QtCore.Qt.AlignCenter)
        self._label_alphabet.setObjectName("_label_alphabet")
        self.gridLayout.addWidget(self._label_alphabet, 2, 0, 1, 1)
        self._label_end_states = QtWidgets.QLabel(self.layoutWidget_2)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self._label_end_states.setFont(font)
        self._label_end_states.setAlignment(QtCore.Qt.AlignCenter)
        self._label_end_states.setObjectName("_label_end_states")
        self.gridLayout.addWidget(self._label_end_states, 5, 0, 1, 1)
        self.end_states = QtWidgets.QLineEdit(self.layoutWidget_2)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.end_states.sizePolicy().hasHeightForWidth()
)
        self.end_states.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self.end_states.setFont(font)
        self.end_states.setText("")
        self.end_states.setObjectName("end_states")
        self.gridLayout.addWidget(self.end_states, 5, 1, 1, 1)
        self.start_state = QtWidgets.QComboBox(self.layoutWidget_2)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.start_state.sizePolicy().hasHeightForWidth(
))
```

```
        self.start_state.setSizePolicy(sizePolicy)
        self.start_state.setObjectName("start_state")
        self.gridLayout.addWidget(self.start_state, 4, 1, 1, 1)
        self._label_states = QtWidgets.QLabel(self.layoutWidget_2)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self._label_states.setFont(font)
        self._label_states.setAlignment(QtCore.Qt.AlignCenter)
        self._label_states.setObjectName("_label_states")
        self.gridLayout.addWidget(self._label_states, 0, 0, 1, 1)
        self._label_start_state = QtWidgets.QLabel(self.layoutWidget_2)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self._label_start_state.setFont(font)
        self._label_start_state.setAlignment(QtCore.Qt.AlignCenter)
        self._label_start_state.setObjectName("_label_start_state")
        self.gridLayout.addWidget(self._label_start_state, 4, 0, 1, 1)
        self.alphabet = QtWidgets.QLineEdit(self.layoutWidget_2)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.alphabet.sizePolicy().hasHeightForWidth())
        self.alphabet.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self.alphabet.setFont(font)
        self.alphabet.setText("")
        self.alphabet.setObjectName("alphabet")
        self.gridLayout.addWidget(self.alphabet, 2, 1, 1, 1)
        self.states = QtWidgets.QLineEdit(self.layoutWidget_2)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.states.sizePolicy().hasHeightForWidth())
        self.states.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self.states.setFont(font)
        self.states.setText("")
        self.states.setObjectName("states")
        self.gridLayout.addWidget(self.states, 0, 1, 1, 1)
        self.check_chain_btn = QtWidgets.QPushButton(MainWindow)
        self.check_chain_btn.setGeometry(QtCore.QRect(40, 310, 359, 31))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.check_chain_btn.sizePolicy().hasHeightForWi
dth())
        self.check_chain_btn.setSizePolicy(sizePolicy)
        self.check_chain_btn.setStyleSheet("background-color: rgb(254, 255, 160)")
        self.check_chain_btn.setObjectName("check_chain_btn")
        self.layoutWidget1 = QtWidgets.QWidget(MainWindow)
        self.layoutWidget1.setGeometry(QtCore.QRect(40, 270, 361, 31))
        self.layoutWidget1.setObjectName("layoutWidget1")
        self.horizontalLayoutBtn = QtWidgets.QHBoxLayout(self.layoutWidget1)
        self.horizontalLayoutBtn.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstr
aint)
```

```
        self.horizontalLayoutBtn.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayoutBtn.setSpacing(10)
        self.horizontalLayoutBtn.setObjectName("horizontalLayoutBtn")
        self.load_dka_btn = QtWidgets.QPushButton(self.layoutWidget1)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.load_dka_btn.sizePolicy().hasHeightForWidth
())
        self.load_dka_btn.setSizePolicy(sizePolicy)
        self.load_dka_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.load_dka_btn.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.load_dka_btn.setStyleSheet("background-color: rgb(142, 179, 248)")
        self.load_dka_btn.setObjectName("load_dka_btn")
        self.horizontalLayoutBtn.addWidget(self.load_dka_btn)
        self.save_dka_btn = QtWidgets.QPushButton(self.layoutWidget1)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.save_dka_btn.sizePolicy().hasHeightForWidth
())
        self.save_dka_btn.setSizePolicy(sizePolicy)
        self.save_dka_btn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.save_dka_btn.setStyleSheet("background-color: rgb(12, 255, 81)")
        self.save_dka_btn.setObjectName("save_dka_btn")
        self.horizontalLayoutBtn.addWidget(self.save_dka_btn)
        self.check_chain = QtWidgets.QLineEdit(MainWindow)
        self.check_chain.setGeometry(QtCore.QRect(430, 310, 541, 31))
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self.check_chain.setFont(font)
        self.check_chain.setText("")
        self.check_chain.setObjectName("check_chain")
        self.log = QtWidgets.QTextEdit(MainWindow)
        self.log.setGeometry(QtCore.QRect(360, 370, 611, 181))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.log.setFont(font)
        self.log.setReadOnly(True)
        self.log.setObjectName("log")
        self.horizontalLayoutWidget = QtWidgets.QWidget(MainWindow)
        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(30, 370, 301, 21))
        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setSpacing(30)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.radioButton_RL_regular_grammar =
QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        self.radioButton_RL_regular_grammar.setChecked(True)
        self.radioButton_RL_regular_grammar.setObjectName("radioButton_RL_regular_gra
mmar")
        self.horizontalLayout.addWidget(self.radioButton_RL_regular_grammar)
        self.radioButton_LL_regular_grammar =
QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        self.radioButton_LL_regular_grammar.setEnabled(True)
        self.radioButton_LL_regular_grammar.setChecked(False)
        self.radioButton_LL_regular_grammar.setObjectName("radioButton_LL_regular_gra
mmar")
        self.horizontalLayout.addWidget(self.radioButton_LL_regular_grammar)
        self.generate_regular_grammar_btn = QtWidgets.QPushButton(MainWindow)
```

```
        self.generate_regular_grammar_btn.setGeometry(QtCore.QRect(30, 520, 301, 31))
        self.generate_regular_grammar_btn.setObjectName("generate_regular_grammar_btn
")
        self.save_log_bnt = QtWidgets.QPushButton(MainWindow)
        self.save_log_bnt.setGeometry(QtCore.QRect(860, 560, 111, 31))
        self.save_log_bnt.setObjectName("save_log_bnt")
        self.verticalLayoutWidget = QtWidgets.QWidget(MainWindow)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(30, 400, 301, 107))
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(10, 10, 10, 10)
        self.verticalLayout.setObjectName("verticalLayout")
        self._label_end_states_2 = QtWidgets.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPointSize(11)
        self._label_end_states_2.setFont(font)
        self._label_end_states_2.setAlignment(QtCore.Qt.AlignCenter)
        self._label_end_states_2.setObjectName("_label_end_states_2")
        self.verticalLayout.addWidget(self._label_end_states_2)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setSpacing(40)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.chain_lenght_from = QtWidgets.QSpinBox(self.verticalLayoutWidget)
        self.chain_lenght_from.setMaximum(20)
        self.chain_lenght_from.setObjectName("chain_lenght_from")
        self.horizontalLayout_2.addWidget(self.chain_lenght_from)
        self.chain_lenght_to = QtWidgets.QSpinBox(self.verticalLayoutWidget)
        self.chain_lenght_to.setMaximum(20)
        self.chain_lenght_to.setProperty("value", 4)
        self.chain_lenght_to.setObjectName("chain_lenght_to")
        self.horizontalLayout_2.addWidget(self.chain_lenght_to)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.clear_log_bnt = QtWidgets.QPushButton(MainWindow)
        self.clear_log_bnt.setGeometry(QtCore.QRect(730, 560, 111, 31))
        self.clear_log_bnt.setObjectName("clear_log_bnt")

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "❤️ ТЯПиМ ❤️"))
        self.label_table.setText(_translate("MainWindow", "Таблица переходов:"))
        self._label_alphabet.setText(_translate("MainWindow", "Алфавит"))
        self._label_end_states.setText(_translate("MainWindow", "Конечные\n"
" состояния"))
        self._label_states.setText(_translate("MainWindow", "Состояния"))
        self._label_start_state.setText(_translate("MainWindow", "Начальное \n"
"состояние"))
        self.check_chain_btn.setText(_translate("MainWindow", "Проверить цепочку"))
        self.load_dka_btn.setText(_translate("MainWindow", "Загрузить ДКА"))
        self.save_dka_btn.setText(_translate("MainWindow", "Сохранить ДКА"))
        self.radioButton_RL_regular_grammar.setText(_translate("MainWindow",
"Праволинейная"))
        self.radioButton_LL_regular_grammar.setText(_translate("MainWindow",
"Леволинейная"))
        self.generate_regular_grammar_btn.setText(_translate("MainWindow",
"Сгенерировать регулярную грамматику"))
        self.save_log_bnt.setText(_translate("MainWindow", "Сохранить лог"))
        self._label_end_states_2.setText(_translate("MainWindow", "Диапазон длин
цепочек\n"
" регулярной грамматики"))
        self.clear_log_bnt.setText(_translate("MainWindow", "Очистить лог"))
```