## 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

« » 2021 г.

Проверил:

/Мироненко К.А./

### ФИО студента

Ассистент кафедры ПМиК /Павлова У. В./

### ФИО преподавателя

« » 2021 г. Оценка

Новосибирск, 2021г

Оглавление

[Постановка задачи 3](#_Toc93864216)

[Описание алгоритма 4](#_Toc93864217)

[Описание основных блоков программы 6](#_Toc93864218)

[Результаты работы программы 7](#_Toc93864219)

[Текст программы 11](#_Toc93864220)

# 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 («Программма») 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

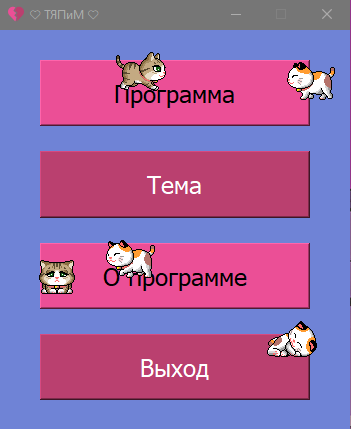
Основные блоки программы:

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

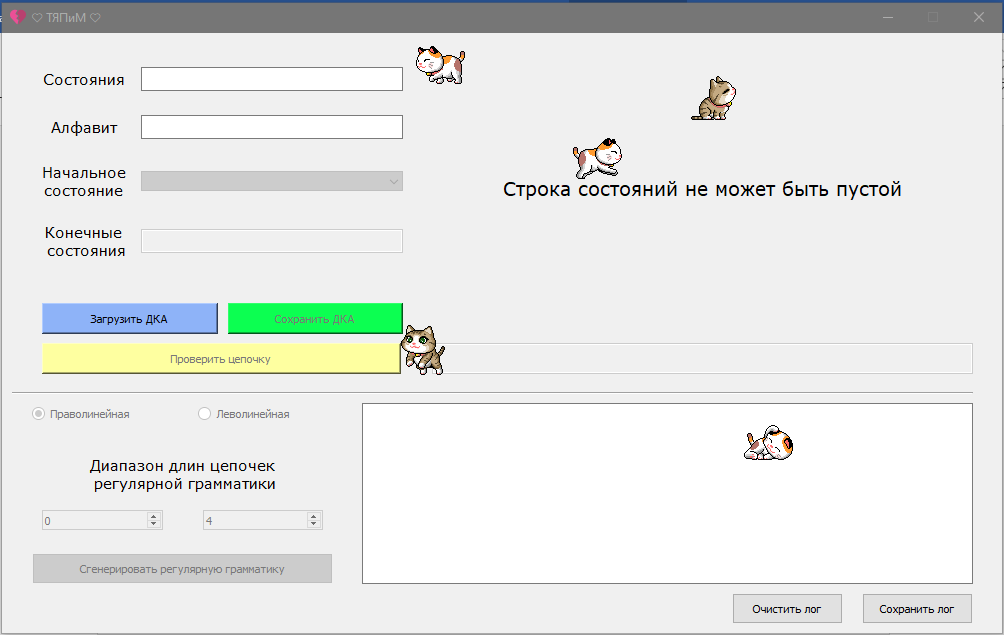
Основные функции программы:

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

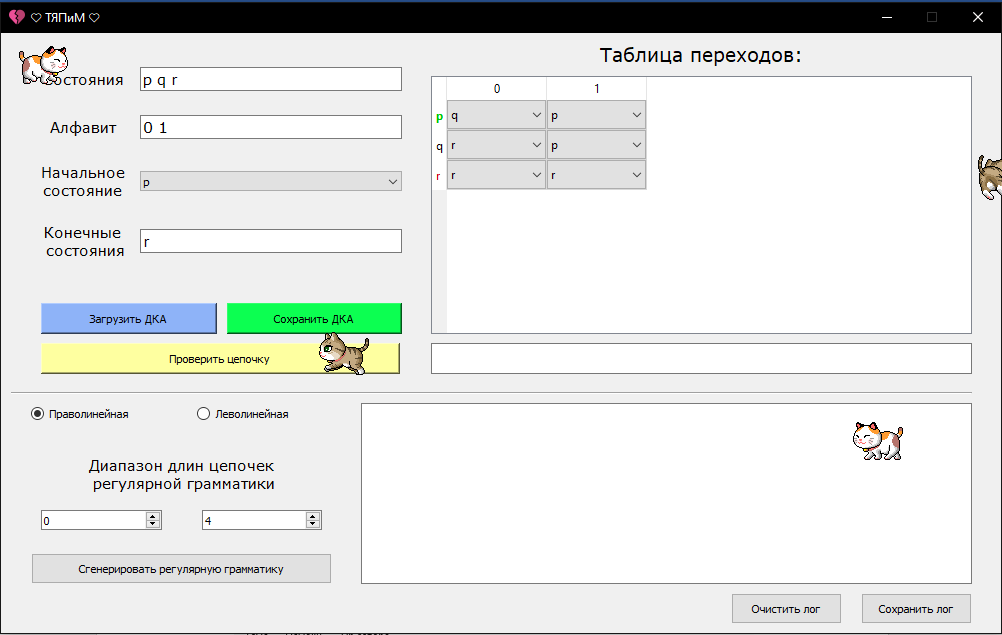
# Результаты работы программы



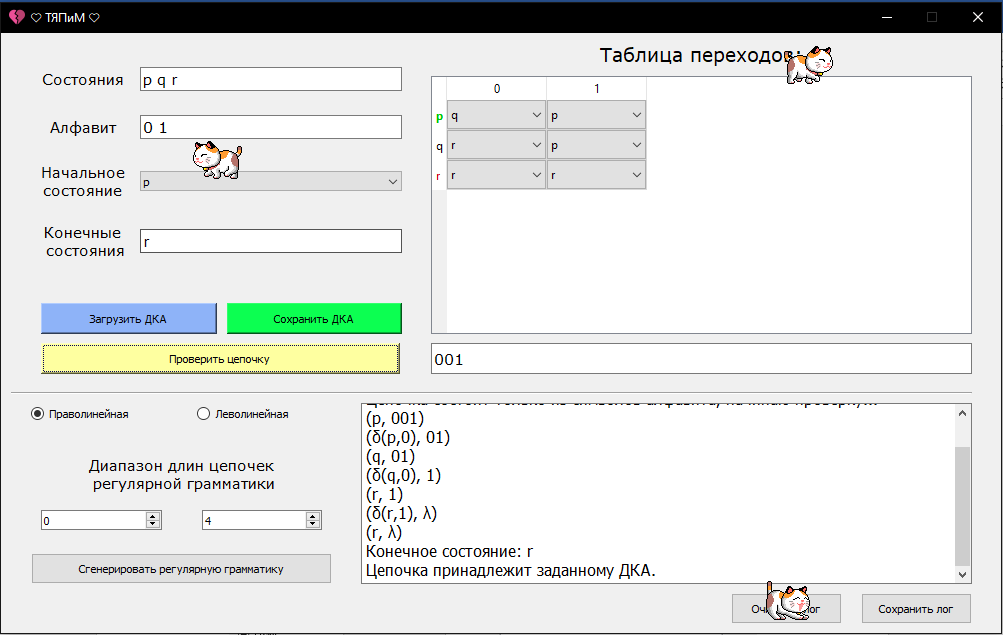
## Рисунок 1. Интерфейс меню программы.



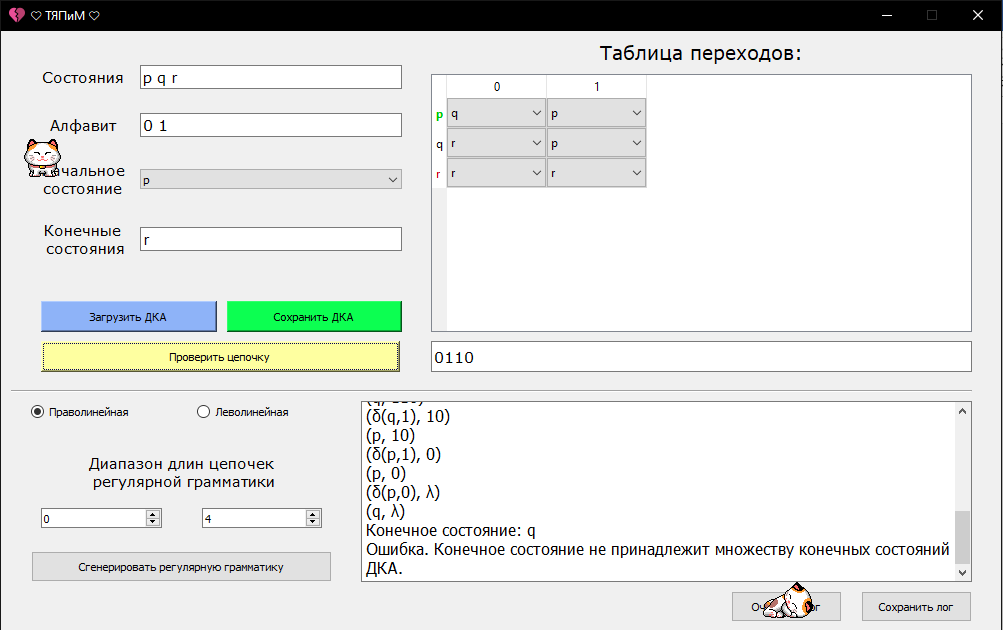
## Рисунок 2. Интерфейс программы.



## Рисунок 3. Интерфейс программы.



## Рисунок 4. Проверка цепочки (принадлежащей языку).



## Рисунок 5. Проверка цепочки (не принадлежащей языку).

## 

## Рисунок 6-7. Герерация 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%BB%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%B2\_%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().hasHeightForWidth())

        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.SetDefaultConstraint)

        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\_grammar")

        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\_grammar")

        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"))