Муниципальное бюджетное общеобразовательное учреждение
средняя общеобразовательная школа № 30 г. Пензы

VIII открытый региональный конкурс
исследовательских и проектных работ школьников
«Высший пилотаж – Пенза» 2026

ПРОЕКТ
на тему: «**АНАЛИЗАТОР ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ**»

Выполнил:
Поликарпов Арсений Витальевич,
учащийся 10А класса
Научный руководитель:
Зайцев Владимир Анатольевич,
учитель информатики

Пенза, 2025 г.

Содержание

Введение

Логика - философская дисциплина и нормативная наука о законах, формах и приёмах интеллектуальной деятельности [1]. Эта наука используется во всех предметных областях, ее основам следуют практически все науки. Например, во время выполнения домашнего задания логика поможет оптимизировать рабочий процесс, распределив время и последовательность каждой задачи или руководствуясь логикой, можно составить наиболее выгодный маршрут передвижения для путешествия, который поможет избежать дополнительных трат.

Практическое решение какой-либо задачи часто требует формального описания на языке алгебры логики. Составленное выражение может иметь довольно сложный вид и для его обработки иногда требуется большая затрата времени, поэтому исследование возможностей автоматизации данного процесса очень важна, а хороший аппарат реализации будет обязательно востребован. Статистика 2025 года показывает, что 68% профессионалов регулярно сталкиваются с логическими ошибками в рабочих документах и презентациях, а 73% признают, что не всегда могут их выявить [2].

Решение сложных логических выражений представляет собой большой алгоритм действий и доставляет не мало проблем. Для этого и используется программа, способная решать логическое выражение, записанное на формальном языке алгебры логики. Разработанная программа позволит автоматически анализировать логическое выражение и определять его истинность.

Объектом данного исследования выступает автоматический анализ синтаксиса языка, а *предметом* – синтаксический анализатор логических исследований.

Цель исследования: создание синтаксического анализатора логических выражений с построением таблицы истинности и возможностью построения логических выражений по известной таблице.

Задачи:

- Провести анализ существующих программ и приложений, работающих с логическими выражениями.
- Изучить принципы анализа синтаксиса формального языка.
- Написать программу на языке Python для анализа логических выражений и построения его таблицы истинности.
- Расширить возможности программы для построения логических выражений по таблице истинности.

Для решения поставленных задач использовались следующие *методы*: поиск и анализ программ и приложений, обобщение и систематизация изученного материала для создания компьютерной программы.

Практическая значимость заключается в возможности использования программы учениками при выполнении домашней работы, учителями при проверке самостоятельных и домашних работ, а также в других областях науки, где необходим анализ сложных логических выражений.

Новизна нашего проекта заключается в учебном эффекте. Анализ синтаксиса формального языка – это довольно сложная задача для программирования, нужно учитывать много факторов: приоритет операторов, особенности синтаксиса, возможные ошибки и неточности ввода. Работа над проектом даёт возможность изучить принципы программирования на языке высокого уровня, выработать практические навыки создания собственной практически применимой программы с графическим интерфейсом и возможность реализации творческого потенциала. Навыки, полученные в ходе проекта, могут быть применены в дальнейшем.

1. Теоретическая часть

1.1 Анализ существующих синтаксических анализаторов логических выражений

На первом этапе работы был проведён поиск существующих программных решений для работы с логическими выражениями и функциями.

Наиболее интересные решения:

- ”Построение таблицы истинности онлайн | СКНФ | СДНФ” (https://programforyou.ru/calculators/postroenie-tablitci-istinnosti-sknf-sdnf).
- ”Таблица истинности онлайн” (https://mathforyou.net/online/discrete/truthtable/).
- ”Таблица истинности онлайн” (https://math.semestr.ru/inf/table.php).

Недостатки всех рассмотренных сервисов: много навязчивой рекламы, ограничение в количестве переменных и длине выражения, не очень удобный интерфейс.

Собственный продукт будет создаваться с учётом недостатков конкурентов, а также с целью проверки собственных возможностей в программировании таких задач и приобретения опыта разработчика.

1.2 Логические операции

В логике есть множество логических операций, такие как инверсия, конъюнкция, дизъюнкция, импликация, эквивалентность. Давайте рассмотрим каждую поподробнее.

Инверсия — логическое отрицание.

Инверсию обозначают несколькими равносильными символами: НЕ, ¬, Ā, not A (A — логическое высказывание). Если высказывание A истинно, то после инверсии оно станет ложным, и наоборот [3].

Таблица истинности для логической функции «инверсия»:

A	неA
1	0
0	1

Конъюнкция — логическое умножение.

Конъюнкцию обозначают символами И, ∧, &, A and B (A, B — простые логические высказывания). Эту логическую операцию выполняют минимум для двух простых высказываний. Она будет истинна только в том случае, когда все высказывания, для которых выполняется конъюнкция, истинны [3].

Таблица истинности для логической функции «конъюнкция»:

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Дизъюнкция — логическое сложение.

Дизъюнкцию обозначают как ИЛИ, ∨, |, A or B. Эту логическую операцию выполняют минимум для двух простых логических высказываний. Она будет истинна в том случае, когда хотя бы одно из высказываний истинно [3].

Таблица истинности для логической функции «дизъюнкция»:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

[2]

Импликация — логическое следование.

Импликация - это сложное логическое выражение, которое истинно во всех случаях, кроме как из истины следует ложь. То есть, данная логическая операция связывает два простых логических выражения, из которых первое является условием (A), а второе (A) является следствием условия (A) [4]. Обозначения: →

Таблица истинности для логической функции «импликация»

A	B	$A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

Эквивалентность — логическая равнозначность.

Эквивалентность — это сложное логическое выражение, которое истинно на равных значениях переменных A и B [4].

Обозначения: ↔

Таблица истинности для логической функции «импликация»:

A	B	$A \equiv B$
0	0	1
0	1	0
1	0	0
1	1	1

Строгая дизъюнкция — сложение по модулю 2 (в теории множеств это объединение двух множеств без их пересечения).

Строгая дизъюнкция истинна, если значения аргументов не равны. Для функции трёх и более переменных результат выполнения операции будет истинным только тогда, когда количество аргументов равных 1, составляющих текущий набор — нечетное. Такая операция естественным образом возникает в кольце вычетов по модулю 2, откуда и происходит название операции [4].

Обозначения: $\oplus$

Таблица истинности для логической функции «импликация»:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Также в логике есть свой порядок действий – приоритет:

1. инверсия

2. конъюнкция
3. дизъюнкция и исключающая дизъюнкция
4. импликация и эквиваленция.

Операции в скобках имеют высший приоритет. Операции с равным приоритетом выполняются слева направо.

Для большего понимания логики разберем пример:

$$0 \vee \neg(1 \wedge 1 \vee 0)$$

1) Первое действие происходит в скобках $(1 \wedge 1 \vee 0)$, так как ”И” имеет больший приоритет выполним выражение $1 \wedge 1$ и получим 1. Подставим его в скобки и получим $0 \vee \neg(1 \vee 0)$.
2) Следующим действием будет логическое сложение в скобках $(1 \vee 0)$. Значением этого выражения является 1. Получаем $0 \vee \neg(1)$.
3) Третьим действием будет логическое отрицание (не 1), ответом на которое является 0. Получаем $0 \vee 0$.
4) Значением этого выражения является 0.

1.3 Построение таблицы истинности

Таблица истинности — таблица, описывающая логическую функцию. Под «логической функцией» в данном случае понимается функция, у которой значения переменных (параметров функции) и значение самой функции выражают логическую истинность. Например, в двузначной логике они могут принимать значения «истина» либо «ложь» (true либо false, 1 либо 0) [5]. Таблица истинности позволяет наглядно показать значение логического выражения при разных значениях переменной.

Пример таблицы истинности логической функции $A \vee \neg(B \wedge A)$:

A	B	$B \wedge A$	$\neg(B \wedge A)$	$A \vee \neg(B \wedge A)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	1
1	1	1	0	1

1.4 Построение СКНФ и СДНФ

Совершенная конъюнктивная нормальная форма (СКНФ) - это КНФ, удовлетворяющая трем условиям:

- не содержит одинаковых элементарных дизъюнкций;
- ни одна из дизъюнкций не содержит одинаковых переменных;
- каждая элементарная дизъюнкция содержит каждую переменную из входящих в данную КНФ.

Любая булева формула, которая не является тождественно истинной, может быть представлена в СКНФ [6].

Правила построения СКНФ по таблице истинности

Для каждого набора переменных, при котором функция равна 0, записывается дизъюнкция, причем переменные, которые имеют значение 1, берутся с отрицанием. Полученное выражение является конъюнкцией этих дизъюнкций [6].

Совершенная дизъюнктивная нормальная форма (СДНФ) - это ДНФ, удовлетворяющая трем условиям:

- не содержит одинаковых элементарных конъюнкций;

- ни одна из конъюнкций не содержит одинаковых переменных;
- каждая элементарная конъюнкция содержит каждую переменную из входящих в данную ДНФ, к тому же в одинаковом порядке.

Любая булева формула, которая не является тождественно ложной, может быть представлена в СДНФ, к тому же единственным образом [6].

Правила построения СДНФ по таблице истинности

Для каждого набора переменных, при котором функция равна 1, записывается произведение, причем переменные, которые имеют значение 0 берут с отрицанием. Полученное выражение является дизъюнкцией этих конъюнкций [6].

Пример построения СКНФ и СДНФ:

A	B	C	F	Выражения для СКНФ	Выражения для СДНФ
0	0	0	**0**	$A \vee B \vee C$	
0	0	1	**1**		$\neg A \wedge \neg B \wedge C$
0	1	0	**0**	$A \vee \neg B \vee C$	
0	1	1	**1**		$\neg A \wedge B \wedge C$
1	0	0	**1**		$A \wedge \neg B \wedge \neg C$
1	0	1	**0**	$\neg A \vee B \vee \neg C$	
1	1	0	**1**		$A \wedge B \wedge \neg C$
1	1	1	**0**	$\neg A \vee \neg B \vee \neg C$	
СКНФ: $F = (A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C)$ ***СДНФ:*** $F = (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge \neg C)$					

2. Практическая часть

2.1 Синтаксический анализатор логических выражений. Построение таблицы истинности

Одной из моих задач по данному проекту было написание Синтаксического анализатора логических выражений.

Для написания программы я выбрал язык Python из-за следующих его особенностей:

- **Лёгкость освоения**. Python прост, логичен, имеет понятный синтаксис, поэтому подходит даже для новичков.
- **Кроссплатформенность**. Программы, написанные на Python, могут запускаться и функционировать на всех типах операционных систем.
- **Скорость разработки**. Чтобы написать программу на Python, нужно значительно меньше кода, чем при разработке, например, на Java.
- **Универсальность**. Python применим практически во всех основных областях, включая веб-разработку, науку о данных, автоматизацию, Интернет вещей, производство и многое другое.
- **Множество инструментов**. Стандартные библиотеки Python способны решать даже сложные задачи. Установка дополнительных модулей, созданных для конкретных целей, помогает при разработке специальных проектов.
- **Масштабируемость**. Возможность адаптации высокоуровневой логики позволяет проектам, разработанным на Python, масштабироваться и расширяться.
- **Широкая поддержка сообщества**. Python имеет активное сообщество разработчиков, которые предлагают помощь, документацию, учебные ресурсы и готовые решения.

2.2 Основные функции программы для построения таблицы истинности

Основную вычислительную задачу выполняет функция **`calcLogic().`**

Это рекурсивная функция. В информатике рекурсия — это метод решения вычислительной задачи, при котором решение зависит от решений более мелких задач того же типа. Рекурсия решает такие рекурсивные задачи с помощью функций, которые вызывают сами себя из собственного кода. Этот подход можно применять ко многим типам задач, и рекурсия является одной из центральных идей информатики [7].

Эта функция делит все логическое выражение на подвыражения и упрощает их до бинарных данных (0 или 1). Первым ее действием является нахождение скобок, если скобки есть в выражении, а после вызов рекурсии, счет и замена скобки на получившееся значение. Следующим шагом является нахождение импликации и эквиваленции, выбирается ближайшая от начала строки операция, и деление выражения на две части до этой операции и после, каждая часть попадает опять в рекурсию и в конечном итоге, когда две части становятся бинарными значениями (0 или 1), вызывается функция **`LogicOR()`**, **`LogicAND()`**, **`LogicEK()`**, **`LogicIM()`**,**`LogicSTROR()`**, которые досчитывают выражение. Отрицание работает по другой схеме. В **`calcLogic()`** есть обработка инверсии, как поиск значений -1 и -0, которые заменяются соответственно на 0 и 1. Рекурсия заканчивается тогда, когда выражение содержит бинарное данное или ошибку – это считается результатом работы функции **`calcLogic()`**.

Код функции **`calcLogic()`**:

```
def calcLogic(s):
  rp = s.find(")")
  if rp > -1:
    lp = s.rfind("(",0,rp)
```

```
        if lp > -1:
          return  calcLogic(s[:lp]+calcLogic(s[lp+1:rp])+s[rp+1:])
        else:
          return "@"
      else:
        k = s.find("#")
        k1 = s.find("=")
        if k < k1:
          return LogicEK(calcLogic(s[:k1]),calcLogic(s[k1+1:]))
        elif k1 < k:
          return LogicIM(calcLogic(s[:k]),calcLogic(s[k+1:]))
        else:
          k = s.find("|")
          k1 = s.find("/")
          if k > k1:
            return LogicOR(calcLogic(s[:k]),calcLogic(s[k+1:]))
          elif k1 > k:
            return LogicSTROR(calcLogic(s[:k1]),calcLogic(s[k1+1:]))
          else:
            k = s.find("^")
            if k > -1:
              return LogicAND(calcLogic(s[:k]),calcLogic(s[k+1:]))
            else:
              if s=="-0":
                return "1"
              elif s=="-1":
                return "0"
              elif "@" in s or len(s) != 1:
                return "@"
              else:
                return s
```

LogicOR() и **LogicAND()** это две вспомогательные функции для функции **calcLogic()**. Они вычисляют значения простых выражений, состоящих из двух значений и одной операции (OR или AND), а также возвращают ошибку (в программе ошибка обозначается значком @), если аргументы не являются бинарными данными (1 или 0) или уже содержат ошибку @.

Код функции **LogicOR()**:

```
def LogicOR(a,b):
  if "@" in a+b or len(max(a,b)) > 1:
    return "@"
  elif int(a)+int(b) == 0:
    return "0"
  else:
    return "1"
```

Код функции **LogicAND()**:

```
def LogicAND(a,b):
  if "@" in a+b or len(max(a,b)) > 1:
    return "@"
  elif int(a)*int(b) == 1:
    return "1"
  else:
    return "0"
```

Код функции **LogicIM()**:

```
def LogicIM(a,b):
  if "@" in a+b or len(max(a,b)) > 1:
    return "@"
  elif a == '1' and b == '0':
    return "0"
  else:
    return "1"
```

Код функции **LogicEK()**:

```
def LogicEK(a,b):
  if "@" in a+b or len(max(a,b)) > 1:
    return "@"
  elif a == b:
    return "1"
  else:
    return "0"
```

Код функции **LogicSTROR()**:

```
def LogicSTROR(a,b):
  if "@" in a+b or len(max(a,b)) > 1:
    return "@"
  elif int(a)+int(b) == 0 or int(a)+int(b) == 2:
    return "0"
  else:
    return "1"
```

Помимо вычислительных функций есть еще две.

Функция **clicked()** выполняется после нажатия на кнопку ввода в графическом интерфейсе программы. Сама функция представляет собой создание и удаление таблицы истинности, перевод переменных в двоичную систему, вызов функции **calcLogic()**.

Код функции **clicked()**:

```
def clicked():
    s=txt.get()
l='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdifghjklmnopqrstuvwxyz'
    s2=s
    T1=[]
    P=[]
    N=[]
    A=[]
    PER=[]
```

```
    lst=[]
    ans=''
    for i in range(len(l)):
        if s.find(l[i])>=0:
            P.append(l[i])
            T1.append(l[i])
    Z='9'+'0'*len(P)
    for i in range(100):
        for j in range(100):
            list = window.grid_slaves(column=i, row=j+6)
            for l in list:
                l.destroy()
    for i in range(2**len(P)):
        s=s2
        if i!=0:
            Z=str(int(Z)+1)
            while Z.find('2')>=0:
                n=Z.find('2')
                Z=Z[:n] + '0' + Z[n+1:]
                Z=Z[:n-1]+str(int(Z[n-1])+1)+Z[n:]
        for x in range(len(P)):
            while s.find(P[x])>=0:
                s=s[:s.find(P[x])] + Z[x+1] + s[s.find(P[x])+1:]
        while s.find(' ')>-1:
            s=s[:s.find(' ')]+s[s.find(' ')+1:]
        if i==0:
            T1.append(s2)
            lst.append(T1)
        T1=[]
        for j in range(1,len(Z)):
            T1.append(Z[j])
        T1.append(calcLogic(s))
        lst.append(T1)
    total_rows = len(lst)
    total_columns = len(lst[0])
    for x in range(total_rows):
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=7)
        for y in range(total_columns):
            self = Entry(window, width=18, fg='blue',
font=('Arial',10,'bold'))
            self.grid(row=x+7, column=y+1)
            self.insert(END, lst[x][y])
```

Функция **`clicked2()`** вызывается по нажатию на кнопку “Информация” в графическом интерфейсе и выводит информацию по использованию программы.

2.3 Графический интерфейс для таблицы истинности

Для создания графического интерфейса я использовал библиотеку Python – **tkinter**. Благодаря ей был создан интерфейс для пользователя.

Возможности программы:

- Ввод логической функции, содержащей:
  - переменные, обозначаемые буквами латинского алфавита;
  - логические операторы:
    - конъюнкция (обозначается ^);
    - дизъюнкция (обозначается |);
    - строгая дизъюнкция (обозначается /);
    - импликация (обозначается #);
    - эквиваленция (обозначается =);
    - инверсия (обозначается –);
  - скобки ( ) для повышения приоритета.
- Определение истинности логической функции для всех возможных значений и построение таблицы истинности.

Внешний вид интерфейса представлен на рисунке № 1.

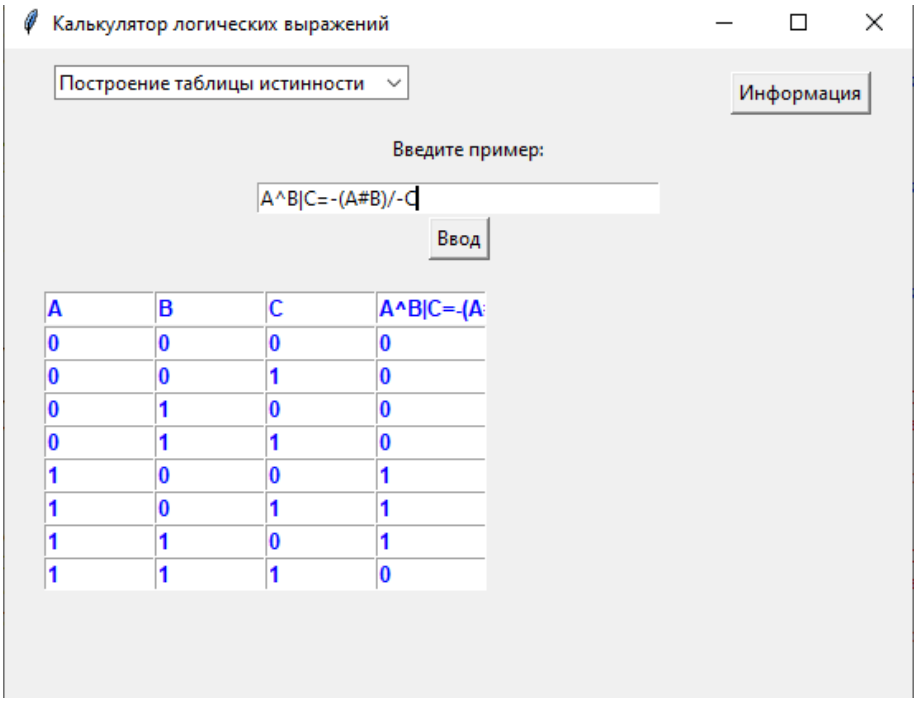

Рис. 1. Интерфейс программы «Калькулятор логических выражений»

2.4 Синтаксический анализатор логических выражений. Построение СКНФ И СДНФ

Основные вычислительные задачи выполняют функции **`SKNF()`** и **`SDNF()`**.

Рассмотрим функцию **`SKNF()`**. Эта функция собирает все данные из таблицы истинности и ищет все строки, в последнем столбце которых есть 0, после если переменные переменная является 1, то инверсирует эту переменную, а с переменными, являющимися 0, ничего не делает. Далее функция “соединяет” все переменные дизъюнкцией и так происходит со всеми строками. По итогу функция “соединяет” все получившиеся выражения конъюнкцией и выводит СКНФ на экран.

Функция **`SDNF()`** поступает аналогично **`SKNF()`**. Эта функция собирает все данные из таблицы истинности и ищет все строки, в последнем столбце которых есть 1, после если переменные переменная является 0, то инверсирует эту переменную, а с переменными,

являющимися 1, ничего не делает. Далее функция “соединяет” все переменные конъюнкцией и так происходит со всеми строками. По итогу функция “соединяет” все получившиеся выражения дизъюнкцией и выводит СДНФ на экран.

Код функции **SKNF()**:

```
def SKNF():
  sknfN=[]
  sknfE=''
  H=[]
  h=[]
  s=''
  for i in range(1,total_rows):
    if self[i*total_columns+total_columns-1].get()=='0':
      sknfN.append(i)
  for i in range(len(sknfN)):
    for x in range(total_columns-1):
      if self[sknfN[i]*total_columns+x].get()=='1':
        h.append('-'+self[x].get())
      else:
        h.append(self[x].get())
    for k in range(len(h)):
      if k!=0:
        s+='|'
      s+=h[k]
    H.append(s)
    s=''
    h=[]
  for i in range(len(H)):
    if i!=0:
      sknfE+='^'
    sknfE+='('+H[i]+')'
  return sknfE
```

Код функции **SDNF()**:

```
def SDNF():
  sdnfN=[]
  sdnfE=''
  H=[]
  h=[]
  s=''
  for i in range(1,total_rows):
    if self[i*total_columns+total_columns-1].get()=='1':
      sdnfN.append(i)
  for i in range(len(sdnfN)):
    for x in range(total_columns-1):
      if self[sdnfN[i]*total_columns+x].get()=='0':
```

```
        h.append('-'+self[x].get())
      else:
        h.append(self[x].get())
    for k in range(len(h)):
      if k!=0:
        s+='^'
      s+=h[k]
    H.append(s)
    s=''
    h=[]
  for i in range(len(H)):
    if i!=0:
      sdnfE+='|'
    sdnfE+='('+H[i]+')'
  return sdnfE
```

Помимо вычислительных функций есть еще несколько.

Функция **`clicked3()`** строит таблицу истинности, а **`clicked4()`** проверяет правильность вписанных в таблицу данных и вывод СКНФ И СДНФ.

Функция **`clicked2()`** вызывается по нажатию на кнопку “Информация” в графическом интерфейсе и выводит информацию по использованию программы.

2.3 Графический интерфейс для СКНФ и СДНФ

Для создания графического интерфейса я использовал библиотеку Python – **tkinter**. Благодаря ей был создан интерфейс для пользователя.

Внешний вид интерфейса представлен на рисунке № 2.

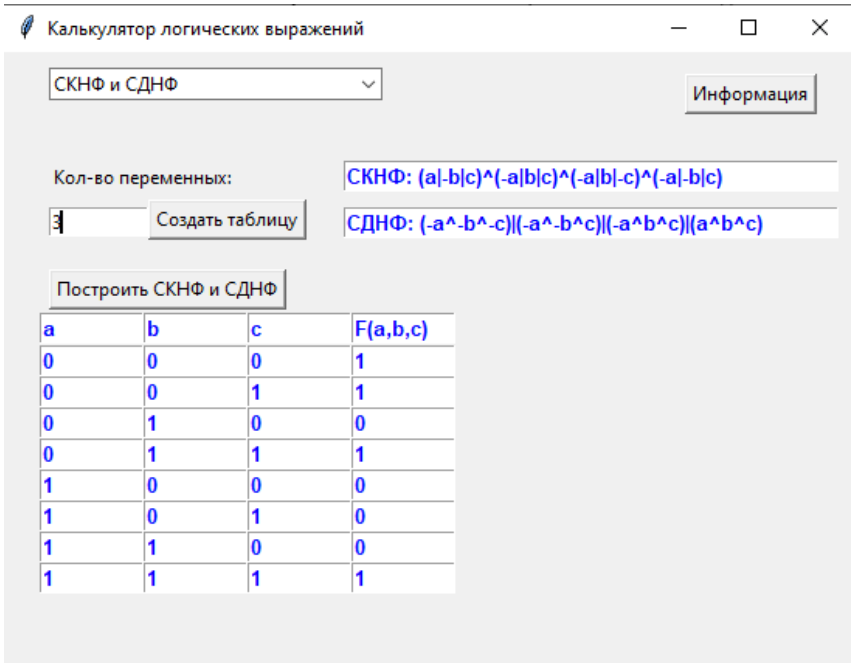

Рис. 2. Интерфейс программы «Калькулятор логических выражений»

Код остальных функций программы представлен в приложении № 1.

Заключение

Анализ существующих решений на рынке позволил нам ознакомиться с разнообразными подходами к обработке логических выражений, а также выявить их сильные и слабые стороны. Это знание стало основой для проектирования нашего анализатора, который не имеет рекламы, может работать оффлайн, умеет работать с большим числом переменных и сочетает в себе простоту и эффективность.

В процессе изучения основ синтаксического анализа формальных языков были рассмотрены различные алгоритмы, такие как основные логические операции (инверсия, конъюнкция, дизъюнкция), рекурсивный алгоритм – это позволило выбрать необходимые инструменты для построения корректной логической модели. Эти знания легли в основу программы и обеспечили её надёжность и точность.

Реализация синтаксического анализатора на языке Python продемонстрировала практическое применение теоретических знаний. Была создана программа, способная не только анализировать введённые логические выражения, но и строить соответствующие таблицы истинности, что является конечной целью нашего проекта. Так же была добавлена возможность строить логические выражения по таблице в СДНФ и СКНФ. Данное приложение может быть полезным для студентов, изучающих основы логики и программирования, а также для разработчиков в области автоматизации обработки данных.

В ходе выполнения проекта были приобретены ценные навыки в области программирования, синтаксического анализа и работы с логическими выражениями. Кроме того, проект стал отличной площадкой для углубленного изучения принципов разработки программного обеспечения, учета пользовательского опыта и оптимизации алгоритмов.

Подводя итог, можно сказать, что данный проект успешно достиг поставленной цели и задач, предоставив функциональный инструмент для работы с логическими выражениями. Результаты нашей работы могут послужить основой для последующих исследований и разработок в данной области, а также вдохновить других специалистов на создание инновационных решений в сфере логики и программирования.

Список источников

1. Логика // Википедия. Свободная энциклопедия: [сайт]. – URL: https://ru.wikipedia.org/wiki/Логика (дата обращения: 11.11.2024).
2. Статистика // Skypro. [сайт]. – URL: https://sky.pro/wiki/analytics/tipichnye-logicheskie-oshibki-kak-ih-raspoznat-i-ne-dopuskat/
3. Логические выражения (инверсия, дизъюнкция, конъюнкция) // Фоксфорд. Свободная энциклопедия: [сайт]. – URL: https://foxford.ru/wiki/informatika/logicheskie-vyrazheniya?utm_referrer=https%3A%2F%2Fyandex.ru%2F(дата обращения: 08.04.2025)
4. Логические выражения (импликация, эквиваленция, строгая дизъюнкция) // Свободная энциклопедия: [сайт]. – URL: https://spravochnick.ru/informatika/algebra_logiki_logika_kak_nauka/logicheskie_operacii_i_ih_svoystva/
5. Таблица истинности // Википедия. Свободная энциклопедия: [сайт]. – URL: https://ru.wikipedia.org/wiki/Таблица_истинности(дата обращения: 15.04.2025)
6. Построение СКНФ и СДНФ по таблице истинности. // Свободная энциклопедия: [сайт]. – URL: https://spravochnick.ru/informatika/algebra_logiki_logika_kak_nauka/postroenie_sknf_i_sdnf_po_tablice_istinnosti/
7. Рекурсия // Википедия. Свободная энциклопедия: [сайт]. – URL: https://en.wikipedia.org/wiki/Recursion_(computer_science) (дата обращения: 15.04.2025)

Приложение 1. Код функций программы «Калькулятор логических выражений» которые не были рассмотрены в основном тексте работы

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter.ttk import Combobox

def clicked2():
   messagebox.showinfo('Информация', 'Калькулятор логических выражений - это программа созданная для быстрого счета логических выражений. \nПрограмма допускает такие символы как: \n «a»,«b»...«Z» - Переменные\n «|» - Логическое сложение(дизъюнкция)\n «^» - Логическое умножение(конъюнкция)\n «#» - Импликация\n «=» - Эквиваленция\n «/» - Исключа́ющее или(строгая дизъюнкция)\n «-» - Отрицание (инверсия)\n «(...)» - Скобки')

def clicked22():
   messagebox.showinfo('Информация', 'Калькулятор логических выражений - это программа созданная для быстрого счета логических выражений. \nСКНФ и СДНФ — понятия в логике, которые обозначают совершенные конъюнктивную и дизъюнктивную нормальные формы. \n СКНФ — это конъюнктивная нормальная форма (КНФ), в которой нет одинаковых элементарных дизъюнкций и все дизъюнкции состоят из одного и того же набора переменных, в который каждая переменная входит только один раз. \n СДНФ — это дизъюнктивная нормальная форма (ДНФ), в которой нет одинаковых элементарных конъюнкций и все конъюнкции состоят из одного и того же набора переменных, в который каждая переменная входит только один раз.')
def clicked3():
  btn4.place(x=30, y=140)
  s=txt2.get()
  A1=[['']*int(s)+['F()']]
  A=[]
  KKK=1
  global self,entry_widgets,total_columns,total_rows,A2
  A1=[['']*int(s)+['F()']]
  A2=[]
  self=[]
  entry_widgets = []
  total_columns=int(s)+1
  total_rows=2**int(s)+1
  for i in range(100):
      for j in range(100):
          list = window.grid_slaves(column=i, row=j+8)
          for l in list:
              l.destroy()
  Z='9'+'0'*int(s)
  for i in range(2**int(s)):
      if i!=0:
          Z=str(int(Z)+1)
```

```
            while Z.find('2')>=0:
                n=Z.find('2')
                Z=Z[:n] + '0' + Z[n+1:]
                Z=Z[:n-1]+str(int(Z[n-1])+1)+Z[n:]
        for i in range(len(Z)-1):
          A.append(Z[i+1])
        A+=['']
        A1.append(A)
        A=[]
  for i in range(len(A1)):
    for j in range(len(A1[i])):
      A2.append(A1[i][j])
  #print(A1)
  for x in range(total_rows):
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=8)
        for y in range(total_columns):

            #self[x+y*len(x)] = Entry(window, width=18, fg='blue',
            #                 font=('Arial',10,'bold'))
            self.append(Entry(window, width=9, fg='blue',font=('Arial',10,'bold')))
            self[y+x*total_columns].grid(row=x+8, column=y+1)
            self[y+x*total_columns].insert(END, A1[x][y])
            entry_widgets.append(self)
def clicked4():
  l=1
  for i in range(total_columns):
    if self[i].get()=='':
      print('Введите название для переменных')
      l=0
  for i in range(1,total_rows):

    if     self[i*total_columns+total_columns-1].get()!='1'     and     self[i*total_columns+total_columns-
1].get()!='0':
      print('Введите значение для переменных')
      l=0
  if l==1:
    for i in range(total_columns):
      A2[i]=self[i].get()
    for i in range(1,total_rows):
      A2[i*total_columns+total_columns-1]=self[i*total_columns+total_columns-1].get()
    A2[total_columns-1]='F('
    for i in range(total_columns-1):
      if i!=0:
        A2[total_columns-1]+=','
```

```
      A2[total_columns-1]+=str(A2[i])
    A2[total_columns-1]+=')'
    self1=Entry(window, width=9, fg='blue',font=('Arial',10,'bold'))
    self1.grid(row=8,column=total_columns)
    self1.insert(END, A2[total_columns-1])
    global S1,S2
    S1=Entry(window, width=45, fg='blue',font=('Arial',10,'bold'))
    S2=Entry(window, width=45, fg='blue',font=('Arial',10,'bold'))
    S1.place(x=220, y=70)
    S2.place(x=220, y=100)
    S1.insert(0, 'СКНФ: '+str(SKNF())+'                                    ')
    S2.insert(0, 'СДНФ: '+str(SDNF())+'                                    ')

def COMBO(*arg):
  if combo.get()=="Построение таблицы истинности" or arg==11:
        for i in range(100):
                for j in range(100):
                   list = window.grid_slaves(column=i, row=j)
                   for l in list:
                      l.destroy()
        lbl2.place_forget()
        if 'S1' in globals() or 'S2' in globals():
          S1.place_forget()
          S2.place_forget()
        txt2.place_forget()
        btn4.place_forget()
        btn33.place_forget()
        btn6.place_forget()
        btn3.place_forget()
        lbl1 = Label(window, text="")
        lbl1.grid(column=0, row=0)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=1)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=2)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=3)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=4)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=5)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=6)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=7)
```

```
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=8)

        #lbl = Label(window, text="     Введите пример: ")
        lbl.place(relx=.5, y=60, anchor="c")
        #txt = Entry(window,width=40)
        txt.place(relx=.5, y=90, anchor="c")
        btn5.place(relx=.5, y=115, anchor="c")
        #txt.bind("<Return>", clicked)

        btn2.place(relx=.8, rely=.1, anchor="sw")
        table_frame = ttk.Frame(window, padding="1")
        table_frame.grid(row=1, column=0)
        lst = [('A','B','A^B'),(0,0,0),(0,1,0),(1,0,0),(1,1,1)]
        total_rows = len(lst)
        total_columns = len(lst[0])

if combo.get()=="СКНФ и СДНФ":
        for i in range(100):
            for j in range(100):
                list = window.grid_slaves(column=i, row=j)
                for l in list:
                    l.destroy()
        lbl.place_forget()
        txt.place_forget()
        btn2.place_forget()
        btn3.place_forget()
        btn5.place_forget()
        lbl2.place(x=30, y=70)
        txt2.place(x=30, y=100)
        #txt2.bind("<Return>", clicked3)
        btn6.place(x=94, y=95)
        btn33.place(relx=.8, rely=.1, anchor="sw")
        #btn4.grid(column=1, row=6)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=0)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=1)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=2)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=3)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=4)
        lbl1 = Label(window, text="      ")
```

```
        lbl1.grid(column=0, row=5)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=6)
        lbl1 = Label(window, text="      ")
        lbl1.grid(column=0, row=7)
window = Tk()
window.title("Калькулятор логических выражений")
window.geometry('550x400')
btn4 = Button(window, text="Построить СКНФ и СДНФ", command=clicked4)
lbl = Label(window, text="     Введите пример: ")
txt = Entry(window,width=40)
btn2 = Button(window, text="Информация", command=clicked2)
#global btn4
lbl2 = Label(window, text="Кол-во переменных:                          ")
txt2 = Entry(window,width=10)
btn3 = Button(window, text="Информация", command=clicked2)
btn33 = Button(window, text="Информация", command=clicked22)
#btn4 = Button(window, text="Построить СКНФ и СДНФ", command=clicked4)
btn5 = Button(window, text="Ввод", command=clicked)
btn6 = Button(window, text="Создать таблицу", command=clicked3)
var = StringVar()
combo = Combobox(window,width=32, textvariable=var)
combo['values'] = ("Построение таблицы истинности","СКНФ и СДНФ")
combo['state'] = 'readonly'
combo.current(0)  # установите вариант по умолчанию
combo.place(x=30, y=10)
var.trace_add('write',COMBO)
print(COMBO(11))

window.mainloop()
#scrollbar = ttk.Scrollbar(orient="vertical", command=window.yview)
#scrollbar.pack(side=RIGHT, fill=Y)

#window["yscrollcommand"]=scrollbar.set
```

Рецензия

учителя высшей квалификационной категории Зайцева Владимира Анатольевича

на работу «Анализатор логических выражений»

учащегося 10 класса МБОУ СОШ №30 г. Пензы,

Поликарпова Арсения Витальевича

Тема научной работы представляет высокий учебно-исследовательский интерес, она актуальна и разработанное приложение может быть практически применимо.

В процессе работы была проведена большая теоретическая и практическая деятельность и разработана компьютерная программа для работы с логическими выражениями.

Программа представляет из себя по сути анализатор синтаксиса формального языка, что для учащегося средней школы является нетривиальной задачей. Поэтому разработка представляет большой интерес также и с точки зрения учебного эффекта.

Продукт, созданный в рамках работы (программа), может быть доработан в будущем: улучшен дизайн интерфейса, эргономика взаимодействий с целью повышения востребовательности программы в реальных режимах использования.

Работа имеет четкую структуру, содержание её основных частей изложено логично и соответствует заявленной теме. Работа выполнена с большой долей самостоятельности и рекомендуется к высокой оценке.

Зайцев В. А., 12.01.2026

учитель высшей квалификационной категории

МБОУ СОШ № 30 г. Пензы