Міністерство освіти і науки України

Нацональний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Лабораторна робота № 1

з дисципліни

**«Дискретна математика»**

Виконав:

Студент групи КН-114

**Кратко Денис**

Викладач:

**Мельникова Н.І.**

Львів – 2019

**Тема:** Моделювання основних логічних операцій

**Мета роботи:** Ознайомитись на практиці із основними поняттями

математичної логіки, навчитись будувати складні висловлювання за

допомогою логічних операцій та знаходити їхні істинностні значення

таблицями істинності, використовувати закони алгебри логіки, освоїти

методи доведень

**Теоретичні відомості:**

1.1. Основні поняття математичної логіки. Логічні операції

Просте висловлювання (атомарна формула, атом) – це розповідне

речення, про яке можна сказати, що воно істинне (T або 1) або хибне (F

або 0), але не те й інше водночас.

Складне висловлювання – це висловлювання, побудоване з простих

за допомогою логічних операцій (логічних зв’язок). Найчастіше вживаними

операціями є 6: заперечення (читають «не», позначають

,

–

), кон’юнкція

(читають «і», позначають



), диз’юнкція (читають «або», позначають



),

імплікація (читають «якщо ..., то», позначають



), альтернативне «або»

(читають «додавання за модулем 2», позначають



), еквівалентність

(читають «тоді і лише тоді», позначають



).

Запереченням довільного висловлювання Р називають таке

висловлювання

P

, істиносне значення якого строго протилежне

значенню Р. Кон’юнкцією або логічним множенням двох висловлювань

P та Q називають складне висловлювання P Q, яке набуває істинного

значення тільки в тому випадку, коли істинні обидві його складові.

Диз’юнкцією або логічним додаванням двох висловлювань P та Q

називають складне висловлювання P Q, яке набуває істинного значення в

тому випадку, коли істинною є хоча б одна його складова. Імплікацією

двох висловлювань P та Q називають умовне висловлювання «якщо P, то

Q» (P



Q), яке прийнято вважати хибним тільки в тому випадку, коли

передумова (антецедент) P істинна, а висновок (консеквент) Q хибний. У

будь-якому іншому випадку його вважають істинним. Альтернативним

“або” двох висловлювань P та Q називають складне висловлювання P



Q,

яке набуває істинного значення тоді і лише тоді, коли P та Q мають різні

логічні значення, і є хибним в протилежному випадку. Еквіваленцією

двох висловлювань P та Q називають складне висловлювання P



Q, яке





набуває істинного значення тоді і лише тоді, коли P та Q мають однакові

логічні значення, і є хибним в протилежному випадку, тобто логічно

еквівалентні складні висловлювання – це висловлювання, які набувають

однакових значень істинності на будь-якому наборі істиносних значень

своїх складових.

Тавтологія – формула, що виконується у всіх інтерпретаціях

(тотожно істинна формула). Протиріччя – формула, що не виконується у

жодній інтерпретації (тотожно хибна формула). Формулу називають

нейтральною, якщо вона не є ні тавтологією, ні протиріччям (для неї існує

принаймні один набір пропозиційних змінних, на якому вона приймає

значення Т, і принаймні один набір, на якому вона приймає значення F).

Виконана формула – це формула, що не є протиріччям (інакше кажучи,

вона принаймні на одному наборі пропозиційних змінних набуває

значення Т).

1.2. Закони логіки висловлювань

А В

Закони асоціативності

(P  Q)  R  P  (Q  R) (P  Q)  R  P  (Q  R)

Закони комутативності

P  Q  Q  P P  Q  Q  P

Закони ідемпотентності

P P  P P P  P

Закони дистрибутивності

P  (Q  R)  (P  Q)  (P  R) P  (Q  R)  (P  Q)  (P  R)

Закони доповнення

закон виключення третього:

P  (P)  T

закон протиріччя:

P  (P)  F

закон подвійного заперечення

P  P

Закони де Моргана

(P  Q)  P  Q (P  Q)  P  Q

Закони поглинання

(P  Q)  P  P (P  Q)  P  P

Співвідношення для сталих (закони тотожності та домінування)

PT T

PT  P

(тот)

PF  P

(тот)

PF  F

1.3. Логіка першого ступеня. Предикати і квантори. Закони

логіки першого ступеня

Предикат – це твердження, яке містить змінні та приймає значення

істини чи фальші залежно від значень змінних; п-місний предикат – це

предикат, що містить п змінних х1,..., хп.

Квантор - логічний оператор, що перетворює будь-який предикат на

предикат меншої місності, зв'язуючи деякі змінні початкового предиката.

Вживаються два квантори: узагальнення (універсальний) (позначається )

та приналежності (екзистенціальний) (позначається ). Для будь-якого

предиката Р(х) вирази читаються як «всі x мають

властивість Р(х)» та «існує (бодай один) х, що має властивість Р(х)»

відповідно.

Перехід від P(x) до х P(x) або х P(x) називають зв’язуванням

предметної змінної х, а саму змінну х – зв’язаною (заквантованою).

Незв’язану змінну називають вільною. У виразах х P(x) або х P(x)

предикат належить області дії відповідного квантора. Формулу, що не

містить вільних змінних, називають замкненою.

Якщо D={a1,..., aп} – скінченна предметна область змінної х у

предикаті P(x), то можна скористатись логічними еквівалентностями

х P(x)=

( ) ... ( ) P a1   P an

та х P(x)= ( ) ... ( ) P a1   P an

.

Обчислення предикатів, у якому квантори можуть зв’язувати лише

предметні змінні, але не можуть зв’язувати предикати, називають

обчисленням першого порядку. Обчислення, у яких квантори можуть

зв’язувати не лише предметні змінні, але й предикати, функціональні

символи чи інші множини об’єктів, називають обчисленнями вищих

порядків.

Основні закони логіки першого ступеня (логіки предикатів):

1.

(xP(x))  x(P(x)),xP(x)  x(P(x)) .

2.

(xP(x))  x(P(x)), xP(x))  x(P(x)).

3.

x(P(x) Q(x))  xP(x) xQ(x).

4.

x(P(x) Q(x))= xP(x) xQ(x)  

 

 

5.

x(P(x) Q)= xP(x) Q .

6.

x(P(x) Q)= xP(x) Q

7.  x ( P ( x )  Q ) =  x P ( x )  Q.

8.  x ( P ( x )  Q ) =  x P ( x )  Q.

9.  x  y P ( x, y ) =  y  x P ( x, y ).

10.  x  y P( x, y ) =  y  x P( x, y ).

11.

xP(x)  tP(t), xP(x)  tP(t).

12.

xP  P, xP  P.

Випереджена нормальна форма – формула, записана у вигляді

Q1x1Q2x2...QnxnM, де кожне Qixi (i = 1,2,...,n) – це xi або xi

, а формула M

не містить кванторів. Вираз Q1x1...Qnxn називають префіксом, а M –

матрицею формули, записаної у випередженій нормальній формі.

1.4. Методи доведень

При доведенні теорем застосовують логічну аргументацію. Доведення

в інформатиці – невід’ємна частина перевірки коректності алгоритмів.

Необхідність доведення виникає, коли нам потрібно встановити істинність

висловлювання виду (P



Q ). Існує декілька стандартних типів доведень.

1. Пряме міркування. Допускаємо, що висловлювання Р істинне і

показуємо справедливість Q. Такий спосіб доведення виключає

ситуацію, коли Р істинне, а Q хибне, оскільки саме в цьому і лише в

цьому випадку імплікація P



Q набуває хибного значення (див. табл.

1.1).

2. Обернене міркування. Допускаємо, що висловлювання Q хибне і

показуємо помилковість Р. Фактично прямим способом перевіряємо

істинність імплікації (



Q



P), що згідно з прикладом 1.5 (правилом

контрапозиції) логічно еквівалентне істинності вихідного твердження

(P



Q ).

3. Метод «від протилежного». У допущенні, що висловлювання Р

істинне, а Q хибне, використовуючи аргументоване міркування,

одержимо протиріччя. Цей спосіб заснований на тому, що імплікація (P



Q) набуває хибного значення лише тоді, коли Р істинне, а Q хибне.

4. Принцип математичної індукції – це така теорема:



Теорема. Нехай Р(п) – предикат, визначений для всіх натуральних п.

Допустимо, що

1) Р(1) істинне і

2) k



1 імплікація (P(k)



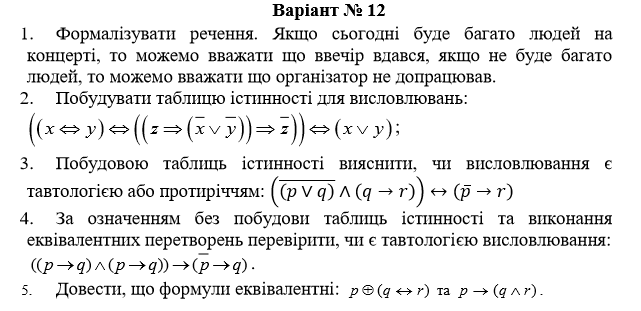
P(k+1)) є вірною.

Тоді Р(п) істинне при будь-якому натуральному п.

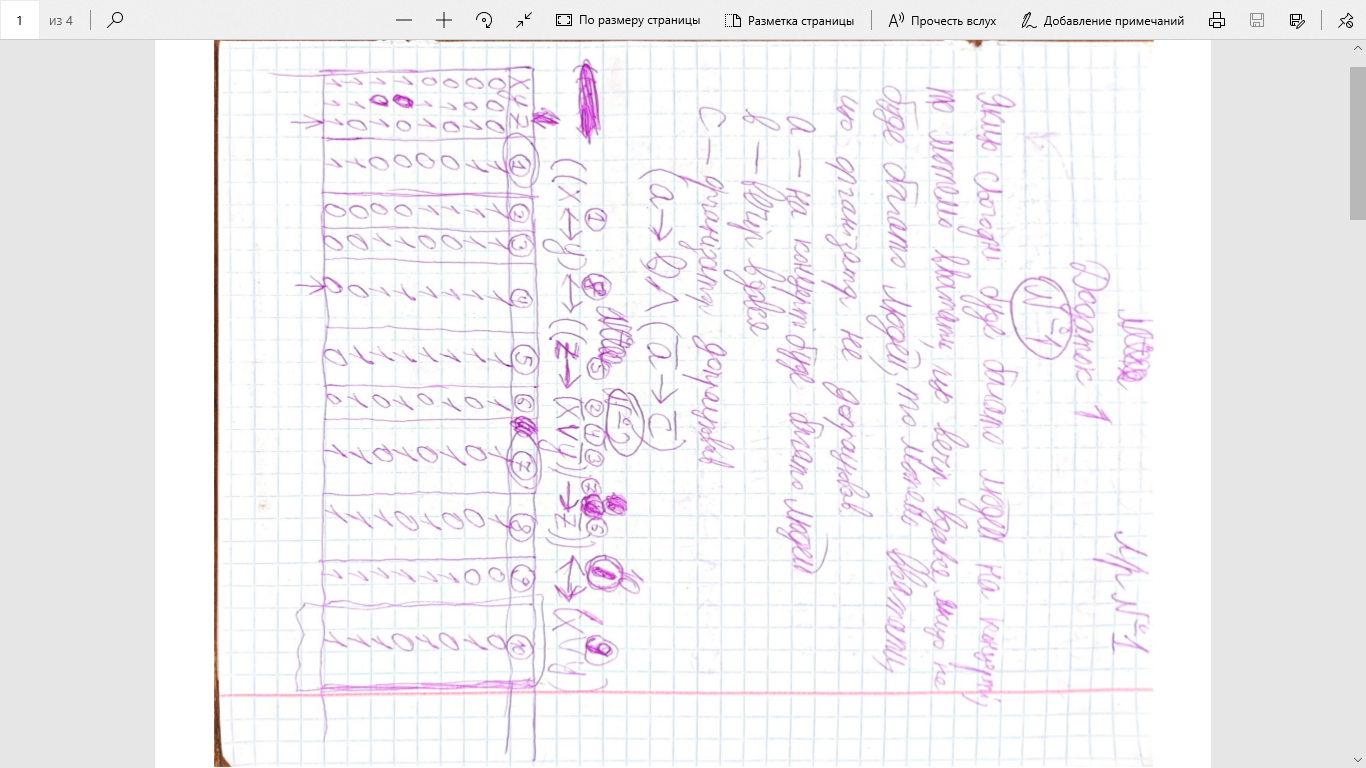
Завдання (12 варіант)

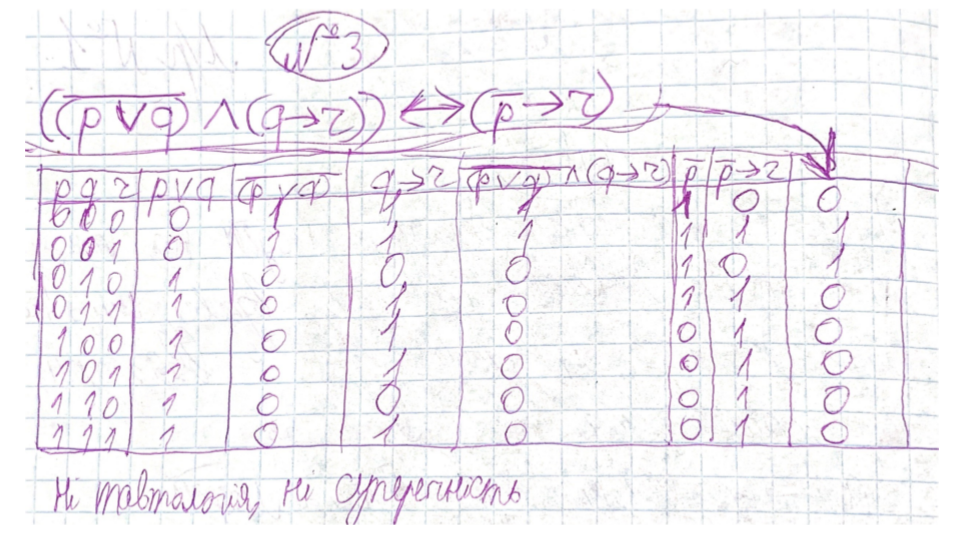
Додаток 1

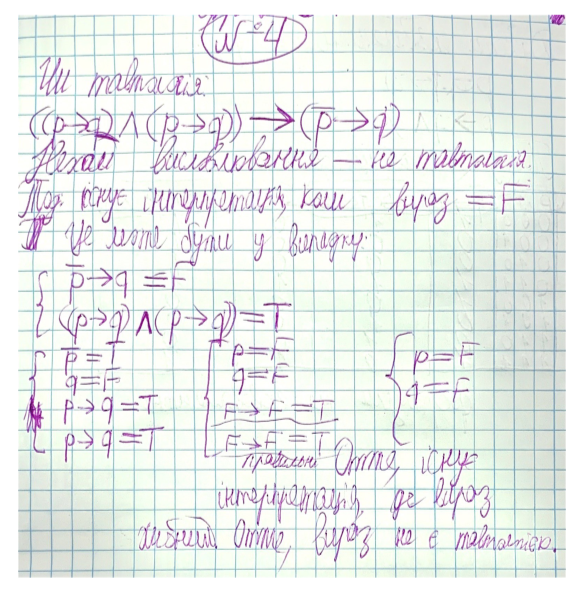
Завдання

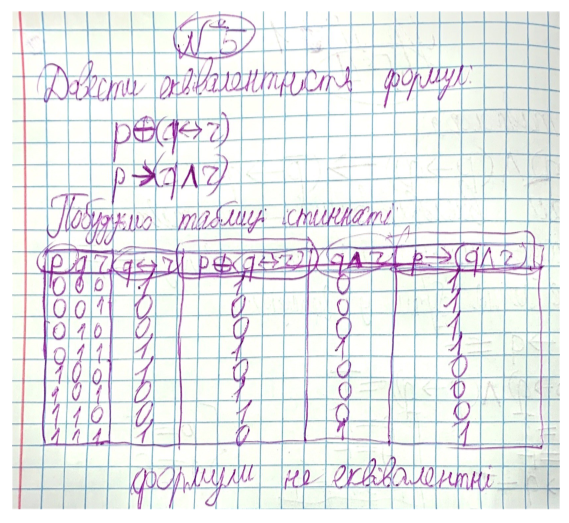


Розв’язок



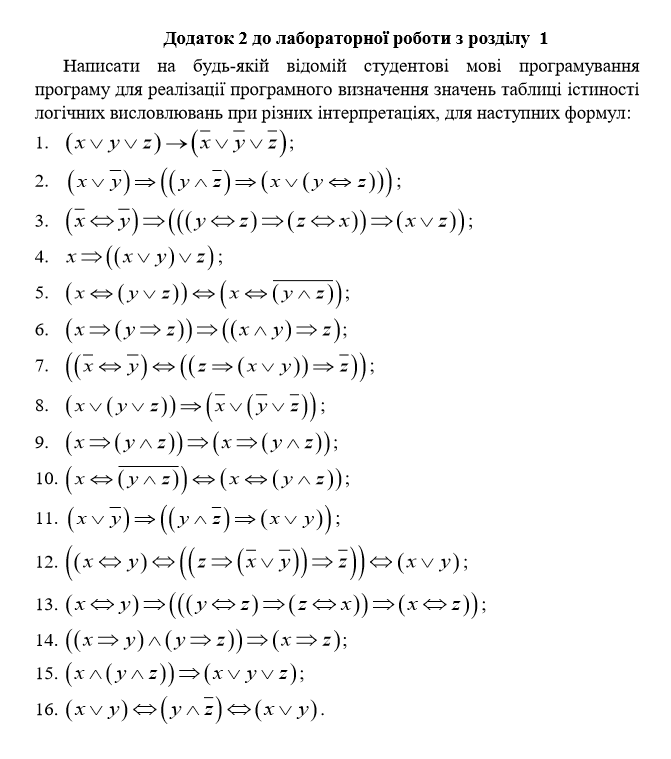






Додаток 2

Завдання



Програмна реалізація

# ((((x)=(y))=(((z)>((-(x))+(-(y))))>(-(z))))=((x)+(-(y))))

class Expression:

def \_\_init\_\_(self, left, right, repr):

self.\_left = left

self.\_right = right

self.\_repr = repr

def left(self, atoms\_values):

return self.\_left.value(atoms\_values)

def right(self, atoms\_values):

return self.\_right.value(atoms\_values)

def value(self, atoms\_values):

pass

class Negative(Expression):

def value(self, atoms\_values):

return not self.right(atoms\_values)

class Conjunction(Expression):

def value(self, atoms\_values):

return self.left(atoms\_values) and self.right(atoms\_values)

class Disjunction(Expression):

def value(self, atoms\_values):

return self.left(atoms\_values) or self.right(atoms\_values)

class Implication(Expression):

def value(self, atoms\_values):

return not self.left(atoms\_values) or self.right(atoms\_values)

class Equivalence(Expression):

def value(self, atoms\_values):

return self.left(atoms\_values) == self.right(atoms\_values)

class T(Expression):

def value(self, atoms\_values):

return True

class F(Expression):

def value(self, atoms\_values):

return False

class NonePrimitive(Expression):

def \_\_init\_\_(self):

self.\_left = None

self.\_right = None

def value(self, atoms\_values):

return None

class Atom(Expression):

def \_\_init\_\_(self, left, right, repr, letter):

super().\_\_init\_\_(left, right, repr)

self.\_letter = letter

def value(self, atoms\_values):

return atoms\_values[self.\_letter]

def str\_to\_expression\_recurse(s):

# print('Рекурсія', s)

# перевір мінус

# врахуй дужки

# врахуй представлення

# print("ВХІД:", s)

operators = {'-': Negative, '\*': Conjunction, '+': Disjunction, '>': Implication, '=': Equivalence}

primitives = {'0': F, '1': T}

if len(s) >= 4:

inset = 0

for i, c in enumerate(s):

if c == '(':

inset += 1

elif c == ')':

inset -= 1

elif (inset == 1) and c in operators:

a = s[1:i]

op = c

b = s[i+1:-1]

try:

return operators[op](str\_to\_expression\_recurse(a), str\_to\_expression\_recurse(b), s[1:-1])

except KeyError:

raise Exception('Містить недопустимі оператори')

elif s:

try:

return primitives[s](NonePrimitive(), NonePrimitive(), s[1])

except KeyError:

return Atom(NonePrimitive(), NonePrimitive(), s[1], s[1])

else:

return NonePrimitive();

def str\_to\_expression(s):

return str\_to\_expression\_recurse(s)

def print\_full\_table(expression, expression\_atoms):

print(''.join(expression\_atoms))

for i in range(2 \*\* len(expression\_atoms)):

prepare\_bin = bin(i)[2:].rjust(len(expression\_atoms), '0')

prepare\_dict = {}

for j in range(len(expression\_atoms)):

prepare\_dict[expression\_atoms[j]] = bool(int(prepare\_bin[j]))

print(prepare\_bin, expression.value(prepare\_dict))

def main():

print('''---------- ІНСТРУКЦІЯ ----------

Ця програма будує таблицю істинності для введеного виразу. Атоми і їх кількість визначаються автоматично.

Вводьте вираз без пробілів, усі висловлювання загортайте в дужки.

Наприклад: ((((x)=(y))=(((z)>((-(x))+(-(y))))>(-(z))))=((x)+(-(y))))

Оператори:

- заперечення

\* кон\'юнкція

+ диз\'юнкція

> імплікація

= еквівалентність

--------------------------------

''')

entered\_expression = input('Введіть вираз: ')

expression\_atoms = set()

for i in entered\_expression:

if i in 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ':

expression\_atoms.add(i)

expression\_atoms = sorted(list(expression\_atoms))

try:

expression = str\_to\_expression(entered\_expression)

print\_full\_table(expression, expression\_atoms)

except:

print('Неправильно введений вираз')

if \_\_name\_\_ == '\_\_main\_\_':

main()

Результат виконання програми

