*Титульна сторінка*

АННОТАЦІЯ

Зміст

[Вступ 5](#_Toc126503238)

[Постановка задачі 6](#_Toc126503239)

[Основна частина 7](#_Toc126503240)

[1. Теоретичні основи формальних мов і граматик 7](#_Toc126503241)

[1.1. Означення формальних мов і граматик 7](#_Toc126503242)

[1.2. Класифікація граматик 8](#_Toc126503243)

[1.3. Нотація Бекуса — Наура 9](#_Toc126503244)

[1.4. Синтаксичний аналіз 10](#_Toc126503245)

[1.5. Рекурсивний спуск 12](#_Toc126503246)

[2. Існуючі інструменти для роботи з формулами 13](#_Toc126503247)

[3. Розробка web-додатку 14](#_Toc126503248)

[3.1. Загальний опис програми 14](#_Toc126503249)

[3.2. Опис формули 15](#_Toc126503250)

[3.3. Дії з формулами 17](#_Toc126503251)

[3.4. Інтерфейс користувача 17](#_Toc126503252)

[4. Застосування результатів розробки 18](#_Toc126503253)

[Висновки 19](#_Toc126503254)

[Література 20](#_Toc126503255)

[Додатки 21](#_Toc126503256)

[Додаток А 21](#_Toc126503257)

[Додаток Б 22](#_Toc126503258)

# Вступ

# Постановка задачі

1. Вивчити теоретичні основи формальних мов і граматик.
2. Ознайомитися з існуючими інструментами для роботи з формулами
3. Розробити web-додаток
4. Розглянути застосування результатів розробки

# Основна частина

## Теоретичні основи формальних мов і граматик

### Означення формальних мов і граматик

В формальних язиках ***алфавітом***називають непорожню множину символів. ***Словом*** в алфавіті може бути будь-яка скінченна послідовність символів. Порожнім словом **ε** називають послідовність символів нульової довжини.

Якщо заданий алфавіт Σ, то множину усіх слів в алфавіті **Σ** позначають **Σ\***. Наприклад, якщо Σ = {a, b}, то Σ\* = { ε, a, b, aa, ab, bb, ba, aaa …}.

***Язиком*** в алфавіті Σ називають множину слів в алфавіті, тобто язик є підмножиною Σ\*.

**Визначення**. Формальною граматикою G, що породжує множину символів, називається наступна сукупність чотирьох об'єктів: G = { Σ, N , P , S }, де

* Σ – алфавіт термінальних символів; терміналом може бути будь-який об’єкт, який не змінює своє значення; зазвичай в якості терміналів використовуються ASCII або Unicode символи.
* N – алфавіт нетермінальних символів (іноді їх називають *синтаксичними* *змінними*); з нетерміналів будуються ланцюжки породжувані граматикою.
* P – множина правил або продукцій вигляду α→ β , де α і β – слова з алфавіту Σ ∪ N.
* S - початковий символ граматики; S ∈ N.

Ми можемо сприймати продукції в граматиці як способи «перетворення» одних слів з алфавіту Σ ∪ N в інші слова з Σ ∪ N.

**Визначення**. Ми можемо *безпосередньо вивести v* з *u* вграматиціG ={Σ, N, P, S }(позначається u ⇒ v), якщо:

* y→ y’ ∈ P
* x, z ∈ (Σ ∪ N)∗
* *u* = xyz та *v* = xy’z.

Ми можемо *вивести v* з *u* (позначається u ⇒\* v) вграматиціG ={Σ, N, P, S}*,* якщо *u ⇒ w1 ⇒ w2 ⇒ … wn ⇒ v.* Таке перетворення позначається *u* ⇒\* *v.*

### Класифікація граматик

За ієрархією Чомскі граматики поділяються на 4 типи. Кожен наступний тип є підтипом попереднього. Далі будемо використовувати малі літери a, b, c,… для позначення терміналів, великі літери A, B, С… – для нетерміналів, грецькі літери α, β, γ,… для послідовностей з терміналів та нетерміналів.

1. Необмежені – на продукції не накладаються ніякі обмеження. Продукції мають вигляд α→ β.
2. Контекстно-залежні. Ліва частина може містити нетермінал, оточений послідовністю символів, які присутні в правій частині. Сам термінал замінюється послідовність символів. Продукції мають вигляд α A β → α γ β.
3. Контекстно-вільні. Ліва частина складається з одного нетермінала. Продукції мають вигляд A→ β.
4. Регулярні граматики. Продукції мають вигляд A → αA або A → α

**Приклад 1.1**. Наступна контекстно-вільна граматика генерує правильні вирази, які складаються зі знаків додавання та множення, групуючих дужок та змінних x та y.

Термінальний алфавіт Σ = { x, +, ∗, (, )}

Нетермінальний алфавіт – N = {E, T, F, V}

Продукції P= {

E → T

E → T + E

T→ F

T→ F\*T

F→(E)

F→V

V→ x

V→ y

}

Стартовий нетермінал – E

Виведемо вираз y \*(x+y) за допомогою перерахованих правил.

E ⇒ T ⇒ F\*T ⇒ V\*T ⇒ y\*T ⇒ y\*F ⇒ y\*(E) ⇒ y\*(T+E) ⇒ y\*(F+E) ⇒ y\*(V + E) ⇒ y\*(x + E) ⇒ y\*(x + T) ⇒ y\*(x + F) ⇒ y\*(x + V) ⇒ y\*(x + y)

### Нотація Бекуса — Наура

При описі складних конструкцій за допомогою формальних граматик доведеться використовувати багато нетермінальних символів, через це такий опис втрачає наочність. В цьому можна використати форму Бекуса-Наура (БНФ). В БНФ в якості нетерміналів використовуються слова природної мови, узяті в кутові дужки, а в якості знака переходу від лівої частини продукції до правої ­­– дві двокрапки та знак дорівнює. Наприклад, продукцію

E→N+N

де E позначає «вираз», а N – число, можна записати наступним чином:

<додавання> ::= <число>+<число>

Якщо декілька продукцій мають однакові ліву частину, то їх можна об’єднати в одну продукцію, в правій частині якої – праві частини об’єднуваних продукцій, розділені знаком «|». Наприклад продукції

<змінна> ::= a

<змінна> ::= b

можна поєднати в

<змінна> ::= a | b

Зазвичай за допомогою БНФ описують контекстно-вільні граматики, але також її можна використовувати для регулярних граматик.

### Синтаксичний аналіз

Задачею синтаксичного аналізу є зіставлення формальної граматики з послідовністю символів в природній або формальній мові. В процесі синтаксичного аналізу створюється дерево розбору або структура даних, яка представляє синтаксичну структуру вхідної послідовності, і яку зручно потім обробляти.

Дерево синтаксичного розбору – це впорядковане кореневе дерево, яке зображує синтаксичну структуру рядка відповідно до формальної граматики. Вт такому дереві листя зображують нетермінали, а листя – термінали.

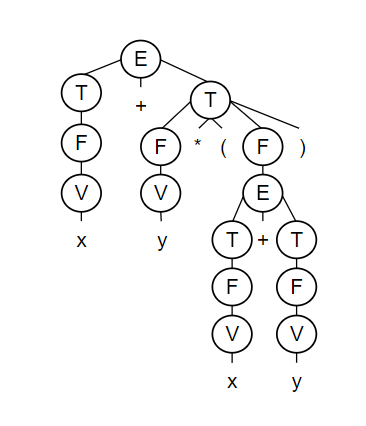
**Приклад 1.2**. На Рисунку 1 можна побачити приклад синтаксичного розбору виразу x + y \* (x + y) в дерево синтаксичного розбору відповідно до граматики, описаної в прикладі 1.1

Рисунок Дерево синтаксичного розбору виразу x + y \* (x + y)

Алгоритми синтаксичного аналізу поділяють на такі групи:

• Спадні (низхідні). Починаючи зі стартового нетерміналу намагаємось перетворити усі нетермінали, використовуючи продукції зліва-направо, поки не отримаємо потрібну послідовність терміналів. Іншими словами будуємо дерево починаючи з коріння та закінчуючи листями.

• Висхідні. Для кожного термінала використовуємо продукції справа-наліво, потім виконуємо такі ж дії для отриманих нетерміналів, поки не залишиться стартовий нетермінал. Іншими словами будуємо дерево починаючи з листів та закінчуючи корінням.

### Рекурсивний спуск

Програма синтаксичного аналізу методом рекурсивного спуску складається із набору процедур, по одній для кожного нетерміналу. Робота програми починається із виклику процедури для початкового символу і успішно завершується у випадку сканування всієї вхідної стрічки.() Нетермінали розкриваються послідовно зліва-направо.

Варто зауважити, що рекурсивний спуск є вразливим для граматик, в яких в лівій частині продукції та напочатку правої частини стоїть один і той же нетермінал. В такому випадку програма-аналізатор зациклюється, через те, що процедура, намагаючись розкрити поточний нетермінал, викликає таку ж процедуру, яка намагається розкрити той же нетермінал.

При аналізі методом рекурсивного може виникнути проблема, яка полягає в тому, що неможливо визначити за яким правилом перетворювати поточний нетермінал. В такому випадку можна перебирати усі правила, в яких в правій частині стоїть поточний нетермінал, та визначати чи відповідають нетермінали в тілі продукції вхідному рядку. Якщо не відповідають, то потрібно повернутись та обрати іншу продукцію.

Щоб уникнути проблем, пов’язаних з «поверненнями», використовують LL(k) граматики. Такі граматики дозволяють, однозначно визначити необхідну продукцію, переглянувши лише k поточних символів.

## Існуючі інструменти для роботи з формулами

**Maple**

**Wolfram Mathematica**

**Matlab**

**Microsoft Math Solver**

## Розробка web-додатку

### Загальний опис програми

Створювана програма має робити наступні речі:

1. Вводити формули у зручний спосіб
2. Відображати формули.
3. Надавати користувачу можливість формули або їх частини
4. Виконувати певні математичні перетворення з виділеними формулами або їх частинами.

#### Введення формули

Для зручного введення формул будемо використовувати JavaScript бібліотеку MathQuill.

За допомогою MathQuill можна створити поле, в яке можна вводити звичайні символи, AsciiMath або Latex-команди. Після закінчення введення команди, текст команди замінюється відповідним зображенням. Також поле вводу підтримує і автоматичні команди, наприклад, при наборі символу «/» в полі вводу створюється дріб, а при наборі символу «\_» автоматично створюється нижній індекс.

Після того, як користувач закінчив введення формули, програма може легко отримати Latex введеної формули.

#### Виведення формули

Найпопулярнішою бібліотекою для відображення математики є MathJax. MathJax – це крос-платформна бібліотека, яка дозволяє відображати математичну нотацію, використовуючи LaTeX, AsciiMath або MathML розмітку. MathJax завантажується як частина вебсторінки, сканує сторінку на наявність розмітки та відповідним чином відображає математичну інформацію.

### Опис формули

#### Формальна граматика

За допомогою LaTeX-розмітки можна описати будь-які математичні конструкції, також її використовують обрані бібліотеки для введення та виведення формул. Тому в створюваній програмі математичні вирази будемо описувати LaTeX-розміткою.

Далі нам потрібно аналізувати обрану розмітку. Для цього опишемо синтаксис LaTeX-розмітки за допомогою Форми Бекуса-Наура (БНФ). Ми будемо описувати не весь синтаксис LaTeX-розмітки, а тільки його частину, яку будемо використовувати в додатку, а саме: дроби, тригонометричні функції, нижні індекси, верхні індекси (ступінь), вектори та грецькі та латинські літери.

У додатку А розміщено опис синтаксису LaTeX-розмітки, у вигляді БНФ нотації.

#### Структура даних

Для зображення формули в мові програмуванні була створена структура даних, яка ґрунтується на вищеописаній формальній граматиці .

Для структури формули створено декілька типів об'єктів (класів), які відповідають основним частинам формули. В об'єкті існує набір атрибутів, які описують з чого складається описувана частина формули. Кожному атрибуту може бути присвоєний інший з створених об'єктів, масив із створених об'єктів, або об'єкти примітивних типів *string* та *number*. Кожній властивості може бути присвоєний лише об'єкт визначеного типу, отже властивості є типізованими.

В додатку Б описана структура формули. Назва типу об’єкту починається з великої літери. Далі в фігурних дужках йде опис властивостей: спочатку назва властивості потім через двокрапку тип властивості; якщо після типу написані квадратні дужки «[]», то властивості має присвоюватись масив, який складається з елементів вказаного типу. Якщо після назви типу об’єкту написано «(Mult)», це означає, що даний тип може бути присвоєним будь-якій властивості, яка має тип Mult.

Тип *Fomula* є основним, в ньому описується вся формула. Його властивість *equalityParts* – це набір виразів, які на письмі поєднуються знаком «=». Якщо структура описує звичайний вираз, то equalityParts складається з одного елемента, якщо рівняння – то з двох елементів (правої та лівої частини рівняння), якщо послідовність тотожних перетворень виразу – то кількість елементів може бути довільною.

Тип *Expr* описує будь-який вираз. Властивість *content* – це набір доданків, з яких складається описуваний вираз. Наприклад, вираз a2+b2-2ab складається з 3 доданків: +a2, +b2 та -2ab, а вираз -(a+b)2 складається з одного доданка: -(a+b)2.

Тип Term описує будь-який доданок. Властивість *sign* описує знак доданка, може бути «+» або «-», на письмі знак доданка ставиться перед множниками або не ставиться, якщо доданок стоїть на початку виразу та має знак «-». Властивість *content* – це набір множників, які поєднуються операцією множення «\*». Множниками можуть бути вирази (записуються у дужках), числа, дроби, змінні тощо. Наприклад в виразі a2 – 2ab + b­­­2 перший та третій доданки мають знак «+», другий має знак «-». Наприклад, доданок 2a2(a+b) має три множники: 2, a2 та (a+b); доданок (-1.5a)\*b має два множники: (-1.5a) та b; доданок +(-1.5) має єдиний множник (-1.5), який є виразом.

Тип *Fraction* описує дроби та має дві властивості: numerator (числівник) та denominator (знаменник). В формальній граматиці та в звичайному записі дробу числівник та знаменник є виразами. Однак, при виконанні основних математичних дій з дробами, таких як множення та ділення, зручніше описувати числівник та знаменник за допомогою набору множників, тобто за допомогою типу Term. Якщо числівник та знаменник дробу мають тип Term, то операцію множення (ділення) дробів можна описати, як додавання множників числівника та знаменника одного дробу до іншого дробу.

#### Відтворення початкового рядка

#### Синтаксичний аналіз

### Дії з формулами

#### Тотожні перетворення виразу

#### Дії з рівняннями

### Інтерфейс користувача

Вставлення

Дії з формулами

Контекстне меню

Збереження або завантаження

## Застосування результатів розробки

# Висновки

# Література

# Додатки

## Додаток А

Нотація БНФ, яка описує синтаксис LaTeX

<*formula*> ::= <*expr*> = <*formula*> | <*expr*>

<*expr*> ::= <term><expr> | <term>

<*term*> ::= <pm><mult-group>

<pm>::= + | - | <empty>

<mult-group>::= <mult>\*<mult-group> | <mult><mult-group> | <mult>

<mult>::= <frac> | <function> | <number> | <sqrt> | <exponent> | <variable> | <vector> | (<expr>)

<fraction>::= \frac{<expr>}{<expr>}

<function>::= \<functionName>(<expr>)

<functionName>::= \sin | \cos | \tan | \ctg | \arcsin | \arccos | \arctan | \arcctg

<number>::= <integer> | <integer>.<integer>

<integer> ::= <digit> | <digit><integer>

<sqrt>::= \sqrt{<expr>} | \sqrt[<expr>]{<expr>}

<exponent>::= <mult>^{<expr>}

<vector>::= \vec{<variable>}

<variable>::= <var-name><sub><primes>

<primes>::= '<primes> | <empty>

<sub>::= \_{<expr>} | <empty>

<var-name>::= <latin> | <greek>

В цій нотації використовуються наступні невизначені нетермінали:

<*empty*> – позначає порожній рядок.

<*greek*> – позначає будь-яку літеру грецького алфівіту у LaTeX запису (“\” + назва літери). Наприклад: *\alpha* (α), *\beta* (β), *\Sigma* (Σ), *\Omega* (**Ω**).

<*latin*> – позначає будь-яку літеру літинського алфавіту в нижньому або верхньому регістрі.

<*digit*> – позначає будь-яку цифрк від 0 до 9

## Додаток Б

Структура формули

Formula{

equality\_parts: Expr[]

}

Expr{

content: Term[]

}

Term{

sign: "+"|"-"

content: Mult[]

}

Fraction{

numerator: Term

denominator: Term

}

Function(Mult){

name: string

Content: Block

}

Number(Mult){

value: number

}

Sqrt(Mult){

root: Block

content: Mult

}

Exponent(Mult){

exponent: Block

base: Mult

}

Variable(Mult){

name: string

index: string

vector: bool

primeCount: number;

}