КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Лабораторна робота №4
«LL(K)-СИНТАКСИЧНИЙ АНАЛІЗАТОР»

Виконав студент
3-го курсу, групи ТТП-32
ОПП "Інформатика"
Ковалик Матвій
Науковий керівник:
Шишацька Олена Володимирівна

КИЇВ - 2023

ЗМІСТ

ПОСТАНОВКА ЗАВДАННЯ

Розробити LL(1)-синтаксичний аналізатор для заданої граматики, який будує AST або визначає та локалізує синтаксичну помилку:

1) запрограмувати всі необхідні функції: Firsk(k), Follow(k), побудова таблиці управління, власне аналізатор по таблиці

2) запрограмувати допоміжні функції: пошук епсилон-нетерміналів, читання і розбір введеної граматики, тощо

на додаткові бали:

3) LL(k) аналізатор для k>1 = +6 балів

4) також запрограмувати аналізатор методом рекурсивного спуску = +4 бали

5) реалізувати LALR-аналізатор (на прикладі граматики мови C) = +10 балів

6) візуалізація дерева виводу (AST) = +3 бали.

На додаткові бали я створив LL(k) аналізатор, аналізатор методом рекурсивного спуску та візуалізував дерево виводу.

РЕАЛІЗАЦІЯ ЗАВДАННЯ

Посилання на написаний код програми:
https://github.com/kkommatt/SP_Lab4_LLk

1. Створюємо екземпляри граматик та виконуємо обчислення, зокрема читаємо вміст вхідного файлу, де записані правила граматики.
2. Основа програми клас Grammar. Цей клас зберігає правила граматики, набір нетерміналів та терміналів, має методи для отримання терміналів та нетерміналів, правил.
3. Виводимо на консоль правила граматики для того, щоб користувач впевнився, що граматика зберігається в пам’яті комп’ютера, так як це бажає користувач. Це можна побачити на малюнку 1.
4. Обчислюємо множину FIRST для символів граматики за допомогою визначеної функції(окрема для k=1) та виводимо результат на консоль. Це можна побачити на малюнку 2.
5. Обчислюємо множину FOLLOW для символів граматики за допомогою визначеної функції, результат виводимо на консоль. Це можна побачити на малюнку 2.
6. Створюємо LL(1) таблицю розбору, яка визначає, яке правило використовувати на основі поточного нетермінала та входу, виводимо для користувача на консоль. Це можна побачити на малюнку 2
7. Створюємо LL(k) таблицю розбору для k>1. Малюнок 4.
8. Виконуємо синтаксичний аналіз вхідного рядка токенів, використовуючи метод рекурсивного спуску за допомогою визначеної для цього функції, спочатку відображаємо чи можемо ми це зробити, тобто чи виводить граматика такого вигляду слова. Малюнок 1.
9. Візуалізуємо AST у консолі, використовуючи різні рівні рядків та пропуски для представлення ієрархії вузлів. Малюнок 3.
10. Виводимо результати обчислень Follow і First k>1 та побудовану таблиці розбору в консоль.

ТЕСТУВАННЯ

```
Rules of parsed grammar
A -> BA'
A' -> +BA' | e
B -> CB'
B' -> *CB' | e
C -> (A) | id
Is parsed?
Success
```

```
First_1
A: {( i }
A': {+ e }
B: {( i }
B': {* e }
C: {( i }
Follow_1
A: {' ) }
A': {' }
B: {' ( i }
B': {' }
C: {( i }
Parsing Table
Non-terminal: A, Terminal: ( => Production: BA'
Non-terminal: A, Terminal: i => Production: BA'
Non-terminal: A', Terminal: ' => Production: e
Non-terminal: A', Terminal: + => Production: +BA'
Non-terminal: B, Terminal: ( => Production: CB'
Non-terminal: B, Terminal: i => Production: CB'
Non-terminal: B', Terminal: ' => Production: e
Non-terminal: B', Terminal: * => Production: *CB'
Non-terminal: C, Terminal: ( => Production: (A)
Non-terminal: C, Terminal: i => Production: id
Epsilon producing non terminals
A' B'
```

```
AST:
A
  B
    C
      A
        B
          C
            id
          B'
            e
        A'
          op (+)
          B
            C
              id
            B'
              e
          A'
            e
    B'
      op (*)
      C
        id
      B'
        e
  A'
    e
```

```
First_k:
A: {( i }
A': {+ e }
B: {( i }
B': {* e }
C: {( i }
First_k:
A: {c }
B: {c }
S: {a b }
Follow_k:
A: {d f }
B: {d f }
S: {$ }
LL(2) Parsing Table:
Non-terminal: A, Lookahead: c => Productions: c
Non-terminal: B, Lookahead: c => Productions: c

Process finished with exit code 0
```

ВИСНОВОК

Під час виконання лабораторної роботи я реалізував аналізатор для LL(1) та LL(k) граматик, використовуючи мову програмування C++. Центральним аспектом роботи було створення програмного забезпечення, здатного опрацьовувати задані граматики. Мною було розроблено клас, який є основою для представлення граматики в пам'яті комп'ютера. Він зберігає правила у вигляді відображення від нетерміналів до їх продукцій та надає методи для отримання множини терміналів, нетерміналів та правил.

Я визначив функції для знаходження first та follow. Виконано тестування аналізатора на практичному прикладі, перевірено його здатність правильно розбирати вхідну послідовність символів.

Варто також додати, що я в цілому поглибив знання з системного програмування та розвинув навички програмування мовою C++.