Лабораторная работа №2. Ручное построение нисходящих синтаксических анализаторов

3 сентября 2021 г.

Введение

Цель данной лабораторной работы — научиться разрабатывать грамматики для заданных неформально языков с учетом семантического смысла и приоритета операторов, разрабатывать лексические анализаторы, разрабатывать вручную нисходящие синтаксические анализаторы, разрабатывать подробные тестовые наборы для анализаторов.

Форма отчетности: программа и текстовый отчет. Программа может быть написана на любом языке программирования общего назначения (рекомендуются Си++ и Java).

1 Разработка грамматики

Разработайте контестно-свободную грамматику для языка, описанного в условии вашего варианта. Сначала разработайте грамматику, исходя из структуры языка, чтобы она максимально близко соответствовала интуитивным представлениям о построении слов из языка. Затем, при необходимости, устраните левую рекурсию и/или правое ветвление.

В отчете приведите исходную и преобразованную грамматику, опишите смысл всех нетерминалов.

Пример задания

Язык правильных скобочных выражений

Пример выполнения задания

Построим грамматику.

$$S \to SS$$
$$S \to (S)$$
$$S \to \varepsilon$$

Нетерминал	**Описание**
S	Правильная скобочная последовательность.

В грамматике есть левая рекурсия. Устраним ее. Получится грамматика:

$$S \to (S)S'$$
$$S \to \varepsilon$$
$$S' \to SS'$$
$$S' \to \varepsilon$$

Нетерминал	**Описание**
S	Правильная скобочная последовательность.
S'	Продолжение правильной скобочной последовательности.

2 Построение лексического анализатора

Лексический анализатор должен получать на вход строку и выдавать последовательность терминалов (токенов). Пробелы и переводы строк должны игнорироваться.

Пример выполнения задания

В нашей грамматике два терминала: ‘(’ и ‘)’.

Построим лексический анализатор. Заведем класс `Token` для хранения терминалов. Не забудем также про конец строки.

```
public enum Token {
    LPAREN, RPAREN, END
}
```

Терминал	Токен
(	LPAREN
)	RPAREN
$	END

Ниже приведен пример простейшего лексического анализатора. Если токены в вашей грамматике могут состоять из нескольких символов, это следует учесть.

```
import java.io.IOException;
import java.io.InputStream;
import java.text.ParseException;

public class LexicalAnalyzer {
    InputStream is;
    int curChar;
    int curPos;
    Token curToken;

    public LexicalAnalyzer(InputStream is) throws ParseException {
        this.is = is;
        curPos = 0;
        nextChar();
    }

    private boolean isBlank(int c) {
        return c == ' ' || c == '\r' || c == '\n' || c == '\t';
    }

    private void nextChar() throws ParseException {
        curPos++;
        try {
            curChar = is.read();
        } catch (IOException e) {
            throw new ParseException(e.getMessage(), curPos);
        }
    }

    public void nextToken() throws ParseException {
        while (isBlank(curChar)) {
            nextChar();
```

```
            }
            switch (curChar) {
                case '(':
                    nextChar();
                    curToken = Token.LPAREN;
                    break;
                case ')':
                    nextChar();
                    curToken = Token.RPAREN;
                    break;
                case -1:
                    curToken = Token.END;
                    break;
                default:
                    throw new ParseException("Illegal character "
                        + (char) curChar, curPos);
            }
        }

        public Token curToken() {
            return curToken;
        }

        public int curPos() {
            return curPos();
        }
}
```

3 Построение синтаксического анализатора

Постройте множества FIRST и FOLLOW для нетерминалов вашей грамматики. Затем напишите синтаксический анализатор с использованием рекурсивного спуска.

Пример выполнения задания

Построим множества FIRST и FOLLOW для нетерминалов нашей грамматики.

Нетерминал	FIRST	FOLLOW
S	$($, ε	), \$
S'	$($, ε	), \$

Заведем структуру данных для хранения дерева.

```
import java.util.Arrays;
import java.util.List;

public class Tree {
    String node;

    List<Tree> children;

    public Tree(String node, Tree... children) {
        this.node = node;
        this.children = Arrays.asList(children);
    }

    public Tree(String node) {
        this.node = node;
    }
}
```

Ниже приведен простейший синтаксический анализатор с использованием рекурсивного спуска.

```
import java.io.InputStream;
import java.text.ParseException;

public class Parser {
    LexicalAnalyzer lex;

    Tree S() throws ParseException {
        switch (lex.curToken()) {
            case LPAREN:
                // (
                lex.nextToken();
                // S
                Tree sub = S();
                // )
                if (lex.curToken() != Token.RPAREN) {
```

```
                    throw new ParseException(")␣expected␣"+
                        "at␣position", lex.curPos());
                }
                lex.nextToken();
                // S'
                Tree cont = SPrime();
                return new Tree("S", new Tree("("), sub,
                    new Tree(")"), cont);
            case RPAREN:
            case END:
                // eps
                return new Tree("S");
            default:
                throw new AssertionError();
        }
    }

    Tree SPrime() throws ParseException {
        switch (lex.curToken()) {
            case LPAREN:
                // S
                Tree sub = S();
                // S'
                Tree cont = SPrime();
                return new Tree("S'", sub, cont);
            case RPAREN:
            case END:
                // eps
                return new Tree("S'");
            default:
                throw new AssertionError();
        }
    }

    Tree parse(InputStream is) throws ParseException {
        lex = new LexicalAnalyzer(is);
        lex.nextToken();
        return S();
    }
}
```

4 Визуализация дерева разбора

Для изучения результата разработайте систему визуализации дерева разбора. Рекомендуется использовать систему GraphViz (`https://graphviz.org/`)

5 Подготовка набора тестов

Подготовьте набор исчерпывающих тестов для вашей программы. По возможности добейтесь, чтобы все возможные ошибки в грамматике или в лексическом/синтаксическом анализаторе ими покрывались. Про каждый тест дайте короткое описание.

Пример выполнения задания

Ниже приведен лишь небольшой набор тестов, исчерпывающий набор для правильных скобочных последовательностей должен содержать больше тестов.

Тест	**Описание**
()()	Тест на правило $S \to SS$
(())	Тест на правило $S \to (S)$
	Тест на правило $S \to \varepsilon$
(()((()))()()))()()	Небольшой случайный тест

Задания

Вариант 1. Арифметические выражения

Арифметические выражения с операциями сложения, вычитания, умножения, скобками, унарным минусом и унарными функциями. Приоритет операций стандартный. Скобки используются для изменения приоритета и передачи аргументов в функции.

В качестве операндов выступают целые числа. Используйте один терминал для всех чисел. Любая последовательность букв задает имя функции. Используйте один терминал для всех функций.

Пример: `(1+2)*sin(-3*(7-4)+2)`

Вариант 2. Регулярные выражения

Регулярные выражения с операциями конкатенации (простая последовательная запись строк), выбора (вертикальная черта), замыкания Клини. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

Для обозначения базовых языков используются маленькие буквы латинского алфавита. Используйте один терминал для всех символов.

Пример: `((abc*b|a)*ab(aa|b*)b)*`

Вариант 3. Логические формулы в стиле Python

Логические формулы. Используются операции `and`, `or`, `xor`, `not`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал (не три 'a', 'n', 'd' для `and`).

Пример: `(a and b) or not (c xor (a or not b))`

Вариант 4. Логические формулы с множествами в стиле Python

Логические формулы. Используются операции `and`, `or`, `xor`, `not`, `in`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета. Предусмотреть возможность оператора `not in`.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал (не три 'a', 'n', 'd' для `and`).

Пример: `(a in b) or (c not in b)`

Вариант 5. Логические формулы в стиле Си

Логические формулы. Используются операции `&`, `|`, `^`, `!`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал.

Пример: `(!a | b) & a & (a | !(b ^ c))`

Вариант 6. Описание переменных в Kotlin

Блок описания переменных в языке Kotlin. Каждое описание начинается ключевым словом "`var`" или "`val`", далее идет описание переменной. Описание содержит имя переменной, затем двоеточие, затем имя типа. Затем может идти начальное значение, предусмотреть инициализацию только для типа `Int` числом, выражения рассматривать не требуется.

Используйте один терминал для всех имен переменных и имен типов. Используйте один терминал для ключевого слова `var` (не три 'v', 'a', 'r').

Пример: `var a: Int; val c: Int = 2;`

Вариант 7. Описание переменных в Си

Описания переменных в Си. Сначала следует имя типа, затем разделенные запятой имена переменных. Переменная может быть указателем, в этом случае перед ней идет звездочка (возможны и указатели на указатели, и т. д.). Описаний может быть несколько.

Используйте один терминал для всех имен переменных и имен типов.

Пример: `int a, *b, ***c, d;`

Вариант 8. Описание лямбда функции в Python

Описание лямбда функции в Python. Описание начинается ключевым словом "`lambda`", далее идет множество аргументов через запятую, двоеточие, выражение. Используйте логические и/или арифметические операции.

Используйте один терминал для всех имен переменных. Используйте один терминал для ключевых слов `lambda` и т. п. (не несколько 'l', 'a', 'm' и т. д.).

Пример: `lambda n : n + 2`

Вариант 9. Описание заголовка функции в Kotlin

Заголовок функции в Kotlin. Заголовок начинается ключевым словом "`fun`", далее идет имя функции, скобка, несколько описаний аргументов через запятую, затем может идти двоеточие и имя возвращаемого типа.

Используйте один терминал для всех имен переменных. Используйте один терминал для ключевых слов `fun` и т. п. (не несколько 'f', 'u', 'n').

Пример: `fun printSum(a: Int, b: Int): Unit`

Вариант 10. Заголовок функции в Си

Заголовок функции в Си. Заголовок начинается именем возвращаемого типа или словом "`void`", далее идет имя функции, скобка, затем разделенные запятой описания аргументов. Переменная может быть указателем, в этом случае перед ней идет звездочка (возможны и указатели на указатели, и т. д.). Аргументов может быть несколько.

Используйте один терминал для всех имен переменных и имен типов.

Пример: `int fib(int n);`

Вариант 11. Описание массивов в Kotlin

Массив в Kotlin. Описание начинается ключевым словом "`var`", далее идет имя массива, двоеточие, имя типа "`Array`", далее в угловых скобках имя типа элементов массива.

Используйте один терминал для всех имен переменных и имен типов. Используйте один терминал для ключевого слова `var` (не несколько 'v', 'a', 'r').

Пример: `var x: Array<Int>;`

Вариант 12. Оператор for в Си

Примитивная версия оператора for в Си. Оператор начинается ключевым словом "`for`", далее в скобках три параметра, разделенные точкой с запятой. Первый параметр: имя типа, имя переменной, начальное значение. Второй параметр: имя переменной, знак сравнения, число. Третий параметр: имя переменной, операция инкремента/декремента.

Используйте один терминал для всех имен переменных и имен типов. Используйте один терминал для ключевого слова `for` (не несколько 'f', 'o', 'r'). Используйте один терминал для оператора `++` (не два плюса).

Пример: `for (int x = 0; x < 10; x++)`