

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ имени Н.Э.БАУМАНА  
(национальный исследовательский университет)»**

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа № 5**

Лексический распознаватель

по дисциплине «Конструирование компиляторов»

Вариант 21

Работу выполнил

студент группы ИУ9-62Б

Жук Дмитрий

Москва, 2022

**Цель работы**

Целью данной работы является изучение использования детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа.

**Задание**

Выполнение лабораторной работы состоит из пяти этапов:

1. Описание лексических доменов модельного языка в виде регулярных выражений;

2. Построение недетерминированного лексического распознавателя для модельного языка;

3. Детерминизация построенного лексического распознавателя и факторизация его алфавита;

4. Построение массива обобщённых символов, матрицы переходов и массива заключительных состояний для полученного детерминированного лексического распознавателя с факторизованным алфавитом;

5. Разработка лексического анализатора, работающего на основе интерпретации построенных структур данных.

**Индивидуальный вариант**

Ключевые слова: **get**, **set**. Операторы: **||**, **+**. Комментарии начинаются с **|+** и заканчиваются на **+|**, могут пересекать границы строк текста, последовательность знаков **|+** внутри комментария является синтаксической ошибкой.

**Реализация**

/\* eslint-disable no-console, import/prefer-default-export \*/

import fs from 'fs';

// eslint-disable-next-line @typescript-eslint/no-unused-vars

const reg = /(?<space>^[ \t\n]+)|(?<number>^[0-9]+)|(?<sign>^(?:\+|\|\|))|(?<comment>^\|\+(?:[^+]|\+[^|])\*\+\|)|(?<key>^(?:get|set)(?![a-zA-Z0-9]))|(?<ident>^[a-zA-Z][a-zA-Z0-9]\*)|(?<error>^.)/;

enum TokenType {

SPACES = 'spaces',

NUMBER = 'number',

SIGN = 'sign',

COMMENT = 'comment',

IDENT = 'ident',

KEY = 'key',

ERROR = 'error',

}

class SmartIterator {

private pos: [number, number] = [1, 1];

private prevPos: [number, number] = [1, 0];

private ind = 0;

// eslint-disable-next-line no-useless-constructor, no-empty-function

constructor(private s: string) { }

savePos() { const { pos, prevPos, ind } = this; return { pos, prevPos, ind }; }

loadPos({ pos, prevPos, ind }: ReturnType<SmartIterator['savePos']>) {

this.ind = ind;

this.pos = pos;

this.prevPos = prevPos;

}

next() {

this.prevPos = this.pos;

this.pos = (this.s[this.ind] === '\n' ? [this.pos[0] + 1, 1] : [this.pos[0], this.pos[1] + 1]);

return this.s[this.ind++];

}

see() { return this.s[this.ind] || ''; }

has() { return this.ind < this.s.length; }

}

class Automat {

// eslint-disable-next-line no-useless-constructor

constructor(

private states: Record<string, { case?: [RegExp, string][], final?: TokenType }>,

private initialState = 'start',

// eslint-disable-next-line no-empty-function

) { }

parseToken(s: string) {

const self = this;

return {

\* [Symbol.iterator](): Generator<

{ type: TokenType, from: [number, number], to: [number, number], value: string }> {

const iter = new SmartIterator(s);

while (iter.has()) {

let state = self.initialState;

const fromPos = iter.savePos();

for (

let find: [RegExp, string] | undefined;

// eslint-disable-next-line no-cond-assign

find = self.states[state]?.case?.find(([r]) => r.test(iter.see()));

) {

// console.log({ see: iter.see(), state, find });

[, state] = find;

iter.next();

}

const final = self.states[state]?.final;

// console.log({ see: iter.see(), state, final });

if (final) {

const nowPos = iter.savePos();

yield {

type: final,

from: fromPos.pos,

to: nowPos.prevPos,

value: s.slice(fromPos.ind, nowPos.ind),

};

} else {

iter.loadPos(fromPos);

yield {

type: TokenType.ERROR,

from: fromPos.pos,

to: fromPos.pos,

value: iter.next(),

};

}

}

},

};

}

}

console.log([...new Automat({

start: {

case: [

[/\s/, 'spaces'],

[/\d/, 'number'],

[/\+/, 'sign+'],

[/\|/, 'presignorcomment'],

[/[gs]/, 'prekey1'],

[/[a-zA-Z]/, 'ident'],

],

},

spaces: {

case: [[/\s/, 'spaces']],

final: TokenType.SPACES,

},

number: {

case: [[/\d/, 'number']],

final: TokenType.NUMBER,

},

'sign+': { final: TokenType.SIGN },

presignorcomment: {

case: [

[/\|/, 'sign||'],

[/\+/, 'bodycomment'],

],

},

'sign||': { final: TokenType.SIGN },

bodycomment: { case: [[/\|/, 'preerror'], [/\+/, 'precomment3'], [/[^]/, 'bodycomment']] },

preerror: { case: [[/\+/, 'error'], [/./, 'bodycomment']] },

// error: { final: TokenType.ERROR },

precomment3: { case: [[/\|/, 'comment'], [/\+/, 'precomment3'], [/[^]/, 'bodycomment']] },

comment: { final: TokenType.COMMENT },

prekey1: { case: [[/e/, 'prekey2'], [/[a-zA-Z0-9]/, 'ident']], final: TokenType.IDENT },

prekey2: { case: [[/t/, 'key'], [/[a-zA-Z0-9]/, 'ident']], final: TokenType.IDENT },

key: { case: [[/[a-zA-Z0-9]/, 'ident']], final: TokenType.KEY },

ident: { case: [[/[a-zA-Z0-9]/, 'ident']], final: TokenType.IDENT },

}).parseToken(fs.readFileSync(process.argv[2], { encoding: 'utf8' }))]);

Листинг 1 — Код программы

**Вывод**

В ходе лабораторной работы было приобретен навык использования детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа. Так же был улучшен навык написания регулярных выражений вообще и в TypeScript, в частности.