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 № 7.2

Самоприменимый генератор компиляторов на основе предсказывающего анализа

по дисциплине «Конструирование компиляторов»

Вариант 3

Работу выполнил

студент группы ИУ9-62Б

Жук Дмитрий

Москва, 2022

Цель работы

Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

Задание

Лабораторная работа делается на основе выполненной лабораторной работы 7.1. Выполнение данной лабораторной работы состоит из следующих этапов:

1. Переписыавние нрамматики вхожного языка на самом входном языке.
2. Добавление в программу, написанной на лабораторной работе 7.1, генератора таблицы разбора на основе дерева разбора. Таблица разбора должна представлять собой инициализированный двумерный массив на рабочем языке.
3. Тестирование генератора компиляторов путем написания простейшего калькулятора арифметических выражений на основе грамматики.
4. Раскрутка генератора компиляторов путём подачи на вход грамматики входного языка, написанной на самом входном языке и замены таблицы разбора, написанной вручную, на сгенерированную таблицу разбора.

Индивидуальный вариант

```
; только правила грамматики

(F)  = n | \( (E) \).
(T)  = (F) (T1).
```

```
(T1) = * (F) (T1) | .
(axiom E) = (T) (E1).
(E1) = + (T) (E1) | .
```

Листинг 1 — вариант входного языка в примерах описаний грамматик

Грамматика

- `(axiom Init) = (Nterm) (Nterms) .`
- `(Nterms) = (Nterm) (Nterms) | .`
- `(Nterm) = nterm \= (Rule) (Rules) | axiom \= (Rule) (Rules) .`
- `(Rules) = \| (Rule) (Rules) | \. .`
- `(Rule) = nterm (Rule) | term (Rule) | .`

Реализация

```
export type Term = { type: 'term', value: string };
export type Nterm = { type: 'nterm', value: string };

export type Rule = (Term | Nterm)[]
export type Rules = Rule[]

export type Language = {
  axiom: string,
  nterms: Record<string, Rules>,
}

const join = <T>(a: Set<T>, b: Set<T> | T[]) => b.forEach((v) => a.add(v));

const EPSILON = Symbol('EPSILON');

const smartFirst = (
  first: Record<string, Set<string | typeof EPSILON>>,
) => {
  const F = (
    rule: Rule,
  ): [string] | [typeof EPSILON] | Set<string | typeof EPSILON> => {
    if (rule.length === 0) return [EPSILON];
    if (rule[0].type === 'term') return [rule[0].value];
    if (!first[rule[0].value].has(EPSILON)) return first[rule[0].value];
    const copy = new Set(first[rule[0].value]);
    copy.delete(EPSILON);
    join(copy, F(rule.slice(1)));
    return copy;
  };
  return F;
};
```

```
const createFirst = (lang: Language): Record<string, Set<string | typeof
EPSILON>> => {
  const allNterms = Object.keys(lang.nterms);
  const first: Record<string, Set<string | typeof EPSILON>> =
Object.fromEntries(
    allNterms.map((nameNterm) => [nameNterm, new Set()]),
  );

  const F = smartFirst(first);

  let wasChange;
  do {
    wasChange = false;
    // eslint-disable-next-line no-loop-func
    allNterms.forEach((nameNterm) => lang.nterms[nameNterm]
      .forEach((rule) => F(rule).forEach((v) => {
        if (!first[nameNterm].has(v)) {
          wasChange = true;
          first[nameNterm].add(v);
        }
      })));
  } while (wasChange);

  // console.log('CREATE_FIRST');
  // console.dir(first, { depth: null });

  return first;
};

const EOF = Symbol('EOF');

const createFollow = (
  lang: Language,
  first = createFirst(lang),
): Record<string, Set<string | typeof EOF>> => {
  const allNterms = Object.keys(lang.nterms);
  const follow: Record<string, Set<string | typeof EOF>> =
Object.fromEntries(
    allNterms.map((nameNterm) => [nameNterm, new Set()]),
  );

  follow[lang.axiom].add(EOF);

  const pairHasEpsilon: [string, string][] = [];

  const F = smartFirst(first);
  allNterms.forEach((nameNterm) => lang.nterms[nameNterm]
    .forEach((rule) => rule.forEach(({ type, value }, i) => {
      if (type === 'nterm') {
        const f = F(rule.slice(i + 1));
        let hasEpsilon = false;
        if (Array.isArray(f)) {
          if (f[0] !== EPSILON) join(follow[value], f);
          else hasEpsilon = true;
        } else {
          f.forEach((v) => {
            if (v !== EPSILON) follow[value].add(v);
```

```
            else hasEpsilon = true;
          });
        }
        if (hasEpsilon) {
          pairHasEpsilon.push([nameNterm, value]);
        }
      }
    }));

  let wasChange;
  do {
    wasChange = false;
    // eslint-disable-next-line no-loop-func
    pairHasEpsilon.forEach(([x, y]) => follow[x].forEach((v) => {
      if (!follow[y].has(v)) {
        wasChange = true;
        follow[y].add(v);
      }
    }));
  } while (wasChange);

  // console.log('CREATE_FOLLOW');
  // console.dir(follow, { depth: null });

  return follow;
};

const createFirstAndFollow = (lang: Language) => {
  const first = createFirst(lang);
  return { first, follow: createFollow(lang, first) };
};

export const createTable = (
  lang: Language, { first, follow } = createFirstAndFollow(lang),
) => {
  // const terms = getTerminals(lang);
  const F = smartFirst(first);
  const deta: Record<
    string,
    Record<
      string | typeof EOF,
      (Term | Nterm)[] | undefined
    > | undefined
  > = {};

  const addTable = (X: string, a: string | typeof EOF, u: Rule) => {
    // eslint-disable-next-line no-multi-assign
    const detaX = deta[X] ??= {} as NonNullable<typeof deta[string]>;
    if (detaX[a] !== undefined) {
      // eslint-disable-next-line no-throw-literal
      throw {
        message: 'not LL(1) grammar', X, a, value: [detaX[a], u],
      };
    }
    detaX[a] = u;
  };
```

```
  Object.entries(lang.nterms).forEach(([X, rules]) => rules.forEach(
    (u) => {
      const FIRSTu = F(u);
      let hasEpsilon = false;
      FIRSTu.forEach((a) => {
        if (a === EPSILON) {
          hasEpsilon = true;
        } else {
          addTable(X, a, u);
        }
      });
      if (hasEpsilon) {
        follow[X].forEach((b) => addTable(X, b, u));
      }
    },
  ));

  // console.log('CREATE_TABLE');
  // console.dir(deta, { depth: null });

  return deta;
};
```

Листинг 2 — генератор таблицы

```
enum CalculatorType {
  PLUS = '+',
  MUL = '*',
  N = 'n',
  OPEN = '(',
  CLOSE = ')',
}

type CalculatorLexems =
  | Lexema<'+', string>
  | Lexema<'*', string>
  | Lexema<'n', number>
  | Lexema<'(', string>
  | Lexema<')', string>
  ;

export const calculatorAnalyzer = lexicalAnalyzer<CalculatorLexems>({
  reg:
/^(?:(?<plus>\+)|(?<n>\d+)|(?<mul>\*)|(?<open>\()|(?<close>\))|(?<space>\s+)|
(?<error>.))/,
  rules: {
    plus: (value) => ({ value, type: CalculatorType.PLUS }),
    mul: (value) => ({ value, type: CalculatorType.MUL }),
    n: (value) => ({ value: +value, type: CalculatorType.N }),
    open: (value) => ({ value, type: CalculatorType.OPEN }),
    close: (value) => ({ value, type: CalculatorType.CLOSE }),
  },
});

export const calculatorLang = {
  axiom: 'E',
```

```
  nterms: {
    F: [
      [term('n')],
      [term('('), nterm('E'), term(')')],
    ],
    T: [[nterm('F'), nterm('T1')]],
    T1: [
      [term('*'), nterm('F'), nterm('T1')],
      [],
    ],
    E: [[nterm('T'), nterm('E1')]],
    E1: [
      [term('+'), nterm('T'), nterm('E1')],
      [],
    ],
  },
};

export const calculatorAggregate = {
  F: (...args: [{ value: number }] | [any, number, any]) => (
    args.length === 3 ? args[1] : args[0].value
  ),
  T: (a: number, b: number) => a * b,
  T1: (...args: [] | [any, number, number]) => (args.length === 0 ? 1 :
args[1] * args[2]),
  E: (a: number, b: number) => a + b,
  E1: (...args: [] | [any, number, number]) => (args.length === 0 ? 0 :
args[1] + args[2]),
};

const calculatorCompiler = compiler({
  lexer: calculatorAnalyzer,
  lang: calculatorLang,
  aggregate: calculatorAggregate,
});

console.dir(calculatorCompiler(fs.readFileSync(0, 'utf-8')), { depth: null
});
```

Листинг 3 — простейший калькулятор

```
import fs from 'fs';
import { compiler, loadTable, saveTable } from '.';
import { grammarAggregate, grammarAnalyzer } from './grammar';

const readt = fs.readFileSync(process.argv[2], 'utf-8');
const readf = fs.readFileSync(process.argv[3], 'utf-8');

console.log(saveTable(compiler({
        lexer: grammarAnalyzer,
        table: loadTable(readt),
        axiom: 'Init',
        aggregate: grammarAggregate,
})(readf).res));
```

Листинг 4 — раскрутка генератора компиляторов

Вывод

В ходе лабораторной работы было приобретен навык разработки синтаксического анализатора на основе предсказывающего анализа.