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 № 7.1

Синтаксический анализатор 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

Выполнение лабораторной работы состоит из следующих этапов:

1. Составление описаний лексической структуры и грамматики входного языка на основе примера из листинга 1.
2. Разработка лексического анализатора для входного языка.
3. Составление таблицы предсказывающего разбора для входного языка.
4. Разработка алгоритма предсказывающего разбора, работающего на основе порожденной таблицы. Реализация этого алгоритма.

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

Синтаксис

Неформальное описание: входной язык — это непустая последовательность «определения нетерминалов».

«Определения нетерминала» - начинается либо с названия нетерминала, либо с названия аксиомы, далее содержит символ равно, «перечисление правил» и заканчивается символом точка.

«Перечисление правил» — это непустая последовательность «правил», разделенных знаком ИЛИ.

«Правило» — это последовательность (возможно пустая) нетерминалов и терминалов.

Формальное описание:

Лексическая структура (будет задана в виде регулярных выражений):

- комментарий - `;.*`
- пробелы - `\s+`
- равно - `=`
- точка - `\.`
- ИЛИ - `\|`
- аксиома - `\(axiom[ \t]+[a-zA-Z][a-zA-Z0-9]*\)`
- нетерминал - `\([a-zA-Z][a-zA-Z0-9]*\)`
- терминал - `\\.|[^\s()|.=]+`

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 const lexicalAnalyzer = <L extends { type: string }>({
  reg,
  rules,
  def = () => null,
}: InputLexicalAnalyzer<L>) => (s: string) => ({
    * [Symbol.iterator](): Generator<SmartLexems<L>> {
      let fromLine = 1;
      let fromPos = 1;
      let fromAbs = 0;
      let toLine = 1;
      let toPos = 1;
      let toAbs = 0;

      const buildToken = (l: L): SmartLexems<L> => ({
        ...l,
        from: { line: fromLine, pos: fromPos, abs: fromAbs },
        to: { line: toLine, pos: toPos, abs: toAbs },
      });

      for (let e = reg.exec(s); e?.groups; e = reg.exec(s)) {
        const groups = Object.entries(e.groups).filter(([, value]) => value);
        if (groups.length !== 1 || e[0].length === 0 || e.groups.error) {
          throw {
            message: 'match not one group or value zero or error',
            groups,
            from: { line: fromLine, pos: fromPos, abs: fromAbs },
            to: { line: toLine, pos: toPos, abs: toAbs },
          };
        }

        fromLine = toLine;
        fromPos = toPos;
        fromAbs = toAbs;

        s = s.slice(e.index + e[0].length);
        toAbs += e[0].length;
        const split = e[0].split('\n');
        if (split.length === 1) {
          toPos += e[0].length;
        } else {
```

```
          toLine += split.length - 1;
          toPos = split[split.length - 1].length + 1;
        }

        const [[key, value]] = groups;
        const res = rules[key]?.(value) ?? def(value);
        if (res !== null) {
          yield buildToken(res);
        }
      }
    },
  });

enum TokenType {
  EQ = '=',
  END = '.',
  OR = '|',
  AXIOM = 'axiom',
  NTERM = 'nterm',
  TERM = 'term',
}

export const grammarAnalyzer = lexicalAnalyzer<Lexems>({
  reg:
/^(?:(?<comment>;.*)|(?<space>\s+)|(?<eq>=)|(?<end>\.)|(?<or>\|)|\(axiom[
\t]+(?<axiom>[a-zA-Z][a-zA-Z0-9]*)\)|\((?<nterm>[a-zA-Z][a-zA-Z0-
9]*)\)|(?<term>\\.|[^\s()|.=]+)|(?<error>.))/,
  rules: {
    eq: (value) => ({ value, type: TokenType.EQ }),
    end: (value) => ({ value, type: TokenType.END }),
    or: (value) => ({ value, type: TokenType.OR }),
    axiom: (value) => ({ value, type: TokenType.AXIOM }),
    nterm: (value) => ({ value, type: TokenType.NTERM }),
    term: (value) => ({ value: value.replace(/^\\/, ''), type: TokenType.TERM
}),
  },
});
```

Листинг 2 — лексической анализатор для входного языка

```
{
  Init: {
    nterm: [
      { type: 'nterm', value: 'Nterm' },
      { type: 'nterm', value: 'Nterms' }
    ],
    axiom: [
      { type: 'nterm', value: 'Nterm' },
      { type: 'nterm', value: 'Nterms' }
    ]
  },
  Nterms: {
    nterm: [
      { type: 'nterm', value: 'Nterm' },
      { type: 'nterm', value: 'Nterms' }
    ],
```

```
    axiom: [
      { type: 'nterm', value: 'Nterm' },
      { type: 'nterm', value: 'Nterms' }
    ],
    [Symbol(EOF)]: []
  },
  Nterm: {
    nterm: [
      { type: 'term', value: 'nterm' },
      { type: 'term', value: '=' },
      { type: 'nterm', value: 'Rule' },
      { type: 'nterm', value: 'Rules' }
    ],
    axiom: [
      { type: 'term', value: 'axiom' },
      { type: 'term', value: '=' },
      { type: 'nterm', value: 'Rule' },
      { type: 'nterm', value: 'Rules' }
    ]
  },
  Rules: {
    '|': [
      { type: 'term', value: '|' },
      { type: 'nterm', value: 'Rule' },
      { type: 'nterm', value: 'Rules' }
    ],
    '.': [ { type: 'term', value: '.' } ]
  },
  Rule: {
    nterm: [
      { type: 'term', value: 'nterm' },
      { type: 'nterm', value: 'Rule' }
    ],
    term: [
      { type: 'term', value: 'term' },
      { type: 'nterm', value: 'Rule' }
    ],
    '|': [],
    '.': []
  }
}
```

Листинг 3 — таблица предсказывающего разбора

```
export const compiler = <L extends { type: string }>({
  lexer, lang, aggregate = {}, print = (s) => console.log(s.replace(/^/gm, '>
')),
}: {
  lexer: ReturnType<Wrapper<L>['lexicalAnalyzer']>,
  lang: Language,
  aggregate?: Record<string, ((...args: any[]) => any) | undefined>,
  print?: (s: string) => void,
}) => (s: string): { ok: true, res: any } | { ok: false, res: undefined } =>
{
    let lexems: SmartLexems<L>[];
    try {
```

```
  lexems = [...lexer(s)];
} catch ({
  message, groups, from, to,
}) {
  print(`Lexer Error: ${message}}`);
  print(`             groups: ${JSON.stringify(groups)}`);
  print(`             from: ${JSON.stringify(from)}`);
  print(`             to: ${JSON.stringify(to)}`);
  return { ok: false, res: undefined };
}

let table: ReturnType<typeof createTable>;
try {
  table = createTable(lang);
} catch ({
  message, X, a, value,
}) {
  print(`Table Error: ${message}`);
  print(`             X: ${JSON.stringify(X)}`);
  print(`             a: ${JSON.stringify(a)}`);
  print(`             value: ${JSON.stringify(value)}`);
  return { ok: false, res: undefined };
}

let indexLexem = 0;
const calc = (nterm: string): any => {
  const currentLexem = lexems.at(indexLexem);
  const currentLexemType = currentLexem ? currentLexem.type : EOF;
  const detaXA = table[nterm]?.[currentLexemType];

  if (detaXA === undefined) {
    throw {
      message: 'no transition between nterm & term',
      nterm,
      term: currentLexem,
    };
  }

  const res = detaXA.map((v) => {
    if (v.type === 'term') {
      const now = lexems.at(indexLexem);
      if (v.value !== now?.type) {
        throw {
          message: 'unexpected term',
          nterm,
          term: [v.value, now],
        };
      }
      indexLexem++;
      return now;
    }
    return calc(v.value);
  });

  const aggreg = aggregate[nterm];
  if (aggreg) {
    return aggreg(...res);
```

```
    }
    // @ts-ignore
    res.nterm = nterm;
    return res;
  };

  let res;
  try {
    res = calc(lang.axiom);
  } catch ({ message, nterm, currentLexem }) {
    print(`Calc Error: ${message}`);
    print(`            nterm: ${JSON.stringify(nterm)}`);
    print(`            term: ${JSON.stringify(currentLexem)}`);
    return { ok: false, res: undefined };
  }

  return { ok: true, res };
};
```

Листинг 4 — алгоритм предсказывающего разбора, работающего на основе порожденной таблицы

Вывод

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