Implementação de Analisador Sintático para Linguagem *Lang*

Diego Paiva - 201565516C
Thaynara Ferreira - 201565254C

Outubro, 2020

Resumo

O presente documento tem como objetivo esclarecer as principais decisões de projeto tomadas na implementação do analisador sintático para a linguagem fictícia *Lang*, referente ao trabalho prático da disciplina *DCC045 - Teoria dos Compiladores*.

1 Estrutura Léxica

A saber, na implementação deste trabalho, existem dois tipos de *tokens* em *Lang*, aqueles com estrutura fixa (separadores, operadores, etc) e os que seguem uma das formas a seguir:

- `ID`: Sequência de letras, dígitos e sobrescritos (underscore) que, obrigatoriamente, começa com uma letra. Ex: `var`, `var1`, `fun10`, etc.
- `TYPE_NAME`: Identificador cuja primeira letra é maiúscula. Ex: `Racional`, `Point`, etc.
- `INT`: Sequência de um ou mais dígitos. Ex: `0`, `1`, `10`, `128`, etc.
- `FLOAT`: Sequência de zero ou mais dígitos, seguido por um ponto e uma sequência de um ou mais dígitos. Ex: `3.141526535`, `1.0`, `.12345`, etc.
- `CHAR`: Um único caractere delimitado por aspas simples. Os caracteres especiais quebra-de-linha, tabulação, *backspace* e *carriage return* são definidos usando os caracteres de escape `\n`, `\t`, `\b` e `\r`, respectivamente. Para especificar um caractere `\` , é usado `\\` e para a aspas simples o `\'`. Ex: `'a'`, `'\n'`, `'\t'`, `'\\'` e `'\''`, etc.

2 Estrutura Sintática

A linguagem fictícia *Lang* é uma linguagem com propósitos meramente educacionais. Ela possui construções simples como tipos de dados, declarações, funções, comandos e expressões. Sua estrutura pode ser representada pela gramática a seguir.

$$
\begin{aligned}
\text{prog} &\rightarrow \text{data}^* \ \text{func}^* \\
\text{data} &\rightarrow \textbf{data} \ \text{TYPE\_NAME} \ \text{'\{'} \ \text{decl}^* \ \text{'\}'} \\
\text{decl} &\rightarrow \text{ID} \ \text{'::'} \ \text{type} \ \text{';'} \\
\text{func} &\rightarrow \text{ID} \ \text{'('} \ \text{params}^? \ \text{')'} \ (\text{':'} \ \text{type} \ (\text{','} \ \text{type})^*)^? \ \text{'\{'} \ \text{cmd}^* \ \text{'\}'} \\
\text{params} &\rightarrow \text{ID} \ \text{'::'} \ \text{type} \ (\text{','} \ \text{ID} \ \text{'::'} \ \text{type})^* \\
\text{type} &\rightarrow \text{type} \ \text{'['} \ \text{']'} \\
&\quad | \ \text{btype} \\
\text{btype} &\rightarrow \textbf{Int} \\
&\quad | \ \textbf{Char}
\end{aligned}
$$

$$
\begin{aligned}
&\mid \textbf{Bool}\\
&\mid \textbf{Float}\\
&\mid \text{TYPE\_NAME}\\
\text{cmd} \rightarrow\ &\text{'\{'}\ \text{cmd}^*\ \text{'\}'}\\
&\mid \textbf{if}\ \text{'('}\ \text{exp}\ \text{')'}\ \text{cmd}\\
&\mid \textbf{if}\ \text{'('}\ \text{exp}\ \text{')'}\ \text{cmd}\ \textbf{else}\ \text{cmd}\\
&\mid \textbf{iterate}\ \text{'('}\ \text{exp}\ \text{')'}\ \text{cmd}\\
&\mid \textbf{read}\ \text{lvalue}\ \text{';'}\\
&\mid \textbf{print}\ \text{exp}\ \text{';'}\\
&\mid \textbf{return}\ \text{exp}\ (\text{','}\ \text{exp})^*\ \text{';'}\\
&\mid \text{lvalue}\ \text{'='}\ \text{exp}\ \text{';'}\\
&\mid \text{ID}\ \text{'('}\ \text{exps}^?\ \text{')'}\ (\text{'<'}\ \text{lvalue}\ (\text{','}\ \text{lvalue})^*\ \text{'>'})^?\ \text{';'}\\
\text{exp} \rightarrow\ &\text{exp}\ \text{'\&\&'}\ \text{exp}\\
&\mid \text{rexp}\\
\text{rexp} \rightarrow\ &\text{aexp}\ \text{'<'}\ \text{aexp}\\
&\mid \text{rexp}\ \text{'=='}\ \text{aexp}\\
&\mid \text{rexp}\ \text{'!='}\ \text{aexp}\\
&\mid \text{aexp}\\
\text{aexp} \rightarrow\ &\text{aexp}\ \text{'+'}\ \text{mexp}\\
&\mid \text{aexp}\ \text{'-'}\ \text{mexp}\\
&\mid \text{mexp}\\
\text{mexp} \rightarrow\ &\text{mexp}\ \text{'*'}\ \text{sexp}\\
&\mid \text{mexp}\ \text{'/'}\ \text{sexp}\\
&\mid \text{mexp}\ \text{'\%'}\ \text{sexp}\\
&\mid \text{sexp}\\
\text{sexp} \rightarrow\ &\text{'!'}\ \text{sexp}\\
&\mid \text{'-'}\ \text{sexp}\\
&\mid \textbf{true}\\
&\mid \textbf{false}\\
&\mid \textbf{null}\\
&\mid \text{INT}\\
&\mid \text{FLOAT}\\
&\mid \text{CHAR}\\
&\mid \text{pexp}\\
\text{pexp} \rightarrow\ &\text{lvalue}\\
&\mid \text{'('}\ \text{exp}\ \text{')'}\\
&\mid \text{'new'}\ \text{btype}\ (\text{'['}\ \text{sexp}\ \text{']'})^?\\
&\mid \text{ID}\ \text{'('}\ \text{exps}^?\ \text{')'}\ \text{'['}\ \text{exp}\ \text{']'}\\
\text{lvalue} \rightarrow\ &\text{ID}\\
&\mid \text{lvalue}\ \text{'['}\ \text{exp}\ \text{']'}\\
&\mid \text{lvalue}\ \text{'.'}\ \text{ID}\\
\text{exps} \rightarrow\ &\text{exp}\ (\text{','}\ \text{exp})^*
\end{aligned}
$$

Onde a^* e $a^?$ denotam, respectivamente, zero ou mais ocorrências e zero ou uma ocorrência

da regra *a*. Cadeias entre aspas simples representam *tokens* e palavras em negrito são palavras reservadas da linguagem.

3 Ajuste da Gramática

É possível perceber que a linguagem *Lang*, assim como muitas outras, possui regras que são recursivas à esquerda, e.g., aexp → aexp '+' mexp. Além disso, também há regras que contém prefixos em comum (fatoração à esquerda), e.g., mexp → mexp '*' sexp | mexp '/' sexp | mexp '%' sexp, o que acaba gerando ambiguidade na construção da linguagem.

Para tratar estes problemas, é necessário que sejam feitos ajustes nas regras que contém tais inconsistências. Nas Seções a seguir mostraremos como foram feitos estes tratamentos na gramática em questão.

3.1 Remoção da Recursão à Esquerda

Para uma determinada regra $A \rightarrow A\alpha \mid \beta$, é possível remover sua recursividade à esquerda substituindo A pelas regras $A \rightarrow \beta A'$ e $A' \rightarrow \alpha A' \mid \varepsilon$.

3.2 Refatoração à Esquerda

Para uma determinada regra $A \rightarrow \alpha\,\beta \mid \alpha\,\gamma$, é possível refatorá-la à esquerda substituindo A pelas regras $A \rightarrow \alpha A'$ e $A' \rightarrow \beta \mid \gamma$.

3.3 Ajustes em *Lang*

Apresentados os métodos para tratamento de recursão à esquerda e de fatoração à esquerda, enumeramos as regras que foram modificadas em *Lang*.

1. type → btype type′
   type′ → '[' ']' type′
   | ε
2. cmd → '{' cmd* '}'
   | **if** '(' exp ')' cmd cmd′
   | **iterate** '(' exp ')' cmd
   | **read** lvalue ';'
   | **print** exp ';'
   | **return** exp (',' exp)* ';'
   | lvalue '=' exp ';'
   | ID '(' exps? ')' ('<' lvalue (',' lvalue)* '>')? ';'
   cmd′ → **else** cmd
   | ε
3. exp → rexp exp′
   exp′ → '&&' exp exp′
   | ε

4. rexp → aexp rexp′ rexp‴
   rexp‴ → rexp″ rexp‴
   | ε
   rexp′ → '<' aexp
   | ε
   rexp″ → '==' aexp
   | '!=' aexp

5. aexp → mexp aexp″
   aexp″ → aexp′ aexp″
   | ε
   aexp′ → '+' mexp
   | '-' mexp

6. mexp → sexp mexp″
   mexp″ → mexp′ mexp″
   | ε
   mexp′ → '*' sexp
   | '/' sexp
   | '%' sexp

7. lvalue → ID lvalue″
   lvalue″ → lvalue′ lvalue″
   | ε
   lvalue′ → '[' exp ']'
   | '.' ID

4 Implementação

Para gerar um analisador sintático para a linguagem, utilizou-se a ferramenta **ANTLR**. Segundo [1], "ANTLR é um gerador de analisador poderoso para ler, processar, executar ou traduzir texto estruturado ou arquivos binários. É amplamente utilizado para construir linguagens, ferramentas e estruturas. A partir de uma gramática, ANTLR gera um analisador que pode construir e percorrer árvores de análise."

5 *Building*

Para realizar o *build* da aplicação, recomenda-se a utilização da ferramenta de gerenciamento **Maven** [2]. Com o Maven instalado, navegue até o diretório do projeto que contém o arquivo `pom.xml`. Após isso, rode o comando:

```
mvn clean compile assembly:single
```

Este comando irá compilar o código, incluir as dependências necessárias (ANTLR) e empacotar em um formato distribuível `jar` dentro da pasta `target`. Para executar o programa principal, rode o seguinte comando:

```
java -jar target/lang-compiler-1.0-jar-with-dependencies.jar -bs
```

Onde `-bs` é a opção para realizar a bateria de testes sintáticos.

Referências

[1] "Antlr." `https://www.antlr.org/`. Acesso em: 25/10/2020.

[2] "Apache maven project." `http://maven.apache.org/`. Acesso em: 04/10/2020.