Compilador para a Linguagem TIGER usando a Linguagem JAVA como Ferramenta de Desenvolvimento

1 A Gramática de TIGER

$$
\begin{array}{rcl}
\textit{exp} & ::= & \textit{l-value} \mid \textbf{nil} \mid \texttt{"("}\ \ \textit{expseq}\ \ \texttt{")"}\ \mid \textit{num} \mid \textit{string} \\
& & \mid \texttt{-}\ \textit{exp} \\
& & \mid \textit{id}\ \ \texttt{"("}\ \ \textit{args}\ \ \texttt{")"} \\
& & \mid \textit{exp}\ \ \texttt{"+"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"-"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"*"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"/"}\ \ \textit{exp} \\
& & \mid \textit{exp}\ \ \texttt{"="}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"<>"}\ \ \textit{exp} \\
& & \mid \textit{exp}\ \ \texttt{"<"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{">"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"<="}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{">="}\ \ \textit{exp} \\
& & \mid \textit{exp}\ \ \texttt{"\&"}\ \ \textit{exp} \mid \textit{exp}\ \ \texttt{"|"}\ \ \textit{exp} \\
& & \mid \textit{type-id}\ \ \texttt{"\{"}\ \ \textit{id}\ \ \texttt{"="}\ \ \textit{exp}\ \textit{idexps}\ \ \texttt{"\}"} \\
& & \mid \textit{type-id}\ \ \texttt{"["}\ \ \textit{exp}\ \ \texttt{"]"}\ \ \textbf{of}\ \textit{exp} \\
& & \mid \textit{l-value}\ \ \texttt{":="}\ \ \textit{exp} \\
& & \mid \textbf{if}\ \textit{exp}\ \textbf{then}\ \textit{exp}\ \textbf{else}\ \textit{exp} \\
& & \mid \textbf{if}\ \textit{exp}\ \textbf{then}\ \textit{exp} \\
& & \mid \textbf{while}\ \textit{exp}\ \textbf{do}\ \textit{exp} \\
& & \mid \textbf{for}\ \textit{id}\ \ \texttt{":="}\ \ \textit{exp}\ \textbf{to}\ \textit{exp}\ \textbf{do}\ \textit{exp} \\
& & \mid \textbf{break} \\
& & \mid \textbf{let}\ \textit{decs}\ \textbf{in}\ \textit{expseq}\ \textbf{end}
\end{array}
$$

$$
\begin{array}{rcl}
\textit{decs} & ::= & \textit{dec}\ \textit{decs} \mid \varepsilon \\
\textit{dec} & ::= & \textit{tydec} \mid \textit{vardec} \mid \textit{fundec} \\
\textit{tydec} & ::= & \textbf{type}\ \textit{id}\ \ \texttt{"="}\ \ \textit{ty} \\
\textit{ty} & ::= & \textit{id} \mid \texttt{"\{"}\ \ \textit{id}\ \ \texttt{":"}\ \ \textit{type-id}\ \textit{tyfields}_1\ \texttt{"\}"}\ \mid \textbf{array of}\ \textit{id} \\
\textit{tyfields} & ::= & \textit{id}\ \ \texttt{":"}\ \ \textit{type-id}\ \textit{tyfields}_1 \mid \varepsilon \\
\textit{tyfields}_1 & ::= & \texttt{","}\ \ \textit{id}\ \ \texttt{":"}\ \ \textit{type-id}\ \textit{tyfields}_1 \mid \varepsilon \\
\textit{vardec} & ::= & \textbf{var}\ \textit{id}\ \ \texttt{":="}\ \ \textit{exp} \mid \textbf{var}\ \textit{id}\ \ \texttt{":"}\ \ \textit{type-id}\ \ \texttt{":="}\ \ \textit{exp} \\
\textit{fundec} & ::= & \textbf{function}\ \textit{id}\ \ \texttt{"("}\ \ \textit{tyfields}\ \texttt{")"}\ \ \texttt{"="}\ \ \textit{exp} \\
& & \mid \textbf{function}\ \textit{id}\ \ \texttt{"("}\ \ \textit{tyfields}\ \ \texttt{")"}\ \ \texttt{":"}\ \ \textit{type-id}\ \ \texttt{"="}\ \ \textit{exp}
\end{array}
$$

$$
\begin{array}{rcl}
\textit{l-value} & ::= & \textit{id} \mid \textit{l-value}\ \ \texttt{"."}\ \ \textit{id} \mid \textit{l-value}\ \ \texttt{"["}\ \ \textit{exp}\ \ \texttt{"]"} \\
\textit{type-id} & ::= & \textit{id}
\end{array}
$$

$$
\begin{array}{rcl}
\textit{expseq} & ::= & \textit{exp}\ \textit{expseq}_1 \mid \varepsilon \\
\textit{expseq}_1 & ::= & \texttt{";"}\ \ \textit{exp}\ \textit{expseq}_1 \mid \varepsilon
\end{array}
$$

$$
\begin{array}{rcl}
\textit{args} & ::= & \textit{exp}\ \textit{args}_1 \mid \varepsilon \\
\textit{args}_1 & ::= & \texttt{","}\ \ \textit{exp}\ \textit{args}_1 \mid \varepsilon
\end{array}
$$

$$
\begin{array}{rcl}
\textit{idexps} & ::= & \texttt{","}\ \ \textit{id}\ \ \texttt{"="}\ \ \textit{exp}\ \textit{idexps} \mid \varepsilon
\end{array}
$$

2 O Compilador para TIGER

O nosso compilador TIGER foi implementado usando o paradigma orientado por objeto. A escolha por este paradigma se explica pela necessidade de desenvolver a ferramenta o mais modular possível, o que, na orientação por objeto, vem a ser uma premissa básica. Organizamos a implementação de TIGER em seis etapas, as quais listamos a seguir:

- Análise Léxica
- Análise Sintática
- Tabela de Símbolos
- Sintaxe Abstrata
- Análise Semântica
- Geração de Código Intermediário

3 Metodologia para o Desenvolvimento do seu Compilador TIGER

O seu compilador para TIGER pode ser implementado na linguagem JAVA, C ou C++, de preferência em JAVA, mas antes de começar a projetá-lo, você deve se familiarizar com a linguagem TIGER. Para isto está disponível no endereço eletrônico:

`/~mariza/Cursos/CompiladoresI/Geral/Projeto2021-2/Proj2021-2`

os pacotes com as classes responsáveis pelas funcionalidades do compilador TIGER. A Seção 3.1 descreve cada um destes pacotes e mostra como usar o compilador.

3.1 Interface com o usuário

A execução do compilador pode ser feita a partir de qualquer plataforma que contenha um interpretador *java* instalado.

O compilador está disponibilizado em diversos pacotes, cada um contendo classes relacionadas e responsáveis por determinada funcionalidade do compilador. Listamos em seguida estes pacotes, descrevendo a funcionalidade principal de cada um:

/Absyn: camada de abstração que permite separar as ações sintáticas das ações semânticas.

/Assem: provê uma estrutura de tipos de dados para instruções de linguagem *assembly* sem designação de registradores.

/ErrorMsg: responsável pela produção de mensagens de erro.

/Frame: provê a estrutura de dados referente ao registro de ativação, sem ater-se a detalhes de implementação.

/JavaVM: implementa detalhes referentes à Máquina Virtual JAVA.

/Main: responsável pelo direcionamento do fluxo de dados no compilador.

/Parse: contém os analisadores léxico e sintático.

/Semant: contém o analisador semântico.

/Symbol: provê a tabela de símbolos em conjunto com suas operações de acesso.

/Temp: provê os temporários e os *labels*.

/Translate: provê a tradução para código intermediário.

/Tree: provê a linguagem da árvore de representação intermediária.

/Types: descreve os tipos de dados da linguagem TIGER.

/Util: provê a classe de lista de booleanos.

Para executar o compilador deve-se digitar a seguinte linha de comando:

```
java Main.Main <NOME DO ARQUIVO> [-opções]
```

onde:

`<NOME DO ARQUIVO>:` é o arquivo contendo o programa TIGER a ser compilado e opções pode ser:

absyn: imprime na saída padrão a árvore de sintaxe abstrata.

listinput: imprime na saída padrão a listagem do arquivo de entrada.

listparse: imprime na saída padrão os estados percorridos pelo analisador sintático.

intermcode: imprime na saída padrão o código intermediário gerado.

Para redirecionar a saída das opções para um arquivo basta digitar a seguinte linha de comando:

`java Main.Main <NOME DO ARQUIVO> [-opções] > <nome-arquivo-saida>`

onde: `<nome-arquivo-saida>`: é o arquivo contendo a saída das opções requisitadas.

3.2 Ferramentas de Apoio

3.2.1 JLex

A ferramenta JLex é um gerador de analisador léxico para JAVA que recebe como entrada um arquivo com a especificação léxica da linguagem na forma de expressões regulares e gera o código fonte JAVA correspondente ao analisador léxico.

Esta ferramenta encontra-se disponível no endereço [1] em formato de código fonte JAVA e, portanto, deve ser compilada antes de sua execução. Deve-se, ainda, ajustar-se o CLASSPATH para que a mesma possa ser referenciada a partir de qualquer caminho do ambiente em questão. A seguir apresentamos como isto pode ser feito em ambiente Unix com bash `cshell`. Supondo que os arquivos correspondentes ao JLex estejam no caminho `/java/JLex`, inclua no arquivo `.cshrc` a diretiva:

`setenv CLASSPATH ./:/java` Para executar o JLex entre com a linha de comando:

`java JLex.Main <NOME DO ARQUIVO>` onde:
`<NOME DO ARQUIVO>` é o arquivo de especificação léxica para a linguagem em questão. Como resultado da execução desta linha de comando, será gerado o arquivo `<NOME DO ARQUIVO>.java` se não houver erros no arquivo de entrada.

3.2.2 CUP

A ferramenta CUP é um gerador de analisador sintático LALR para JAVA. Como resultado, ela gera o código fonte JAVA correspondente ao analisador sintático.

Esta ferramenta pode ser obtida no endereço [2] e deve ser compilada pelo interpretador JAVA antes de ser executada. Tal como na ferramenta

JLex, ela deve ser incluída no CLASSPATH para ser referenciada a partir de qualquer caminho do ambiente. Supondo-se que a ferramenta foi instalada no caminho `/java/java_cup`, podemos referenciar a variável CLASSPATH da seguinte forma:

```
setenv CLASSPATH ./:/java
```
[1]

Para executar o CUP entre com a linha de comando:

`java java_cup.Main < <NOME DO ARQUIVO>` onde `<NOME DO ARQUIVO>` é o arquivo contendo a especificação da gramática para a linguagem em questão. Não havendo erros durante a execução do CUP, são gerados dois arquivos fonte em JAVA, `parser.java` e `sym.java`, além do arquivo binário `CUP$Grm$actions.class`.

Referências

[1] Berk, Elliot, *JLex: A Lexical Analyser Generator for JAVA$^{(TM)}$*, http://www.cs.princeton.edu/~appel/modern/java/JLex, 1997.

[2] Hudson, Scott, *LALR Parser Generator for JAVA^{TM}*, http://www.cs.princeton.edu/~appel/modern/java/CUP, 1998.

[1] o caminho `/java/java_cup` descrito é aquele que contém o arquivo `Main.class`.