INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

Curso de Engenharia de Computação
Linguagens Formais, Autômatos e Compiladores

Gerador de analisadores sintáticos ascendentes CUP

Slides da disciplina ECM253 – Linguagens Formais, Autômatos e Compiladores
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia – Escola de Engenharia Mauá
Prof. Marco Antonio Furlan de Souza

CUP

- **Conceitos**
  - **CUP** = *Constructor of Useful Parsers*;
  - **URL**: http://www2.cs.tum.edu/projects/cup/docs.php;
  - **CUP** é um sistema para **gerar analisadores sintáticos LALR** a partir de especificações simples;
  - Basta **criar** uma **especificação baseada** na **gramática desejada** e **também** com a **construção** de um **analisador léxico** capaz de **tokenizar** a entrada.

CUP

- **Especificação do arquivo de entrada**
  - **Especificação do pacote e de importações**
    - **Opcional** – definição do pacote que conterá o código a ser gerado e de importações necessárias. Define-se, também o nome da classe do analisador:
  - **Componentes do código-fonte do usuário**
    - Dividido nas partes:
      - `action code {: … :} code` (opcional – código a ser adicionado em classe de suporte privada);
      - `parser code {: … :}` (opcional – métodos e variáveis que podem ser adicionados à classe do analisador);
      - `init with {: … :}` (inicialização pré-análise léxica. Por exemplo, inicializar o analisador léxico);
      - `scan with {: … :}` (opcional – código a ser gerado no corpo do analisador que especificará como obter o próximo token).

CUP

- **Especificação do arquivo de entrada**
  - **Lista de símbolos**
    - Define os símbolos da gramática – terminais e não terminais:
      - `terminal classname name1, name2, … ;`
      - `non terminal classname name1, name2, … ;`
      - `terminal name1, name2, … ;`
      - `non terminal name1, name2, … ;`
  - **Declarações de precedência e associatividade**
    - Opcional, especifica a precedência e associatividade dos terminais:
      - `precedence left terminal[, terminal...];`
      - `precedence right terminal[, terminal...];`
      - `precedence nonassoc terminal[, terminal...];`
  - **Gramática**
    - Regras na forma `X ::= α;`, seguidas ou não de ações `{: … :}`.

CUP

- **Exemplo**
  - Considerar a gramática:

```
expr_list ::= expr_list expr_part | expr_part
expr_part ::= expr ';'
expr      ::= expr '+' expr | expr '-' expr | expr '*' expr
            | expr '/' expr | expr '%' expr | '(' expr ')'
            | '-' expr | number
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Baixar a distribuição de http://www2.cs.tum.edu/projects/cup/install.php.
    - Descompactar os arquivos. Existem **dois arquivos** do tipo jar na distribuição:
      - `java-cup-11b.jar`: deve ser **incorporado** ao **projeto** durante o **desenvolvimento**;
      - `java-cup-11b-runtime.jar`: arquivo que, se desejado, **pode substituir** `java-cup-11b.jar` no **aplicativo final** (ele é menor em tamanho) a ser distribuído nos clientes.

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar um **novo projeto Java** no **Eclipse** com nome `CUPExpParser`;
    - **Adicionar** a **biblioteca** do **CUP** `java-cup-11b.jar` ao projeto:
      - Criar a pasta `tools`, abaixo da pasta do projeto;
      - Copiar o arquivo `java-cup-11b.jar` para esta pasta;
    - **Adicionar** a **biblioteca** do **JFlex** `jflex-1.6.1.jar` na pasta `tools` do passo anterior;
    - Criar as pastas `src/jflex` e `src/cup` no projeto.

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - No **Eclipse**, criar arquivo do **Ant**, `build.xml`:

```
<project name="ExpressionInterpreter" default="dist" basedir=".">
    <description>
        Ant build file for a simple expression interpreter.
    </description>
    <!-- set global properties for this build -->
    <property name="src" location="src" />
    <property name="tools" location="tools" />
    <property name="jflex" location="src/jflex" />
    <property name="cup" location="src/cup" />
    <property name="build" location="build" />
    <property name="dist" location="dist" />

    <taskdef name="jflex" classname="jflex.anttask.JFlexTask"
        classpath="${tools}/jflex-1.6.1.jar" />

    <taskdef name="cup" classname="java_cup.anttask.CUPTask"
        classpath="${tools}/java-cup-11b.jar" />

    <target name="init">
        <!-- Create the build directory structure used by compile -->
        <mkdir dir="${build}" />
    </target>
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - No **Eclipse**, criar arquivo do **Ant**, `build.xml` (cont.):

```
<target name="compile" depends="init" description="compile the source">
    <!-- Run jflex from jflex dir -->
    <jflex file="${jflex}/Scanner.jflex" destdir="${src}" />
    <!-- Run cup from cup dir -->
    <cup srcfile="${cup}/Parser.cup" destdir="${src}"
        parser="MyParser" interface="true" locations="false" />
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}" classpath="${tools}/java-cup-11b.jar"/>
</target>

<target name="dist" depends="compile" description="generate the distribution">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}" />
    <!-- Put everything in ${build} into the jar file -->
    <jar jarfile="${dist}/cup_exp_parser.jar" basedir="${build}">
        <manifest>
            <attribute name="Main-Class" value="parser.Main" />
    </manifest>
    </jar>
</target>
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - No **Eclipse**, criar arquivo do **Ant**, `build.xml` (cont.):

```
    <target name="clean" description="clean up">
        <!-- Delete the ${build} and ${dist} directory trees -->
        <delete dir="${build}" />
        <delete dir="${dist}" />
     </target>
</project>
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar arquivo do **JFlex**, `src/jflex/Scanner.jflex`:

```
/* JFlex Scanner for CUP example. */
package parser;
import java_cup.runtime.*;
%%
%class MyScanner
%cup
%unicode
%line
%column
%{
    private Symbol symbol(int type) {
        return new Symbol(type, yyline, yycolumn);
    }
    private Symbol symbol(int type, Object value) {
        return new Symbol(type, yyline, yycolumn, value);
    }
%}

ws = [ \t\f\r\n]
number = [0-9]+
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar arquivo do **JFlex**, `src/jflex/Scanner.jflex` (cont.):

```
%%
";"           { return symbol(sym.SEMI); }
"+"           { return symbol(sym.PLUS); }
"-"           { return symbol(sym.MINUS); }
"*"           { return symbol(sym.TIMES); }
"/"           { return symbol(sym.DIVIDE); }
"%"           { return symbol(sym.MOD); }
"("           { return symbol(sym.LPAREN); }
")"           { return symbol(sym.RPAREN); }
{number} { return symbol(sym.NUMBER, new Integer(yytext()) ); }
{ws}     { /* ignore */ }
.        { return symbol(sym.ERROR, yytext()); }
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar arquivo do **CUP**, `src/cup/Parser.cup`:

```
// CUP specification for a simple expression evaluator (w/ actions)

package parser;
import java_cup.runtime.*;

/* Terminals (tokens returned by the scanner). */
terminal           ERROR, SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal           UMINUS, LPAREN, RPAREN;
terminal Integer   NUMBER;

/* Non-terminals */
non terminal            expr_list, expr_part;
non terminal Integer    expr;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar arquivo do **CUP**, `src/cup/Parser.cup` (cont.):

```
/* The grammar */

expr_list ::= expr_list expr_part
    | expr_part
    ;

expr_part ::= expr:e {: System.out.println("gives " + e); :} SEMI
    | error SEMI
    ;

expr ::= expr:e1 PLUS expr:e2 {: RESULT = new Integer(e1.intValue() + e2.intValue()); :}
    | expr:e1 MINUS expr:e2 {: RESULT = new Integer(e1.intValue() - e2.intValue()); :}
    | expr:e1 TIMES expr:e2 {: RESULT = new Integer(e1.intValue() * e2.intValue()); :}
    | expr:e1 DIVIDE expr:e2{: RESULT = new Integer(e1.intValue() / e2.intValue()); :}
    | expr:e1 MOD expr:e2 {: RESULT = new Integer(e1.intValue() % e2.intValue()); :}
    | NUMBER:n {: RESULT = n; :}
    | MINUS expr:e {: RESULT = new Integer(0 - e.intValue()); :} %prec UMINUS
    | LPAREN expr:e RPAREN {: RESULT = e; :}
    ;
```

CUP

- **Exemplo**
  - **Preparação do ambiente**
    - Criar a classe principal, `src/Main.java`:

```
package parser;

import java.io.InputStreamReader;

class Main {
    public static void main(String[] args) {
        MyParser parser = new MyParser(new MyScanner(new InputStreamReader(System.in)));
        try {
            parser.parse();
        } catch (Exception e) {
            System.out.println("Caught an exception.");
        }
    }
}
```

CUP

- **Exemplo**
  - **Compilação e execução**
    - Construir o projeto com `Ant build`;
    - **Executar**
      - Assumindo que se está no terminal, na pasta `dist`:
      - No Linux:

```
java -cp cup_exp_parser.jar:../tools/java-cup-11b.jar parser.Main
```

      - No Windows:

```
java -cp cup_exp_parser.jar;../tools/java-cup-11b.jar parser.Main
```