Progetto di
Linguaggi e Traduttori
2015/2016

Damiano Di Stefano – Marco Giuseppe Salafia

AST

Il progetto richiedeva la traduzione di un sorgente scritto in un formato ridotto del linguaggio C (MINI C).
Per generare l'analizzatore lessicale abbiamo utilizzato JFlex mentre come generatore di Parser abbiamo fatto uso di Java Cup.

Il Progetto è organizzato nelle seguenti directory:

- *jflex*
- *cup*
- *xsl*
- *output*

1. ./jflex
Questa directory contiene il file jflex *minic.jflex* per generare il Lexer necessario al Parser Cup.
Nel primo settore (*user code*) abbiamo messo solamente gli import che saranno ricopiati nella java class Lexer che sarà generata. In questa sezione è necessario inserire gli import di java_cup.runtime necessarie al funzionamento del lexer.

Nella seconda sezione (*options and declarations*) oltre alle varie opzioni abbiamo incluso tra i tag "%{ %}" (necessari per l'inserimento di codice che sarà copiato tale e quale nella classe java) i costruttori dei simboli e della classe Lexer. Quest'ultimo ha come argomenti un Reader e una ComplexSymbolFactory.
Come macro abbiamo definito:

```
D = 0 | [1-9][0-9]*                 Cifre Numeriche
L=[A-Za-z]                          Letterali
AN= [0-9A-Za-z]                     Alfanumerici
new_line = \r|\n|\r\n
white_space = {new_line} | [ \t\f]
```

Infine, è fondamentale l'inserimento dello scanning method %eofval { … %eofval} che ritorna al parser il simbolo EOF quando l'input è stato completamente analizzato.

Nel terzo ed ultimo settore (*lexical rules*) abbiamo inserito le direttive per identificare tutti i lessemi necessari al Parser, come ad esempio:

```
"if"            { return symbol(IF, yytext()); }
 {L}({AN})*     { return symbol(IDENT,yytext()); }
```

Queste due regole sono state riportate proprio in questo ordine in quanto si deve dare la precedenza al lessema <if> inteso come parola chiave che ad <if> inteso come identificatore.

2. ./jcup
In questa directory è presente il file il file minic.cup per generare il Parser che utilizzando il Lexer creato con Jflex, generando un albero di parsing che provvederà a trasformare in AST (XML) e successivamente in un file HTML per visualizzare l'albero in SVG.

Nella prima parte sono presenti tuti gli import fra cui anche i package javax.xml per gestire la creazione degli XML.
Il codice inserito all'interno di *parser code {: … :};* viene copiato direttamente nel file Parser.java che verrà generato. In questa sezione abbiamo inserito:

- Le funzioni predefinite che vengono chiamate da cup in seguito agli errori di sintassi.
- La funzione main dove viene istanziato il Lexer creato con JFlex che legge il file sorgente da analizzare come parametro. Inoltre all'interno del main è contenuto tutto il codice necessario alla creazione dell'XML e la successiva traduzione in HTML con SVG.

Successivamente abbiamo inserito tutte le specifiche della grammatica: terminali e non terminali, seguite poi dalle precedenze degli operatori. Infine abbiam inserito tutte le produzioni della grammatica che definisce il linguaggio MINI C.

La grammatica che ci è stata fornita nelle specifiche di progetto utilizzava una notazione di Backus-Naur estesa che presentava una stella di Kleene. Non sapendo se Java Cup potesse gestire la stella di Kleene e non avendo trovato nessuna soluzione, abbiamo aggirato il problema aggiungendo una nuova produzione con un nuovo non terminale. Abbiamo cambiato la grammatica nel seguente modo:

<program> ::= { <stmt>* } => <program> ::= { <stmtlist> }
<stmtlist> ::= <stmtlist> <stmt> | ε

3. Visualizzazione dell'albero
Per effettuare la traduzione dell'albero di parsing siamo partiti dall'esempio MINI JAVA del sito ufficiale di JCup, modificando opportunamente il codice per visualizzare l'albero in SVG. JCup crea l'albero di parsing su un file XML che passiamo come argomento (*simple.xml*). Tramite un XSL (*tree.xsl*) questo albero di parsing viene trasformato nell'AST corrispondente (*output.xml*) che poi sarà quello che verrà rappresentato graficamente con SVG in una pagina HTML (svg.html).
Il file svg.html è stato generato tramite il file *tree-view-svg.xsl* che si occupa di definire come l'AST deve essere rappresentato graficamente in un html.
Il collegamento dei nodi è stato effettuato tramite il tag:

```
<svg>
        <line> … </line>
</svg>
```

I nodi invece sono rappresentati tramite il tag :

```
<svg>
        <rect> … </rect>
        <text> Nome_del_nodo </text>
</svg>
```

Nel file tree-view-svg.xsl inoltre è definito il template per generare una pagina html collegata a un foglio di stile CSS che abbiamo utilizzato per personalizzare il rendering della pagina.

Inoltre, per dare una valore aggiunto, abbiamo inserito uno script JQuery che legge semplicemente il file sorgente di input del Parser e visualizza il codice in una sezione della pagina HTML accanto all'AST.

```
$(document).ready(function(){
            $.get('../input.c', function(data)
            {
               $("code").text(data);
               Prism.highlightElement($('code')[0]);
            }, 'text');
});
```