Istruzioni per l'utilizzo di Bison/JFlex - Java

Sia data la seguente grammatica:

S → aSb

S → ab

Definizione delle regole lessicali e sintattiche

1. Creare un file `MyGrammar.y` dove andremo a definire le regole per l'analisi sintattica:

- Questo file deve rispettare la seguente struttura:

```
// Definizioni e librerie

// main
%code{ }

// Dichiarazione dei token
%token

%%

// Grammatica

%%

// Codice Java addizionale
```

  - **Definizioni e librerie**: in questa sezione vanno inseriti i prototipi delle funzioni che verranno utilizzate dal parser e le librerie necessarie (già presenti nella distribuzione).
  - **Main**: il main va definito all'interno del blocco ***code***. Qui andremo a leggere l'input da file, eseguire l'analisi lessicale e sintattica, ed infine stampare il risultato del parsing:

```
...
%code {
    public static void main(String args[]) throws IOException {
        MyGrammarLexer lexer = new MyGrammarLexer(System.in);
        MyGrammar parser = new MyGrammar(lexer);
        if(parser.parse())
        System.out.println("Parsing Result = SUCCESS");
        return;
    }
}
...
```

  - **Dichiarazione dei token**: In questo esempio, essendo i token singoli caratteri, potremmo saltare questa sezione ed inserirli direttamente nella grammatica. Nel nostro caso quindi inseriremo solo il *UNKNOWN_TOKEN* che riconosce un token non facente parte del linguaggio.

```
...
%token UNKNOWN_TOKEN
...
```

  - **Grammatica**: si tratta di riscrivere la grammatica nella forma EBNF (Extended Backus–Naur form):

```
rule: prod_1 | prod_2 | ... | prod_n;
```

    Per gestire il problema dei prefissi, aggiungere un regola iniziale che produce l'assioma della nostra grammatica. Nel nostro caso:

```
...
%%

prog:
    srule
;

srule
    : 'a' srule 'b'
    | 'a' 'b'
;

%%
...
```

- **Codice Java addizionale**: In questa sezione vanno inserite le funzioni per la gestione degli errori e del Lexer (già presenti nella distribuzione).

2. Creare un file `MyGrammar.l` dove andremo a definire le regole per l'analisi lessicale:

- Questo file deve rispettare la seguente struttura:

```
%%

// Opzioni

// Dichiarazioni token

%%
<YYINITIAL> {
    // Regole per il lexer
}
```

- **Opzioni**: in questa sezione vanno inserite le opzioni di default per utilizzare JFlex (già presenti nella distribuzione).
- **Dichiarazioni token**: In questo esempio, essendo i token singoli caratteri, li abbiamo inseriti direttamente nelle regole per il lexer. Ma nel caso in cui i token siano più complessi allora questa sezione è necessaria. Infatti qui possiamo definire i token tramite espressioni regolari. Se ad esempio al posto del carattere 'a' volessimo un numero qualsiasi allora dovremmo definire un token in questo modo: `num ([0-9])+;` . Oltre a quelli della grammatica in input è necessario definire anche un token che equivalga a tutto quello che il Lexer deve segnalare come "token sconosciuto": Il ' . ' indica, infatti, "tutto tranne i token già definiti".

```
...
UNKNOWN_TOKEN = .

%%
...
```

- **Regole per il Lexer**: qui vanno definite le operazioni da effettuare quando viene riconosciuto un token. Un'operazione necessaria è fare il *return* del token. Questo deve ritornare il nominativo del token e dovrà essere lo stesso che verrà definito nel file 'MyGrammar.y'. Nel nostro esempio inseriamo anche una print per ogni token letto:

```
...
<YYINITIAL> {
    "a"     {System.out.println("[token at line " + yyline + ":" + yycolumn + " = \"" + yytext() + "\"]"); return 'a';}
    "b"     {System.out.println("[token at line " + yyline + ":" + yycolumn + " = \"" + yytext() + "\"]"); return 'b';}
    {UNKNOWN_TOKEN} {return MyGrammarLexer.UNKNOWN_TOKEN;}
}
```

Compilazione ed esecuzione

1. Generare l'analizzatore lessicale (lexer) definito in *MyGrammar.l* con flex:

```
jflex MyGrammar.l
```

2. Generare l'analizzatore sintattico (parser) definito in *MyGrammar.y* con bison:

```
bison MyGrammar.y -L java
```

3. Compilare i file appena generati:

```
javac *.java
```

4. Eseguire passando come parametro di input il file a cui applicare il parsing:

```
java MyGrammar < input-file
```