

POLITECNICO DI TORINO

**Laboratorio di Compilatori**  
Corso di Linguaggi e Traduttori

Esercitazione 3



Stefano Scanzio  
mail: [stefano.scanzio@polito.it](mailto:stefano.scanzio@polito.it)  
sito: <http://www.skenz.it/traduttori>

a.a 2007 / 2008

## Uso avanzato di Cup

- Precedenze degli operatori
- Gestione degli errori sintattici
- Debugging del parser

<http://www.skenz.it/traduttori> Esercitazione 3 2

## Sezione delle Precedenze: Grammatiche volutamente ambigue

- In certi casi la grammatica può essere resa volutamente ambigua al fine di limitare il numero delle regole.
- È necessario però fornire indicazioni sulla risoluzione delle ambiguità.
- Un caso tipico è dato dalle espressioni algebriche:

Grammatica non ambigua	Grammatica Fortemente ambigua
S → E	E → E '+' E
E → E '*' T	E → E '-' E
E → E '/' T	E → E '** E
E → T	E → E '/' E
T → T '*' F	E → '(' E ')'
T → T '/' F	E → INTEGER
T → F	
F → '(' E ')'	
F → INTEGER	

<http://www.skenz.it/traduttori> Esercitazione 3 3

## Associatività

- Operatore associativo a sinistra ( E ::= E '+' E )
  - 1+2+3+4 → 3+3+4 → 6+4 → 10
- Operatore associativo a destra ( E ::= E '+' E )
  - 1+2+3+4 → 1+2+7 → 1+9 → 10
- L'operatore '=' in un assegnazione è un operatore associativo a destra:
  - a = b = 3
  - Un altro operatore associativo a destra è l'elevazione a potenza
  - 3^2^2 → 3^4 → 81

<http://www.skenz.it/traduttori> Esercitazione 3 4

## Sezione delle Precedenze: Operatori

- 1.) E → E '+' E
- 2.) E → E '\*' E
- 3.) E → '(' E ')'
- 4.) E → INTEGER

- La regola 1 (così come la 2) è ambigua in quanto non specifica l'associatività dell'operatore '+' ('\*').
- Inoltre le regole 1 e 2 non specificano la precedenza tra gli operatori '+' e '\*'.
- È possibile suggerire a Cup come comportarsi aggiungendo due informazioni nella sezione delle precedenze.
- La parola chiave precedence left introduce un operatore associativo a sinistra, precedence right introduce un operatore associativo a destra, precedence nonassoc introduce un operatore non associativo.
- L'ordine con cui gli operatori sono dichiarati è inverso alla loro priorità.

<http://www.skenz.it/traduttori> Esercitazione 3 5

## Sezione delle Precedenze: Regole di risoluzione dell'ambiguità

- Ad ogni regola che contiene almeno un terminale definito come operatore, Cup associa la precedenza e l'associatività dell'operatore più a destra.
- Se la regola è seguita dalla parola chiave %prec, la precedenza e l'associatività sono quelle dell'operatore specificato.
- In caso di conflitto shift-reduce, viene favorita l'azione della regola con la precedenza maggiore.
- Se la precedenza è la stessa, si usa l'associatività: sinistra implica reduce, destra shift.

<http://www.skenz.it/traduttori> Esercitazione 3 6

### Sezione delle Precedenze: Esempio

```
precedence left '+', '-'; /* Bassa priorit  *
precedence left '*', '/';
precedence left uminus; /* Alta priorit  */

start with E;

E ::= E '+' E
    | E '-' E
    | E '*' E
    | E '/' E
    | '-' E %prec uminus
    | '(' E ')
    | INTEGER
;
```

http://www.skenz.it/traduttori

Esercitazione 3

7

### Codice utente

- Esistono alcune direttive che permettono di inserire codice utente direttamente nel parser
- Servono per:
  - Personalizzare il comportamento del parser
  - Aggiungere codice direttamente all'interno della classe che realizza il parser
  - Utilizzare uno scanner diverso da quello di default (JFlex)
- Sono:
  - `init with { ... ; }`
    - ▲ Il codice viene eseguito prima di ogni chiamata allo scanner, quindi prima che qualsiasi simbolo terminale venga fornito al parser
    - ▲ Utilizzata per inizializzare variabili o per inizializzare lo scanner nel caso non si utilizzi JFlex

http://www.skenz.it/traduttori

Esercitazione 3

8

### Codice utente

- `scan with { ... ; }`
  - ▲ Indica al parser quale procedura utilizzare per richiedere il successivo terminale allo scanner
  - ▲ Deve restituire un oggetto di tipo `java_cup.runtime.Symbol`
  - ▲ Anche questa direttiva serve per l'utilizzo di scanner diversi da quello di default (JFlex)
- `parser code { ... ; }`
  - ▲ Il codice viene incluso all'interno della classe che realizza il parser
  - ▲ Includere metodi di scanning all'interno del parser, eseguire l'overriding di funzioni del parser (Ad es. riscrivere le funzioni di gestione degli errori)
- `action code { ... ; }`
  - ▲ Codice incluso nella classe privata in cui   definita la grammatica
  - ▲ Procedure e variabili da utilizzare all'interno delle azioni associate alle regole della grammatica (Symbol table,...)


http://www.skenz.it/traduttori

Esercitazione 3

9

### Errori:

#### stampa linea e colonna



```
import java_cup.runtime.*;
...
%%
%cup
%line
%column

%{
    private Symbol symb(int type){
        return new Symbol(type, yline, ycolumn);
    }
    private Symbol symb(int type, Object value){ //Per la gestione semantica
        return new Symbol(type, yline, ycolumn, value);
    }
}%
...
%%
[a-z]          {return symb(sym.EL); }
' '           {return symb(sym.VIRGOLA); }
```

**Costuttori di Symbol:**

```
public Symbol( int sym_id)
public Symbol( int sym_id, int left, int right)
public Symbol( int sym_id, Object o)
public Symbol( int sym_id, int left, int right, Object o)
```

http://www.skenz.it/traduttori

Esercitazione 3

10



### Errori: stampa linea e colonna

```
import java_cup.runtime.*;

parser code {
    public void syntax_error(Symbol simbolo_attuale) {
        StringBuffer m = new StringBuffer("Errore");

        if (simbolo_attuale.left != -1) {
            m.append(" in linea " + (simbolo_attuale.left+1));
            m.append(", colonna " + (simbolo_attuale.right+1));
        }
        m.append(", simbolo: " + (simbolo_attuale));
        m.append(" : Errore di Sintassi");

        System.err.println(m);
    }
};
```

http://www.skenz.it/traduttori

Esercitazione 3

11

### Gestione degli errori sintattici

- In genere quando un parser incontra un errore non dovrebbe terminare brutalmente l'esecuzione
  - Un compilatore in genere cerca di provvedere alla situazione per poter analizzare il resto del file, in modo da segnalare il maggior numero possibile di errori
- Per default, un parser generato da CUP, quando incontra un errore:
  - segnala tramite la funzione `public void report_fatal_error(String message, Object info)` un "syntax error" e termina l'esecuzione
  - N.B. `report_fatal_error(String, Object)` nella sua implementazione di default chiama la funzione `report_error(String, Object)` la quale chiama `syntax_error(Symbol)` e termina l'esecuzione del parser lanciando un'eccezione

http://www.skenz.it/traduttori

Esercitazione 3

12

## Simbolo predefinito 'error'

- Il simbolo predefinito `'error'` indica una condizione di errore. Esso può essere usato all'interno della grammatica per consentire al parser di riprendere l'esecuzione dopo un eventuale errore.
- Un errore gestito dal simbolo predefinito `'error'` invocherà la funzione `public void report_error(String, Object)` invece di `report_fatal_error(String, Object)`, la quale riporterà l'errore e continuerà il parsing

```
expr ::= E UGUALE
      | error UGUALE
;
```

<http://www.skenz.it/traduttori>

Esercitazione 3

13

## Come viene gestito il simbolo 'error' da Cup?

- Quando il parser generato da Cup incontra un errore, comincia a svuotare lo stack fino a che incontra uno stato in cui il simbolo `'error'` è lecito.
  - Nell'esempio di prima viene svuotato lo stack da `exp` scorrette (cioè da quella sequenza di simboli che non riesce a ridursi in `exp`), finché non si giunge ad ottenere nello stack `stmts`
- Fa lo *shift* del simbolo `error`.
- Se il simbolo successivo è accettabile il parser procede nell'analisi.
- Altrimenti il parser continua a leggere e scartare simboli finché non ne trova uno accettabile
  - Nell'esempio precedente il parser leggerà e scarterà simboli finché non trova un simbolo di NL

<http://www.skenz.it/traduttori>

oEsercitazione 3

14

## Alcune regole generali

- Una semplice strategia per la gestione degli errori è quella di saltare la *statement* corrente:

```
stmt ::= error ';' ;
```

- A volte può essere utile recuperare un delimitatore di chiusura corrispondente ad uno di apertura:

```
expr ::= '(' expr ')'
      | '(' error ')'
```

- NB: Per limitare la proliferazione di errori spuri, dopo il riconoscimento di un errore, la segnalazione è sospesa finché non vengono *shiftati* almeno tre simboli consecutivi

<http://www.skenz.it/traduttori>

Esercitazione 3

15

## Grammatica

```
file ::= funcs
      ;

funcs ::= /* empty */
      | funcs func
      ;

func ::= ID '(' ')'
      | compound
      ;

stmts ::= /* empty */
      | stmts stmt
      ;

stmt ::= exp ';'
      | compound
      ;

compound ::= '{' stmts '}'
          ;

exp ::= NUM
     | exp '+' exp
     | exp '-' exp
     | exp '*' exp
     | exp '/' exp
     | '-' exp %prec NEG
     | '(' exp ')'
```

<http://www.skenz.it/traduttori>

oEsercitazione 3

16

## Statement ed espressioni

```
stmt ::= exp ';'
      | compound
      | error ';' { parser.report_error("syntax error in statement\n",null); };

compound ::= '{' stmts '}'
          | '{' stmts error '}' { parser.report_error("missing ; before }\n",null); };

exp ::= ...
     | '(' error ')' { parser.report_error(" syntax error in expression\n",null); };
```

<http://www.skenz.it/traduttori>

Esercitazione 3

17

## Debugging del parser

- Cup mette a disposizione una serie di opzioni per visualizzare le strutture interne che costituiscono il parser:
  - `-dump_grammar` : Stampa l'elenco dei simboli terminali, non terminali e le produzioni
  - `-dump_states` : Riporta il grafo degli stati
  - `-dump_table` : Riporta la ACTION TABLE e la REDUCE TABLE
  - `-dump` : Stampa tutte le informazioni
- Può essere eseguito in modalità debug, stampando in uscita tutte le azioni effettuate per analizzare una sequenza di simboli in ingresso

### Modalità normale:

```
Yylex l = new Yylex(new FileReader(file));
parser p = new parser(l);
Object result = p.parse();
```

### Modalità debug:

```
Yylex l = new Yylex(new FileReader(file));
parser p = new parser(l);
Object result = p.debug_parse();
```

<http://www.skenz.it/traduttori>

oEsercitazione 3

18

### Grammatica

- dump\_grammar**

```

==== Terminals =====
[0]EOF [1]error [2]NUMERO [3]PIU
==== Non terminals =====
[0]$START [1]exp [2]T
==== Productions =====
[0] $START ::= exp EOF
[1] exp ::= exp PIU T
[2] exp ::= T
[3] T ::= NUMERO
    
```

```

exp → exp PIU T
exp → T
T → NUMERO
    
```

<http://www.skenz.it/traduttori> Esercitazione 3 19

### Stati

- dump\_states**

```

==== Viable Prefix Recognizer ====
START lair_state [0]: {
[exp ::= (*) T , (EOF PIU )]
[exp ::= (*) exp PIU T , (EOF PIU )]
[T ::= (*) NUMERO , (EOF PIU )]
[$START ::= (*) exp EOF , (EOF )]
}
transition on exp to state [3]
transition on T to state [2]
transition on NUMERO to state [1]
-----
lair_state [1]: {
[T ::= NUMERO (*) , (EOF PIU )]
}
-----
lair_state [2]: {
[exp ::= T (*) , (EOF PIU )]
}
-----
lair_state [3]: {
[exp ::= exp (*) PIU T , (EOF PIU )]
[$START ::= exp (*) EOF , (EOF )]
}
transition on EOF to state [5]
transition on PIU to state [4]
-----
lair_state [4]: {
[exp ::= exp PIU (*) T , (EOF PIU )]
[T ::= (*) NUMERO , (EOF PIU )]
}
transition on T to state [6]
transition on NUMERO to state [1]
-----
lair_state [5]: {
[$START ::= exp EOF (*) , (EOF )]
}
-----
lair_state [6]: {
[exp ::= exp PIU T (*) , (EOF PIU )]
}
    
```

<http://www.skenz.it/traduttori> Esercitazione 3 20

### Action / Reduce Tables

- dump\_tables**

<pre> ----- ACTION_TABLE ----- From state #0 [term 2:SHIFT(to state 1)] From state #1 [term 0:REDUCE(with prod 3)] [term 3:REDUCE(with prod 3)] From state #2 [term 0:REDUCE(with prod 2)] [term 3:REDUCE(with prod 2)] From state #3 [term 0:SHIFT(to state 5)] [term 3:SHIFT(to state 4)] From state #4 [term 2:SHIFT(to state 1)] From state #5 [term 0:REDUCE(with prod 0)] From state #6 [term 0:REDUCE(with prod 1)] [term 3:REDUCE(with prod 1)]     </pre>	<pre> ----- REDUCE_TABLE ----- From state #0 [non term 1-&gt;state 3] [non term 2-&gt;state 2] From state #1 From state #2 From state #3 From state #4 [non term 2-&gt;state 6] From state #5 From state #6     </pre>
--	--

<http://www.skenz.it/traduttori> Esercitazione 3 21

### Debugging

- debug\_parse()**

```

# Initializing parser
RICONOSCIUTO: 3
# Current Symbol is #2
# Shift under term #2 to state #1
RICONOSCIUTO: +
# Current token is #3
# Reduce with prod #3 [NT=2, SZ=1]
# Reduce rule: top state 0, lhs sym 2 -> state 2
# Goto state #2
# Reduce with prod #2 [NT=1, SZ=1]
# Reduce rule: top state 0, lhs sym 1 -> state 3
# Goto state #3
# Shift under term #0 to state #5
# Current token is #0
# Reduce with prod #0 [NT=0, SZ=2]
# Reduce rule: top state 0, lhs sym 0 -> state -1
# Goto state #1
    
```

**Stringa di ingresso: 3+5**

<http://www.skenz.it/traduttori> Esercitazione 3 22

```

# Initializing parser
# Current Symbol is #2
# Shift under term #2 to state #1
# Current token is #3
    
```

```

----- ACTION_TABLE -----
From state #0
[term 2:SHIFT(to state 1)]
    
```

<http://www.skenz.it/traduttori> Esercitazione 3 23

```

# Reduce with prod #3 [NT=2, SZ=1]
# Reduce rule: top state 0, lhs sym 2 -> state 2
# Goto state #2
    
```

```

----- ACTION_TABLE -----
From state #1
[term 0:REDUCE(with prod 3)]
[term 3:REDUCE(with prod 3)]
----- REDUCE_TABLE -----
From state #0
[non term 1->state 3] [non term 2->state 2]
    
```

<http://www.skenz.it/traduttori> Esercitazione 3 24

