

POLITECNICO DI TORINO

Laboratorio di Compilatori
Corso di Linguaggi e Traduttori

Esercitazione 4



Stefano Scanzio
mail: stefano.scanzio@polito.it
sito: <http://www.skenz.it/traduttori>

a.a 2010 / 2011

Definizioni Guidate dalla Sintassi

- Ad ogni simbolo viene associato un insieme di attributi:
 - **sintetizzati**: calcolati in base al valore degli attributi dei figli di un nodo nell'albero di derivazione,
 - **ereditati**: calcolati in base al valore degli attributi dei nodi fratelli e del nodo padre nell'albero di derivazione,
- Ad ogni produzione viene associato un insieme di regole semantiche che specificano come vengono calcolati gli attributi.
- Lo scanner passa i valori semantici dei simboli terminali al parser il quale, mentre riconosce la grammatica, aggiorna i nodi dell'albero di derivazione

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

Attributi sintetizzati

- Se una definizione guidata dalla sintassi usa solo attributi sintetizzati è detta definizione ad *attributi S*.
- È possibile calcolare i valori degli attributi di una definizione ad attributi S bottom-up, dalle foglie alla radice dell'albero di derivazione.

$E \rightarrow E_1 '+' T$	$E.value ::= E_1.value + T.value$
$E \rightarrow T$	$E.value ::= T.value$
$T \rightarrow number$	$T.value ::= number.value$

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

Semantica in Cup: La classe Symbol

- Ogni simbolo presente nello stack di Cup è un oggetto di tipo Symbol (`cup/java_cup/runtime/Symbol.java`)
- Ad esso sono associate diverse informazioni:
 - Un numero che identifica univocamente il simbolo
↳ `public int sym;`
 - Lo stato in cui si trova il parser
↳ `public int parse_state;`
 - Due numeri interi che servono per il passaggio di numero di linea e di colonna dallo scanner verso il parser
↳ `public int left, right;`
 - Una classe di tipo Object che serve a gestire la semantica
↳ `public Object value;`

<http://www.skenz.it/traduttori> Esercitazione 4 5

scanner.jflex

Passaggio dei valori semantici al parser

- Simbolo e valore semantico:
`[a-zA-Z][a-zA-Z0-9_]* { return new Symbol(sym.ID, new String(yytext())); }`
- Simbolo, numero di linea, numero di colonna e valore semantico:

```
%{
private Symbol symbol(int type, Object value){
return new Symbol(type, yyline, yycolumn, value);
}
%%
[a-zA-Z][a-zA-Z0-9_]* { return symbol(sym.ID, new String(yytext())); }
```
- o in modo equivalente:
`[a-zA-Z][a-zA-Z0-9_]* { return new Symbol(sym.ID, yyline, yycolumn, new String(yytext())); }`

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 4 6

Semantica in Cup: definizione simboli (non)? terminali

- Cup deve conoscere il tipo di informazione semantica associata ad ogni simbolo
- Si usa la seguente implementazione dei costrutti terminal e non terminal di Cup:
 - `terminal <Object> <elenco_simboli_terminali> ;`
 - `non terminal <Object> <elenco_simboli_non_terminali> ;`
- `<Object>` rappresenta il tipo di oggetto associato ad un determinato simbolo
- Es.
 - `terminal String ID;`
↳ In ID verrà memorizzato un contenuto semantico di tipo stringa
 - `terminal Integer NUM;`
 - `non terminal MyObject var;`

Per esempio:
`class MyObject {`
`public String var_name;`
`public String var_type;`
`}`

<http://www.skenz.it/traduttori> Esercitazione 4 7

Semantica in Cup: Utilizzo valori semantici, le etichette

- Dati un insieme di regole:

```
E ::= E PIU T
      | E MENO T ;
```

- Si può far riferimento ai valori semantici dei simboli aggiungendo delle etichette ai simboli che interessano:

- Un etichetta è il carattere ':' seguito da un nome

```
E ::= E:n1 PIU T:n2
      | E:n1 MENO T:n2 ;
```

- All'interno delle regole le etichette potranno essere usate normalmente come oggetti del tipo specificato nei costrutti terminal e non terminal :

```
E ::= E:n1 PIU T:n2 {
      System.out.print(n1 + " + " + n2 + " =");
    }
    | E:n1 MENO T:n2 {
      System.out.print(n1 + " - " + n2 + " =");
    } ;
```

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

Esercitazione 4

8

Semantica in Cup: Azioni e oggetto RESULT

- Ad ogni regola può essere associata un'azione ({ : /* Codice Java */ ; }) da eseguirsi ogni qualvolta la regola è applicata nel processo di analisi
- Tale azione provvede ad aggiornare i valori semantici associati a ciascun simbolo
- Cup definisce per ogni regola un oggetto di tipo Object chiamato RESULT
- RESULT rappresenta il risultato delle regole semantiche contenute nell'azione ed è perciò associato al simbolo non terminale che compare nella parte sinistra di una regola

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

Esercitazione 4

9

Esempio: Calcolo di attributi sintetizzati

- Data la grammatica delle espressioni algebriche, la regola seguente associa al simbolo 'E' la somma o la sottrazione dei valori degli addendi/sottraendi:

```
non terminal Integer E;
E ::= E:n1 PIU T:n2 {
      RESULT = new Integer(n1.intValue() + n2.intValue());
    }
    | E:n1 MENO T:n2 {
      RESULT = new Integer(n1.intValue() - n2.intValue());
    } ;
```

- Nota:

- A RESULT deve essere assegnato un oggetto di tipo Integer: new Integer()
- Gli operatori matematici devono operare su numeri e non su oggetti: n1.intValue()

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

Esercitazione 4

10

Esempio: Calcolo di attributi sintetizzati

- Se attraverso la classe RESULT si vogliono propagare più valori semantici si può procedere nel seguente modo:

```
terminal TO, TC;
terminal String identifier;
terminal Integer Args;
non terminal Object[ ] Func;
non terminal goal;
```

```
goal ::= Func:a {
      System.out.println( "Nome Funzione: " + a[0] + "Numero argomenti: " + a[1] );
    } ;
```

```
Func ::= identifier:a TO Args:b TC {
      RESULT = new Object[2];
      RESULT[0] = new String(a);
      RESULT[1] = new Integer(b);
    } ;
```

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

Esercitazione 4

11

Esempio: Calcolo di attributi sintetizzati

- Alternativamente si può costruire un oggetto che contenga tutte le informazioni necessarie:

```
action code {
  class MyFunc {
    MyFunc(String id, int args) {
      this.id = new String(id);
      this.args = args;
    }
    public String id;
    public int args;
  }
}

non terminal MyFunc Func;

goal ::= Func:a {
      System.out.println( "Nome Funzione: " + a.id + "Numero argomenti: " + a.args );
    } ;

Func ::= identifier:a TO Args:b TC {
      RESULT = new MyFunc( a, b.intValue() );
    } ;
```

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

Esercitazione 4

12

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();
Object result = p.parse();
```

Modalità debug:

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

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

oEsercitazione 4

13

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 4 14

Stati

- dump_states**

```

==== Viable Prefix Recognizer ====
START lalr_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]

lalr_state [1]: {
  [T ::= NUMERO (*) , (EOF PIU )]
}

lalr_state [2]: {
  [exp ::= T (*) , (EOF PIU )]
}

lalr_state [3]: {
  [exp ::= exp (*) PIU T , (EOF PIU )]
  [$START ::= exp (*) EOF , (EOF )]
}
transition on EOF to state [5]
transition on PIU to state [4]

lalr_state [4]: {
  [exp ::= exp PIU (*) T , (EOF PIU )]
  [T ::= (*) NUMERO , (EOF PIU )]
}
transition on T to state [6]
transition on NUMERO to state [1]

lalr_state [5]: {
  [$START ::= exp EOF (*) , (EOF )]
}

lalr_state [6]: {
  [exp ::= exp PIU T (*) , (EOF PIU )]
}
    
```

<http://www.skenz.it/traduttori> Esercitazione 4 15

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->state 3] [non term 2->state 2] From state #1 From state #2 From state #3 From state #4 [non term 2->state 6] From state #5 From state #6 </pre>
--	--

<http://www.skenz.it/traduttori> Esercitazione 4 16

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 4 17

```

# 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 4 18

```

# 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 4 19

