


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 2007 / 2008

Attributi ereditati

- Sono utili per esprimere la dipendenza di costrutti di un linguaggio dal contesto in cui si trovano.
- $a, b : \text{int};$

$D \rightarrow L \text{' ' T '}$ $L \rightarrow L_1 \text{' ' id}$ $L \rightarrow \text{id}$ $T \rightarrow \text{'integer'}$	$L.type = T.type$ $L_1.type = L.type$ $\text{new_var}(id.name, L.type)$ $\text{new_var}(id.name, L.type)$ $T.type = \text{type_int}$
--	---

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

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 2

Attributi L

- L'ordine di valutazione degli attributi dipende dall'ordine con cui vengono creati o visitati i nodi dell'albero di derivazione.
- Comunemente i parser seguono l'ordine di una visita in profondità dell'albero.
- Una grammatica è detta ad *attributi L* se è possibile il calcolo dei valori degli attributi con una visita in profondità dell'albero di derivazione.
- In tali grammatiche si ha una propagazione delle informazioni da sinistra a destra (dell'albero di derivazione).
- La grammatica precedente non è una grammatica ad *attributi L*
 - Il passaggio delle informazioni avviene da destra a sinistra

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

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.

$E \rightarrow E_1 \text{'+' T}$ $E \rightarrow T$ $T \rightarrow \text{number}$	$E.value ::= E_1.value + T.value$ $E.value ::= T.value$ $T.value ::= \text{number.value}$
--	---

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

Attributi ereditati di tipo L

- $\text{int } a, b;$

$D \rightarrow T L \text{' '}$ $L \rightarrow L_1 \text{' ' id}$ $L \rightarrow \text{id}$ $T \rightarrow \text{'integer'}$	$L.type = T.type$ $L_1.type = L.type$ $\text{new_var}(id.name, L.type)$ $\text{new_var}(id.name, L.type)$ $T.type = \text{type_int}$
--	---

http://www.skenz.it/traduttori Esercitazione 4 6

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

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

10



## 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())); }
```

**Costitutori 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

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

11

## 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

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

12

### 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

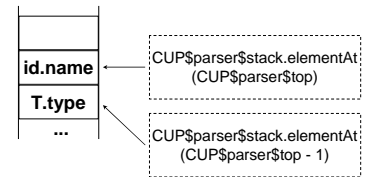
13

### Calcolo di attributi ereditati

- Una produzione che usa una copy-rule:

```
Decl → T Decl1 | Decl1.type = T.type
Decl1 → id | var (id.name, Decl1.type)
```

Situazione dello stack prima di ridurre Decl1



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

Esercitazione 4

16

### 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);
};
```

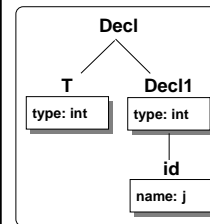
```
Func ::= identifier:a TO Args:b TC {
 RESULT = new MyFunc(a, b.int(Value));
};
```

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

Esercitazione 4

14

### Calcolo di attributi ereditati



- Il simbolo 'Decl1' ha l'attributo ereditato 'type'.
- Il valore di tale attributo è presente sullo stack semantico (nella posizione associata a 'T') prima che venga creato il simbolo 'Decl1'.
- Tuttavia esso è al di fuori del contesto semantico della regola relativa a 'Decl1'.

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

Esercitazione 4

17

### Calcolo di attributi ereditati

- In un riconoscitore bottom-up, non viene riservato spazio sullo stack semantico fino a che il corrispondente simbolo sintattico non è riconosciuto.
- Ciò rende problematica la gestione degli attributi ereditati.
- Se la grammatica è ad attributi L, e le regole semantiche per il calcolo degli attributi ereditati sono *copy-rules* è possibile "aggirare l'ostacolo" in maniera semplice.
- Se le regole sono più complesse è possibile inserire dei *marker*, cioè dei non-terminali che si espandono in  $\epsilon$ .

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

Esercitazione 4

15

### Calcolo di attributi ereditati

- È possibile accedere ai valori semantici precedentemente memorizzati nello stack purché se ne conosca la posizione relativa alla regola corrente.
  - `CUP$parser$stack.elementAt(CUP$parser$top)` rappresenta il simbolo in cima allo stack
  - `CUP$parser$stack.elementAt(CUP$parser$top-1)` indica il simbolo che precede quello collocato in cima allo stack
  - `CUP$parser$stack.elementAt(CUP$parser$top-1).value` indica il valore semantico associato al simbolo in posizione top-1
- Assumendo che il simbolo 'Decl1' sia sempre preceduto da un identificatore di tipo:
 

```
Decl1 ::= id:name {
 string type = (String)((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 RESULT = new String(type);
 add_id(name, RESULT);
};
```

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

Esercitazione 4

18

### Calcolo di attributi ereditati mediante marker

- Aggiungendo la regola `Decl1 ::= id VIR Decl1`; non è più vero che 'Decl1' è sempre preceduto da un identificatore di tipo.
- Nel caso della regola: `Decl1 ::= id`; Il simbolo che precede 'id' nello stack prima dell'operazione di reduce è 'VIR'
- Si aggiunge una regola vuota che fa in modo che entrambe le regole siano precedute da un valore semantico di tipo

http://www.skenz.it/traduttori
Esercitazione 4
19

scanner.jflex

### Esempio marker

```

import java_cup.runtime.*;
%%
%cup
%unicode

nl = '\n | \r | \n'
id = '[a-zA-Z][a-zA-Z0-9]*'
tipo = int | float | char | double

%%

"." { return new Symbol(sym.VIR);}
"." { return new Symbol(sym.PV);}

(tipo) { return new Symbol(sym.TYPE, new String(yytext())); }

(id) { return new Symbol(sym.ID, new String(yytext())); }

(nl) | "." | t (:)

```

http://www.skenz.it/traduttori
Esercitazione 4
22

### Esempio: Calcolo degli attributi ereditati mediante marker

```

Decl1 ::= id:name { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 add_id(name, RESULT);
};

Decl1 ::= id:name VIR Empty Decl1 { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 add_id(name, RESULT);
};

Empty ::= { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-2)).value;
};

```

http://www.skenz.it/traduttori
Esercitazione 4
20

parser.cup

### Esempio marker

```

import java_cup.runtime.*;

terminal VIR, PV;
terminal String TYPE, ID;
non terminal goal, lista_dich;
non terminal String dich, lista_id;

start with goal;

goal ::= lista_dich { : System.out.println("PARSER: GRAMMATICA RICONOSCIUTA
CORRETTAMENTE!!!"); : }

lista_dich ::= | lista_dich dich
;

dich ::= TYPE lista_id x PV { :
 System.out.println("PARSER: Riconosciuta dichiarazione di tipo: " + x);
};

```

http://www.skenz.it/traduttori
Esercitazione 4
23

### Azioni intermedie

- Si può evitare di introdurre esplicitamente un non terminale riscritto come stringa nulla, utilizzando, nella parte destra delle regole, le **azioni intermedie**.
- Esse vengono automaticamente sostituite da CUP con un simbolo non terminale, a sua volta riscritto come `c`.
- Il codice relativo alle regole:
 

```

Decl1 ::= id:name VIR Empty Decl1 ;
Empty ::= ;

```
- Potrà essere così riscritto:
 

```

Decl1 ::= id:name VIR { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-2)).value;
 ;
}
Decl1 { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 add_id(name, RESULT);
};

```

http://www.skenz.it/traduttori
Esercitazione 4
21

parser.cup

### Esempio marker

```

lista_id ::= ID:name VIR { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-2)).value;
 ;
}
lista_id { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 System.out.println("PARSER: var(" + name + ", " + RESULT + ")");
};

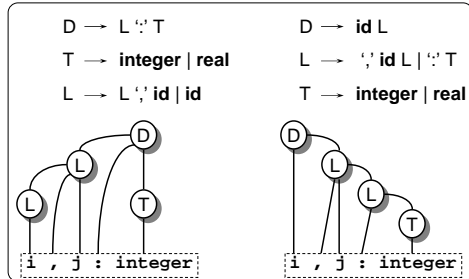
lista_id ::= ID:name { :
 RESULT = (String) ((java_cup.runtime.Symbol)
 CUP$parser$stack.elementAt(CUP$parser$top-1)).value;
 System.out.println("PARSER: var(" + name + ", " + RESULT + ")");
};

```

http://www.skenz.it/traduttori
Esercitazione 4
24

### Trasformazione della grammatica

- È possibile evitare l'uso degli attributi ereditati trasformando la grammatica.



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

Esercitazione 4

25

### Gestione degli errori semantici

- La verifica degli errori semantici viene di solito effettuata all'interno delle azioni associate alle regole
- Di solito vengono verificati:
  - La giusta corrispondenza tra gli operandi
  - Se le variabili e le funzioni sono state dichiarate
  - La coerenza tra i prototipi delle funzioni e i parametri passati quando vengono utilizzate

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

Esercitazione 4

26