

POLITECNICO DI TORINO

Laboratorio di Compilatori

Corso di Linguaggi e Traduttori

Esercitazione 2



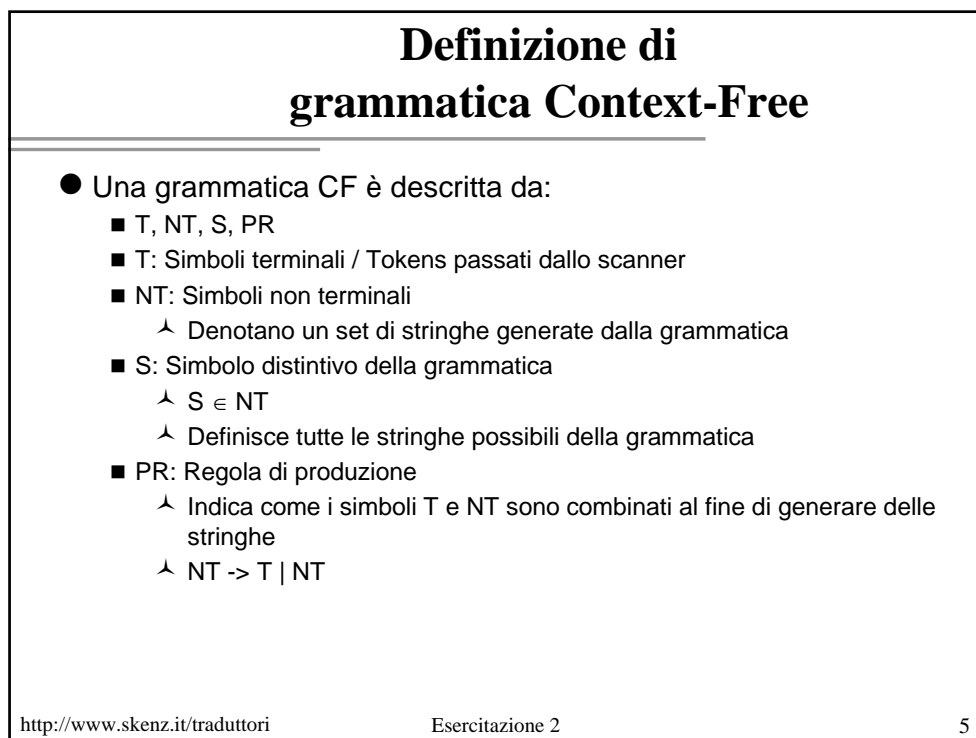
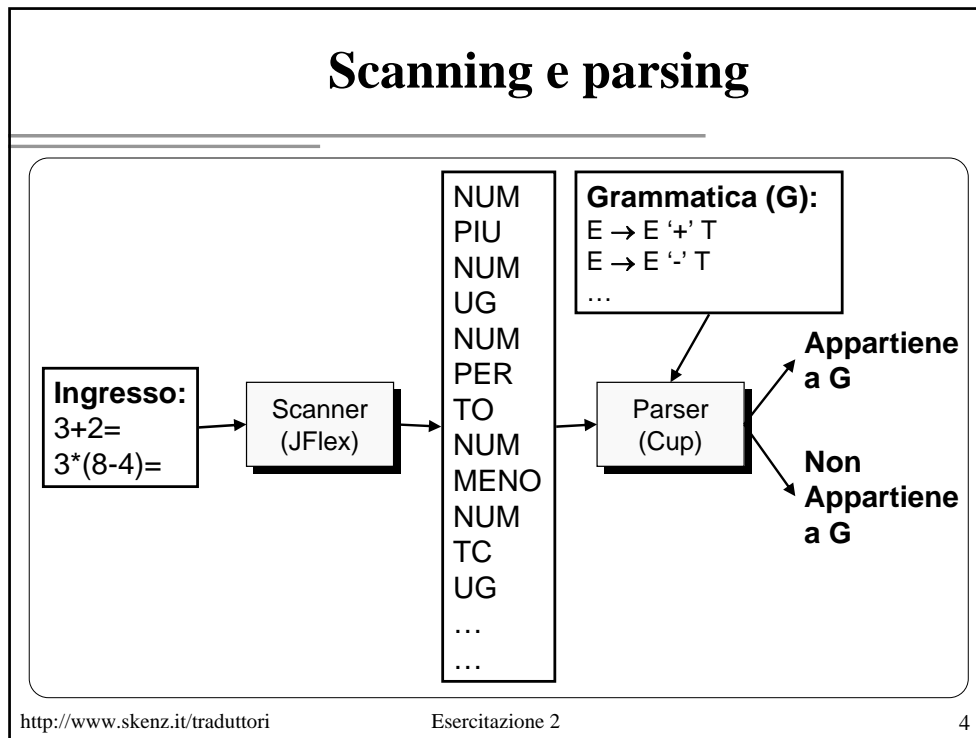
Stefano Scanzio

mail: stefano.scanzio@polito.itsito: <http://www.skenz.it/traduttori>

a.a 2007 / 2008

Riconoscitori e analizzatori sintattici

- Data una grammatica non ambigua ed una sequenza di simboli in ingresso, un riconoscitore è un programma che verifica se la sequenza appartiene o no al linguaggio definito dalla grammatica.
- Un analizzatore sintattico (*parser*) è un programma che è in grado di associare alla sequenza di simboli in ingresso il relativo albero di derivazione.
- Gli algoritmi di analisi possono essere
 - top-down (dalla radice alle foglie)
 - bottom-up (dalle foglie alla radice) : CUP



Esempio

Produzioni (regole grammaticali):

```
assign_stmt -> id := expr ;  
expr -> expr operator term  
expr -> term  
term -> id  
term -> real  
term -> integer  
operator -> +  
operator -> -
```

● Derivazione:

- Una sequenza di regole grammaticali che, applicate, trasformano un simbolo non terminale (NT) in una sequenza di simboli terminali (T) che prendono il nome di tokens

Funzionamento di un parser: la tecnica Shift/Reduce

- Si usa uno stack, inizialmente vuoto, per memorizzare i simboli già riconosciuti.
- I simboli terminali vengono immessi sullo stack (azione di shift), fino a che la cima dello stack non contiene un handle (parte destra della regola): quando ciò avviene, l'handle viene ridotto (azione di reduce) con il non-terminale relativo.
- Si ha successo se esaminati tutti i simboli in ingresso, lo stack contiene solo il simbolo distintivo della grammatica.

Alberi di derivazione e tecnica Shift/Reduce

Stringa di ingresso:

a1 , a2 , a3

Scanner:

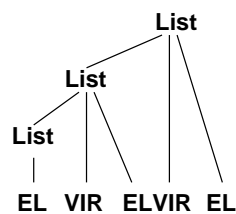
a1 , a2 , a3 → EL VIR EL VIR EL

Grammatica Ricorsiva Sinistra

List → List VIR EL

List → EL

Albero di derivazione



Azione:

Shift:

Reduce:

Shift:

Shift:

Reduce:

Shift:

Shift:

Reduce:

Stack:

ε

EL

List

List VIR

List VIR EL

List

List VIR

List VIR EL

List

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

Esercitazione 2

8

Grammatica Ambigua

- Una grammatica si dice ambigua se esiste almeno una sequenza di simboli del linguaggio per cui esistono due o più alberi di derivazione distinti.
- Esercizio: trovare gli alberi di derivazione per
if (i=1) then if (j=2) then a:=0 else a:=1

data la grammatica:

- S → M
- M → 'if' C 'then' M
- M → 'if' C 'then' M 'else' M
- M → VAR ':=' NUM | VAR ':=' VAR
- M → '(' VAR ':=' NUM ')'

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

Esercitazione 2

9

Esempio grammatica non ambigua: Costrutto if-then-else

- La grammatica per i costrutti del tipo if-then-else può essere resa non ambigua come segue:
 - $S \rightarrow M \mid U$
 - $U \rightarrow \text{'if' } C \text{'then' } S$
 - $U \rightarrow \text{'if' } C \text{'then' } M \text{'else' } U$
 - $M \rightarrow \text{'if' } C \text{'then' } M \text{'else' } M$
 - $M \rightarrow \text{VAR ':='} \text{ NUM} \mid \text{VAR ':='} \text{ VAR}$
 - $C \rightarrow \text{'(' VAR ':='} \text{ NUM ')'}$
- `if (i=1) then if (j=2) then a:=0 else a:=1`

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

Esercitazione 2

10

Esempio grammatica non ambigua: Espressioni algebriche

- La grammatica che descrive le espressioni algebriche è context-free ed usa la ricorsione:
 - $S \rightarrow E$
 - $E \rightarrow E \text{'+' } T$
 - $E \rightarrow E \text{'-' } T$
 - $E \rightarrow T$
 - $T \rightarrow T \text{'*' } F$
 - $T \rightarrow T \text{'/' } F$
 - $T \rightarrow F$
 - $F \rightarrow \text{'(' } E \text{')'}$
 - $F \rightarrow \textit{number}$
- I simboli T e F della grammatica algebrica servono a togliere l'ambiguità sulla priorità degli operatori '+' e '-' rispetto agli operatori '*' e '/'.

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

Esercitazione 2

11

Ambiguità e conflitti in Cup

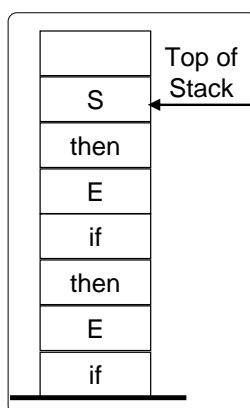
- Se la grammatica è ambigua possono verificarsi dei conflitti.
- Ciò significa che l'analizzatore deve scegliere tra più azioni alternative plausibili.
- Il problema viene, di solito, risolto modificando la grammatica per renderla non ambigua oppure fornendo indicazioni a Cup su come comportarsi in caso di ambiguità.
- La seconda ipotesi richiede una comprensione adeguata dell'algoritmo di analisi, per evitare di generare comportamenti scorretti.

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

Esercitazione 2

12

Ambiguità e conflitti in Cup: Conflitti shift-reduce



- 1) $S \rightarrow \text{if } E \text{ then } S$
- 2) $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- 3) $S \rightarrow \text{var } = E$

Il prossimo simbolo in ingresso è 'else'.

Possono succedere due cose:

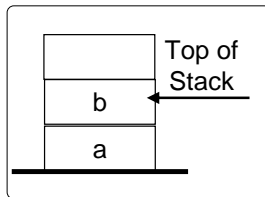
- ridurre le prime quattro voci dello stack, secondo la produzione 1;
 - introdurre 'else' nello stack secondo quanto previsto dalla produzione 2.
- In queste situazioni, Cup decide, in mancanza di altri suggerimenti, di eseguire lo **shift**.

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

Esercitazione 2

13

Ambiguità e conflitti in Cup: Conflitti reduce-reduce



Il prossimo simbolo in ingresso è '\$'.

Possono succedere due cose:

- 1) $S \rightarrow a B$
- 2) $S \rightarrow B$
- 3) $B \rightarrow a b$
- 4) $B \rightarrow b$

- ridurre le prime due voci dello stack, secondo la produzione 3;
 - ridurre la prima voce secondo quanto previsto dalla produzione 4.
- In queste situazioni, Cup decide, in mancanza di altri suggerimenti, di *ridurre la regola che è stata definita per prima* (la n°3).

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

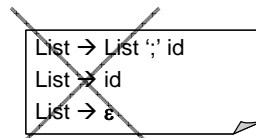
Esercitazione 2

14

Le Liste

- Una lista, eventualmente vuota, di identificatori **separati** da “;”

- $List \rightarrow id_list \mid \epsilon$
- $id_list \rightarrow id_list \text{ ; } id$
- $id_list \rightarrow id$



- Una lista, eventualmente vuota, di identificatori **terminati** da “;”

- $List \rightarrow id_list \text{ ; } \mid \epsilon$
- $id_list \rightarrow id_list id$
- $id_list \rightarrow id$

- Alternativamente:

- $List \rightarrow List id \text{ ; } \mid \epsilon$
- $List \rightarrow \epsilon$

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

Esercitazione 2

15

Introduzione a Cup

- Cup è un generatore di analizzatori sintattici che trasforma la descrizione di una grammatica context-free in un programma Java che riconosce ed analizza la grammatica stessa.
- Oltre alle regole sintattiche, è possibile specificare quali azioni devono essere eseguite in corrispondenza del riconoscimento dei vari simboli della grammatica.
- È necessario integrare il parser così generato con un analizzatore lessicale: alcune convenzioni permettono di eseguire l'integrazione di Cup con lo scanner JFlex in modo semplificato.
- Manuale Ufficiale:
<http://www.skenz.it/traduttori/risorse/manualeCup.html>

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

Esercitazione 2

16

Il formato del file di ingresso

- Il file di ingresso di Cup ha una sintassi molto simile a quella di un programma Java
- Può essere idealmente diviso nelle seguenti sezioni:
 - Setup
 - Terminali e Non Terminali
 - Precedenze (Prossima esercitazione)
 - Regole
- Possono essere introdotti commenti racchiusi dai simboli /* e */ o preceduti dal simbolo //

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

Esercitazione 2

17

Sezione di Setup

- Vengono specificate tutte le direttive utili ad un corretto funzionamento del parser
- Inclusione della libreria di Cup e di altre librerie:
`import java_cup.runtime.*;`
- Codice utente: (Prossima esercitazione)
 - Ridefinizione di funzioni interne a Cup
 - Integrazione con scanner diversi da JFlex

Sezione dei Terminali / Non-Terminali

- Vengono definiti
 - Simbolo distintivo della grammatica
 - Simboli terminali: passati da Jflex
 - Simboli Non-Terminali
- Simbolo distintivo della grammatica:
 - start with <nome_non_terminale>;
 - Rappresenta la radice dell'albero di derivazione
 - E' lecita una sola occorrenza di questa parola chiave

Sezione dei Terminali / Non-Terminali

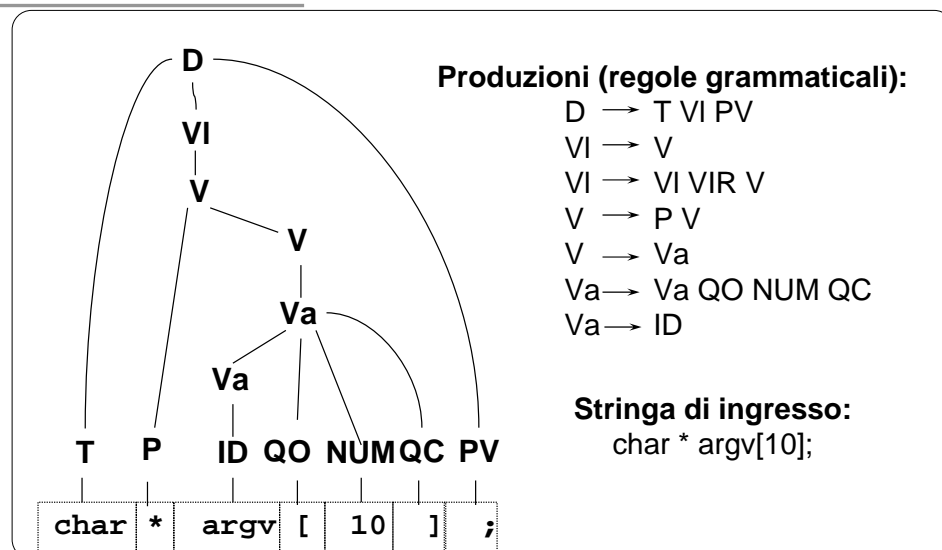
- Simboli Terminali
 - terminal <simbolo_terminale_1>, ..., <simbolo_terminale_n>;
 - <simbolo_terminale> : Un nome formato da lettere, '_', '.' e cifre (non iniziali).
 - I simboli terminali vengono passati da JFlex
- Simboli Non-Terminali
 - non terminal <simbolo_non_terminale_1>, ..., <simbolo_non_terminale_n>;
 - <simbolo_non_terminale> : Un nome formato da lettere, '_', '.' e cifre (non iniziali).

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

Esercitazione 2

20

Sezione dei Terminali / Non-Terminali

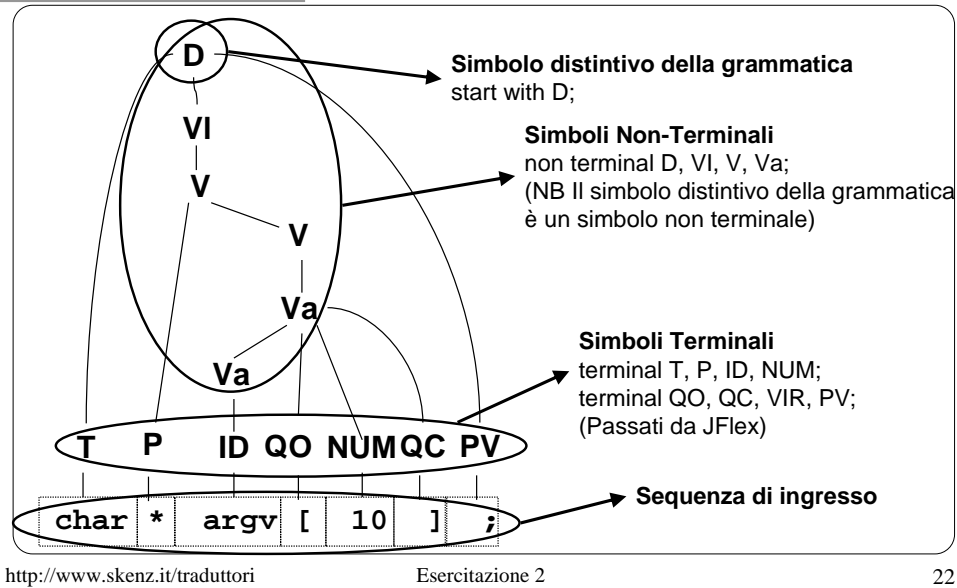


<http://www.skenz.it/traduttori>

Esercitazione 2

21

Sezione dei Terminali / Non-Terminali



Sezione della Regole

- La sezione delle regole è costituita da una o più regole del tipo:
`<simbolo_non_terminale> ::= CorpoDellaRegola ;`
- dove *CorpoDellaRegola* è una sequenza di 0 o più simboli.
- Se per un dato non terminale esistono più produzioni, queste possono essere raggruppate tra loro e separate dal carattere '|'.
- Ad ogni regola grammaticale può essere associata un'azione racchiusa tra i simboli '{: e :}'
 - NB. L'azione viene effettuata subito prima dell'operazione di reduce
- Es:
`D ::= T VI PV`
`{: System.out.println("Riconosciuta una dichiarazione di tipo"); :}`
`;`

Sezione della Regole: Esempio

```
//Sezione dei Terminali / Non-Terminali
terminal T, P, ID, NUM, PV, VIR, QO, QC;
non terminal D, V, VI, Va;
start with D;
```

```
//Sezione delle regole
D ::= T VI PV;
```

```
VI ::= V
      |VI VIR VI;
```

```
V ::= P V
     |Va
```

```
Va ::= Va QO NUM QC
      |Va
```

Produzioni (regole grammaticali):

```
D → T VI PV
VI → V
VI → VI VIR V
V → P V
V → Va
Va → Va QO NUM QC
Va → ID
```

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

Esercitazione 2

24

Integrazione di JFlex e Cup

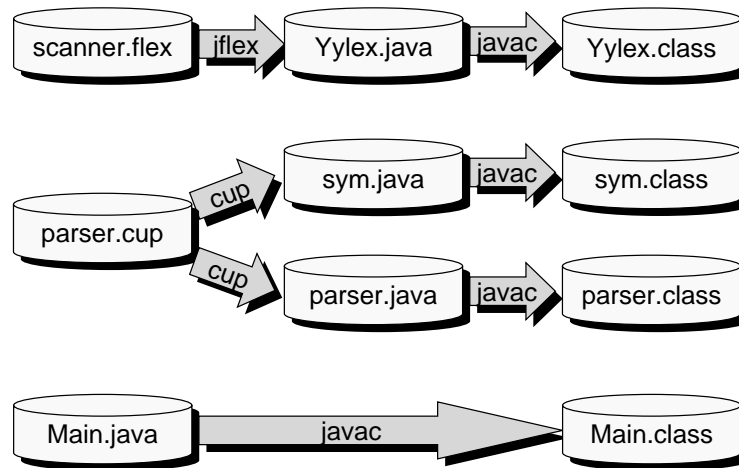
- Parser e scanner devono accordarsi sui valori associati ai token (simboli terminali della grammatica)
- Lo scanner, ogni volta riconosciuto un simbolo terminale lo deve passare al parser
 - Viene eseguito tramite la classe Symbol i cui costruttori fondamentali sono i seguenti:
 - public Symbol(int 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)
- Quando si introduce un terminale per mezzo della parola chiave terminal, Cup associa a tale simbolo un valore intero

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

Esercitazione 2

25

Integrazione di JFlex e Cup



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

Esercitazione 2

26



Modifiche allo scanner

List → List VIRGOLA EL
List → EL

- Includere la libreria di Cup (`java_cup.runtime.*`) nella sezione del codice
- Attivare la compatibilità con Cup tramite la direttiva `%cup` nella sezione delle dichiarazioni

```

import java_cup.runtime.*;
...
%%
%cup
...
%%
[a-z]      {return new Symbol(sym.EL); }
','       {return new Symbol(sym.VIRGOLA); }
  
```

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

Esercitazione 2

27

parser.cup

Il parser Cup

```

import java_cup.runtime.*;

terminal EL, VIRGOLA;
non terminal List;

start with Lista;

Lista ::= List    { : System.out.println("Lista riconosciuta correttamente"); : } |
                { : System.out.println("Lista vuota"); : }
;

List ::= List VIRGOLA EL
;

List ::= EL
;

```

List → List VIRGOLA EL
List → EL

<http://www.skenz.it/traduttori>
Esercitazione 2
28

Main.java

Programma principale: Main

```

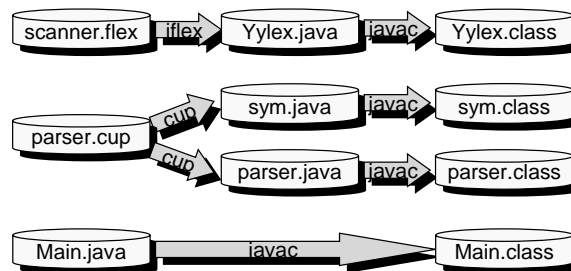
import java.io.*;

public class Main {
    static public void main(String argv[]) {
        try {
            /* Istanzio lo scanner aprendo il file di ingresso argv[0] */
            Yylex l = new Yylex(new FileReader(argv[0]));
            /* Istanzio il parser */
            parser p = new parser(l);
            /* Avvio il parser */
            Object result = p.parse();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

<http://www.skenz.it/traduttori>
Esercitazione 2
29

Compilazione



- `jflex scanner.jflex`
- `java java_cup.Main parser.cup`
 - Nel caso di conflitti shift/reduce o reduce/reduce:
 - `java java_cup.Main -expect <numero> parser.cup`
- `javac Yylex.java sym.java parser.java Main.java`
- `java Main <file>`