

POLITECNICO DI TORINO

## Laboratorio di Compilatori

Corso di Linguaggi e Traduttori

Esercitazione 2

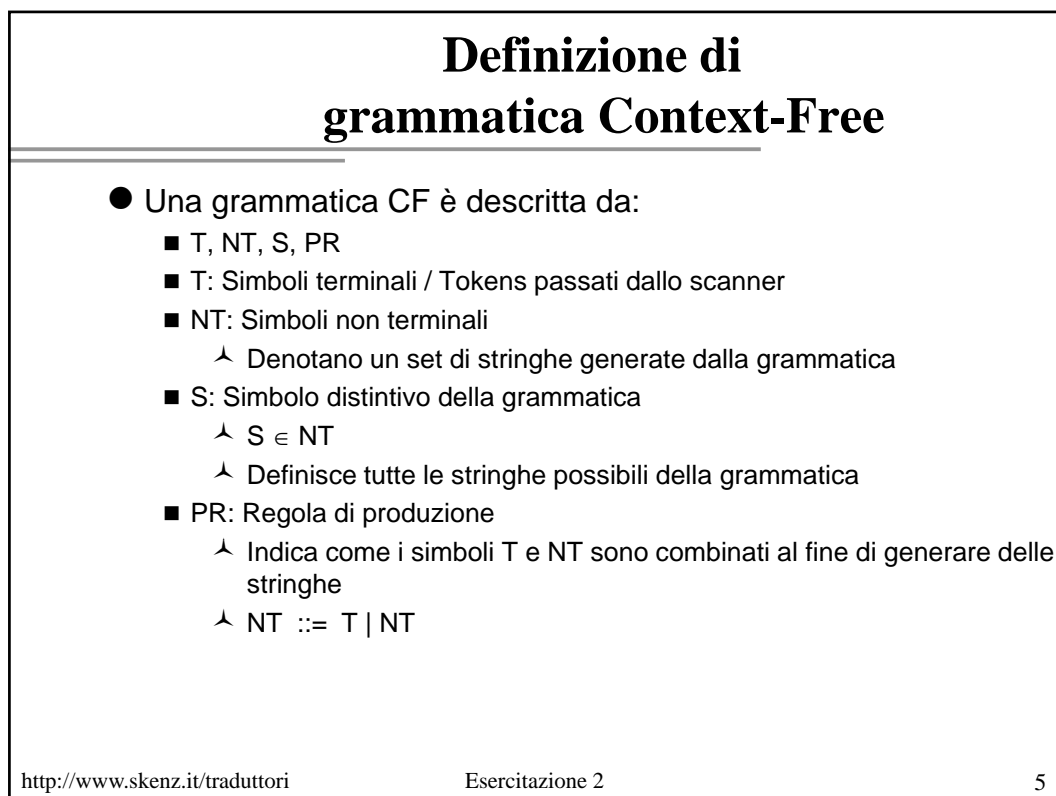
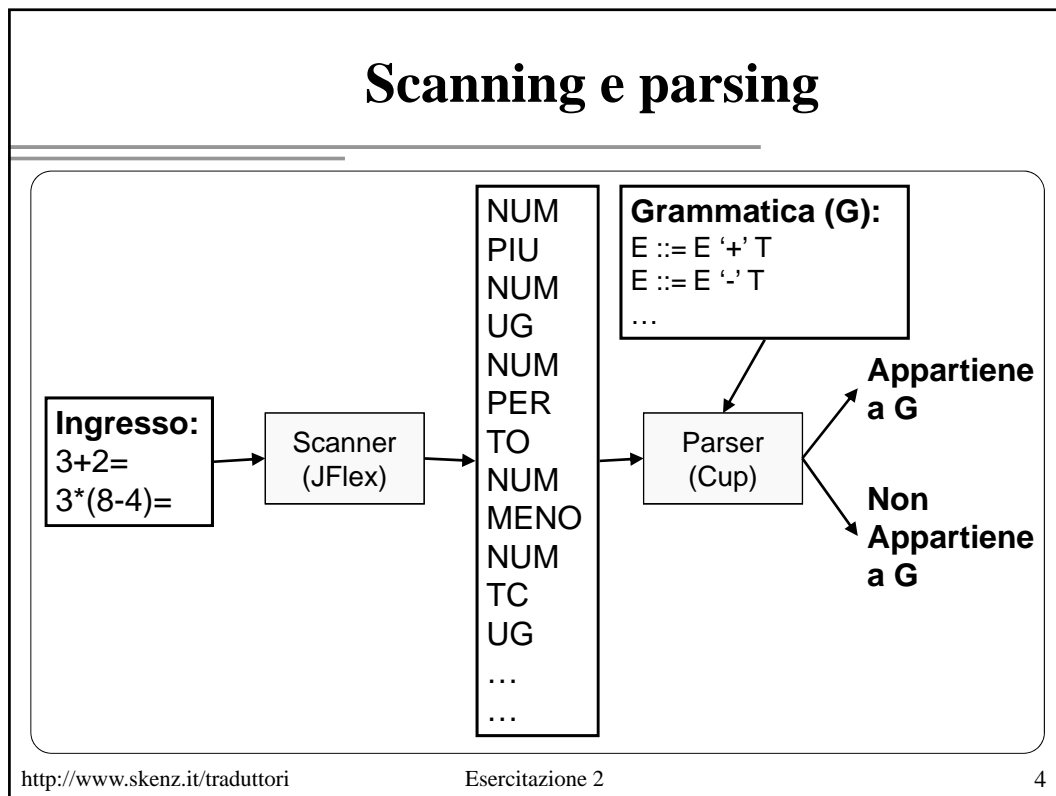


Stefano Scanzio  
mail: [stefano.scanzio@polito.it](mailto:stefano.scanzio@polito.it)  
sito: <http://www.skenz.it/traduttori>

a.a 2010 / 2011

## 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 UG expr PV;  
expr ::= expr operatore term;  
expr ::= term;  
term ::= ID;  
term ::= REAL;  
term ::= INT;  
operator ::= PIU;  
operator ::= MENO;
```

- 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

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

Esercitazione 2

6

## Funzionamento di un parser: la tecnica Shift/Reduce

- Si usa uno stack, inizialmente vuoto, per memorizzare i simboli richiesti dal parser allo scanner.
- 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) e sostituito con il simbolo non-terminale relativo.
- Si ha successo se esaminati tutti i simboli in ingresso, lo stack contiene solo il simbolo distintivo della grammatica.

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

Esercitazione 2

7

## Alberi di derivazione e tecnica Shift/Reduce

**Stringa di ingresso:**

a1 , a2 , a3

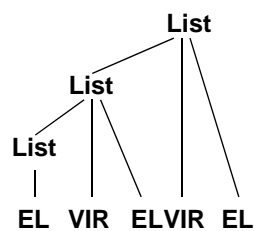
**Scanner:**

a1 , a2 , a3 ::= EL VIR EL VIR EL

**Grammatica Ricorsiva Sinistra**

List ::= List VIR EL  
List ::= EL

**Albero di derivazione**



**Azione:**

**Stack:**

	ε
Shift:	EL
Reduce:	List
Shift:	List VIR
Shift:	List VIR EL
Reduce:	List
Shift:	List VIR
Shift:	List VIR EL
Reduce:	List

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

Esercitazione 2

8

## 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

9

## 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

10

## 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

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

Esercitazione 2

11

## 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

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

Esercitazione 2

12

## 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 o da un altro scanner quando richiesti dal parser
  
- 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

13

## Sezione dei Terminali / Non-Terminali

**Produzioni (regole grammaticali):**

- D ::= T VI PV
- VL ::= V
- VL ::= VL VIR V
- V ::= P V
- V ::= Va
- Va ::= Va QO NUM QC
- Va ::= ID

**Stringa di ingresso:**  
char \* argv [ 10 ] ;

<http://www.skenz.it/traduttori>      Esercitazione 2      14

## Sezione dei Terminali / Non-Terminali

**Simbolo distintivo della grammatica**  
start with D;

**Simboli Non-Terminali**  
non terminal D, VL, V, Va;  
(NB Il simbolo distintivo della grammatica è un simbolo non terminale)

**Simboli Terminali**  
terminal T, P, ID, NUM;  
terminal QO, QC, VIR, PV;  
(Passati da JFlex)

**Sequenza di ingresso**

<http://www.skenz.it/traduttori>      Esercitazione 2      15

## 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.
- 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 VL PV`  
`{ System.out.println("Riconosciuta una dichiarazione di tipo"); ;}`  
`;`

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

Esercitazione 2

16

## Sezione della Regole (2)

- Se per un dato non terminale esistono più produzioni, queste possono essere raggruppate tra loro e separate dal carattere '|'
- Es:  
`funz ::= tipo ID TO VL TC PV`  
`{ System.out.println("Riconosciuto prototipo funzione"); ;}`  
`| tipo ID TO VL TC GO stmt_list GC`  
`{ System.out.println("Riconosciuta funzione"); ;}`  
`;`
- NB: A tutti gli effetti queste due regole è come se fossero due regole separate, quindi se volessi eseguire la stessa azione per entrambe le regole, dovrei scrivere due azioni uguali, una per regola

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

Esercitazione 2

17



## Sezione della Regole: Esempio

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

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

```
VL ::= V
      | VL VIR V;
```

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

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

### Produzioni (regole grammaticali):

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

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

Esercitazione 2

18

## 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
  - Tale simbolo è una classe di tipo 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)
  - La classe Symbol si trova nel seguente file dell'installazione di cup:
    - ↳ java\_cup/runtime/Symbol.java
- Quando si dichiara un simbolo terminale per mezzo della parola chiave terminal, Cup associa a tale simbolo un valore intero
  - Contenuti nel file sym.java generato da cup durante la compilazione

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

Esercitazione 2

19

## Integrazione di JFlex e Cup

- Nel caso in cui nel parser siano dichiarati i seguenti simboli terminali:

```
terminal T, P, ID, NUM, PV, VIR, QO, QC;
```

- Potranno essere passarli da scanner a parser, quanto il parser li richiede, nel seguente modo:

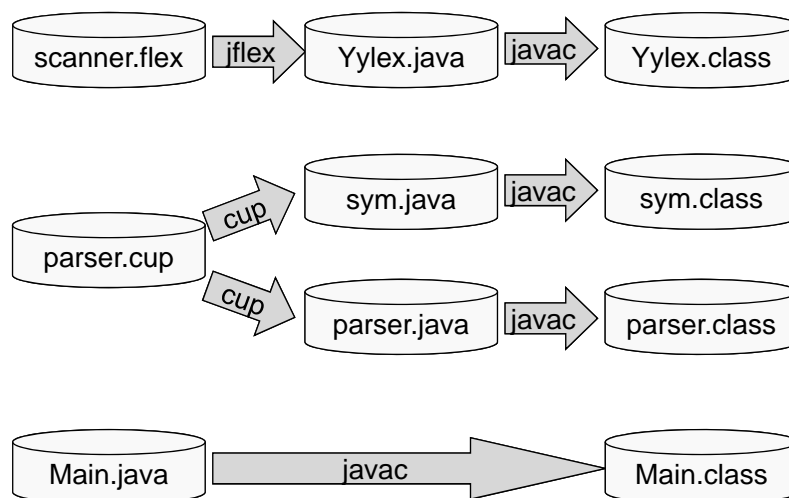
```
...
%%
...
%%
[a-zA-Z_][a-zA-Z0-9_]* {return new Symbol(sym.ID);}
\[                      {return new Symbol(sym.QO);}
\[                      {return new Symbol(sym.QC);}
...
```

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

Esercitazione 2

20

## Integrazione di JFlex e Cup



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

Esercitazione 2

21

scanner.flex

## 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
22

parser.cup

## Il parser Cup

---

List → List VIRGOLA EL  
 List → EL

```
import java_cup.runtime.*;

terminal EL, VIRGOLA;
non terminal Lista, List;

start with Lista;

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

List ::= List VIRGOLA EL
;

List ::= EL
;
```

<http://www.skenz.it/traduttori>
Esercitazione 2
23

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
24

## Compilazione

---

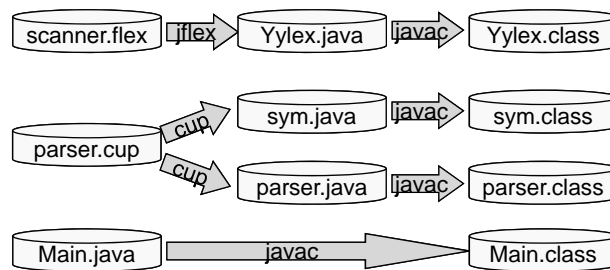
```

graph LR
    scanner[scanner.flex] -- jflex --> Yylex[Yylex.java]
    Yylex -- javac --> Yylex_class[Yylex.class]
    parser[parser.cup] -- cup --> sym[sym.java]
    sym -- javac --> sym_class[sym.class]
    parser -- cup --> parser_java[parser.java]
    parser_java -- javac --> parser_class[parser.class]
    Main[Main.java] -- javac --> Main_class[Main.class]
    
```

- jflex scanner.jflex
- java java\_cup.Main parser.cup
  - Nel caso di conflitti shift/reduce o reduce/reduce:
  - java java\_cup.Main -expect <numero\_conflitti> parser.cup
  - java java\_cup.MainDrawTree parser.cup
    - ✦ Può essere usata in laboratorio o installando una versione modificata di cup scaricabile dal sito del corso
    - ✦ Lanciando il programma verrà generato anche l'albero di derivazione (funzione utilissima per il debugging della grammatica)

<http://www.skenz.it/traduttori>
Esercitazione 2
25

## Compilazione



- `javac Yylex.java sym.java parser.java Main.java`
  - O semplicemente `javac *.java`
  - Per compilare tutti i file del progetto
- `java Main <file>`
  - Per eseguire il programma su <file>