

① Creare un Nuovo Progetto (nome = Progetto dest.)

② Crea file Main + PSVM

File Che Contiene dettagli Su Grammatiche Context free (\*.g4)

Grammar [NomeFile]

Per TESTARE le Regole Scritte Basta fare Tasto Destro 'Test Rule'

## PARTE TEORICA: (CENNI TEORICI)

**L'interprete** è un Programma che Dato un Programma ed un INPUT Restituisce l'OUTPUT dell'INPUT Eseguito sul quel Programma.

**Linguaggio formale** è un Insieme di Stringhe finite, Costituite a partire da un Alfabeto di Simboli anch'Esso finito

**Linguaggio formale** è un Sottinsieme di TUTTE le Stringhe di lunghezza finita che possono Essere Create a Partire dall' Alfabeto

SEMANTICA  $\Rightarrow$  Una fz. Matematica che Associa degli INPUT gli OUTPUT del Programma, ad Esempio:

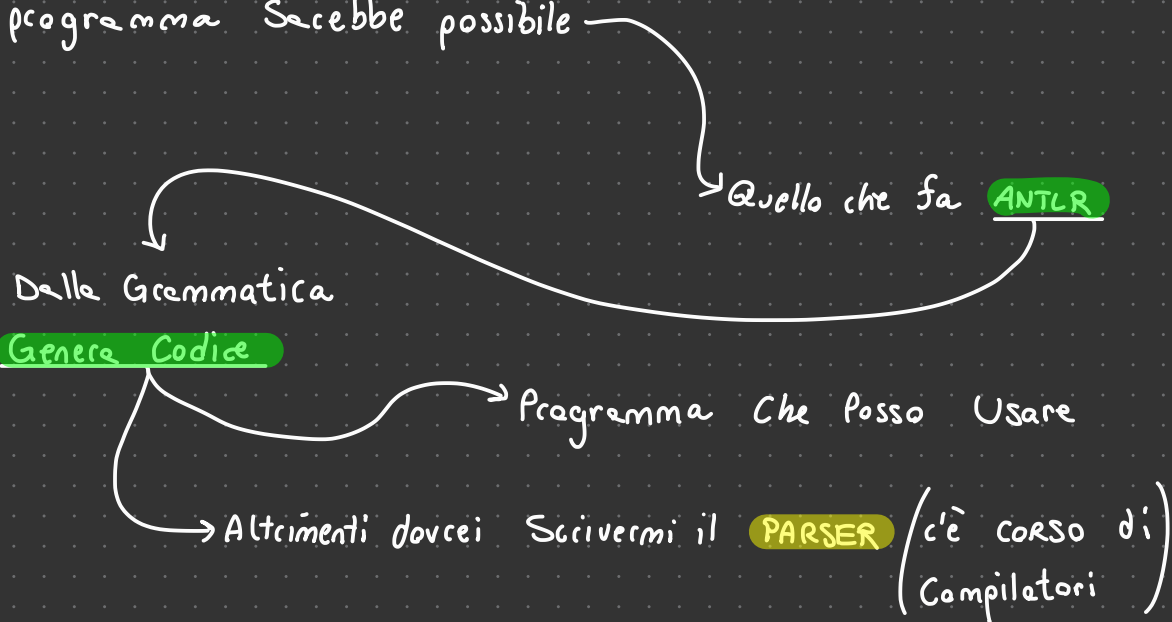
$$\text{Programma con fatt.} \quad [IP] (i) = \begin{cases} 1 & \text{Altrimenti} \\ \text{fattoriale}(i) & \text{se } i \geq 0 \end{cases}$$

**INTERPRETE:** Semantica Deve Essere Solo una Tra tutti,  
perche ci sono  $\infty$  Modi per Scriverlo

Non ci deve ESSERE NECESSARIAMENTE un legame Tra linguaggio  
in cui ho Scrritto il PROGRAMMA e Quello dell' INTERPRETE

Come prima Cosa fa un **Controllo SINTATTICO** per Capire  
Se e Linguaggio in cui è Scrritto il programma o Meno

IL problema è che dire Se  $w \in L$  è un Problema **NON**  
**DECIDIBILE** e Quindi devo Restringere il Campo... Se fosse  
descritto da una Grammatica CONTEXT-FREE allora Questo  
programma Sarebbe possibile



Con l'**IPOTESI** che  $L$  sia in Grado di Essere Generato da una **GRAMMATICA C.F.** allora l'Algoritmo per dire se  $\in L$  o Meno Esiste e Soprattutto è **DECIDIBILE**

**ALBERO DI PARSING**  $\Rightarrow$  Derivazioni della Grammatica per Riconoscere la STRINGA

Non ci deve ESSERE la **RICORSIONE INDIRETTA a Sinistra**, ad Esempio:

~~///~~  $S \rightarrow aS$  } Consumo Sempre il CARATTERE  $a$ , Quindi  
~~///~~  $S \rightarrow Sa$  } è ACCETTABILE  
~~///~~  $S \rightarrow S$

$\rightarrow$  NON può Andare Bene Visto che NON Consumo Nessun Carattere

## AMBIGUITÀ LESSICALI:

~~///~~ Sceglie **Quello che Arriva prima** (Se la Stringa è in più di una Categoria SINTATTICA)

~~///~~ **Longest-Match-Rule**  $\Rightarrow$  Uso il MATCH più lungo

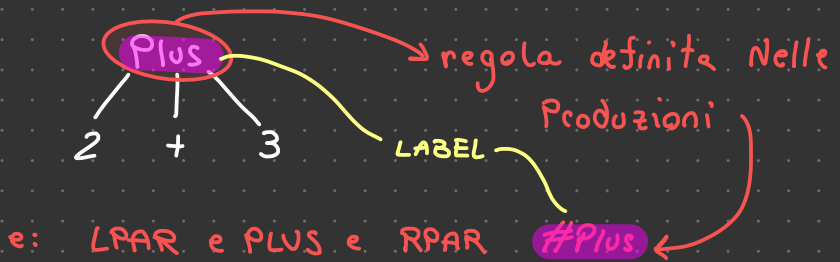
In fase di DEFINIZIONE è molto  
Importante per non Costringere a  
Parentesizzare TUTTO (per EX. ARITMETICHE)

**LEXER:** Analizzatore **Lessicale**, Trasforma una **Stringa in Seq. di TOKEN**, ad Esempio:

Da **'(2+3)'** a

LPAR	'('
NAT	'2'
PLUS	'+'
NAT	'3'
RPAR	')

**PARSER:** Analizzatore **Sintattico**, usa i TOKEN Generati dal LEXER e Costruirà dai TOKEN Creati prima Questo albero Qui:



**TERMINAL NODE**  $\Rightarrow$  Una foglia dell'Albero Sintattico. Rappresenta un **TOKEN Prodotto dal LEXER**

**NODO INTERNO** (regola Grammaticale del Parser) di TIPO **CONTEXT**

**CONTEXT** viene Generato ad ogni Regola Riconosciuta da PARSER

**CLASSE VISITOR**  $\Rightarrow$  Implementazione di un **INTERFACCIA** che **Interpreta il Nostro Programma (ALBERO)**

Noi DOBBIAMO Solo riscrivere le interfacce in modo che facciano Effettivamente il Nostro Caso

Se ho dei Valori parametrici devo usare uno 'STATO', che lo posso Implementare Tramite la HASMAP.

## LINGUAGGIO ESOTERICO BRAINFUCK:

→ Essere per Nulla Chiaro a Meno a chi lo ha Scritto, mi Serve Sapere la SEMANTICA

Ce Ne Sono Molti

BRAINFUCK è Relativamente Semplice:

Simboli  $\Rightarrow >, <, +, -, \cdot, ,, [, ]$

Esempio Programma Valido  $\Rightarrow P \equiv [< + > -]$

### SEMANTICA

$>/<$  Incremento/Decremento Puntatore

$+/-$  Incremento/Decremento valore che PUNTO

$\cdot$  Stampa a Video

$,$  Input

$[$  Se  $INT=0$  allora Esegui il Comando Successivo alla Parentesi Chiusa

$]$  Se  $INT \neq 0$ , Salto alla Parentesi Aperta Precedente

Array NON può ESSERE Realmente  $\infty$ , Quindi devo fare una Scelta:

- Imposto una dimensione MAX (vincolato)
- Uso una lista (non Posso indicizzare a dx) ma Spreco Tanta Memoria

Mapa <Integer, Integer> che è Efficiente e funziona:

- v.put(1000, 5)
- v.put(-5, 2)

1000	↦	5
-5	↦	2

I metodi che Sovrascrivo Sono Quelli che Eseguono la Semantica del Nostro Linguaggio

↓  
Nel caso di BRAINFUCK, modifico lo Stato della Memoria

## GRAMMAR ADVANCED DESIGN:

Sequenza di TOKEN e Sottofrasi (con Terminatore ';')

- ~~///~~  $(\text{regola})^* \Rightarrow$  Una o più Occorrenze (no  $\epsilon$ )
  - ~~///~~  $(\text{regola})^+ \Rightarrow$  Zero o Più Occorrenze (si  $\epsilon$ )
  - ~~///~~  $((\text{regola}))^? \Rightarrow$  Zero o 1 Occorrenza (si  $\epsilon$ )
- } NOT SURE

TOKEN DEPENDENCY Mettere Nelle Regole La coppia di Tok.  
Come ad Esempio '(' e ')'

FRASI INNESTATE (caso 1 | caso 2 | caso 3)

Prestare Attenzione Alla precedenza

Di Base ANTLR Associa a Sinistra, altrimenti devo forzare:  
exp: <assoc\_right> exp^....

Poi io voglio lavorare Sui Visitor






**IMP:**

→ Imperative

**TOKEN:**




 Naturali } + Tutte le KEYWORD  
 Booleani }

**COMANDI:**

 IF THEN ELSE  
 Assegnamento  
 SKIP  
 While  
 Out per Stampare

[ ; Usato per la  
composizione di Comandi ]

**ESPRESSIONI:**

 Naturale/Booleano  
 Expr. con Parentesi  
 Operazioni Aritmetiche

**IMPLEMENTAZIONE INTERPRETE:**