1 - Introduzione ai linguaggi di programmazione ed alla teoria dei linguaggi formali

La differenza tra un LDP e la programmazione è la possibilità di far conoscere più linguaggi di programmazione di diverso tipo al programmatore con un alta velocità, per far ciò non basta solamente apprendere molti linguaggi differenti ma comprendere le **fondamenta** dietro questi linguaggi.

Introduzione alla teoria dei linguaggi formali

La teoria dei linguaggi formali serve per possedere una buona comprensione dell'operazione di compilazione, questa si fonda pesantemente sulla teoria dei linguaggi formali.
Per poter definire un linguaggio servono diversi livelli di descrizione:

- **Grammatica**: definire quali frase sono corrette
- **Semantica**: cosa significa una frase corretta
- **Pragmatica**: come usare una frase corretta e sensata
- **Implementazione (per i linguaggi di programmazione)**: come eseguire una frase corretta in modo da rispettarne il significato

Concetto intuitivo di grammatica

La grammatica è una forma con una serie di regole che permettono di formulare frasi corrette composto da:

- **Alfabeto del linguaggio**: simboli con cui costruire le parole del linguaggio
- **Lessico**: parole previste dal linguaggio (con delle regole per poter poi costruire tutto)
- **Sintassi**: determina quale sequenze di parole costituiscono frasi legali corrette

Le **grammatiche** quindi sono uno **strumento utile per descrivere una sintassi di un linguaggio**

Studio dei linguaggi

I linguaggi di programmazione, una volta imparati le diverse regole, diventano familiari, qualunque esso sia.

Sintassi e semantica

Nei nostri linguaggi naturali si può comunicare con frasi mal strutturate e incomplete, ma un computer richiede precisione, per questo i programmi devono aderire rigorosamente a regole stringenti e anche delle differenze minori vengono rigettate come errate.

Per un linguaggio macchina è essenziale che **la sintassi sia corretta per comunicare una qualsivoglia semantica**.

Lo studio della semantica non è nient'altro che lo studio della grammatica, quindi della struttura delle frasi, in un linguaggio naturale si usano diverse regole scritte in BNF(Backus Naur Form), metalinguaggio utilizzato per poter definire queste regole in modo semplice e derivabili

The man hits the dog

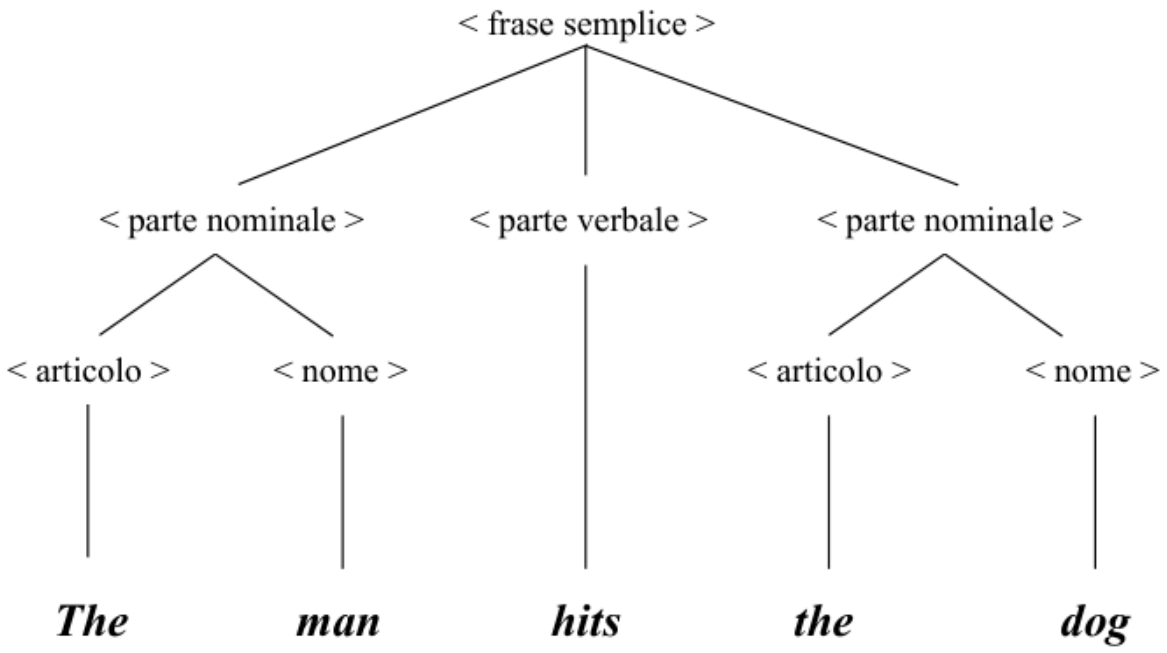
può essere analizzata sintatticamente, cioè risolta nelle parti grammaticali componenti, come segue:

L'insieme di regole è uguale a questo:

< frase semplice > :: =< parte nominale > < parte verbale >
< parte nominale >
< parte nominale > :: =< articolo > < nome >
< nome > :: = ***car*** | ***man*** | ***dog***
< articolo > :: = ***The*** | ***a***
< parte verbale > :: = ***hits*** | ***eats***

Si potrebbe descrivere questa frase con un albero di derivazione

Albero di derivazione



La definizione che da la <frase semplice> permette 72 diverse frasi derivabili, che sebbene alcune siano sintatticamente corrette, non hanno interpretazione errata

Questo esempio illustra un importantissimo concetto nella computazione, cioè nel processo di compilazione avviene come fase cruciale **l'analisi sintattica** come passo essenziale per la sua **interpretazione semantica**

Teoria dei linguaggi formali

Negli anni 50 **Chomsky** cerca di descrivere la sintassi del linguaggio naturale secondo semplici regole di **riscrittura e trasformazione**.
Chomsky considera alcune **restrizioni** sulle regole sintattiche e classifica dei linguaggi in base alle restrizioni imposte alle regole che generano tali linguaggi, una classe importante di regole derivate che genera linguaggi formali si chiama "**grammatiche libere da contesto** (Context-free Grammars)", tale classe dimostra che è uno strumento adeguato a descrivere la sintassi di base di molti linguaggi di programmazione

Un esempio di linguaggio C.F è il linguaggio delle parentesi ben formate che comprende tutte le stringhe di parentesi aperte e chiuse bilanciate correttamente.

Linguaggio delle parentesi ben formate

Questo linguaggio è infinito e permette di contrassegnare il **raggio di azione** nelle espressioni matematiche e, di conseguenza, nei linguaggi di programmazione (dipendente sempre da come questo è strutturato, nel C vengono usate `{}`)

Esempio:

( ) è ben formata;
(( ) ( )) è ben formata;
(( ) ( ) non è ben formata.

Questo linguaggio è definito da un insieme di regole:

1. La stringa $()$ è ben formata
2. Se la stringa di simboli A è ben formata, allora lo è anche (A)
3. Se le stringhe A e B sono ben formate, allora lo è anche la stringa AB (concatenate)

In corrispondenza di questa definizione induttiva, possiamo considerare un sistema di riscrittura che genera esattamente l'insieme di stringhe lecite di parentesi ben formate come

$$1. S \to ()$$

$$2. S \to (S)$$

$$3. S \to SS$$

Queste regole di riscrittura sono dette anche di **produzioni** o **regole di produzione**, in questo caso stabiliscono che "data una stringa, si può formare una nuova stringa sostituendo una S o con una $()$ o con (S) o con SS

- Esempio: Generazione di (( ))(( )( ))
  - Primo modo:

$$S \underset{(3)}{\Rightarrow} SS \underset{(2)}{\Rightarrow} (S)S \underset{(1)}{\Rightarrow} (( ))S \underset{(2)}{\Rightarrow} (( ))(S) \underset{(3)}{\Rightarrow} (( ))(SS) \underset{(1)}{\Rightarrow} (( ))(( )S) \underset{(1)}{\Rightarrow} (( ))(( )( ))$$

  - La corrispondente sequenza di applicazione delle produzioni è:

$$(3) \to (2) \to (1) \to (2) \to (3) \to (1) \to (1)$$

La sequenza di applicazione di queste regole di produzione in successione viene definito come **derivazione** (definito come $\Rightarrow$)

Nell'esempio viene usata la notazione $\alpha \Rightarrow \beta$ (da alpha si produce direttamente beta) per effetto dell'applicazione della regola di riscrittura n