Informatica Grafica

Corso di Laurea in Ingegneria Edile – Architettura

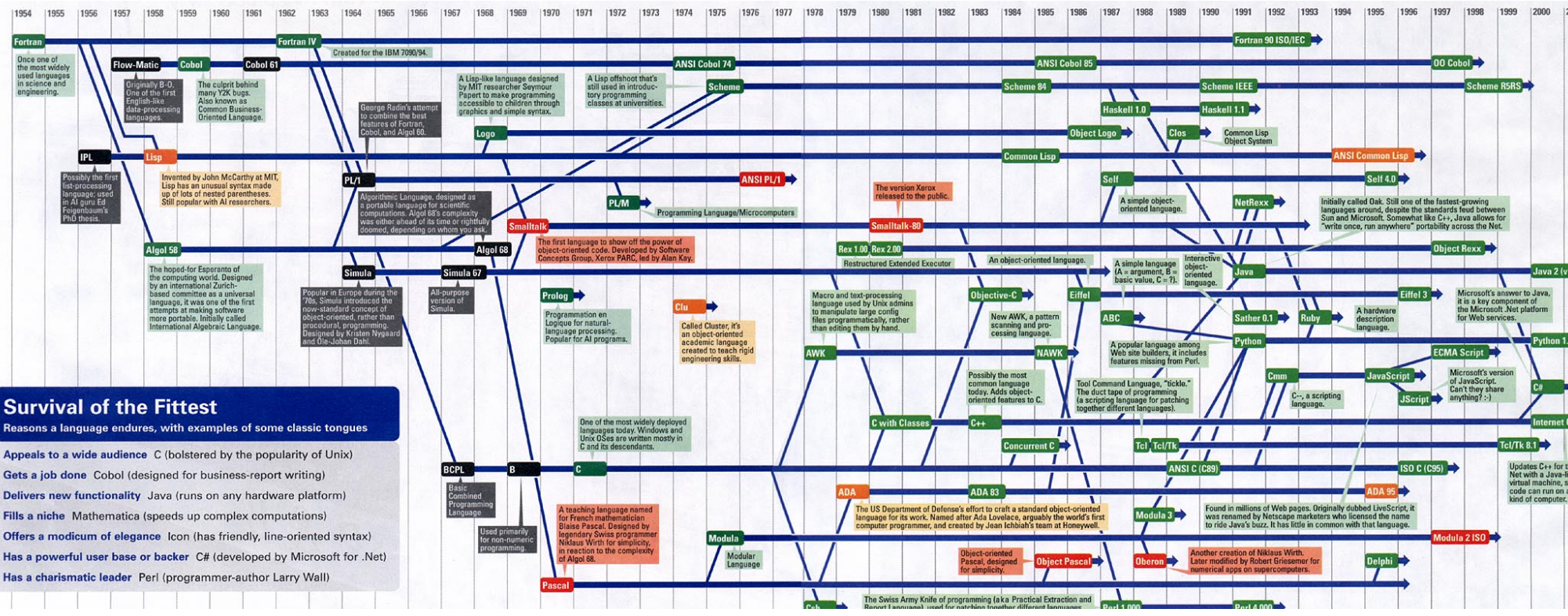
Linguaggi di programmazione

Paolo Torroni

Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
Università degli Studi di Bologna

Anno Accademico 2009/2010

Linguaggi di programmazione



- **Linguaggi di programmazione**
  - Tipologie di linguaggi
  - Dati e controllo
  - Sintassi e semantica
  - Programmazione imperativa e dichiarativa
  - Livelli di un sistema software

Parte I

Programmi e linguaggi di programmazione

Dall'algoritmo al programma

- **Programma**:

  descrizione di un **algoritmo** *scritta in un* **linguaggio di programmazione**

- Caratteristiche di un linguaggio di programmazione:
  - facilità di utilizzo da parte dei programmatori
  - interpretabilità ed eseguibilità dal computer
  - assenza di ambiguità
  - chiarezza e concisione
    - poche strutture sintattiche
    - ciascuna con un significato ben preciso

Dall'algoritmo al programma

- **Programma**:

  rappresentazione eseguibile di un **algoritmo**
  progettato per risolvere un **problema**

- Elementi di un programma:
  - **Logica**: **cosa** vogliamo ottenere dal programma
  - **Controllo**: **come** vogliamo raggiungere tale scopo

$$\textbf{Algoritmo} = \textbf{logica} + \textbf{controllo}$$

- Ad esempio: definizione (logica) di $n!$ (per $n \geq 0$)

$$n! = \begin{cases} 1 & \textbf{se } n \leq 1 \\ n \times (n-1)! & \textbf{altrimenti} \end{cases}$$

- Dove sono logica e controllo?

Due categorie di linguaggi

- **Linguaggi imperativi**
  - definiscono insiemi di **operazioni elementari**
  - definiscono **come** devono essere eseguite (in che ordine e sotto quali condizioni)
  - attenzione sia alla logica sia al controllo

```
int i=1, fattoriale=1;
while( i<=N ) {
    fattoriale=fattoriale*i;
    i=i+1;
}
```

Due categorie di linguaggi

- **Linguaggi dichiarativi**
  - definiscono insiemi di **dichiarazioni** (relazioni tra fatti noti)
  - enfasi sul **cosa**, non sul **come**
  - attenzione soprattutto sulla logica

```
fattoriale(0,1).
fattoriale(1,1).

fattoriale(X,F) :-
    X1 is X-1,
    fattoriale(X1,F1),
    F is X*F1.
```

Linguaggi imperativi

- **Operazioni di base**
  - operazioni **aritmetiche** (somma, sottrazione, etc.)
  - operazioni di **ingresso e uscita** (es. da tastiera, su video)
- **Costrutti linguistici per il controllo di flusso**
  - **sequenza** (esegui A, **poi** esegui B)
  - **scelta** (**se** si verifica una certa condizione C, esegui A)
  - **iterazione**
    - esegui A **N volte**
    - continua a esegure A **finché** non si verifica C
- **Costrutti per lavorare su dati**
  - Possibilità di definire **variabili**
    - nome, tipo, utilizzo, operazioni possibili, ...
  - Operazione di **assegnamento**
  - Possibilità di **costruire** dati per aggregazione

Costruiamo un linguaggio (imperativo) per CR

Caratteristiche di un linguaggio: eseguibilità, compattezza, non ambiguità, facilità d'uso

⇒ **Eseguibilità**: partiamo dalle operazioni eseguibili da CR

Compattezza: nomi simbolici a operazioni e costrutti

Non ambiguità: esiste una unica interpretazione del codice?

Facilità d'uso: tipi di dato, nomi simbolici alle variabili, nuovi costrutti, possibilità di definirne in più.

Costruiamo un linguaggio (imperativo) per CR

Caratteristiche di un linguaggio: eseguibilità, compattezza, non ambiguità, facilità d'uso

$\Rightarrow$ **Eseguibilità**: partiamo dalle operazioni eseguibili da CR

- **somma** 1 alla cella C_n;
- **memorizza** 0 nella cella C_n;
- **leggi** dall'esterno un numero, memorizzalo in C_n;
- **mostra** il contenuto della cella C_n;
- **stop**;
- **se** C_n e C_m contengono lo stesso numero,vai all'istruzione (X).

Compattezza: nomi simbolici a operazioni e costrutti

Non ambiguità: esiste una unica interpretazione del codice?

Facilità d'uso: tipi di dato, nomi simbolici alle variabili, nuovi costrutti, possibilità di definirne in più.

Costruiamo un linguaggio (imperativo) per CR

Caratteristiche di un linguaggio: eseguibilità, compattezza, non ambiguità, facilità d'uso

Eseguibilità: partiamo dalle operazioni eseguibili da CR

- `INC` *`cell_num`*
- `RESET` *`cell_num`*
- `STORE` *`cell_num`*
- `WRITE` *`cell_num`*
- `HALT`
- `JMPEQ` *`cell_num_1 cell_num_2`* `(`*`instruction_num`*`)`

⇒ **Compattezza**: nomi simbolici a operazioni e costrutti

Non ambiguità: esiste una unica interpretazione del codice?

Facilità d'uso: tipi di dato, nomi simbolici alle variabili, nuovi costrutti, possibilità di definirne in più.

Costruiamo un linguaggio (imperativo) per CR

Caratteristiche di un linguaggio: eseguibilità, compattezza, non ambiguità, facilità d'uso

Eseguibilità: partiamo dalle operazioni eseguibili da CR

- `INC` *`cell_num`*
- `RESET` *`cell_num`*
- `STORE` *`cell_num`*
- `WRITE` *`cell_num`*
- `HALT`
- `JMPEQ` *`cell_num_1 cell_num_2`* (*`instruction_num`*)

Compattezza: nomi simbolici a operazioni e costrutti

⇒ **Non ambiguità**: esiste una unica interpretazione del codice?

Facilità d'uso: tipi di dato, nomi simbolici alle variabili, nuovi costrutti, possibilità di definirne in più.

Costruiamo un linguaggio (imperativo) per CR

Caratteristiche di un linguaggio: eseguibilità, compattezza, non ambiguità, facilità d'uso

Eseguibilità: partiamo dalle operazioni eseguibili da CR

- `int x, y;`
- `x=x+y;`
- `read( x, keyboard );`
- `write( screen, x );`
- ~~`HALT`~~
- `if( ( x>0 )AND( x+y < z ) ) { ... }`
- `while( x==y ) { ... }`
- `int fattoriale( x ) { ... }`

Compattezza: nomi simbolici a operazioni e costrutti

Non ambiguità: esiste una unica interpretazione del codice?

$\Rightarrow$ **Facilità d'uso**: tipi di dato, nomi simbolici alle variabili, nuovi costrutti, possibilità di definirne in più.

Eseguiamo un programma con CR

- **Linguaggio macchina** L_{CR} (direttamente eseguibile da CR)
- Linguaggio dei computer: `0` e `1`
- Necessario **codificare** le istruzioni in sequenze di `0` e `1`
- Codice "binario" (non "leggibile")
- CR è un calcolatore rudimentale $\Rightarrow$ possibili solo 6 istruzioni
  - Che formato per i dati?
  - Quanti bit servono?
  - Una possibile codifica?

Un possibile linguaggio macchina per CR

Tabella: Possibile codifica delle istruzioni di CR

`INC`	`001`	**somma** 1 alla cella C_n
`RESET`	`000`	**memorizza** 0 nella cella C_n
`STORE`	`010`	**leggi** dall'esterno un numero, memorizzalo in C_n
`WRITE`	`011`	**mostra** il contenuto della cella C_n
`HALT`	`111`	**stop**
`JMPEQ`	`100`	**se** $C_n = C_m$ vai all'istruzione (X)

Codifica degli operandi per CR

- Gli operandi si riferiscono a celle di memoria
- Che codifica in binario?
  - Quante celle di memoria?
  - Esempio: memoria grande 100 celle $\Rightarrow$ bastano 7 bit

Tabella: Codifica di istruzioni e operandi di CR

INC *cell_num*	001 xxxxxxx
RESET *cell_num*	000 xxxxxxx
STORE *cell_num*	010 xxxxxxx
WRITE *cell_num*	011 xxxxxxx
HALT	111
JMPEQ *c_1 c_2* (*i*)	100 xxxxxxx xxxxxxx xxxxxxx

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?

```
1.  010 0001010
2.  010 0001011
3.  000 0001100
4.  100 0001011 0001100 0001000
5.  001 0001010
6.  001 0001100
7.  100 0001011 0001011 0000100
8.  011 0001010
9.  111
10.
11.
12.
```

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?

```
1.  010 0001010
2.  010 0001011
3.  000 0001100
4.  100 0001011 0001100 0001000
5.  001 0001010
6.  001 0001100
7.  100 0001011 0001011 0000100
8.  011 0001010
9.  111
10.
11.
12.
```

- **Codice macchina** per CR (L_{CR})

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?

```
1.  STORE 0001010
2.  STORE 0001011
3.  RESET 0001100
4.  JMPEQ 0001011 0001100 (0001000)
5.  INC 0001010
6.  INC 0001100
7.  JMPEQ 0001011 0001011 (0000100)
8.  WRITE 0001010
9.  HALT
10.
11.
12.
```

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?

```
1.  STORE 10
2.  STORE 11
3.  RESET 12
4.  JMPEQ 11 12 (8)
5.  INC 10
6.  INC 12
7.  JMPEQ 11 11 (4)
8.  WRITE 10
9.  HALT
10.
11.
12.
```

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?
  1. `STORE 10`
  2. `STORE 11`
  3. `RESET 12`
  4. `JMPEQ 11 12 (8)`
  5. `INC 10`
  6. `INC 12`
  7. `JMPEQ 11 11 (4)`
  8. `WRITE 10`
  9. `HALT`
  10. (*C1*)
  11. (*C2*)
  12. (*C3*)

Esempio di programma in linguaggio macchina

- Cosa fa il seguente programma?
  1. `STORE 10`
  2. `STORE 11`
  3. `RESET 12`
  4. `JMPEQ 11 12 (8)`
  5. `INC 10`
  6. `INC 12`
  7. `JMPEQ 11 11 (4)`
  8. `WRITE 10`
  9. `HALT`
  10. (*C1*)
  11. (*C2*)
  12. (*C3*)
- **Assembly** per CR (ASM_{CR})

Linguaggi di basso livello

- I primi linguaggi per i computer assomigliavano a L_{CR}
- Per semplificare la scrittura di programmi in linguaggi macchina sono stati introdotti i **linguaggi assembly**
  - Rappresentazioni simboliche di linguaggi macchina
  - Corrispondenza 1-1 istruzioni L_{CR} e ASM_{CR}
  - **Traduzione** da ASM_{CR} a L_{CR} fatta da un **programma assemblatore** (*assembler*)
- ASM_{CR} e L_{CR} sono **linguaggi di basso livello**
- Distanza dall'utente
  - È possibile una codifica più astratta dell'algoritmo?

Un linguaggio imperativo di più alto livello

- Codifica dell'algoritmo in **pseudo-codice**
  - linguaggio inventato, ma con un chiaro significato

```
1. int C1, C2, C3=0;
2. read( C1, keyboard );
3. read( C2, keyboard );
4. while( C2!=C3 )
   {
5.     C1++;
6.     C3++;
   }
7. write( screen, C1 );
```

Un linguaggio imperativo di più alto livello

- Codifica dell'algoritmo in **pseudo-codice**
  - linguaggio inventato, ma con un chiaro significato

```
1. int C1, C2, C3=0;
2. read( C1, keyboard );
3. read( C2, keyboard );
4. while( C2!=C3 )
   {
5.     C1++;
6.     C3++;
   }
7. write( screen, C1 );
```

- **Linguaggio** L_1 (indipendente da CR)

Caratteristiche di un linguaggio di alto livello

- In cosa si distingue L_1 da ASM_{CR}?
  - Minore rigidità nel formato delle istruzioni (spazi, indentazioni, a-capo, numerazione delle istruzioni ininfluente)
  - Possibilià di usare nomi simbolici (`screen`, `keyboard`)
  - Possibilià di definire variabili (`C1`, `C2`, `C3`)
  - Possibilià di specificare il tipo delle variabili (`int`)
  - Rappresentazione esplicita del costrutto di iterazione (`while`)
  - Possibilità di raggruppare istruzioni ({ `...` })
- Possibilità aggiuntiva: definire **funzioni**

Funzioni

- **Definizione** di una funzione `somma`

```
1.  int somma( int A, int B )
    {
2.      int S=0;
3.      while( B!=S )
        {
4.          A++;
5.          B++;
        }
    }
    ...
```

- **Uso** della funzione `somma`

```
11. int C1, C2=0;
12. read( C1, keyboard );
13. read( C2, keyboard );
14. write( screen, somma( C1, C2 ) );
```

Traduzione di L_1

- Per essere usato da CR, un programma in L_1 deve essere tradotto
- Varie possibilità:
  - **compilazione** di L_1 in un programma in L_{CR}
  - **compilazione** di L_1 in un programma in ASM_{CR} e poi uso dell'assembler
  - **interpretazione** di ciascuna istruzione di L_1 da parte di un programma in L_{CR}
  - traduzione (compilazione/interpretazione) in un **codice intermedio** con successiva compilazione/interpretazione

Linguaggi di più alto livello

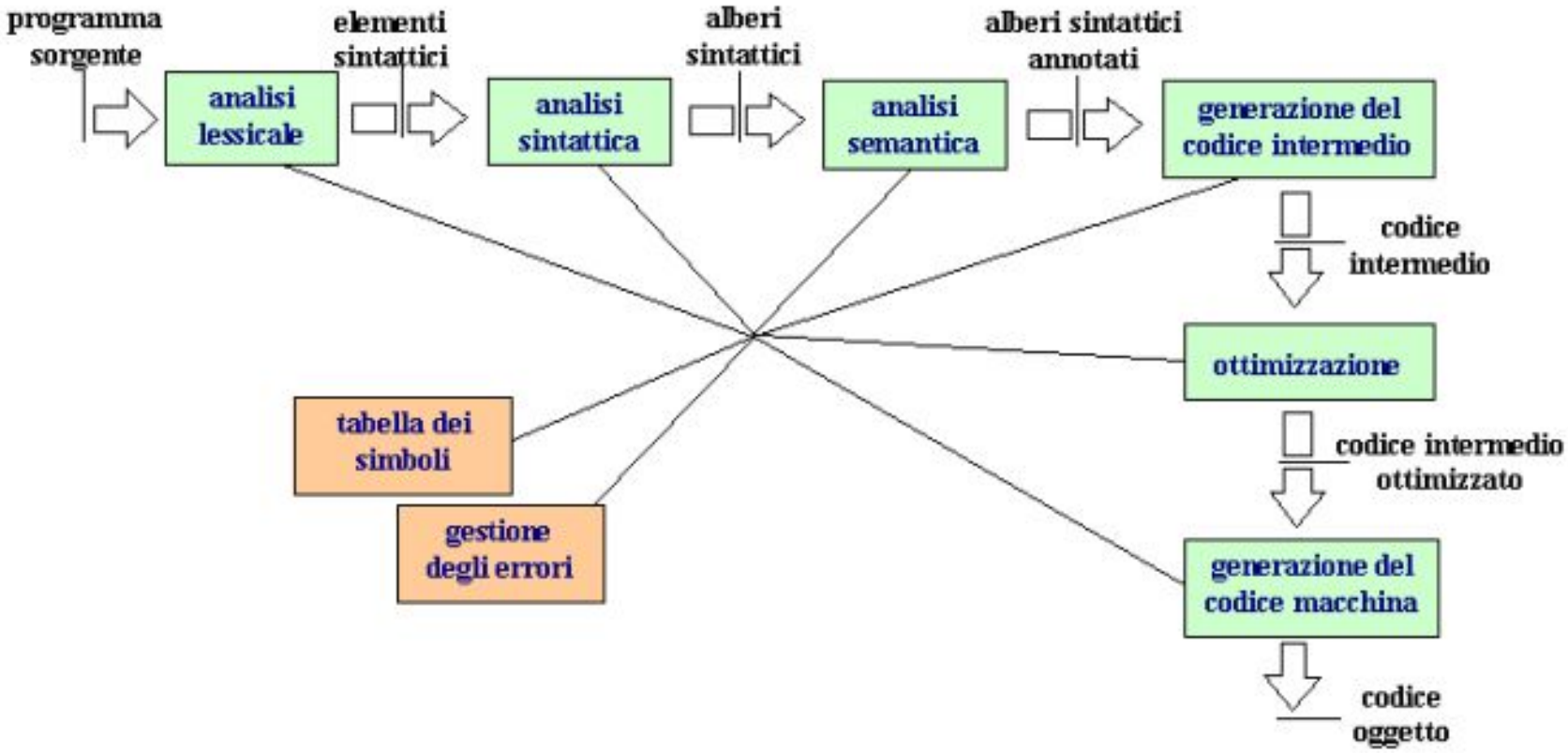
- Linguaggi di programmazione **dichiarativi**
  - Lingauggi logici e a vincoli, usati in *intelligenza artificiale*
  - Usano la logica come base per la definizione di un programma
  - Utili per ragionare sulla conoscenza
  - Usati anche in *ambito commerciale* per risolvere problemi "difficili" (ottimizzazione combinatoria)
- Linguaggi di **marcatura**
  - Usati per scrivere pagine Web
  - HTML
- Linguaggi per le **basi di dati**
  - Consentono di definire e interrogare basi di dati
  - SQL
- Spesso i linguaggi di più alto livello sono **interpretati**

Come definire un linguaggio di programmazione?

- Esistono degli elementi comuni a **tutti** i linguaggi
  - **Alfabeto**: definisce quali simboli usare per scrivere programmi
  - **Sintassi**: definisce le regole che i simboli devono rispettare per ottenere un programma corretto
  - **Semantica**: definisce il significato dei programmi

Implementazione di un linguaggio

- Implementare: definire una trasformazione (traduzione) che prende in input il codice ad alto livello, e lo rende eseguibile
- Occorre definire varie trasformazioni (passi di traduzione)



Parte II

Sintassi

Categorie sintattiche, simboli terminali, produzioni e scopo

- Una grammatica consiste di 4 elementi:
  - $\mathcal{N}$ Insieme dei **simboli non terminali**, detti anche **categorie sintattiche**.
    - Rappresentano **categorie linguistiche**
    - Rappresentate tra parentesi angolate: `<frase>`, `<soggetto>`, `<verbo>`, `<avverbio>`, ...
  - $\mathcal{T}$ Insieme dei **simboli terminali**.
    - Sequenze di caratteri che appaiono nel linguaggio e che vanno **considerati come un tutt'uno**
    - Esempio: `Marco`, `lavora`, `bene`
  - $\mathcal{P}$ Insieme delle **regole**, dette anche **produzioni**.
    - Specificano in che modo, in una stringa, un non terminale può essere **sostituito**
  - σ Lo **scopo** della grammatica, detto anche **simbolo iniziale**
    - È un simbolo non terminale.
    - Specifica la **radice** da cui partire per applicare le produzioni.

Formato delle regole di produzione

- Nei linguaggi di programmazione il formato delle regole ha questa forma:

$$\text{simbolo n.t.} \gg \text{sequenza di simboli t. e n.t.}$$

- Grammatiche **context-free**
  - Un terminale può essere rimpiazzato indipendentemente dai simboli che lo precedono o seguono (contesto)
- Esempio: grammatica della lingua italiana.

$\mathcal{N}$: { `<frase>`, `<soggetto>`, `<verbo>`, `<avverbio>` }
$\mathcal{T}$: { `Marco`, `lavora`, `bene` }
$\mathcal{P}$: { `<frase>` $\gg$ `<soggetto>` `<verbo>` `<avverbio>` (1)
`<soggetto>` $\gg$ `Mario` (2)
`<verbo>` $\gg$ `lavora` (3)
`<avverbio>` $\gg$ `bene` } (4)
σ: `<frase>`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<frase>` $\gg$ `<soggetto> <verbo> <avverbio>`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `<soggetto>` $\gg$ `Mario`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

$\mathcal{P}_{(3)}$: `<verbo>` $\gg$ `lavora`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

$\mathcal{P}_{(3)}$: `Mario lavora <avverbio>`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

$\mathcal{P}_{(3)}$: `Mario lavora <avverbio>`

$\mathcal{P}_{(4)}$: `<avverbio>` $\gg$ `bene`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

$\mathcal{P}_{(3)}$: `Mario lavora <avverbio>`

$\mathcal{P}_{(4)}$: `Mario lavora bene`

Derivazione

- Come facciamo a sapere che `Mario lavora bene` è una **frase corretta** della lingua italiana?
- Seguiamo un processo di **derivazione**, a partire da σ.

$$\sigma : \texttt{<frase>}$$

$\mathcal{P}_{(1)}$: `<soggetto> <verbo> <avverbio>`

$\mathcal{P}_{(2)}$: `Mario <verbo> <avverbio>`

$\mathcal{P}_{(3)}$: `Mario lavora <avverbio>`

$\mathcal{P}_{(4)}$: `Mario lavora bene`

- Le frasi **lecite** (sintatticamente corrette) sono tutte e sole quelle **derivabili** attraverso le regole della grammatica.

Linguaggi generati da grammatiche

- Possiamo definire l'insieme di stringhe derivabili da un simbolo non-terminale di una grammatica.

Definizione (linguaggio di un simbolo non terminale)

Data una grammatica, ed un simbolo non terminale di essa `<non-term>`*, diciamo* **linguaggio di** `<non-term>` *nella grammatica considerata l'insieme di stringhe di* **soli simboli terminali derivabili da** `<non-term>`*.*

- Il linguaggio di una grammatica è quello del suo scopo o simbolo iniziale (σ).

EBNF (Extended Bakus-Naur Form)

- EBNF è una **notazione** per definire una grammatica.
  - Produzioni: <simbolo n.t.> $\gg$ <sequenza di simboli>
  - **Possibile assenza** di un elemento sintattico: [...]
  - **Ripetizione** di un elemento almeno una volta: (...)$^{+}$
  - **Ripetizione** di un elemento zero o più volte: (...)*
  - **Alternativa** tra due elementi: ... | ...

Grammatiche regolari

- Sono una classe importante di grammatiche
- Esiste un algoritmo efficiente per riconoscere i **linguaggi regolari**
  - Solo due formati possibili di regole
    - `<non-term>` ≫ `term <non-term>`
    - `<non-term>` ≫ `term`
  - Esempio: numeri binari
    - `<bin>` ≫ `0 | 1 | 0 <bin> | 1 <bin> |`
- **Espressioni regolari**: una notazione alternativa delle grammatiche regolari, molto usate
  - editor per manipolare testi in base a *pattern*
    - alternativa, parentesi tonde, quadre, graffe
    - quantificazione: `?`, `*`, `+`
    - inizio/fine stringa: `^`, `$`