

Sommario: 25 febbraio, 2020

- Linguaggi di Programmazione: Cosa Sono? Perché Esistono?
- Funzioni Calcolabili e Funzioni Parziali.
- Linguaggio, Programma, Algoritmo e Funzione Calcolabile
- Relazione tra L. di Programmazione e Funzioni Calcolabili
- Esercizi

# Linguaggi di Programmazione: Cosa sono? Perché Esistono?

- Rispondiamo mostrando che sono lo strumento di congiunzione tra alcune nozioni correlate che includono: Funzioni Calcolabili, Algoritmi, Computer Applications.

## Definizione (Linguaggi di Programmazione LP)

Sono i **Formalismi** con cui scrivere **formule**, chiamate **Programmi**, per:

- **Esprimere** *tutte*<sup>a</sup> e **solo** le Computer Applications
- **Definire** *tutte* e **solo** le Funzioni Calcolabili
- **Implementare** *gli* Algoritmi e *renderli* Processi Automatici

---

<sup>a</sup>Tesi di Church-Turing 1936, o tesi dell'equivalenza dei modelli di calcolo, M.Mirsky, Computation: Finite and Infinite Machines, 1972, pag.108-109

I Linguaggi di Programmazione sono i **Formalismi** per:

- **Esprimere** tutte e **solo** le *Computer Applications*:
  - *Computer Applications* sono tutte le applicazioni che:
    - *sono state, sono e saranno*<sup>1</sup> realizzabili in Processi Automatici (eseguibili oggi, su computers *isolati* e/o *interconnessi*)
    - *pervadono ogni comparto e attività* della nostra esistenza: Information, Production, Education, Research, Culture, Health...
- **Definire** tutte e **solo** le *Funzioni Calcolabili*,  $\mathcal{F}$ .

---

<sup>1</sup>Ganchev, H.,..., Clements, K., ..., Coecke, B., Challenging Turing: Extended Models of Computation. The In-computable: Journey Beyond the Turing Barrier, Springer-Verlag, 2017, pag.11-82

I Linguaggi di Programmazione sono i **Formalismi** per:

- **Esprimere** tutte e **solo** *Computer Applications* ...
- **Definire tutte e solo** le *Funzioni Calcolabili*,  $\mathcal{F}$ .

## Definizione (Funzioni Calcolabili $\mathcal{F}$ )

Sia  $\mathcal{D}$  insieme numerabile,

- $\mathcal{F} \stackrel{\text{def}}{=} [\mathcal{D} \rightarrow \mathcal{D}] \subset \mathcal{D} \rightarrow \mathcal{D}^a$
- *Continue*<sup>b</sup> in Topologia di Scott:  $\mathcal{D} \cong^c [\mathcal{D} \rightarrow \mathcal{D}]$
- *definite*<sup>d</sup> mediante specifici formalismi, che includono:  
*Combinatory Logic,  $\lambda$ -Calculus, Turing Machines, State Machines, ..., Linguaggi di Programmazione (LP)*

---

<sup>a</sup>  $\mathcal{D} \rightarrow \mathcal{D}$  è l'insieme non numerabile contenente tutte le funzioni **Parziali** da  $\mathcal{D}$  in  $\mathcal{D}$

<sup>b</sup>  $(\forall g \in \mathcal{F}, \forall c \in \mathcal{D}, \forall \sqcup \{c_i\} \subset \mathcal{D})$  Se  $c \equiv \sqcup \{c_i\}$  Allora  $g(c) = g(\sqcup \{c_i\}) = \sqcup (g(c_i))$

<sup>c</sup>  $\cong$  sta per *isomorfo*, vedi D.Scott, Continuous Lattices, Lecture Notes in Mathematics 274 (1972), 97-136

<sup>d</sup> "definite", qui, significa *finitamente, completamente definite*.

# Funzioni Calcolabili: I Programmi sono Formule

A sinistra un **programma**, in Linguaggio C, definisce una funzione;  
A destra una **formula**, in notazione equazionale-algebrica, per la stessa funzione:

```
void main(){
  int n, ret = 1;
  scanf("%d",&n);
  if(n<0) ret = 0;
  while (n > 0){
    ret =* n;
    n--;}
  printf("%d",ret);
} //Un programma P
```

definisce e calcola  $fact_0(n) = \begin{cases} n! & \text{se } n \geq 0 \\ 0 & \text{se } n < 0 \end{cases}$

- Anche il programma P è una formula: Il linguaggio C è un **formalismo**;
- La formula a sinistra **definisce ed in aggiunta calcola, utilizzando una Macchina C**, la funzione  $fact_0$  in ogni punto  $n$  del dominio (degli interi di C);
- La formula a destra fornisce una definizione per  $fact_0$  ma non è eseguibile su nessuna Macchina nota.

Anticipiamo alcune fondamentali nozioni che saranno riprese e approfondite nel corso.

## Definizione (Formalismo)

*Un Formalismo è definito da una **Sintassi** e da una **Semantica**.  
Esso consiste nell'insieme di tutte le formule aventi una forma che soddisfa la Sintassi e il significato assegnato a ciascuna formula dalla Semantica*

Esempio.

- Aritmetica di Peano;
- Logica dei Predicati;
- Teoria degli Insiemi di Cantor;
- Linguaggio di Programmazione;

## Definizione (Linguaggio di Programmazione)

Un Linguaggio di Programmazione  $\mathcal{L}$  è un formalismo  $\langle \mathcal{P}_{\mathcal{L}}, [|-]_{\mathcal{L}} \rangle$  che definisce tutte e solo le Funzioni Calcolabili,  $\mathcal{F}$ .

La **Sintassi**  $\mathcal{P}_{\mathcal{L}}$  fissa la forma di ogni programma.

La **Semantica**  $[|-]_{\mathcal{L}}$  associa ad ogni programma la Funzione Calcolabile da esso definita.

Esempio. Riprendiamo la situazione di prima:

```
void main(){
  int n, ret = 1;
  scanf("%d",&n);
  if(n<0) ret = 0;
  while (n> 0){
    ret *= n;
    n--;}
  printf("%d",ret);
} //Un programma P
```

definisce e calcola  $\text{fact}_0(n) = \begin{cases} n! & \text{se } n \geq 0 \\ 0 & \text{se } n < 0 \end{cases}$

Nel linguaggio C, Sintassi e Semantica siano  $\mathcal{P}_C, [|-]_C$ . Allora:

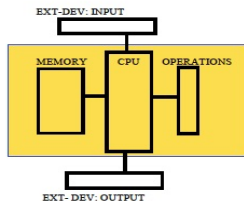
$$P \in \mathcal{P}_C, \quad [P]_C = \text{fact}_0$$

## Definizione (Macchina Astratta ed Esecutore di Programmi)

Una Macchina Astratta  $\mathcal{M}_{\mathcal{L}}$  è un modello di calcolo (Hardware e/o Software) costituito da una coppia  $\langle \mathcal{L}, \mathcal{E}_{\mathcal{L}} \rangle$ .

$\mathcal{L}$  è un Linguaggio di Programmazione, detto Linguaggio Macchina.

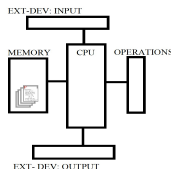
$\mathcal{E}_{\mathcal{L}}$  è un esecutore dei programmi di  $\mathcal{L}$  che calcola in accordo alla semantica di  $\mathcal{L}$ .



- In giallo: La tipica Macchina Astratta di un LP a basso livello (espressivo, es. Linguaggio Assembly)
- Ogni LP ha una propria Macchina Astratta il cui esecutore fornisce una **realizzazione astratta/concreta del modello di calcolo** utilizzato per calcolare i programmi di LP.
- La Macchina Astratta può mostrare anche le **apparecchiature esterne con cui comunica**.

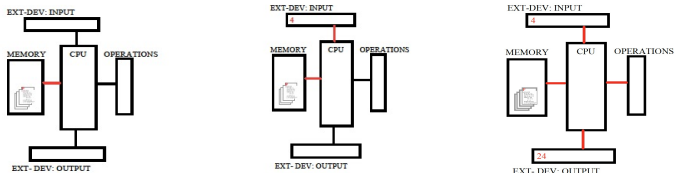


```
void main(){
  int n, ret = 1;
  scanf("%d",&n);
  if(n<0) ret = 0;
  while (n > 0){
    ret = * n;
    n--;}
  printf("%d",ret);
} //Un programma P
```



- Sinistra. Un programma del **Linguaggio C** (il programma P già visto prima)
- Centro. Il programma è contenuto in uno o più files e rappresentato in **Sintassi Astratta** o altro equiv.
- Destra. **Una Macchina Astratta per C**
  - Un Linguaggio può avere più di una Macchina Astratta:
  - Tutte però tra loro equivalenti rispetto alla Semantica del Linguaggio.
  - In particolare, i diversi esecutori forniscono stesso comportamento ad uno stesso programma.
  - Il programma per essere eseguito deve risiedere in Memoria,
  - e le unità che lo compongono, sono selezionate singolarmente ed interpretate dalla CPU.

# Macchina Astratta: Vediamola nella pratica.



- Sinistra. AM inizia l'esecuzione del programma P:  
In rosso la comunicazione degli STm di P, selezionati singolarmente e trasmessi in sequenza;
- Centro. AM legge dall'input:  
In rosso le comunicazioni per:
  - la selezione dello STm `scanf(...)`
  - e la lettura dal buffer di input di un intero da assegnare alla variabile `n`.
- Destra. AM scrive sull'output:  
In rosso le comunicazioni per:
  - la selezione dello STm `while(...)` e della sua esecuzione che coinvolge:
    - operazioni per il confronto, prodotto e decremento
    - il meccanismo di iterazione di STm
  - e la selezione dello STm `print(...)` che coinvolge:
    - la scrittura sul buffer di output dell'ultimo intero assegnato a `ret`

# Funzioni Calcolabili: Calcolo Non Terminante

A sinistra un **programma**, in C, che definisce una funzione il cui calcolo è Non Terminante su alcuni punti del dominio:

A destra una **formulazione** equazionale-algebrica:

```
void main(){
  int n, ret = 1;
  scanf("%d",&n);
  while (n != 0){
    ret *= n;
    n--;}
  printf("%d",ret);
}
```

definisce e calcola  $\text{fact}_{\perp}(n) = \begin{cases} n! & \text{se } n \geq 0 \\ \perp & \text{se } n < 0 \end{cases}$

- La funzione definita dal programma è (strettamente) Parziale:
  - è **non-terminante** sui negativi;
  - è **indefinita** sui negativi;
  - è **divergente** sui negativi;
  - **vale**  $\perp$  (indefinito) sui negativi.

# Funzioni Calcolabili: Funzioni Parziali (e Totali)

## Definizione (Funzioni Parziali su $\mathcal{D}: \mathcal{D} \rightarrow \mathcal{D}$ )

*Sono funzioni che possono essere non definite su alcuni punti del dominio  $\mathcal{D}$ .*

*Include le Totali, definite su tutti i punti.*

- $\text{fact}_0$  e  $\text{fact}_\perp$ : Condividono **Struttura del Dominio  $\mathcal{D}$**  (i.e quella dei **Valori** del Linguaggio C);
- È un problema di **Terminazione del Calcolo della funzione definita**
- Esistono Totali che si sanno calcolare solo utilizzando Parziali

## Esempio (Funzioni Parziali)

*Una stream (illimitata) di interi da leggere fino ai primi due contigui uguali ...*

```
void main(){
    int last, curr;
    scanf("%d",&last);
    scanf("%d",&curr);
    while(last!=curr){last=curr; scanf("%d",curr);}
    printf("%d",curr);
}
```

*Leggerà due contigui uguali? Cos'è una stream in  $\mathcal{D}$ ? Si consideri Esercizio L1.10*

I Linguaggi di Programmazione sono i **Formalismi** per:

- **Esprimere** tutte e **solo** *Computer Applications*...
- **Definire tutte e solo** le *Funzioni Calcolabili*,  $\mathcal{F}$

Definizione (Linguaggio di Programmazione  $\mathcal{L} \in \mathbf{LP}$ )

Sia  $\mathcal{F} \stackrel{\text{def}}{=} [\mathcal{D} \rightarrow \mathcal{D}]$  l'insieme delle funzioni calcolabili.

Sia  $\mathcal{L} \stackrel{\text{def}}{=} \langle \mathcal{P}_{\mathcal{L}}, [|\cdot|]_{\mathcal{L}} \rangle \in \mathbf{LP}$ , un Linguaggio di Programmazione.

Allora,

- *Meaning.*  $[|\cdot|]_{\mathcal{L}} : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{F}$  suriettiva
- *Representation.*  $\exists \bar{\cdot} : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{D} \in \mathcal{F}$  iniettiva <sup>a</sup>
- *Universal.*  $\exists \mathcal{U}_{\mathcal{L}} \in \mathcal{F}$  such that:
  - $(\forall p \in \mathcal{P}_{\mathcal{L}}) \quad \mathcal{U}_{\mathcal{L}}(\bar{p}) = g \in (\mathcal{D} \cong [\mathcal{D} \rightarrow \mathcal{D}] \equiv) \mathcal{F}$
  - $\forall g \in \mathcal{F}, \exists p \in \mathcal{P}_{\mathcal{L}},$  such that:
    - $g = [|p|]_{\mathcal{L}}$
    - $\mathcal{U}_{\mathcal{L}}(\bar{p})(x) = g(x) \quad \forall x \in \mathcal{D}$

---

<sup>a</sup>scrivendo  $\bar{p} \in \mathcal{D}$  per  $\bar{\cdot}(p), \forall p \in \mathcal{P}_{\mathcal{L}}$

# Funzione Universale $\mathcal{U}_{\mathcal{L}}$ , dato un LP $\mathcal{L}$

Cosa Significa e Cosa Implica l'esistenza di questa funzione  $\mathcal{U}_{\mathcal{L}}$ ?

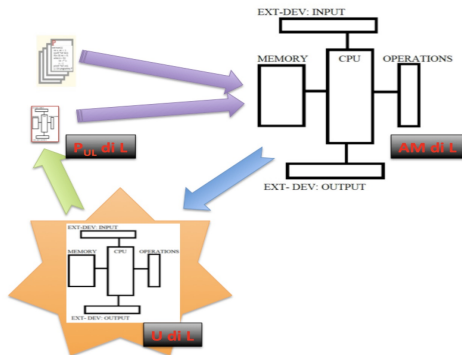
- $\forall \mathcal{L} \in \mathbf{LP}, \exists \mathcal{U}_{\mathcal{L}} \in \mathcal{F}$ ,
- $\forall p \in \mathcal{P}_{\mathcal{L}}, \mathcal{U}_{\mathcal{L}}(\bar{p}) = g \in (\mathcal{D} \cong [\mathcal{D} \rightarrow \mathcal{D}] \equiv) \mathcal{F}$

Ogni Linguaggio di Programmazione  $\mathcal{L}$  ha una propria Funzione Universale  $\mathcal{U}_{\mathcal{L}}$  che:

- È calcolabile,
- Applicata ad ogni programma (opportunamente rappresentato in  $\mathcal{D}$ )  $p$  di  $\mathcal{L}$  calcola la funzione calcolabile definita da  $p$  in  $\mathcal{L}$ .

# Funzione Universale $\mathcal{U}_{\mathcal{L}}$ : Riflessione di $\mathcal{L}$

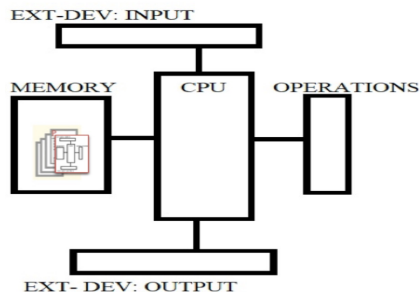
Ogni Linguaggio di Programmazione ha tra i suoi Programmi una Immagine di Se Stesso



- **Blue.** Dalla AM alla Funzione Calcolabile  $\mathcal{U}_{\mathcal{L}}$ .
- **Verde.** Dalla  $\mathcal{U}_{\mathcal{L}}$  ad un Programma  $P_{\mathcal{U}_{\mathcal{L}}}$  di  $\mathcal{L}$  che definisce  $\mathcal{U}_{\mathcal{L}}$  implementando lo Stesso Comportamento Computazionale di AM: Calcola la stessa sequenza sequenza di Stati calcolata da AM, dato un programma di  $\mathcal{L}$ .
- **Viola.** Carichiamo in AM anche  $P_{\mathcal{U}_{\mathcal{L}}}$ :

# Una Lista di $AM_{\mathcal{L}}$ con Riflessione

Ogni Linguaggio di Programmazione ha tra i suoi Programmi una Immagine di Se Stesso e del Comportamento Computazionale della sua AM.



Una sola Immagine? - No, almeno tante quante le differenti AM che possiamo scrivere in  $\mathcal{L}$  per  $\mathcal{L}$  stesso.



## Definizione (Relazione tra $\mathcal{F}$ e **LP**)

- $\mathcal{F} \equiv [\mathcal{D} \rightarrow \mathcal{D}]$ ,  $\mathcal{L} \in \mathbf{LP}$ 
  - (Universale)  $\exists \mathcal{U} \in \mathcal{F}$ ,
  - (Meaning)  $\exists [|-]: \mathcal{P} \rightarrow \mathcal{F}$ , suriettiva,
  - $\forall \bar{p} \in \mathcal{P}^a$ ,  
$$\mathcal{U}(\bar{p}) = g \in (\mathcal{D} \cong [\mathcal{D} \rightarrow \mathcal{D}] \equiv) \mathcal{F}$$
  - $\forall g \in \mathcal{F}$ ,  $\exists \bar{p} \in \mathcal{P}$  :  
$$\mathcal{U}(\bar{p})(x) = g(x) \quad \forall x \in \mathcal{D}$$

dove:

$\mathcal{D}$  numerabile,  $[\mathcal{D} \rightarrow \mathcal{D}]$  funzioni continue su  $\mathcal{D} \rightarrow \mathcal{D}$  con topologia di Scott (quindi,  $\mathcal{D} \cong [\mathcal{D} \rightarrow \mathcal{D}]$ ),  $\mathcal{P}$  insieme programmi di  $\mathcal{L}$ ,  $- : \mathcal{P} \rightarrow \mathcal{D}$  iniettiva.

---

<sup>a</sup> $\mathcal{P}$  dipende da  $\mathcal{L}$ , e scriveremo estesamente  $\mathcal{P}_{\mathcal{L}}$  quando più linguaggi  $\mathcal{L}$  sono considerati e potrebbe esserci ambiguità. Analogamente, è la funzione universale  $\mathcal{U}$  di  $\mathcal{L}$ , e il mapping suriettivo  $[|-]$

Gli **Strumenti Formali, Fondamentali** (che introducono strutture finite: I Programmi) per:

- **Esprimere** tutte e **solo** le *Computer Applications*  
...
- **Definire tutte e solo** le *Funzioni Calcolabili  $\mathcal{F}$* .  
...
- **Implementare** gli *Algoritmi* di calcolo e renderli *Processi Automatici*  
—— Nella prossima Lezione

# Esercizi L1

- 1 Scrivere un programma  $C$  che calcoli la funzione *ordinamento di sequenza* implementando l'algoritmo di QuickSort. Allo scopo completare il testo con eventuali vincoli che si ritenga utile assumere.
- 2 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che  $\mathcal{F}$  è numerabile.
- 3 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che la funzione  $[[\_]]$  è calcolabile. E se sì, cosa calcola.
- 4 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che  $\mathcal{P}$  contiene un programma che calcola  $[[\_]]$ .
- 5 Se e quale relazione, nella formulazione  $\mathcal{F}$ , vediamo tra l'iniettiva  $\_ : \mathcal{P} \rightarrow \mathcal{D}$  e la suriettiva  $[[\_]]$ .
- 6 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che per ogni funzione calcolabile c'è un programma che la calcola.
- 7 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che ogni linguaggio di programmazione definisce tutte e solo le funzioni calcolabili.
- 8 Se e dove, nella formulazione  $\mathcal{F}$ , vediamo che  $[[\_]]$  stabilisce una corrispondenza uno-uno tra  $\mathcal{P}$  e  $\mathcal{F}$ .
- 9 Facendo riferimento all'esercizio 1 sopra. Si commenti la funzione calcolata dal programma scritto, in particolare si dica: Quali sono i valori del numerabile  $\mathcal{D}$ ? Cosa sono le sequenze rispetto a tali valori? Cos'è in  $\mathcal{F}$  un ordinamento di tali sequenze?
- 10 In un esempio dei lucidi abbiamo usato il termine *stream* ad indicare una struttura dati in grado di fornire un'illimitata sequenza di valori. Come potrebbe essere definita una tale struttura in termini di  $\mathcal{F}$  e  $\mathcal{D}$ ?