↳ Computer Science

$(fun\ x \to t)\ s$ $(\lambda x.t)\ s$

Informatica divisa in due mondi:

- Teoria del calcolo (simbolico)
  - Nasce dalla logica
  - Modello algebrico $t \to s$

} Logica: Gödel, Turing, Church, Kleene } 1930-50

Universalità = Dati ≡ Programmi

- Teoria delle macchine calcolatrici
  - Macchine (fisiche e non) per eseguire calcoli
    - Basate su macchina di VN
    - Load-store model
    - Circuiti logici per calcolo (Shannon)

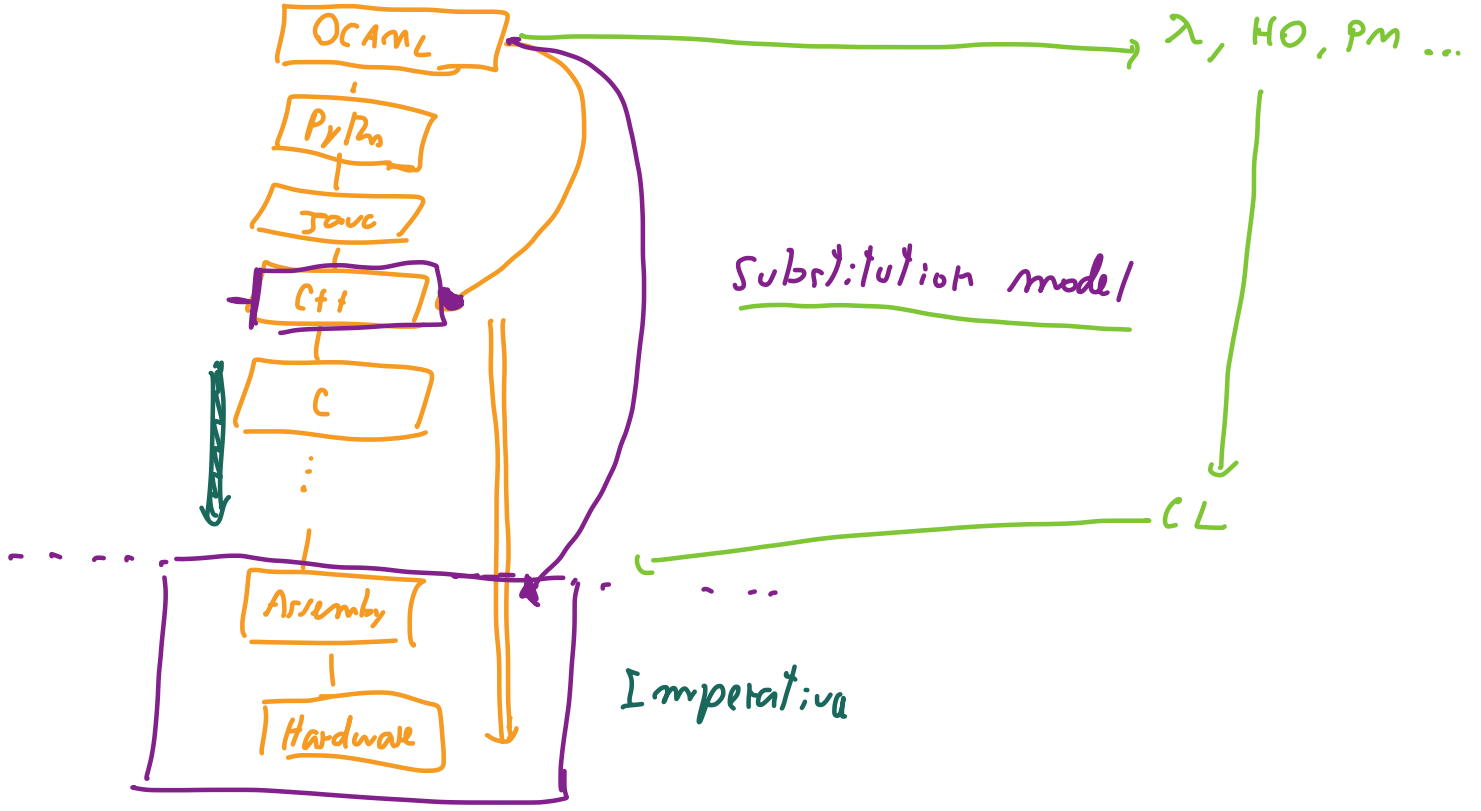
} memoria ⇓ Universalità

$$\frac{t \Rightarrow fun\ x \to t' \quad AND \quad t'[s/x] \Rightarrow v'}{t\ s \Rightarrow v'}$$

Espressioni $e, s, \ldots$

Computazione $e \to s$, $e \Rightarrow v$

} substitution model of computation



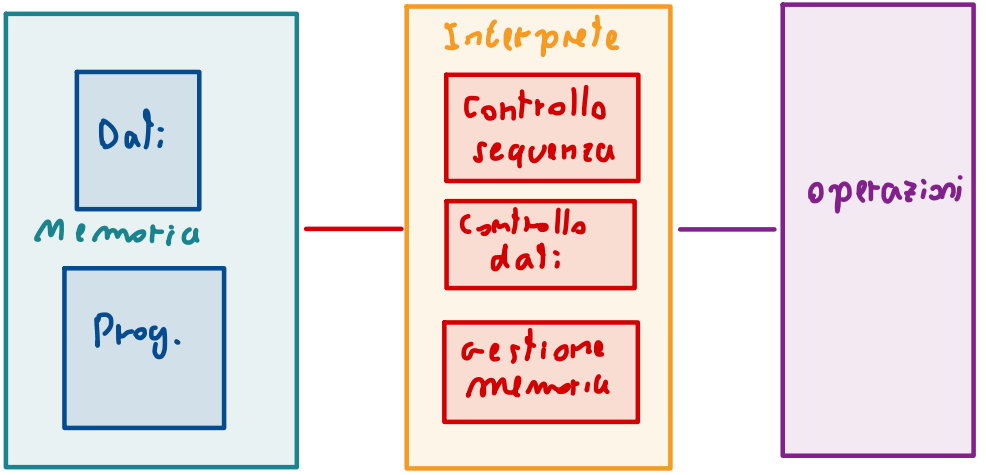
Linguaggio di: programmazione

- sintassi
- semantica
- pragmatica
- ⋮

. linguaggio comprensibile da macchina calcolatrice

Domanda. È la computazione simbolica eseguibile su una macchina?

Def. Una **macchina astratta** per un linguaggio L è un insieme di: strutture dati e algoritmi per eseguire programmi scritti in L



Una macchina M è una macchina per un linguaggio L, il suo **linguaggio macchina**

Implementare L = dare M_L (macchina per L)

Es. Macchina fisica H

La realizzazione di M può poi avvenire a differenti livelli

- Realizzazione in hardware (costruzione fisica)
- Simulazione

Osservazioni: Più espressivo è $\mathcal{L}$, più complessa sarà $\mathcal{M}_{\mathcal{L}}$

- La macchina determina la nozione di: esecuzione di un programma
- La macchina è imperativa $\Rightarrow$ possiamo definire il concetto di esecuzione Tramite il concetto di: stato della macchina

Def. Lo stato di una macchina $\mathcal{M}$ è una rappresentazione astratta di: tutta l'informazione di $\mathcal{M}$ in un certo istante necessaria all'esecuzione di un programma

$\rightsquigarrow$ Più espressivo è $\mathcal{M}_{\mathcal{L}}$, più complessa sarà la nozione di: stato

Ex. stato = memoria + indirizzo PC + ...

Come descrivere l'esecuzione di programmi in $\mathcal{L}$?

① Dare $\mathcal{M}_{\mathcal{L}}$ ⇝ ambigua (quali aspetti di $\mathcal{M}_{\mathcal{L}}$ sono rilevanti? tempo esecuzione?, spazio? ...]
non modulare, ...

② Identificare una nozione di **stato** sufficientemente ricca da descrivere l'esecuzione di programmi in $\mathcal{L}$

$$\langle \sigma, P \rangle \rightarrow \langle \sigma', P' \rangle$$

structural operational semantics (SOS)

NB. La descrizione della SOS dovrebbe essere sufficientemente precisa da specificare l'implementazione dell'interprete di $\mathcal{L}$ in una qualsiasi macchina $\mathcal{M}$ con stato adeguato

Linguaggio di: Programmazione

- sintassi
- semantica
- ⋮

↳ riferimento alla macchina

⇝ model-dependent ✓

(equivalenza estensionale tra macchine)

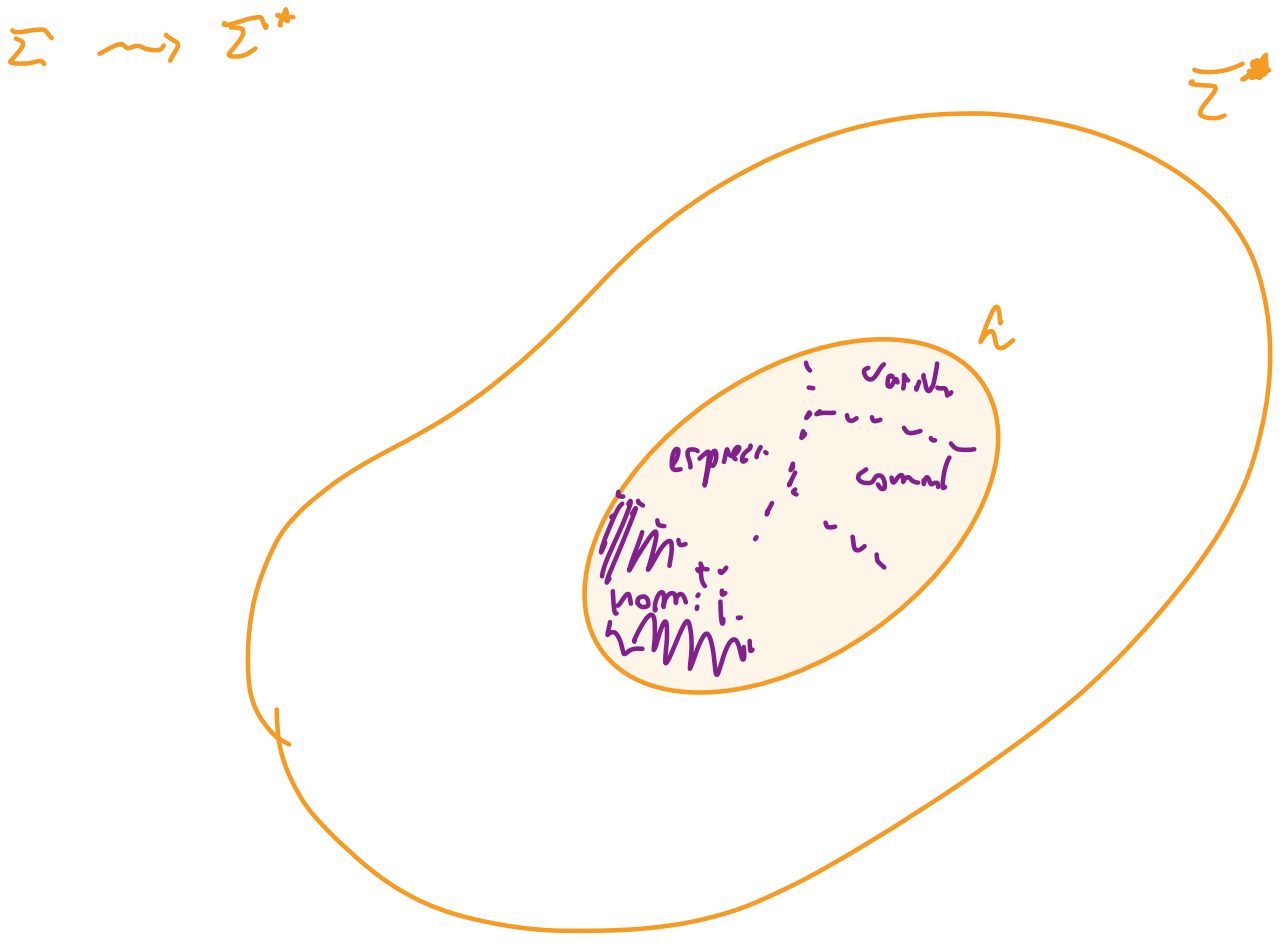
In generale

sintassi ⇝ grammatiche, ...

semantica ⇝ trasformazione stato

Per capire un linguaggio naturale, sintassi non è abbastanza

- nomi
- verbi
- aggettivi
- ⋮


Σ ⇝ Σ*
Σ*
A
variable
expression
command
nom : t

Un linguaggio di programmazione è un meccanismo di astrazione della macchina fisica

Meccanismo di base: nome

La macchina astratta consiste di vari "oggetti" (oggetti (m))

- variabili;
- indirizzi memoria
- operatori
- strutture controllo
- ⋮

} natura imperativa

frase/entità sintattica

Def. Un nome è una espressione in $\mathcal{L}$ (solitamente un identificatore) per rappresentare un oggetto

NB. L'oggetto non è necessariamente un oggetto di $m_{\mathcal{L}}$

Es Se m_L ha una operazione per la somma, l'identificatore + in L è un nome per questa

Es. - Preso un linguaggio imperativo L,

int x

dice:
x è un identificatore per una variabile (oggetto di: m_L)
(nome)

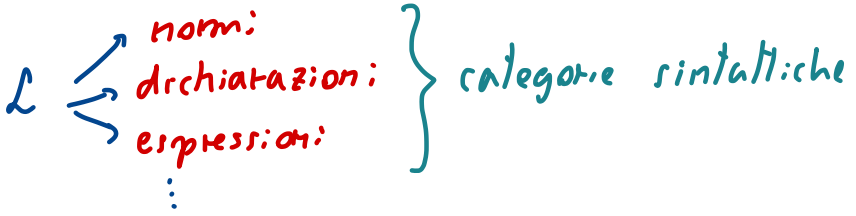
Bisognerebbe parlare di: variabile di: nome x

- in OCaml
def f = fun y → y } la funzione di: nome }

⟹ Abbiamo oggetti: quali: variabili e funzioni

I linguaggi con **nomi** offrono meccanismi di **astrazione** tramite nomi;
Chiamiamo questi meccanismi **dichiarazioni** (o **definizioni**)

Es. In Ocaml
let <identificatore> = <exp> è una **dichiarazione**



Domanda A quali oggetti un linguaggio può dare nome?

Def. Gli oggetti a cui un linguaggio può dare nome sono detti: **oggetti denotabili**

Domanda. Dato un linguaggio $\mathcal{L}$ con dichiarazioni, cosa serve (allo stato della) macchina astratta per eseguire dichiarazioni?

Def. L'ambiente è l'insieme delle associazioni:

(nome, oggetto denotabile)

esistente a run-time in uno specifico punto del programma

NB. L'ambiente non esiste nella macchina fisica. È una caratteristica del linguaggio di alto livello che deve essere simulato dall'implementazione

Se $\mathcal{L}$ ha dichiarazioni, verosimilmente la semantica op userà la nozione di: ambiente

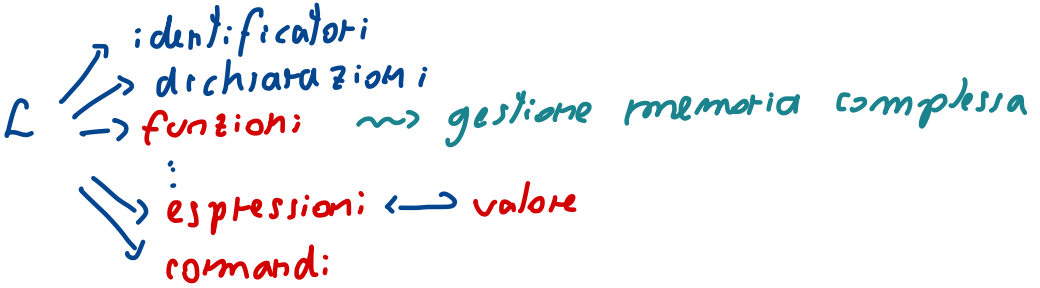
$$\langle \eta, P \rangle \rightarrow \langle \eta', P' \rangle$$

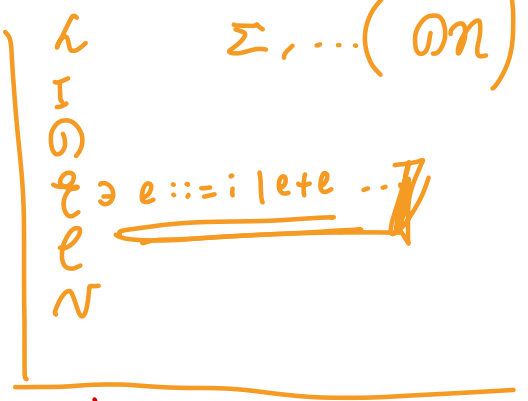
$\mathcal{M}_{\mathcal{L}}$ deve quindi: gestire ambiente e dichiarazioni:

- blocchi (let x = ⟨...⟩ in ⟨...⟩)
- scope (let x = ⟨...⟩ in ⟨...⟩)

⇝ Gestione di queste cose in $\mathcal{M}_{\mathcal{L}}$ non facile.

$$\langle \eta, \underbrace{\text{let } x = v;;}_{\text{dichiarazione}} \rangle \rightarrow \langle \eta[x \mapsto v], \cdot \rangle$$





Def. Una espressione è una entità sintattica la cui valutazione produce un valore oppure non termina

$\mathcal{M}_{\mathcal{L}}$

NB. Def. relativa alla semantica su macchina astratta

Def. Un comando è una entità sintattica la cui valutazione non necessariamente restituisce un valore, ma può avere un effetto collaterale

$\langle \sigma, p \rangle \to \langle \sigma', p' \rangle$

$\langle \sigma, e \rangle \to \langle \sigma, e' \rangle$

Ma cos'è allora un effetto collaterale?

$$\langle (\sigma_1 \dots \sigma_m), P \rangle \rightarrow \langle (\sigma_1', \dots, \sigma_m'), P' \rangle$$

componenti dello stato della macchina

Un effetto collaterale è relativo a una porzione di stato (e.g. memoria)

Diciamo che P ha effetto collaterale se

$$\langle (\sigma, \varepsilon), P \rangle \rightarrow \langle (\sigma', \varepsilon'), P' \rangle$$

e $\varepsilon \neq \varepsilon'$ (modifica stato)

Domanda. Quand'è che problematico avere effetti collaterali?

Risposta. Quando le parti interessate dello stato sono oggetti denotabili

Def. Una variabile è un riferimento (e.g. indirizzo) che può contenere valori

NB. Le variabili sono modificabili

Nei linguaggi imperativi le variabili sono oggetti denotabili
L'assignamento è un comando con effetti collaterali sulla porzione di stato della memoria che associa valori memorizzabili a locazioni

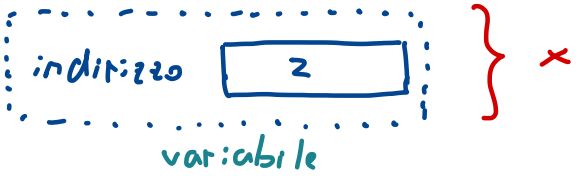
Tra i comandi, l'**assegnamento** è il principale

→ componente atomica per il cambiamento di **stato**

Cosa succede in un assegnamento

$$x := x + 1 \quad ?$$

① Stiamo lavorando con una **variabile** di **nome** x



② Una variabile è una entità composta

Nei linguaggi imperativi abbiamo una distinzione di valori.

l-valori (left-valori)

valori che indicano locazioni → espressioni che valutano ad l-valori possono stare a sinistra di un assegnamento

r-valori (right-valori)

valori che indicano contenuto delle locazioni

Per eseguire $e_1 := e_2$

1. Calcola lo l-valore di e_1, ottenendo una locazione loc
2. Calcola lo r-valore di e_2, ottenendo un valore v
3. Sostituisci il contenuto di loc con v

Quali espressioni di un linguaggio possano denotare l-valori, dipende dal linguaggio

Ex. variabili $(x := x+1)$

array $(a[i] = a[i]+1)$

⋮

Questa complessità porta facilmente a comportamenti "strani"

```
b := 0;
a[f(3)] := a[f(3)] + 1;
```

(a[f(3)] += 1)

```
int f (int n) {
  if b == 0 {
      b := 1;
      return 1;
     }
  else return 2;
```

Altri tipici comandi hanno lo scopo di controllare il flusso d'esecuzione

→ i veri passi computazionali sono fatti dagli assegnamenti

- if_Then_else
- jump, goto, ...
- while_do_
- for ...

Cos'è lo stato della macchina, in concreto?

Linguaggio con
- nomi
- dichiarazioni
- espressioni
- comandi: → variabili sono oggetti denotabili;

Se il linguaggio ha variabili, allora le locazioni dovranno essere denotabili:

stato = Ambiente × Memoria

Ambiente = Nomi → Valori denotabili:
"identificatori" "valori a cui si può dare un nome"

Memoria = Locazioni → Valori memorizzabili
"cosa può essere salvato in memoria"

stato = $\langle \rho, \sigma \rangle$

"Nello stato $\langle \rho, \sigma \rangle$ la variabile x ha valore 5"

→ ① Vediamo nell'ambiente ρ il valore denotabile del nome x

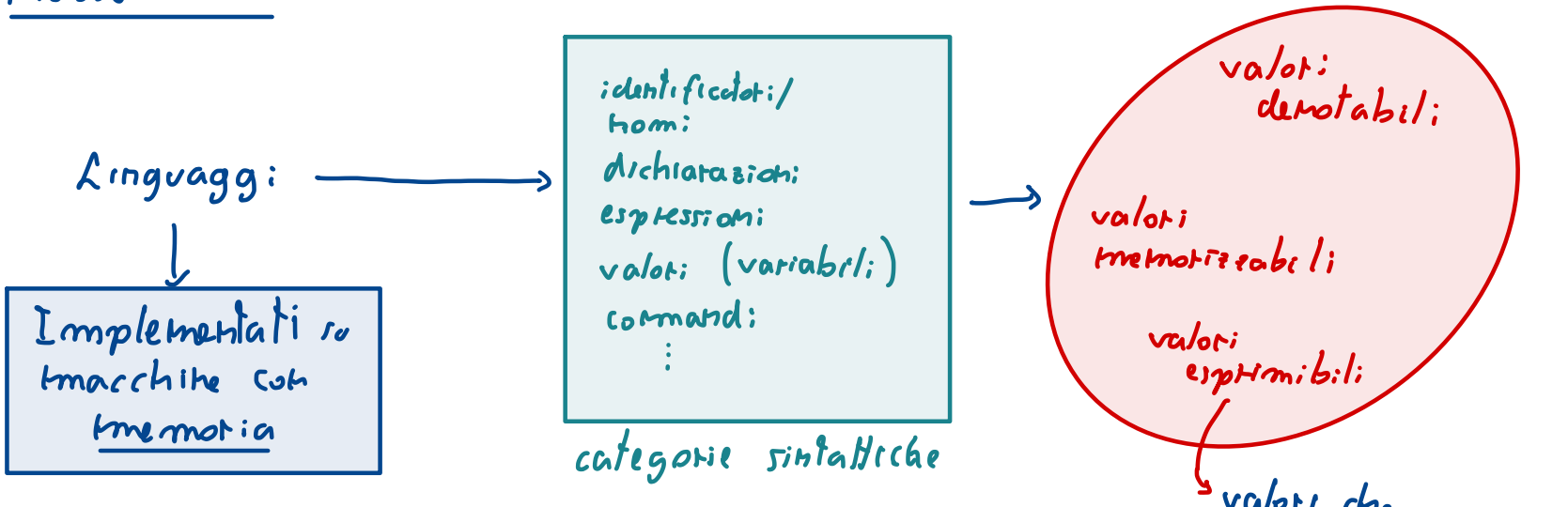
② Questo sarà una locazione $\rho(x)$

③ Vediamo in memoria σ il contenuto di $\rho(x)$

④ Abbiamo che $\sigma(\rho(x))$ è 5

" x := y " ⇝ Scriviamo in $\rho(x)$ il valore di $\sigma(\rho(y))$

Riassumendo



stato = <ambiente, memoria>

espressione	$\langle \rho, \sigma \rangle, e$	$\Downarrow$	$\langle \rho, \sigma, \rangle, v$	
dichiarazione	$\langle \rho, \sigma \rangle, d$	$\Downarrow$	$\langle \rho', \sigma \rangle$	
comando	$\langle \rho, \sigma \rangle, c$	$\Downarrow$	$\langle \rho, \sigma' \rangle$	} effetto collaterale

Specificare un linguaggio = definire questi aspetti

Paradigmi = scelte sistematiche di questi aspetti

Imperativo = locazioni denotabili
- assegnamento tra i comandi
- ⇒ memoria, effetti collaterali, ...

dichiarativo = semantica non necessita di memoria (solo ambiente)

⇒ programmi = dichiarazioni + espressioni

⇒ no variabili

⇒ no comandi

⋮

Es. paradigma funzionale = dichiarazione di espressioni / funzioni con substitution model of computation

paradigma logico