Necessità di Rappresentare Esecuzione di una funzione

f: INPUT → OUTPUT

Mi Serve Strumento per descrivere comE Calcolare f

ovvero Vogliamo Rappresentare ALGORITMI

↳ Sequenza finita di Passi discreti che descrivono comE Calcolare f

Mi dice La "STRATEGIA" per Calcolare La funzione

Mi Serve Strumento che Manipoli dati descrivendo ALGORITMI

PROGRAMMA

Linguaggi di Programmazione ⇒ Linguaggi formali che permett. di Scrivere il PROGRAMMA

Programmi Sono Oggetti finiti, Bene formati di un PL (Programming Language)

Quindi una Rappresentazione (finita) di una Sequenza potenzialmente Infinita di Effetti su una Macchina.

Computazione ∞ ⇒ Sistema Operativo ad Esempio che mi Viene offerta dal ciclo WHILE che dipende da delle

Condizioni per il Quale Terminerà o Continuerà a Ciclare

Linguaggi ce Ne Sono molti e sono Touring Calcolabili e quindi EQUIVALENTI che Sono più o Meno Efficienti in determinati Ambiti

SUDDIVISE PER APPLICAZIONI:

- Scientifiche ⇒ FORTRAN ⇒ Tanti Valori dopo la Virgola
- Economiche ⇒ COBOL ⇒ Con Approssimazioni ed è LEGACY CODE perché NON si Vuole Cambiare linguaggio ma si Aggiusta
- Intelligenza Artificiale ⇒ LISP e TUTTI LINGUAGGI FUNZIONALI
- Sistemi ⇒ C ⇒ Basso lvl ed Efficiente
- Web ⇒ JAVA/PHP/JAVASCRIPT

LIVELLO:

BASSO:

- Linguaggio Macchina
- Assembly

→ CHE HA PORTATO

ALTO: (mix. tra formalità e linguaggio Naturale)

- PARADIGMI
  - IMPERATIVO ⇒ Lavora Sulla CELLA DI MEMORIA come C
  - FUNZIONALE ⇒ ML, LISP.... si Basa sulla composizione di funzioni
  - LOGICO ⇒ PROLOG...
  - PROCEDURALE ⇒ è un IMPERATIVO ma SENZA salti a SUB-ROUTINE
  - OBJECT ORIENTED ⇒ JAVA
  - DICHIARATIVE ⇒ Logico/funzionale

Significato di Variabile è Molto diverso:

- IMPERATIVO ⇒ Astrazione della Cella, modifichiamo il Suo contenuto
- FUNZIONALE ⇒ Classica Variabile MATEMATICA, è un PLACEHOLDER dove tutte le occorrenze hanno Stesso Valore

GENERAZIONALI

- HW PROPRIETARI ⇒ Programma per l'architettura (BINARI)
- ASSEMBLY ⇒ Prima Astrazione dal BINARIO
- ALTO LIVELLO ⇒ Usano linguaggio Naturale ⇒ C, BASIC, JAVA
- DICHIARATIVI ⇒ SQL
- APPL. ORIENTED ⇒ Per Rispondere ad Esigenze Specifiche

CARATTERISTICHE:

- SEQUENZIALI
- CONCORRENTI/MULTI THREAD ⇒ c'è Sincronizzazione e quindi c'è Comunicazione
- Parallelo ⇒ Non Servono costrutti per far Comunicare
- Distribuito ⇒ Su più macchine di Calcolo
- Object Oriented ⇒
- Scripting ⇒

3 ASPETTI IMPORTANTI

1. LEGGIBILITÀ
2. WRITEABILITY
3. AFFIDABILITÀ

} Semplicità, #Limitato di Costrutti, poco e poca Molteplicità

→ modi ≠ di dire la stessa cosa

OVERLOADING → operatore ha più significati

ORTOGONALITÀ DEL LINGUAGGIO (Linguaggi TIPATI)

Astrazione (o.o) ed Espressività

Type Checking, a Tempo Statico di Compilazione Posso Eseguire dei CHECK ed Elimino gli ERRORI di TIPO

Gestione degli ECCEZIONI

ALIASING ⇒ Effetto Negativo per Avere Nomi ≠ per la Stessa Cella (LINGUAGGI con PUNTATORI)

IMPLEMENTARE UN LINGUAGGIO:

(PL) ↳ Programming Language

Costruire **INFRASTRUTTURA** da Installare sulle Macchine (Fisica) per far si che possa Eseguire il PL.

Bisogna Porre Attenzione a:

- **ARCHITETTURA DELLA MACCHINA**
  - CPU
  - MEMORIA

Ho dati e Programmi in MEMORIA (che è SEPARATA dalla CPU) ma anche Istruzioni e dati che viaggiano tra memoria e CPU

ESECUZIONE MACCHINA FISICA

- Program Counter
- Repeat forever
- fetch Instruction Nella MEM Usando PROGRAM COUNTER
- aggiorno Program Counter
- decode Instruction
- Execute Instruction

Implementare PL = Costruire Macchina Astratta per il mio PL

MACCHINA ASTRATTA ⇒ Alto Livello

MACCHINA FISICA ⇒ Programmi linguaggio Macchina

PL ⇒ Linguaggio L con i Suoi Costrutti Sintattici

⇓

Frase Ben formata Nel PL = Programma

Per Eseguirlo Costruiamo La Macchina Astratta per L

MACCHINA ASTRATTA:

Data L Linguaggio di Programmazione, M_L è Macchina Abs. di L è un Insieme di Strutture dati ed Algoritmi che Permettono di Eseguire Programmi Scritti in L.

Quindi ho Max. Velocità ma ZERO FLESSIBILITÀ... cambio Linguaggio (L) e devo Cambiare Macchina (M)

LINGUAGGIO MACCHINA:

Data M Macchina Abs. il Suo linguaggio L_M è il linguaggio macchina di M, ed è L'Insieme di Tutte Le Stringhe di Tutte le Stringhe Eseguibili (riconoscibili) da M.

TEOREMA DEL PADOINO ⇒ Posso Avere ∞ Macchine per l'Implementazione di 1 Linguaggio

Così Posso diventare Indipendente dalla Macchina fisica

ESEMPIO DI LIVELLI:

Implementare un PL ⇒ Realizzare M_L

IMPLEMENTAZIONE:

1. TRADUZIONE IMPLICITA ⇒ Interprete
2. TRADUZIONE ESPLICITA ⇒ Compilatore

Esistono anche degli IBRIDI, ma c'è una PREDOMINANZA di una delle 2 Caratteristiche

1. Simulo le Operazioni, traducendole via via che le Eseguo in linguaggio Sottostante.
2. Realizzo M_L Traducendo Interamente il Programma Scritto in L in un Programma Scritto in linguaggio Sottostante (PENSARE AL PROCESSO DI COMPILAZIONE IN JAVA)

Linguaggi Compilati spesso più EFFICIENTI visto che hanno TRADUZIONE OFFLINE visto che NON è a periodo di ESECUZIONE

Non È decidibile sapere se 2 Programmi Sono Equivalenti, è una Conseguenza del TH. DI RICE

NOTAZIONE:

$prog^L \Rightarrow$ Insieme dei Programmi Scritti in L

$D \Rightarrow$ Insieme dei dati

$P^L \Rightarrow$ Programma Specifico Scritto in L, $P^L : D \to D$

input $\in D$ $\quad P^L(Input) = output$

Esecuzione di P^L su un dato Input fornisce uno specifico OUTPUT

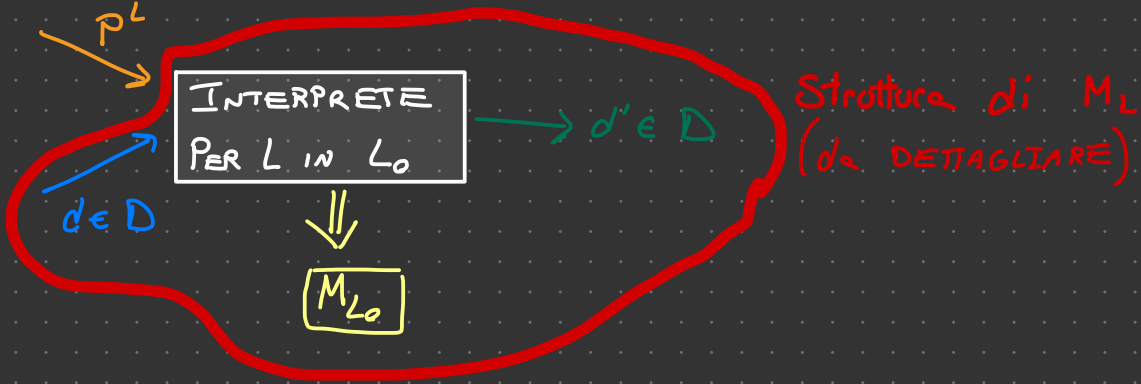
INTERPRETE:

Per il linguaggio L, scritto in L_0 (linguaggio Sottostante) è un programma $I^{L_0 L} : Prog^L \times D \to D$

INPUT ← $Prog^L \times D$ → OUTPUT

$\forall INPUT \in D.\ I^{L_0 L}(P^L, input) = OUTPUT \quad \forall P^L \in PROG^L$

che è una MACCHINA UNIVERSALE



Metodologie di Progettazione di Programmi