Realizzazione Macchina Astratta

Strada Interpretativa

Esecuzione Istantanea e viene Eseguita la **Traduzione Istruzione per Istruzione in L_0**

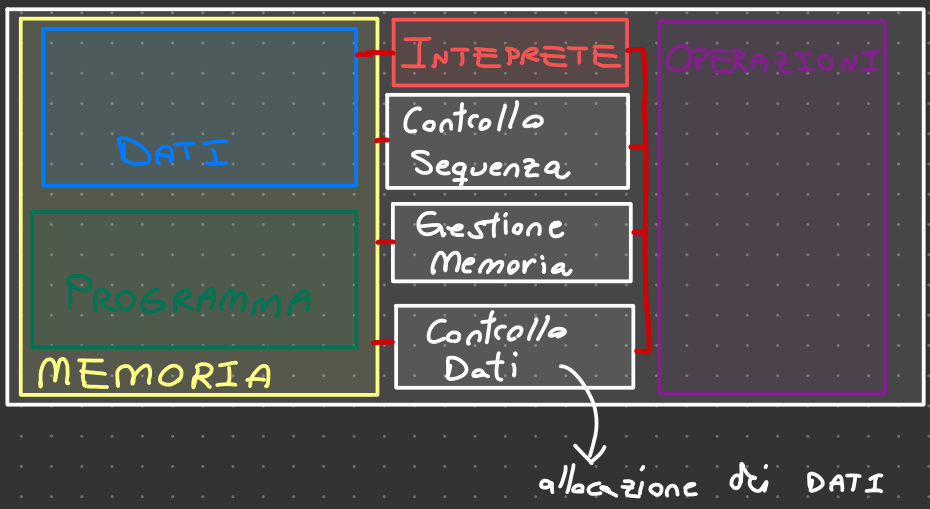
Strada Compilativa

Viene Tradotto Completamente L in L_0 (linguaggio Eseguibile) della Macchine Sottostante

Interprete da L in L_0:

- $I^{L_0}{}_L\,(P^L, Input) = Output$
- $I^{L_0}{}_L : Prog^L \times D \to D$

La Macchina Astratta M_{L_0}:



Interprete dall' Interno:

Simulazione SW di Cosa Succede Sulla Macchina:

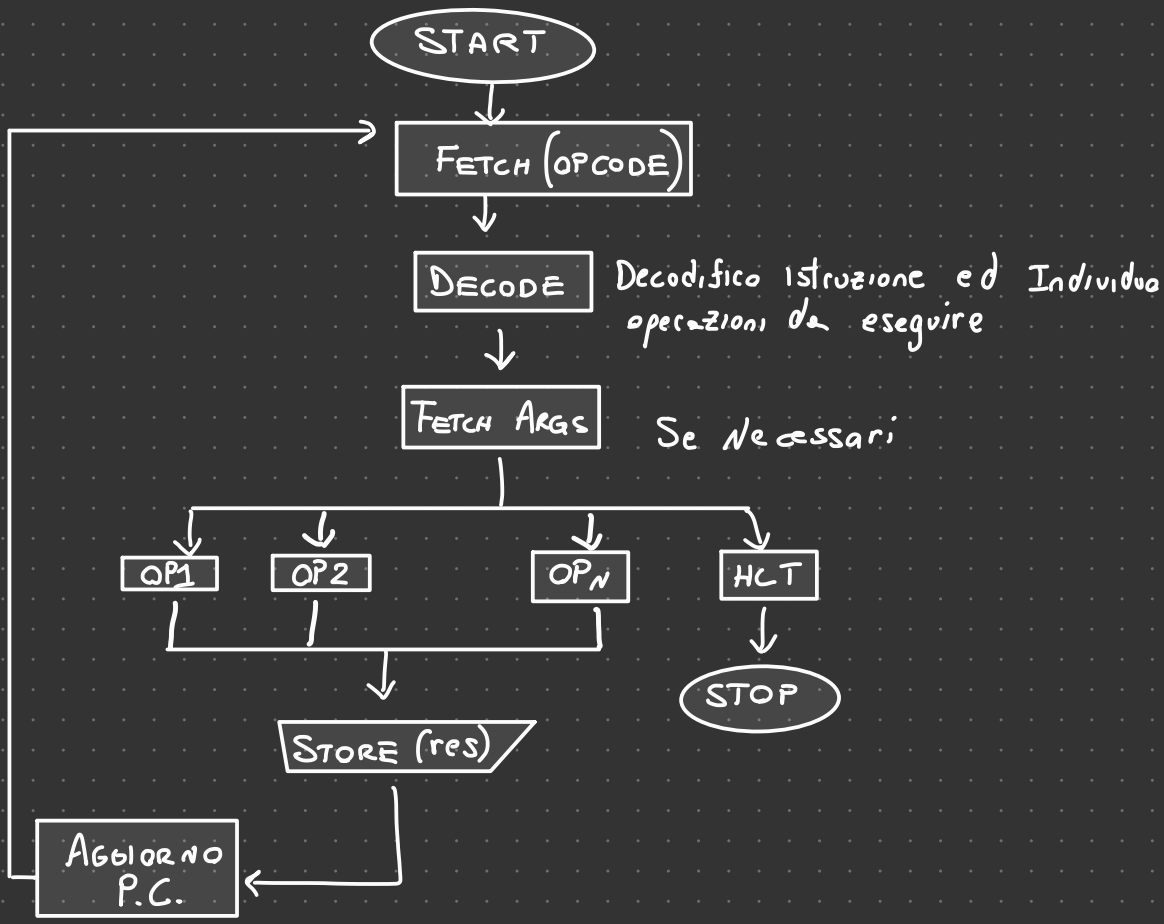
```
BEGIN
go:= TRUE
WHILE go DO:
    FETCH (OPCODE, OPINFO)  AT  PROG. COUNTER   //Estraggo dalla Memoria
    DECODE (OPCODE, OPINFO)
    IF OPCODE needs AGRS   //Devo Estrarli dalla MEMORIA
        THEN fetch (ARGS)
    CASE opcode off:
        op 1: EXECUTE (OP 1, args)
        :
        op N: EXECUTE (OP N, args)
        HLT: go:=false
    IF opcode Salva Results:
        STORE (results)
    P.C. = P.C.+ (SIZE (OPCODE))
END
```

Controllo Dati

Gestione Memoria

Controllo Sequenza

In Flow Chart:



Soluzione Compilativa:

Macchina Astratta che Realizziamo fa Lavoro Solo Sintattico ⇒ DEVE TRADURRE (e BASTA)

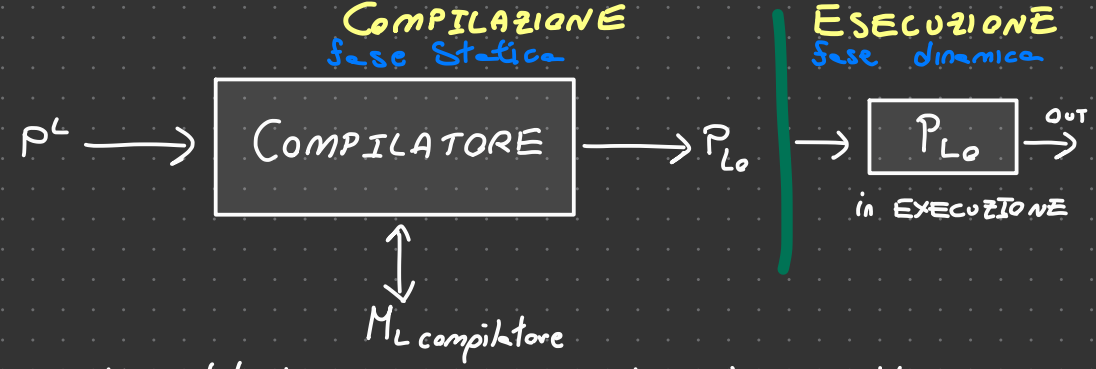
Compilatore ⇒ da L (SORGENTE) a L_o (TARGET) è un PROGRAMMA:

$C^{L L_o}: Prog^L \to Prog^{L_o}$ (non guarda i dati Lui)

$$C^{L L_o}(P^L) = P^{L_o} \quad t.c. \quad \forall d \in D \quad P^L(d) = P^{L_o}(d)$$

Equivalenza SEMANTICA dei linguaggi L ed L_o (ovviamente ≠ SINTATTICAMENTE)

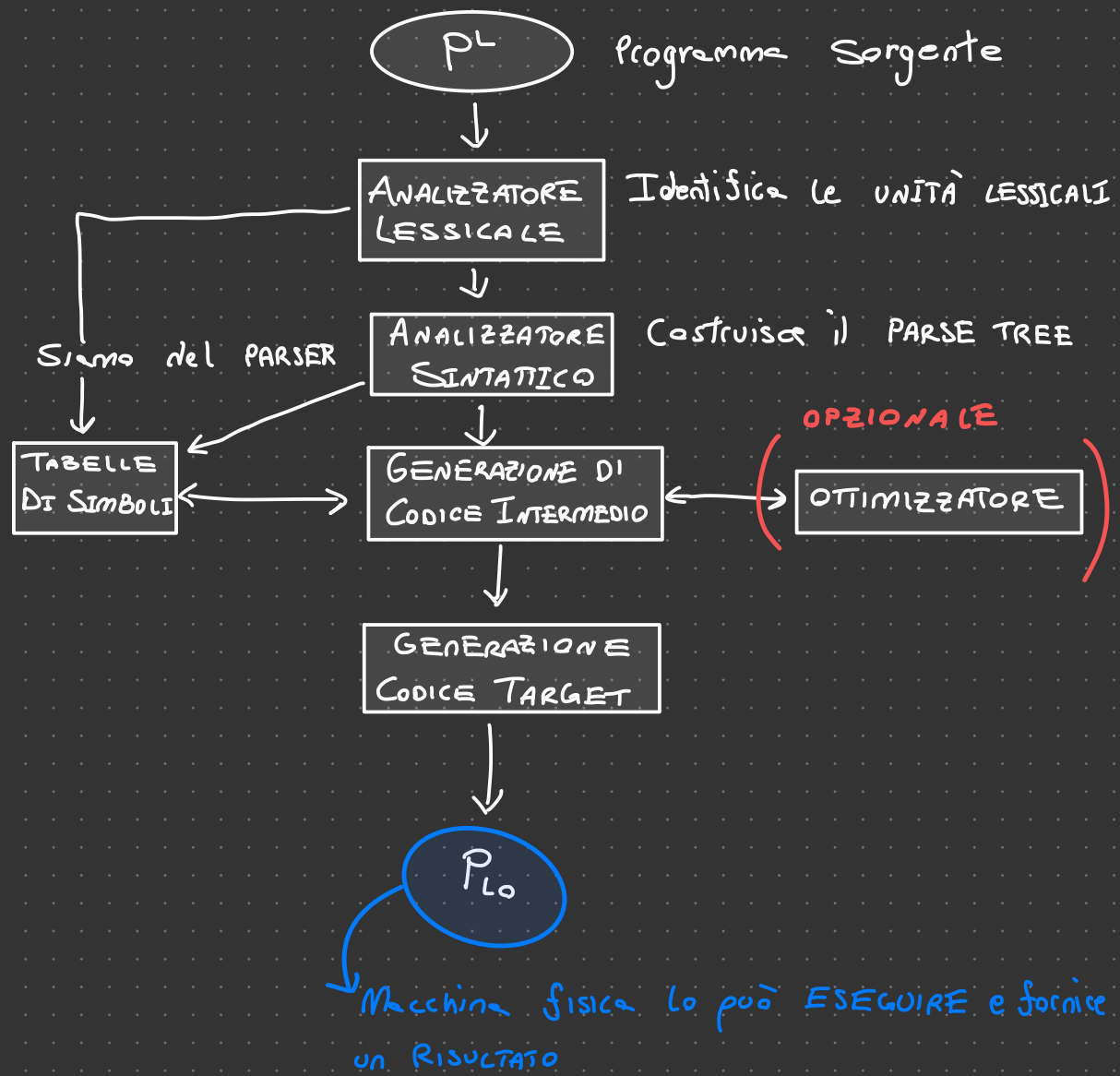
⇒ A INPUT comportamento È uguale



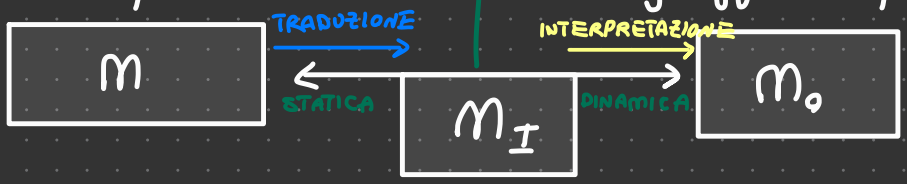
Macchina Abs. del linguaggio per Scrivere il Compilatore

Ci Sono ANALISI Statiche per OTTIMIZZARE il Codice in fase di TRADUZIONE per Avere poi un ESECUZIONE ancora più Veloce

Struttura:


PL
Programma Sorgente
Analizzatore Lessicale
Identifica le unità lessicali
Analizzatore Sintattico
Costruisce il parse tree
Siamo del parser
Tabelle di Simboli
Generazione di Codice Intermedio
opzionale
Ottimizzatore
Generazione Codice Target
PLo
Macchina fisica lo può eseguire e fornire un risultato

L da Implementare | Lo linguaggio Implementato



L_I → linguaggio Intermedio

M_L —COMPILAZIONE→ $M_I \equiv M_0$

$M_I = M_L$ —INTERPRETAZIONE→ M_0

PROIEZIONI DI FITAMURA:

Specializzatore ⇒ $SPEC_L : (prog^L \times Dato) \to prog^L$

$SPEC_L : (p^L, d) = p_d$ → porzione INPUT

$\forall y.\ P_L(y) = P_d^L(y)$ → restante PORZIONE

Usato ad Esempio per Teorema smn

Specializzare un Interprete su un programma da Eseguire restituisce un COMPILATORE

DESCRIVERE UN LINGUAGGIO DI PROGRAMMAZIONE:

SINTASSI ⇒ Regole di formazione/costruzioni frasi Ben formate descrive **RELAZIONI tra Simboli/Segni**

SEMANTICA ⇒ Attribuzione del **Significato ai Simboli**, data una frase che RISPETTA le REGOLE SINTATTICHE

PRAGMATICA ⇒ Sono **REGOLE di BUON UTILIZZO** (ingegneria del SOFTWARE) che NON sono OBBLIGATORIE ⇒ come programmare BENE

IMPLEMENTAZIONE ⇒ Come si Eseguono frasi Ben formate che hanno Significato

SINTASSI:

Consiste nel descrivere le Regole di formazione del Linguaggio

PAROLA ⇒ Stringa di Caratteri

FRASE ⇒ Sequenza di Parole Ben formata (RISPETTA REGOLE SINTASSI)

LINGUAGGIO ⇒ Insieme delle frasi

LESSEMA ⇒ Identifica una Parola con Significato specifico (SIMBOLI TERMINALI DELLA GRAMMATICA).
È UNITÀ SINTATTICA del linguaggio

TOKEN ⇒ È una **Categoria** Sintattica (SIMBOLI NON TERM.

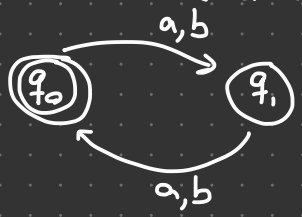
NELLE GRAMMATICHE)

GENERATORI → Grammatiche

RICONOSCITORI → automi lo sono

$L = \{\delta \mid |\delta| \text{ è Pari}\}$ posso Scrivere sia automa Riconoscitore

con $\Sigma = \{a, b\}$



O una Grammatica Generatrice:

$$G: S \to \varepsilon \mid aaS \mid abS \mid bbS \mid baS$$

Descrivo la Grammatica

- ALFABETO ⇒ Simboli
- LESSICO ⇒ Sequenze di Simboli che Costruiscono Parole ovvero i SIMBOLI TERMINALI
- REGOLE SINTATTICHE ⇒ Regole di derivazione della Grammatica che Compongono Tra loro Parole.
- CATEGORIE SINTATTICHE ⇒ Simboli NON TERMINALI della Grammatica

Si usano Grammatiche C.F. anche se NON cattuano TUTTO, come

una chiamata a Procedura.

Catturano UNA PARTE delle Regole ma NON Tutto.

Si usano le Grammatiche C.F. per gli Algoritmi EFficienti che hanno
↳ anche dette BNF

Apertura e chiusura Parentesi è la CAUSA che ci ha fatto Abbandonare gli AUTOMI altrimenti Sarebbero stati ancora più EFFICIENTI

GRAMMATICHE:

È una Quintupla $G = \langle V, T, P, S \rangle$

V insieme finito di Simboli NON Terminali (CATEGORIE SINTATT.)
quindi espressioni, Comandi, Procedure, dichiarazioni, ...
Sono gli Elementi della frase (TOKEN

T Sono i SIMBOLI TERMINALI che hanno già un SIGNIFICATO visto che hanno definizione Nel nostro Vocabolario (LESSEMI) come ad Esempio ⇒ WHILE, IF, :=, {, }, ...

P Produzioni $A \to \alpha$ dove $A \in V$ e $\alpha \in (V \cup T)^*$, Regole di formazione del Programma ad Esempio:
com → .. WHILE exp DO com ...

S ∈ V Simbolo Iniziale è la Categoria dei Programmi

AD ESEMPIO:

Exp $\langle \{E\}, \{or, and, NOT, (,), 0, 1\}, P, E \}$

$E \to 0 \mid 1 \mid (E \text{ or } E) \mid (E \text{ or } E) \mid (\text{NOT } E)$

definisco i Casi Base

Come Combino le Espressioni ⇒ PASSO INDUTTIVO