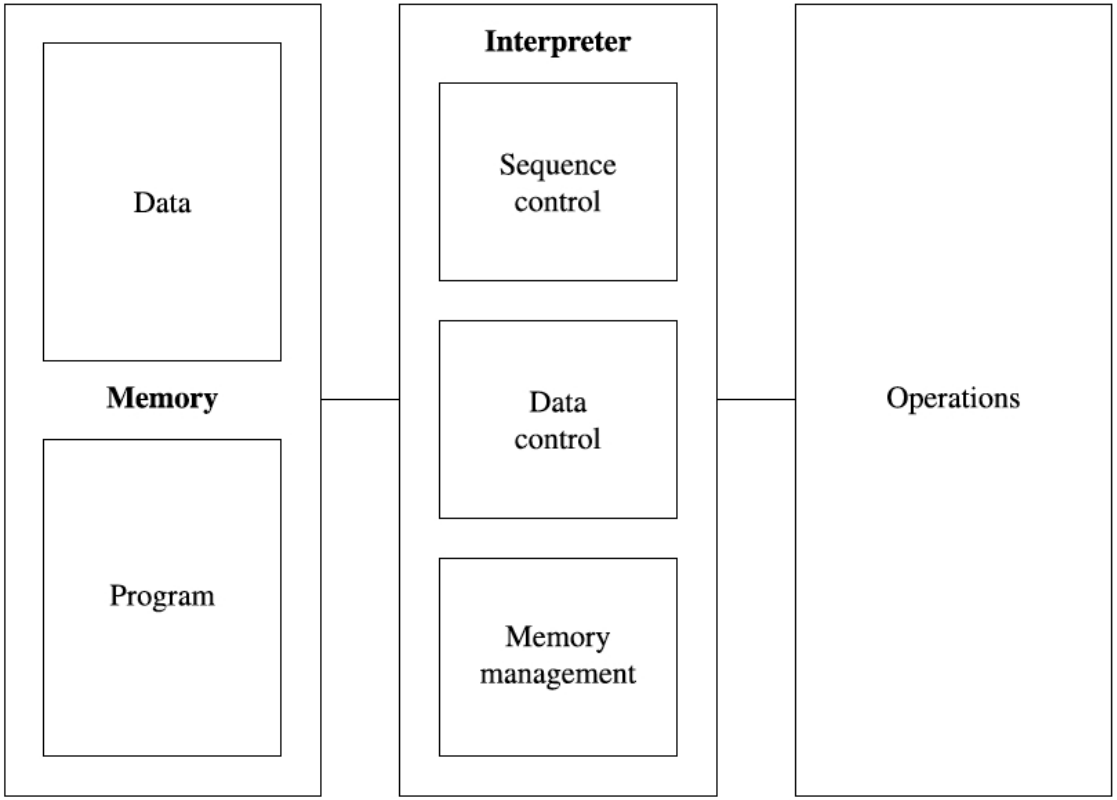
Macchina Astratta: struttura e realizzazione.

Sommario

- Macchina Astratta e l'interprete di Macchina
- Hight e Low Level Languages
- Implementazione di un Linguaggio
- Macchina Intermedia
- Gerarchia di Macchine
- Trasformazioni di Programmi

Struttura di una Abstract Machine /1



- Una AM è l'esecutore di un Linguaggio Eseguibile (inclusi i L. di Programmazione)
- AM di un Linguaggio di Programmazione ha la struttura in figura ed è un'astrazione per un computer
- Tre Componenti principali: **Memoria**, **Interprete** della macchina, **Operazioni**, in comunicazione
- Discussione: Cosa non ci aspetteremmo di trovare se fosse una calcolatrice invece di un computer ?
- Discussione: Analisi dei componenti

Struttura di una Abstract Machine /2

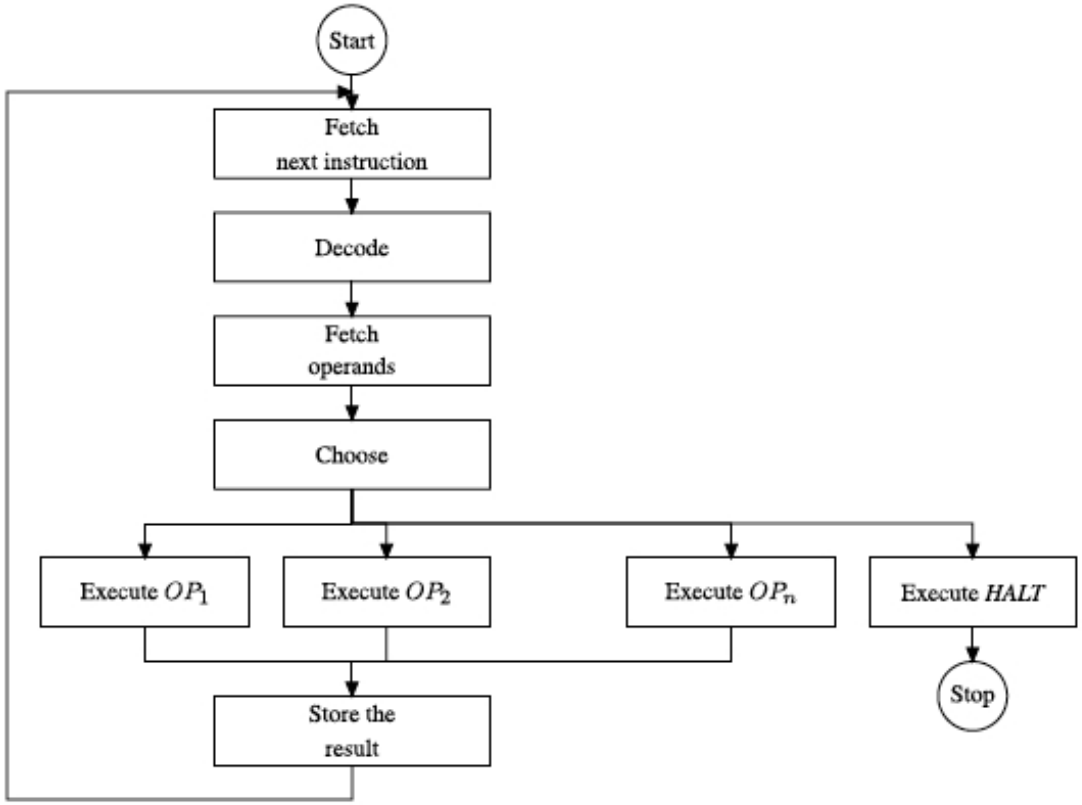
Definition (Macchina Astratta)

Una macchina astratta per un linguaggio $\mathcal{L}$, denotata con $\mathcal{M}_{\mathcal{L}}$ è un qualsiasi insieme di strutture dati e di algoritmi che permettono di memorizzare ed eseguire i programmi di $\mathcal{L}$

Definition (Linguaggio Macchina)

Il linguaggio macchina è il linguaggio $\mathcal{L}$ di una macchina $\mathcal{M}_{\mathcal{L}}$

Ciclo di Esecuzione generico di un Interprete di Macchina



- Discussione: Questa struttura potrebbe essere il nucleo di un chip di un pocessore
- Discussione: Commentiamola su istruzioni di 2AC (2 address code language machine): ADD R5 R0
  - Fetch coinvolge *controllo di sequenza*, *controllo di memoria* e *memoria* (sezione programma),
  - Decode coinvolge *controllo dati* e secondo le modalit di indirizzamento degli operandi ....
  - Fetch operandi coinvolge ... e consiste in ....

Realizzazione di una Macchina Astratta /1

- Quando vogliamo usare un LP $\mathcal{L}$ dobbiamo realizzare il suo esecutore, ovvero $\mathcal{M}_{\mathcal{L}}$
- Implementare $\mathcal{L}$ significa realizzare $\mathcal{M}_{\mathcal{L}}$
- Discussione: Che relazione intercorre tra realizzare $\mathcal{M}_{\mathcal{L}}$ e la funzione $\mathcal{U}_{\mathcal{L}}$, introdotta in slide ...?
- 3 realizzazione principali:
  - **Hardware**: Costruiamo una macchina ad hoc: Una struttura a K-chip che realizza tutti i componenti (RAM, CASHE, CPU e interfaccia BUS)
  - **Software**: Utilizziamo una macchina esistente $\mathcal{M}_{\mathcal{L}_O}$ e scriviamo *opportuni programmi* in $\mathcal{L}_O$ implementare $\mathcal{U}_{\mathcal{L}}$.
  - **Firmware**: Adattiamo ad hoc una macchina esistente molto simile (utilizzando *microcodice* caricato su ROM)

Realizzazione di una Macchina Astratta /2

Definition (Low-Level Programming Language)

È un LP con basso livello espressivo dove la struttura dei programmi e dei costrutti utilizzati è *condizionata dalla realizzazione* della sua AM.

- La costruzione della AM di un Linguaggio Low-Level nasce contestualmente alla definizione del linguaggio
- La sua costruzione non presenta problemi
- La somiglianza della AM con la struttura in 3 componenti e con lo schema di Interprete di macchina, visti nelle slides, è quasi totale.

Definition (Hight-Level Programming Language)

È un LP con alto livello espressivo dove la struttura dei programmi e dei costrutti utilizzati è *condizionata dalle metodologie di programmazione* supportate piuttosto che dalla realizzazione della sua AM.

Realizzazione di una Macchina Astratta /3

Definition (Hight Level Programming Language)

È un LP con alto livello espressivo dove la struttura dei programmi e dei costrutti utilizzati è *condizionata dalle metodologie di programmazione* supportate piuttosto che dalla realizzazione della sua AM.

Per LP ad alto livello solo la soluzione Software è praticabile.

Questa soluzione avviene in due forme principali. Sia $\mathcal{M}_{\mathcal{L}}$ la macchina da realizzare, sia $\mathcal{M}_{\mathcal{L}_O}$ la macchina da utilizzare.

- **Interpretativa Pura**:
  Costruzione di un *Interprete* (di linguaggio) $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_O}$ di $\mathcal{L}$ in $\mathcal{L}_O$.
- **Compilativa Pura**:
  Costruzione di un *Compilatore* che trasforma programmi $\mathcal{L}$ in programmi equivalenti scritti in $\mathcal{L}_O$.

Realizzazione di una Macchina Astratta /4

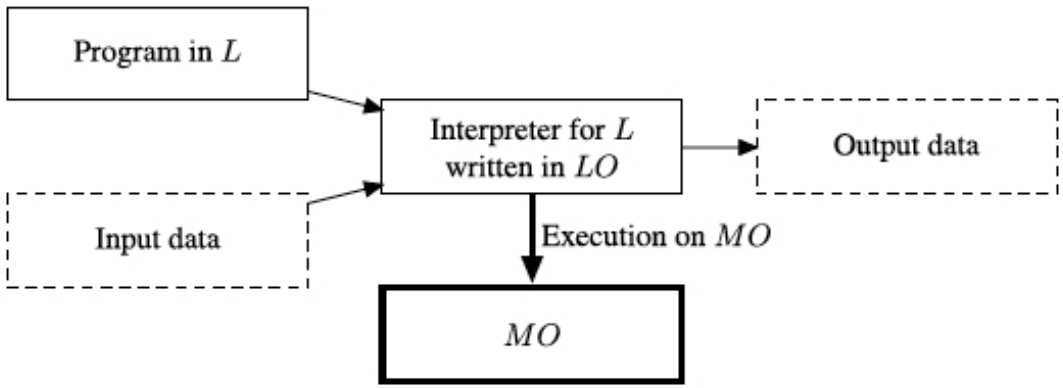
- Sia $\mathcal{M}_{\mathcal{L}}$ la macchina da realizzare, sia $\mathcal{M}_{\mathcal{L}_O}$ la macchina da utilizzare.
- Sia $\mathcal{P}_{\mathcal{L}}$ l'insieme dei programmi di $\mathcal{L}$, sia $\mathcal{P}_{\mathcal{L}_O}$ quello di $\mathcal{L}_O$
- Siano $\mathcal{U}_{\mathcal{L}}$ e $\mathcal{U}_{\mathcal{L}_O}$ le funzioni universali per $\mathcal{L}$ e $\mathcal{L}_O$

Definition (Interprete)

Un interprete per $\mathcal{L}$ in $\mathcal{L}_O$, denotato $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_O}$, è un programma di $\mathcal{L}_O$, i.e. $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_O} \in \mathcal{P}_{\mathcal{L}_O}$:
$\forall p \in \mathcal{P}_{\mathcal{L}}, \forall x \in \mathcal{D}$,

$$\mathcal{U}_{\mathcal{L}_0}(\mathcal{I}_{\mathcal{L}}^{\mathcal{L}_0})(<\mathtt{p},\mathtt{x}>) = \mathcal{U}_{\mathcal{L}}(\overline{\mathtt{p}})(\mathtt{x})$$

dove: $<\_,\_>: (\mathcal{P}_{\mathcal{L}} \times \mathcal{D}) \rightarrow \mathcal{D}$ è iniettiva
$\mathcal{D}$ il dominio dei valori delle funzioni calcolabili $\mathcal{F} \equiv \mathcal{D} \rightarrow \mathcal{D}$



Realizzazione di una Macchina Astratta /5

- Sia $\mathcal{M}_{\mathcal{L}}$ la macchina da realizzare, sia $\mathcal{M}_{\mathcal{L}_O}$ la macchina da utilizzare.
- Sia $\mathcal{P}_{\mathcal{L}}$ l'insieme dei programmi di $\mathcal{L}$, sia $\mathcal{P}_{\mathcal{L}_O}$ quello di $\mathcal{L}_O$
- Siano $\mathcal{U}_{\mathcal{L}}$ e $\mathcal{U}_{\mathcal{L}_O}$ le funzioni universali per $\mathcal{L}$ e $\mathcal{L}_O$

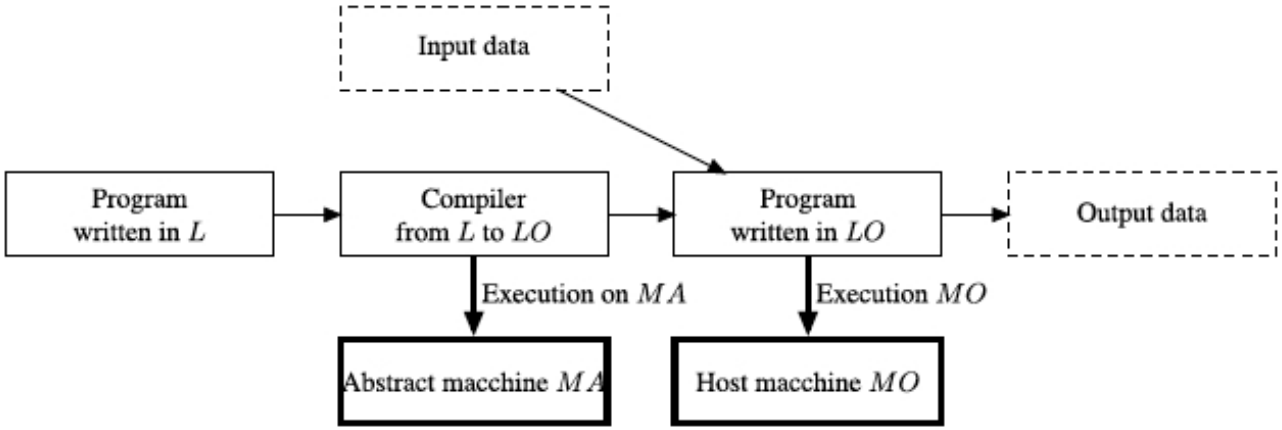
Definition (Compilatore)

Un compilatore da $\mathcal{L}$ in $\mathcal{L}_O$, denotato $\mathcal{C}_{\mathcal{L},\mathcal{L}_O}$, è un programma di un qualche linguaggio $\mathcal{L}_A$, i.e. $\mathcal{C}_{\mathcal{L},\mathcal{L}_O} \in \mathcal{P}_{\mathcal{L}_A}$:

$$\forall p \in \mathcal{P}_{\mathcal{L}}, \text{ sia } \mathcal{U}_{\mathcal{L}_A}(\overline{\mathcal{C}_{\mathcal{L},\mathcal{L}_0}})(<\mathtt{p}>) = \mathtt{q} \in \mathcal{L}_0$$
$$\text{in } \mathcal{U}_{\mathcal{L}_0}(\overline{\mathtt{q}})(\mathtt{x}) = \mathcal{U}_{\mathcal{L}}(\overline{\mathtt{p}})(\mathtt{x}) \quad (\forall \mathtt{x} \in \mathcal{D})$$

dove: $<\_>: \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{D}$ è iniettiva

$\mathcal{D}$ il dominio dei valori delle funzioni calcolabili $\mathcal{F} \equiv \mathcal{D} \rightarrow \mathcal{D}$



Compilatore: Quando usare una Host Machine $\mathcal{M}_{\mathcal{L}_A}$

- I compilatori di un PC non usano Host Machine: $\mathcal{C}_{\mathcal{L},\mathcal{L}_O} \in \mathcal{P}_{\mathcal{L}_O}$
- Quando usare una Host Machine:
  - La macchina $\mathcal{M}_{\mathcal{L}_0}$ non ha risorse adeguate per $\mathcal{C}_{\mathcal{L},\mathcal{L}_O}$
    - Bare Machines, macchine per Servizi, Palmari, ...
  - Si vuole proteggere il codice sorgente dell'applicazione $\mathtt{p} \in \mathcal{P}_{\mathcal{L}}$:
    - Rilasciamo solo il codice oggetto della traduzione di $\mathtt{p}$.
    - $\mathtt{p}$ è un *app* proprietaria e/o non deve essere modificabile
    - Un motivo per usare compilazione (vs. interpretazione)
  - **Bare Machine**: Sistema Operativo è istallato su una "Bare Machine" dotata del solo linguaggio macchina (non assembler) e del suo esecutore.
    - Il linguaggio C (il suo compilatore) e il sistema operativo UNIX, scritto in C

Confronto tra i due approcci

- **Efficienza, costo in tempo.** Compilatore$^+$
  - Compilatore: La decodifica è fatta una volta per tutte le esecuzioni
  - Interprete: a) La decodifica è ri-fatta ad ogni esecuzioni
    b) Uno statement può essere decodificato più volte durante una stessa esecuzione

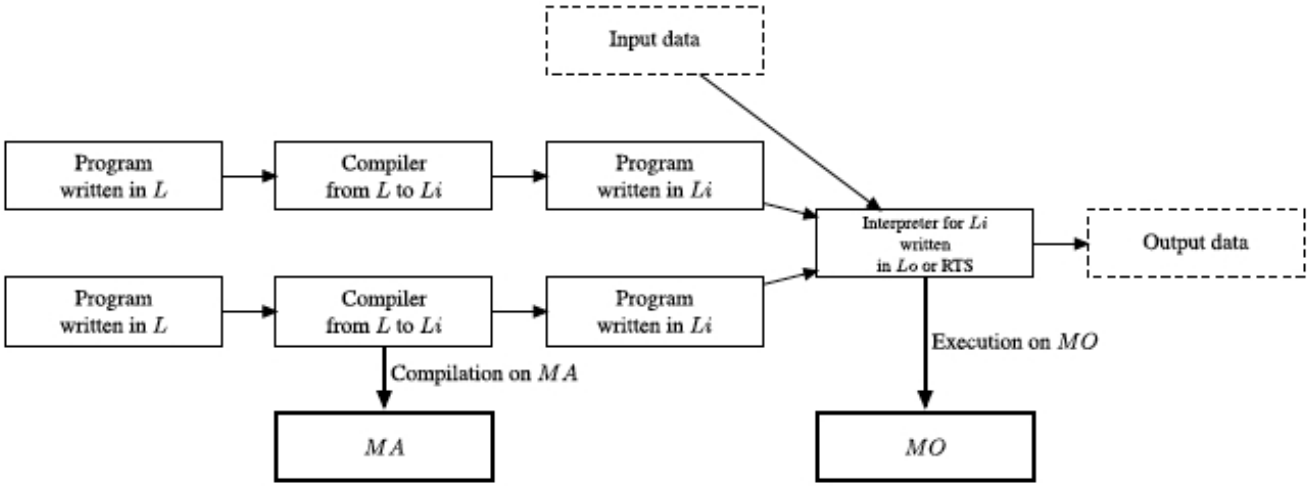
```
P1: for (I = 1, I<=n, I=I+1) C;

P2:
   R1 = 1
   R2 = n
L1: if R1 > R2 then goto L2
   translation of C
   ...
   R1 = R1 + 1
   goto L1
L2: ...
```

- **Semplicità di costruzione.** Interprete $\simeq$ Compilatore
  - Compilatore: Deve utilizzare $\mathcal{U}_{\mathcal{L}}$, $\mathcal{U}_{\mathcal{L}_O}$
  - Interprete: Deve esprimere $\mathcal{U}_{\mathcal{L}}$ con un programma di $\mathcal{L}_O$
- **Occupazione memoria.** Interprete$^+$
  - codice oggetto lineare con dimensione sorgente fattore ... (fig. sopra)
- **Flessibilità e Integrazione con strumenti di esecuzione.** Interprete$^+$
- **Sviluppo**: Interprete$^+$
- **Uso, a sviluppo completato**: Compilatore$^+$

Realizzazione di una Macchina Astratta: Uso di Macchina Intermedia /6

- Uso di una AM intermedia, $\mathcal{M}_{\mathcal{L}_i}$
  - Compiliamo da $\mathcal{L}$ in $\mathcal{L}_i$
  - Interpretiamo $\mathcal{L}_i$ in $\mathcal{L}_O$



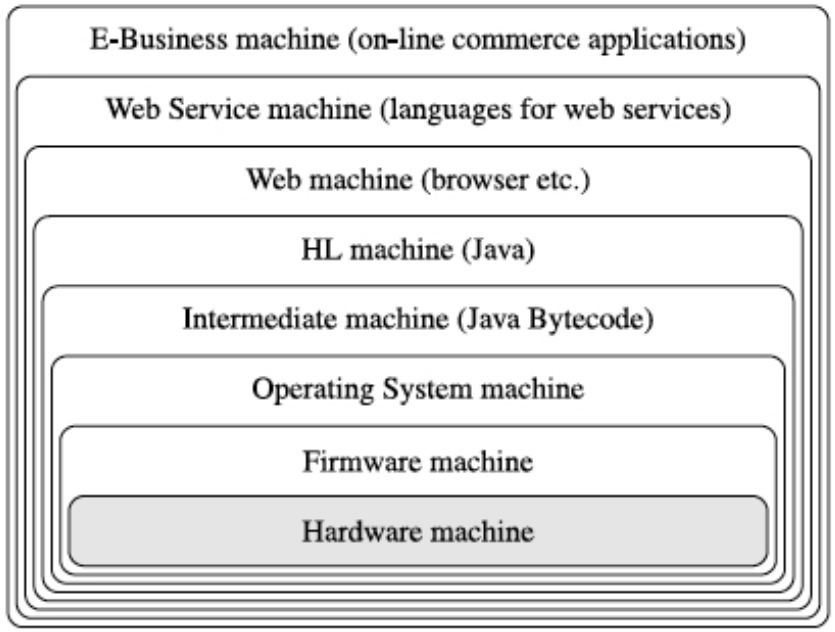
- Interpretativa pura: $\mathcal{L} = \mathcal{L}_i$
- Compilativa pura: $\mathcal{L}_i = \mathcal{L}_O$
- Mista: $\mathcal{L} \neq \mathcal{L}_i \neq \mathcal{L}_O$
  - Interpretativa: $\mathcal{L} > \mathcal{L}_i >> \mathcal{L}_O$ [1] (Java e Java Virtual Machine, 1995)
  - Compilativa: $\mathcal{L} >> \mathcal{L}_i > \mathcal{L}_O$ (Pascal e P-code Machine, 1975)

[1] $>>$ significa molto distante (per livello e/o paradigma), $>$ significa poco distante

Una gerarchia di Macchine Astratte

- Un computer contiene una gerarchia di AM
- Non tutte sono AM di LP
- Oppure, ogni AM estende la AM inferiore con funzionalità per nuovi "servizi"

Fig. 1.8 A hierarchy of abstract machines



Compilatore: Utilizzare le universali nella costruzione

Costruzione di Compilatore:

- Può utilizzare i mappings $\mathcal{U}_{\mathcal{L}}$, $\mathcal{U}_{\mathcal{L}_O}$
- Utilizziamo i mappings nelle frecce in modo da creare un diagramma che commuta definendo $\mathcal{C}_{\mathcal{L},\mathcal{L}_O}$

Compilatore e Interprete: Front-end e Run Time Support /1

- Spesso la realizzazione di un Linguaggio risulta in un kit di strumenti che includono sia C. che I.
- La costruzione di C. e di I. hanno alcune parti che possono essere esattamente le stesse:
  - Front-End
  - RTS
- **Front-End.** Parte iniziale (50 %) del codice di C. e di I.:
  - Analizza il programma e ne genera *Abstract Tree*, `AT`
  - Consiste in A. lessicale, A. sintattica e A. statica e generazione di `AT`
  - `AT` è utilizzato da C. e I. come rappresentazione interna del programma.
- **RTS.**

Compilatore e Interprete: Front-end e Run Time Support /2

- La costruzione di C. e di I. hanno alcune parti che possono essere esattamente le stesse
  - Front-End
  - RTS

- **RTS.** Collezione di librerie che definisce strutture e codice in linguaggio oggetto, $\mathcal{L}_0$, che implementano meccanismi del linguaggio sorgente, $\mathcal{L}$,
  - Sono meccanismi utilizzati dai programmi in modo identico
  - Organizzazione della memoria, Strutture per il controllo dell'esecuzione dei programmi di $\mathcal{L}$.

Esercizi 1.6 - ex1

- Si forniscano 3 esempi, in contesti diversi, di macchine astratte

- Risposta
  1. bare machine (i.e. macchina cruda priva di S. Operativo)
  2. macchina di S.O. (ottenuta dopo l'istallazione di S.O.)
  3. macchina Server di posta elettronica (ed.es. Exchange)