Domande e Risposte orale modulo 1 - capitolo 1

Ndr: le domande `evidenziate` sono quelle da sapere per passare l'esame (citando il prof: "se non sapete rispondervi non presentatevi nemmeno"), le altre domande non sono così determinanti ma saperle non fa male.

Capitolo 1 - Macchine astratte, interpreti, compilatori

1. `Che cosa è una macchina astratta? In cosa si differenzia da una macchina fisica?`

   È un'astrazione del concetto di calcolatore fisico. Una macchina fisica funziona esclusivamente per eseguire il proprio linguaggio. Una macchina fisica corrisponde ad un unico linguaggio.

   > Definizione di macchina astratta:
   >
   > Supponiamo che sia dato un linguaggio di programmazione L, definiamo una macchina astratta per L - indicandola con M_L - un qualsiasi insieme di strutture dati e algoritmi che permettano di memorizzare ed eseguire programmi scritti in L. Una generica macchina astratta M_L è composta da una **memoria** (divisa in memoria dati e memoria programma) e da un **interprete**.

2. `Che cos'è un interprete? In cosa consiste il ciclo fetch-decode-execute?`

   L'interprete è il componente essenziale di una macchina astratta. È il componente che **esegue il ciclo fetch-decode-execute** e interpreta le istruzioni. Esso è costituito da:

   - Operazioni per l'**elaborazione dei dati primitivi** (sulla macchina fisica la ALU);
   - Operazioni e strutture dati per il **controllo della sequenza** di esecuzione delle operazioni (su m.f. il PC);
   - Operazioni e strutture dati per il **controllo del trasferimento dei dati** (su m.f. gestione dei metodi di indirizzamento);
   - Operazioni e strutture dati per la **gestione della memoria** (su m.f. indirizzamento e trasferimento dei blocchi); La struttura di un interprete è la stessa per qualsiasi macchina astratta, ciò che cambia sono i componenti.

   Il ciclo fetch-decode-execute è alla base del funzionamento dei calcolatori basati sulla macchina di Von Neumann, che lo eseguono continuamente. Esso consiste in 3 fasi: nella fase di **fetch** il calcolatore carica dalla memoria le istruzioni del programma; nella fase **decode** il calcolatore decodifica e identifica il tipo di istruzioni da eseguire; infine nella fase **execute** vengono eseguite le istruzioni ricevute.

3. *Cos'è il linguaggio macchina?*

   Def. di **Linguaggio Macchina**: Data una macchina astratta M_L, il linguaggio L "compreso" dall'interprete di M_L è detto linguaggio macchina di M_L.

4. *Possono esistere macchine diverse con lo stesso linguaggio macchina?*

   Sì, poiché ad una macchina corrisponde un unico linguaggio, ma un linguaggio può essere eseguito da più macchine.

5. *In quali modi è possibile implementare una macchina astratta? Elenca vantaggi e svantaggi delle varie tecniche.*

   Una macchina astratta può essere implementata in tre modi:

   - **Hardware**: è sempre possibile realizzare M_L mediante hardware, implementando tutti i costrutti fisicamente (con memorie, porte logiche, bus, ecc).

     PRO: è l'approccio che garantisce le prestazioni migliori;

     CONTRO: una macchina simile è immutabile (una volta realizzata è impossibile da modificare), inoltre è via via più complessa man mano che il livello del linguaggio L è più alto.

   - **Emulazione firmware**: È la via di mezzo fra le tre implementazioni. Gli algoritmi e le strutture dati vengono simulati mediante **microprogrammi** che hanno le stesse possibilità dei programmi scritti in alto livello (vedi simulazione software) ma sono scritti in linguaggi di basso livello, mantenendo così un buon livello di prestazioni. Inoltre i microprogrammi risiedono in memorie di sola lettura (le ROM) per garantire un'alta velocità.

     PRO: prestazioni migliori, best of both worlds.

     CONTRO: flessibilità comunque ridotta in quanto le memorie ROM sono difficili da modificare (va fatto in laboratorio con raggi UV e cazzi vari, uno sbatti assurdo).

   - **Simulazione software**: le strutture dati e gli algoritmi vengono implementati mediante un linguaggio L' (che diamo per già implementato mediante una macchina $M'_{L'}$) di più basso livello rispetto al linguaggio L da implementare.

     PRO: massima flessibilità, possiamo cambiare in ogni momento le implementazioni dei costrutti.

     CONTRO: prestazioni peggiori, dal momento che dobbiamo passare da più livelli di interpretazione.

6. `Che cos'è un compilatore?`

Un compilatore da L a L_0 (che indichiamo con C_{L,L_0} è un programma che realizza una funzione $C_{L,L_0} : Prog^L \leftrightarrow Prog^{L_0}$ tale che dato in input un programma scritto nel linguaggio L (linguaggio sorgente), produce un programma compilato scritto nel linguaggio L_0 (linguaggio oggetto) che potremo eseguire sulla macchina $M_{0_{L_0}}$.

7. `Descrivere la tecnica d'implementazione pura e quella compilativa pura.`

   L'approccio **interpretativo puro** consiste nel realizzare un programma, l'interprete, scritto nel linguaggio L_0 della macchina in cui dobbiamo eseguire il codice, che legge le istruzioni nel nostro codice scritto in linguaggio L e le esegue.

   I **pro** di un approccio interpretativo puro sono:

   - Non bisogna aspettare che il programma venga compilato;
   - È molto flessibile, è facile creare strumenti che interagiscano col programma a runtime ed è facile fare debugging;
   - Più semplice da realizzare rispetto ad un compiler;
   - Occupa meno memoria.

   I **contro** sono:

   - L'esecuzione è più lenta per via della decodifica in tempo reale;
   - La decodifica deve essere eseguita ogni volta. Tipico esempio: **Java**.

   L'approccio **compilativo puro** consiste nel tradurre un programma scritto nel linguaggio L (linguaggio sorgente) in un programmascritto nel linguaggio L_0 (linguaggio oggetto). La traduzione è affidata al compiler, indicato con C_{L,L_0}.

   I **pro** sono:

   - Approccio efficiente, il programma viene decodificato una volta sola e ogni esecuzione è più rapida;

   I **contro** sono:

   - È più difficile da implementare;
   - Poco flessibile (ogni modifica richiede la ricompilazione);
   - Perdita di informazioni sulla struttura del programma, quindi difficile debugging a runtime; Tipico linguaggio compilato: **C**.

8. `Quando un interprete si può dire corretto? Quando un compilatore si può dire corretto?`

   Sia un interprete che un compilatore si dicono corretti quando rispettano la semantica del linguaggio da interpretare/compilare.

9. *Confrontare l'implementazione di una macchina astratta su una macchina ospite per mezzo di un interprete o di un compilatore.*

   - Implementazione interpretativa

     Il principale svantaggio è la scarsa efficienza. Infatti ai tempi di esecuzione del programma bisogna sommare i tempi necessari alla decodifica del codice sorgente. L'interprete non genera codice: il codice prodotto della traduzione non viene prodotto dall'interprete ma descrive solamente le operazioni che questo deve effettuare.

     Gli svantaggi in termini di efficienza sono bilanciati dai vantaggi in termini di flessibilità, per esempio per poter modificare a run-time il funzionamento del programma

   - Implementazione compilativa

     La traduzione di un programma avviene separatamente rispetto alla sua esecuzione. Trascurando il tempo necessario alla compilazione il programma oggetto eseguirà più velocemente della sua versione interpretata. Inoltre ogni istruzione viene tradotta solamente una volta, indipendentemente dal numero di occorrenze all'interno del programma. I principali svantaggi risiedono nella perdita di informazioni riguardo alla struttura del programma sorgente, utili in fase di debug.

10. `Che cos'è la macchina intermedia? (Come vengono implementate nella realtà le macchine astratte?)`

    Una macchina intermedia viene usata per implementare una macchina astratta: fra la macchina M_L del linguaggio che vogliamo implementare e la macchina ospite $M_{0_{L_0}}$ esiste un livello caratterizzato da un proprio linguaggio L_i e la sua relativa macchina astratta $M_{i_{L_i}}$, che sono rispettivamente il linguaggio intermedio e la macchina intermedia.

11. `Quando si dice che una implementazione è di tipo interpretativo e quando di tipo compilativo?`

    Un'implementazione si dice di tipo **interpretativo** nel caso in cui la macchina intermedia sia effettivamente presente e l'interprete di questa sia diverso dall'interprete di $M_{0_{L_0}}$ (ovvero l'interprete della macchina fisica). Esempi: **LISP, ML, Perl, Postscript, Pascal, Prolog, Smalltalk, Java.**

    Un'implementazione è di tipo **compilativo** se la macchina intermedia è più vicina alla macchina ospite e ne condivide l'interprete. Esempi: **C, C++, FORTRAN, Pascal, ADA.** (Si Pascal c'è in entrambi, non è un errore).

12. *L'interprete e il compilatore si possono sempre realizzare?*

    L'esistenza dell'interprete e del compilatore è garantita a patto che il linguaggio L_0 che usiamo per l'implementazione sia sufficientemente espressivo

rispetto al linguaggio L che vogliamo implementare.

Praticamente questo accade sempre perché i linguaggi che usiamo (quelli di uso comune) sono tutti turing-completi.

13. *Cos'è l'implementazione via kernel?*

    L'implementazione via kernel è uno dei due modi per implementare un compilatore (l'altro è l'implementazione via bootstrapping). Per implementare L via kernel devo individuare al suo interno l'insieme minimale di primitive, tale insieme lo chiamo H, ed implemento il compilatore in H; implemento poi a mano un interprete o un compilatore per L.

    È il tipico approccio usato per realizzare i sistemi operativi: viene prima implementato il kernel e poi partendo da questo si implementa il resto del SO.

    In questo modo si semplifica l'implementazione di L e lo si rende facilmente portatile, dal momento che basta reimplementare di volta in volta solo il kernel H nel nuovo linguaggio macchina.

14. *Quando si parla di bootstrapping?*

    L'implementazione di tipo bootstrapping è quella usata ad esempio per il linguaggio pascal.

    In origine pascal era fornito di:

    - Un compilatore in Pascal, da Pascal a P-code: $C^{Pascal}_{Pascal,P-code}$;
    - Lo stesso compilatore, tradotto in P-code: $C^{P-code}_{Pascal,P-code}$;
    - Un interprete per P-code, scritto in Pascal: I^{Pascal}_{P-code};

    Per poter implementare il linguaggio su una specifica macchina M_0 si produce a mano una traduzione dell'interprete I^{Pascal}_{P-code} nel linguaggio M_0 ottenendo $I^{L_0}_{P-code}$.

    A questo punto è già possibile eseguire su M_0 un programma P in Pascal, ma per migliorare l'efficienza realizziamo a mano un compilatore scritto in M_0:

    - Produco C^{Pascal}_{Pascal,L_0}

    E adesso, con tutti gli strumenti realizzati, applico il bootstrapping:

    - $I^{L0}_{P-code}(C^{P-code}_{Pascal,P-code}, C^{Pascal}_{Pascal,L0}) = C^{P-code}_{Pascal,L0}$
    - $I^{L0}_{P-code}(C^{P-code}_{Pascal,L0}, C^{Pascal}_{Pascal,L0}) = C^{L0}_{Pascal,L0}$

    Ovvero do in input all'interprete realizzato a mano i due compilatori (quello tradotto in P-code di prima e quello scritto a mano poco fa) e ottendo un compilatore scritto in P-code da Pascal a L_0. Infine riutilizzo lo stesso compilatore dandogli questa volta in input il compilatore appena prodotto

e quello che avevamo scritto a mano prima, ottendo in output il compilatore finale da Pascal a L_0 e scritto in L_0 per l'appunto.