Traduzione ed Interpretazione

Queste sconosciute

Siano

- L Linguaggio ad alto livello
- M_L Macchina astratta di L
- M_0 Macchina ospite

Implementazione interpretativa di L

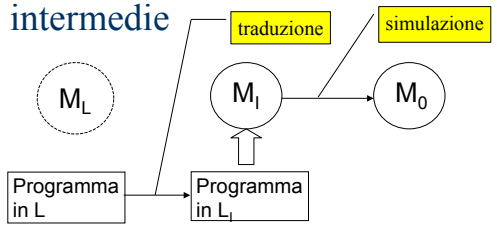
- Simulazione software di M_L su M_0 con interprete di M_L simulato
- Se M_L ed M_0 sono abbastanza diverse (linguaggio ad alto livello vs. hardware) non si riesce ad ottenere una simulazione di M_L per estensione di M_0 (con interprete di M_L = interprete di M_0)
- Efficienza scarsa, soprattutto per “colpa dell’interprete”

Implementazione compilativa di L

- Traduzione dei programmi di L in programmi macchina di L_{M0} funzionalmente **equivalenti**
- Esecuzione su M_0 del programma tradotto
- M_L non viene implementata

- Sono due casi limite che nella realtà non esistono quasi mai
- In una situazione reale c’è quasi sempre una soluzione mista

Traduzione e macchine intermedie



Traduzione in un **linguaggio intermedio** + simulazione della **macchina intermedia**

Pura interpretazione: $M_I = M_L$

Pura traduzione: $M_I = M_0$

Traduzione e macchine intermedie

- La traduzione (compilazione) pura è possibile solo quando la distanza tra M_L ed M_0 è molto limitata
- Es: L = linguaggio assembly di M_0
- In tutti gli altri casi c'è sempre una macchina intermedia che estende come minimo M_0 in alcune componenti

Traduzione intermedia in FORTRAN

- FORTRAN in realtà è un linguaggio di livello molto basso e per questo quasi tutto viene trattato in modo compilativo
- Ma alcuni costrutti richiedono la simulazione: input/output
- Una operazione I/O FORTRAN → svariate centinaia di operazioni nel linguaggio della macchina ospite

Traduzione intermedia in FORTRAN

- Il rimpiazzamento testuale genererebbe programmi di dimensione troppo elevata
- Si definiscono dei sottoprogrammi nel linguaggio di M_0 e nel codice tradotto (dal programma FORTRAN al linguaggio di M_0) si inserisce una chiamata a questi sottoprogrammi
- Tali sottoprogrammi sono chiamati **supporto a tempo di esecuzione**

Macchine intermedie e supporto a tempo di esecuzione

- Macchina intermedia = supporto a tempo di esecuzione + macchina ospite
- Si identifica valutando
  - Efficienza e complessità del traduttore
  - Compattezza del codice prodotto
- Facilità la **portabilità** su diverse macchine ospiti:
  - Bisogna reimplementare il supporto a tempo di esecuzione sulla nuova macchina ospite

Interpretazione e traduzione

- Non esistono implementazioni puramente compilative
- Non esistono implementazioni puramente interpretative
  - C'è sempre almeno la traduzione dei programmi nella loro rappresentazione interna

Interpretazione e traduzione

- La distinzione tradizionale si basa sulla struttura della macchina intermedia
- **Compilazione:** se l'interprete di M_I è l'interprete di M_0
- **Interpretazione:** se l'interprete di M_I è simulato (è un programma nel linguaggio di M_0)
- La distinzione si riferisce alle implementazioni e non ai linguaggi
- E' importante sapere anche il livello di simulazione delle altre componenti di M_I

Vantaggi della traduzione

- Efficienza di esecuzione del codice
- Il costo complessivo della decodifica di un'istruzione per determinare la sequenza di operazioni ed i relativi operandi è in gran parte a carico della fase di traduzione
- La decodifica avviene una sola volta indipendentemente da quante volte l'istruzione viene eseguita

Svantaggi della traduzione

- La dimensione del codice prodotto è tanto elevata quanto maggiore è la distanza tra M_L ed M_0
- Conviene usare la traduzione per quei costrutti di L che hanno un corrispettivo quasi immediato sul linguaggio di M_0
- Conviene usare l'interpretazione in tutti gli altri casi
- Durante il debugging dei programmi si può interagire in maniera più naturale con la macchina a tempo di esecuzione se si ha un'implementazione interpretativa

Programmi macchina rilocabili

- Implementazione puramente compilativa: caricatore o collegatore-caricatore
- Prodotto della traduzione:
  - Codice macchina assoluto in cui sono state risolte le modalità di indirizzamento che dipendono dall'allocazione dei programmi

Linguaggio assembly

- Implementazione compilativa: assemblatore
- Prodotto della traduzione:
  - Le istruzioni in linguaggio assembly, che contengono simboli mnemonici (che denotano codici operativi, indirizzi e registri), sono tradotte nelle corrispondenti istruzioni in linguaggio macchina

FORTRAN

- Compilatore
- Il traduttore produce codice direttamente eseguibile dall'interprete della macchina ospite
- Supporto a tempo di esecuzione:
  - Sottoprogrammi che simulano l'I/O
  - Alcune operazioni aritmetiche non fornite dalla macchina ospite

FORTRAN

- Progettato per mantenere al minimo il supporto a tempo di esecuzione
  - Possibile solo imponendo severe restrizioni che diminuiscono la flessibilità del linguaggio, in particolare per applicazioni di tipo numerico
  - Niente strutture dati dinamiche
  - Niente ricorsione
  - Niente gestione dinamica della memoria

Pascal e C

- Compilatore
- Il traduttore produce codice direttamente eseguibile dall'interprete della macchina ospite
- Se la macchina ospite è una macchina a registri, il supporto a tempo di esecuzione può essere molto complesso

M_{PASCAL} M_C Supporto a tempo di esecuzione

- E' necessaria una estesa simulazione (basata su strutture dati apposite, come le pile) per trattare molte delle componenti di M_{PASCAL} e M_C relativi a:
  - Controllo sequenza (es. Ricorsione)
  - Controllo dati (es. blocchi, sottoprogrammi con ambiente locale dinamico)
  - Gestione della memoria (es. puntatori, heap)
- Sono proprio questi costrutti che rendono C e PASCAL molto più flessibili, potenti ed eleganti di FORTRAN

Implementazione interpretativa del PASCAL

- Il traduttore genera codice nel linguaggio di una macchina intermedia, P_code, con struttura a pila, componenti simili a quelle di M_{PASCAL} e con interprete proprio diverso da quello della macchina ospite
- P_code è simulata (o emulata) sulla macchina ospite
- La differenza con la soluzione compilativa è essenzialmente nell'interprete
- E' una soluzione più portabile

LISP e PROLOG

- Implementazione interpretativa
- Il traduttore realizza semplicemente la trasformazione dei programmi nella loro rappresentazione interna
- La macchina intermedia è sostanzialmente M_{LISP} o M_{PROLOG}

LISP e PROLOG

- L'interprete è simulato
- Tutte le componenti del linguaggio richiedono un'estesa simulazione
- M_{LISP} ed M_{PROLOG} sono profondamente diverse dalle macchine su cui sono normalmente implementate tutte le componenti (incluse le operazioni sui dati primitivi)

Altre implementazioni di LISP e PROLOG

- Meno interpretative (impropriamente chiamate compilatori)
- Soluzioni simili a quella Pascal con P_code
- Macchine intermedie simili a M_{LISP} o M_{PROLOG} , ma simulabili più efficientemente
- La macchina intermedia PROLOG è uno standard: WAM (Warren Abstract Machine)

WAM

- Ha istruzioni di livello più basso di quelle di PROLOG per gestire i costrutti principali di PROLOG (unificazione e backtracking)
- Può essere realizzata in firmware (PROLOG machine)
- Facilita la portabilità
- Si ottiene comunque un miglioramento notevole nelle prestazioni

Java

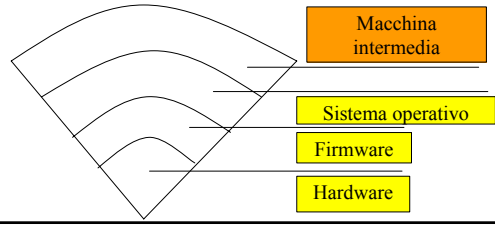
- Implementazione interpretativa
- La macchina intermedia è una macchina a pila che esegue istruzioni tipo assembly (ma che sa gestire la gerarchia delle classi)
- Java Virtual Machine (JVM)
- E' uno standard
- Simulabile su molte architetture (macchine ospiti) tramite il programma emulatore che si ottiene dal sito sun o da altri implementatori

Java

- Il compilatore produce, da un programma scritto in Java, un programma equivalente in bytecode (il linguaggio della JVM)
- La soluzione interpretativa facilita la portabilità
- Efficienza non proprio soddisfacente

Gerarchia di macchine astratte

- La macchina ospite su cui si implementa un linguaggio non è quasi mai la macchina hardware, ma una gerarchia di livelli di astrazione della macchina hardware



Livelli

- Firmware: nuove operazioni emulate con microprogrammi
- Sistema operativo
  - Nuove e più potenti operazioni primitive (ad es. gestione files, memoria virtuale)
  - Maschera alcune componenti delle macchine sottostanti
- Molto spesso la macchina ospite su cui si implementa un linguaggio L_1 è la macchina intermedia di un diverso linguaggio ad alto livello L_2
- Si utilizza L_2 per implementare il supporto a tempo di esecuzione di L_1

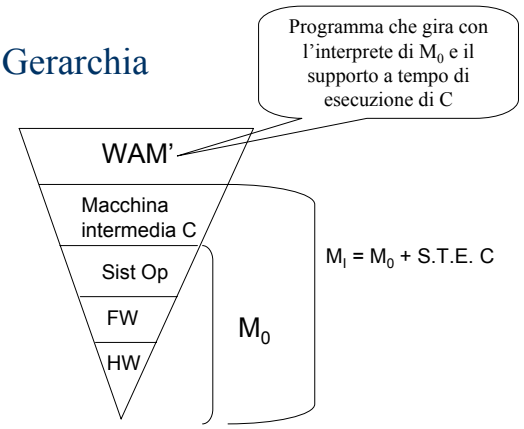
Esempio

- Abbiamo un compilatore C (che genera codice eseguibile dall'interprete della macchina hardware)
- Vogliamo realizzare un "compilatore" PROLOG usando C come linguaggio di implementazione
- Ricordiamo che il "compilatore" PROLOG è in realtà una implementazione interpretativa con la WAM come macchina intermedia

Esempio

- Dobbiamo implementare in C
  - La WAM
  - Un traduttore da programmi PROLOG a programmi WAM funzionalmente equivalenti (non è essenziale comunque che il traduttore sia scritto in C)
- Compiliamo, con il compilatore C, la WAM (che è un programma C) ed otteniamo un programma WAM' (funzionalmente equivalente a WAM) che gira sulla macchina intermedia C

Gerarchia



Esempio

- I programmi generati dal traduttore PROLOG girano su una gerarchia di macchine astratte che contiene anche la macchina intermedia C
- Implementando la WAM in linguaggio macchina, si appoggerebbe direttamente sul sistema operativo

Implementazione

- Perche implementare un linguaggio L_1 in L_2?
- La complessità della realizzazione di una macchina intermedia è direttamente proporzionale alla sua distanza dalla macchina ospite e, quindi, dal livello del linguaggio di implementazione
- È molto più semplice realizzare la WAM in C piuttosto che in linguaggio macchina!

Implementazione via kernel

- Dobbiamo implementare L
- Implementiamo un sottoinsieme L' di L
- L'intera macchina astratta di L viene realizzata utilizzando L' (kernel di L) come linguaggio di implementazione
- È una tecnica molto usata nell'implementazione di sistemi operativi
- Facilita la portabilità

Meccanismi di astrazione

- La gerarchia di macchine astratte non termina con l'implementazione di un linguaggio di alto livello L
- Un programma in L aggiunge un ulteriore livello alla gerarchia, estendendo la macchina intermedia di L con nuovi componenti (di solito operazioni sui tipi di dato)
- In pratica si può vedere la programmazione come definizione e realizzazione di macchine astratte

Programmazione come realizzazione di macchine astratte

- In relazione alle metodologie di programmazione per livelli di astrazione:
  - Dal basso verso l'alto
  - Dall'alto verso il basso
- Mette in evidenza l'importanza dei meccanismi di astrazione offerti dal linguaggio

Meccanismi di astrazione dei linguaggi

- Sottoprogrammi (tutti i linguaggi)
- Definizione di nuovi tipi di dato e tipi di dato astratti (classi) (solo pochi linguaggi moderni)
- Non è chiaro quali siano i meccanismi per astrarre le altre componenti dei linguaggi (gestione memoria, sequenza)
- Alcuni (pochissimi) linguaggi permettono di astrarre tutto con relativa facilità con la metaprogrammazione (es. Implementazione di un linguaggio tramite definizione della sua sintassi e semantica denotazionale in ML)

Astrazione

- In qualunque linguaggio si può realizzare qualsiasi astrazione
- Da considerare
  - Quanto costa realizzare questa astrazione
  - Quanto l'astrazione è facile da usare