Giorno 1
Macchine Astratte, Linguaggi, Interpretazione, Compilazione

Sommario

Una macchina astratta, associata ad un linguaggio, rappresenta il comportamento di una macchina fisica; Può essere implementata su una macchina fisica per realizzare un interprete del linguaggio associato, eventualmente realizzando macchine intermedie, usate anche dai compilatori per il run-time support. Le fasi della compilazione si dividono in front end e back end: la front end consiste di scanner (che genera i token) e parser (che genera gli AST), mentre la back end genera il codice.

1 Macchine Astratte

Una macchina astratta è un sistema virtuale che rappresenta il comportamento di una macchina fisica, individuando:

- L'insieme delle **risorse** necessarie per l'esecuzione dei programmi
- Un insieme di **istruzioni** specificatamente progettato per operare con queste risorse.

Alcuni esempi di macchinie astratte:

- **Algol Object Code**: Macchina astratta per Algol60.
  - *Risorse*: Stack, Heap, Program Store
  - *Istruzioni*: Accesso a variabili e array, scope per variabili e procedure, call by name, call by value
- **Smalltalk-80**: Linguaggio a oggetti compilato su macchina astratta *stack based*
  - *Risorse*: Stack
  - *Istruzioni*: Bytecode per manipolazione dello stack, bytecode per invocazione di metodi, accesso a variabili di istanza di oggetti...
- **Python**: macchina astratta stack-based con numerose istruzioni di base

Macchine astratte per linguaggi funzionali: Sono caratterizzazte da:

- Stack, per gestire chiamate di funzioni
- Environment, per gestire associazione tra variabili e valori
- Closure, per rappresentare il valore di una funzione
- Heap Garbage collector, per la gestione della memoria dinamica.

1.1 Stack machine e bytecode

1.1.1 Stack machine

Una stack machine è una macchina che è capace, ad esempio, di valutare un'espressione in notazione polacca inversa, utilizzando come uniche operazioni di accesso a memoria `push` e `pop`.

1.1.2 Bytecode

Un bytecode è un linguaggio intermedio tra linguaggio macchina e linguaggio di programmazione, usato per descrivere le operazioni che costituiscono un programma.

1.2 Struttura di una macchina astratta

Una macchina astratta è costituita da:

- Un **interprete**, ossia un programma che prende in ingresso il programma da eseguire (il suo *albero di sintassi astratta*) e lo esegue ispezionando la sua struttura.
- Una memoria (dati e programmi)
- Controllo
- Operazioni primitive

1.2.1 Linguaggio macchina di una macchina astratta

Data una macchina astratta M, il linguaggio L_M è il linguaggio che ha come stringhe legali tutti i programmi interpretabili dall'interprete di M.

1.2.2 Implementazione di macchine astratte

L'implementazione di una macchina astratta M consiste nella realizzazione di M all'interno di un'altra macchina detta **macchina ospite** M_O, il cui linguaggio è utilizzato per descrivere le componenti di M.

1.2.3 Interpretazione vs Compilazione

Un linguaggio *interpretato* viene eseguito sulla relativa macchina astratta M, che lavora sulla macchina ospite.

Un linguaggio *compilato* viene tradotto direttamente nel linguaggio macchina della macchina ospite, sulla quale verranno eseguiti direttamente, e quindi **non è necessario realizzare la macchina astratta**.

1.2.4 Macchine intermedie

Talvolta si realizza una *macchina intermedia*, che aggiunge un *supporto a tempo di esecuzione* ad M_O, ossia una collezione di strutture dati e sottoprogrammi che permette l'esecuzione del codice prodotto dal compilatore (e.g. gestione stack, gestione memoria (malloc), codice di debugging)

1.2.5 Ricapitolando

Ci sono tre famiglie di implementazioni:

- **Interprete puro**:
  - La macchina astratta corrisponde alla macchina intermedia
  - L'interprete del linguaggio di M è realizzato su M_0
  - Implementazioni vecchie di alcuni linguaggi logici e funzionali, e.g. LISP, PROLOG
- **Compilatore**
  - Macchina intermedia per estendere la macchina ospite con run-time support
- implementazione mista
  - I programmi sono tradotti dal linguaggio di M a quello della macchina intermedia (bytecode)
  - I programmi in bytecode sono interpretati dalla macchina ospite (Java)

2 Struttura di un compilatore

2.1 Fasi della compilazione

La compilazione si può dividere in due "macrofasi": la fase **front end**, che consiste nell'analisi del programma sorgente e della determinazione della sua struttura sintattica e semantica, e la fase **back end**, che consiste nella generazione del linguaggio macchina.

Le vere e proprie fasi della compilazione sono:

- **Scanner** (analisi lessicale) Genera dei *token* a partire dal programma sorgente. I token consistono in:
  - Parole chiavi (if, while...)
  - operatori, punteggiatura, parentesi...
  - identificatori e costanti

  Esempio:

  ```
  if(x >= y) y = 42;
  ```

  ↧

  $[IF][LPAR][ID(x)][GEQ][ID(y)][RPAR][ID(y)][BECOMES][INT(42)][SCOLON]$

- **Parser** (analisi sintattica) Legge i token, genera albero di sintassi astratta [ok]

  Ambiguità Se non si definisce la precedenza degli operatori, si possono avere più alberi di sintassi astratta diversi per lo stesso programma.

- **Generazione del codice** (back end).

2.2 Parser a discesa ricorsiva

Spiegazione dettagliata in Giorno 2.

2.3 AST in Java

[Struttura albero con programmazione ad oggetti, sulle slide esempio di rappresentazione di albero in javascript, ok]

2.4 Analisi statica

[Taint analysis, vedi slide]