Linguaggi, Interpreti e Compilatori

Indice

1 Lezione 1 - 21/09/2022

1.1 Interpreti e compilatori

Definizione: Interprete (per il linguaggio L)

Un programma che prende in input un programma eseguibile (espresso nel linguaggio L) e lo esegue, producendo l'output corrispondente.

Definizione: Compilatore (per il L verso il linguaggio M)

Un programma che prende in input un programma eseguibile (espresso nel linguaggio L) e lo traduce, producendo in output un programma equivalente (espresso nel linguaggio M.

Nota:

Per eseguire il compilato serve un interprete per il linguaggio M

Definizione: Compilatori ottimizzanti

Il compilatore traduce il programma in modo da ottenere un miglioramento di qualche metrica (tempo di esecuzione, memoria usata, consumo energetico, ...)

Nota:

L'ottimizzazione in senso matematico è impossibile, per cui ci si accontenta di tecniche euristiche che funzionano bene ma non forniscono garanzie di ottimalità.

1.1.1 Interpretazione o compilazione?

Principali motivazioni per la compilazione (attività off-line):

- identificare alcuni errori di programmazione prima dell'esecuzione del programma
- migliorare l'efficienza
- rendere utilizzabili alcuni costrutti dei linguaggi ad alto livello (troppo costosi per l'approccio interpretato)

Linguaggi tipicamente compilati: FORTRAN, Pascal, C, C++, OCaml, ... (possono comunque essere interpretati)

Linguaggi tipicamente interpretati: PHP, R, Matlab, ... (possono comunque essere compilati)

Approcci misti: Java, Python, SQL (varie combinazioni di compilazione e interpretazione)

Esempio: Java

↪ Compilazione da sorgente Java verso bytecode Java

↪ Interpretazione del bytecode Java da parte della JVM

↪ Compilazione JIT di alcune porzioni di bytecode verso linguaggio macchina.

Esempio: SQL

↪ Intepretazione delle query SQL

↪ L'interprete tipicamente include la fase di ottimizzazione

↪ Possibilità di compilare in forma ottimizzata porzioni di SQL (prepared statements, stored procedures, ...)

Compromessi da stabilire:

- Bilanciamento tra attività off-line e on-line
- Il tempo di compilazione deve essere accettabile
- L'occupazione in spazio del programma compilato deve essere accettabile

1.1.2 Perché si studiano i compilatori?

Applicazioni pratiche di concetti teorici:

- Analisi lessicale: regex e fsa
- Analisi sintattica: CFG e automi a pila
- Analisi e ottimizzazione IR: teoria dell'approssimazione, calcoli di punto fisso, equivalenza tra programmi
- Progettazione dei linguaggi di programmazione

Applicazioni di algoritmi e strutture dati sofisticati:

- Tabelle hash, alberi e grafi
- Algoritmi di visita
- Algoritmi greedy, dynamic programming, tecniche euristiche di ricerca in spazi di soluzioni
- Pattern matching, scheduling, colorazione di grafi

Interessanti problemi di system/software engineering:

- Interconnessioni con architettura e SO
- Gestione progetto complesso, organizzazione del codice
- Design pattern
- Compromessi tra efficienza e scalabilità

Implementare interpreti/compilatori per DSL:

- DSL: Domain Specific Language
- Linguaggi di alto livello progettati per una classe specifica di applicazioni
- Es: linguaggi di scripting per librerie grafiche, videogiochi, automazione industriale, robotica, domotica, data science, ...
- Es.: linguaggi per la generazione automatica di documentazione tecnica per il SW (Doxygen, Javadoc)

---

2 Lezione 2 - **/**/2022