Interpreti e compilatori

Enea Zaffanella

enea.zaffanella@unipr.it

21 settembre 2020

Linguaggi, interpreti e compilatori
Laurea Magistrale in Scienze informatiche

Interpreti

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

Compilatori

Compilatore (per il linguaggio 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 bene

Per eseguire il compilato serve un **interprete** per il linguaggio M

Compilatori "ottimizzanti"

Compilatori ottimizzanti

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

Davvero ottimizzanti?

NO. L'ottimizzazione vera e propria (in senso matematico) è in pratica impossibile da ottenersi, per cui si ci accontenta di **tecniche euristiche**, che funzionano bene nei casi comuni ma non forniscono garanzie di ottimalità.

Interpretazione o compilazione? (i)

Compilazione: attività "off-line"

Principali motivazioni:

- identificare alcuni errori di programmazione prima dell'esecuzione del programma
- migliorare l'efficienza (ad es., spostando alcuni calcoli a tempo di compilazione o evitando la ripetizione di calcoli identici)
- rendere utilizzabili alcuni costrutti dei linguaggi ad alto livello (inaccettabilmente costosi per l'approccio interpretato)

Interpretazione o compilazione? (ii)

Linguaggi *tipicamente* compilati

- FORTRAN, Pascal, C, C++, OCaml, ...
- Nota: possono comunque essere interpretati (e.g., `cling`)

Linguaggi *tipicamente* interpretati

- PHP, R, Matlab, ...
- Nota: 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 (Java Virtual Machine)
- Compilazione JIT (Just-In-Time, cioè a tempo di esecuzione) di alcune porzioni di bytecode verso linguaggio macchina

Esempio: SQL

- Interpretazione delle query SQL
- L'interprete tipicamente include la fase di ottimizzazione
- Possibilità di compilare in forma ottimizzata porzioni di SQL (prepared statements, stored procedures, . . .)

Interpretazione o compilazione? (iii)

Necessità di stabilire compromessi

- 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
- Esempio: programmazione generica
  - uso dei **template** in C++
  - uso dei **generics** in Java

Perché si studiano i compilatori? (i)

Una risposta sicuramente **non** appropriata

- Per superare l'esame!
- È sicuramente necessario studiarli per passare l'esame, ma questa non è una motivazione adeguata.

Perché si studiano i compilatori? (ii)

Applicazioni pratiche di concetti teorici

- Analisi lessicale: espressioni regolari e automi a stati finiti
- Analisi sintattica: grammatiche libere da contesto e automi a pila
- Analisi e ottimizzazione IR: teoria dell'approssimazione, calcoli di punto fisso, equivalenza tra programmi
- Progettazione dei linguaggi di programmazione

Perché si studiano i compilatori? (iii)

Applicazioni di algoritmi e strutture dati sofisticati

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

Perché si studiano i compilatori? (iv)

Interessanti problemi di system/software engineering

- Compilatore è parte importante del software di sistema: interconnessioni con architettura e sistema operativo
- Gestione progetto complesso, organizzazione del codice
- Design pattern (ad es., visitor)
- Compromessi tra efficienza e scalabilità

Perché si studiano i compilatori? (v)

Implementare interpreti/compilatori per DSL

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

Perché si studiano i compilatori? (vi)

Una risposta (spesso) non appropriata

- Per implementare un compilatore per un **linguaggio mainstream** (C, C++, Java, Python, . . .)
- Raramente necessario
- Spesso ben oltre le capacità del programmatore medio

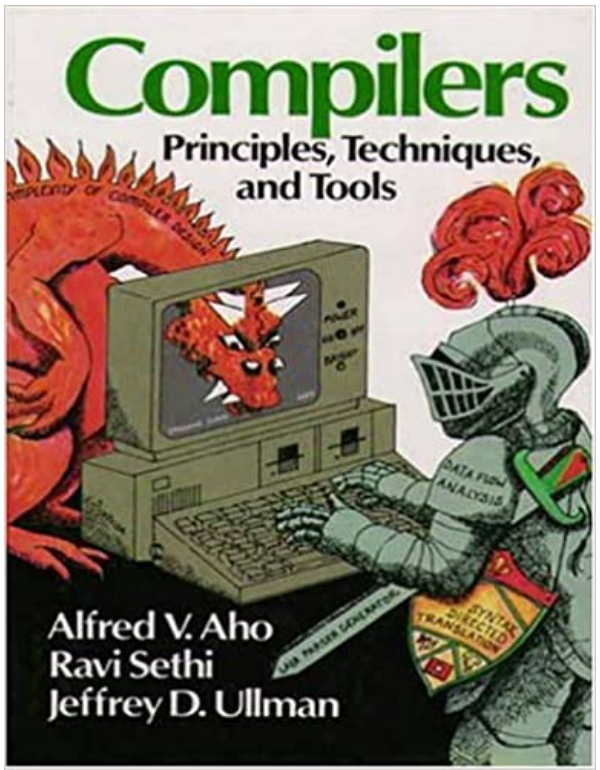
Anche se . . .

- Necessità di **estendere** un compilatore esistente per
  - supportare nuovi costrutti del linguaggio
  - supportare nuove architetture hardware (ad es., GPU)
- Progetti collaborativi come **Clang/LLVM** sono aperti a contributi esterni

Complessità di progetto di un compilatore

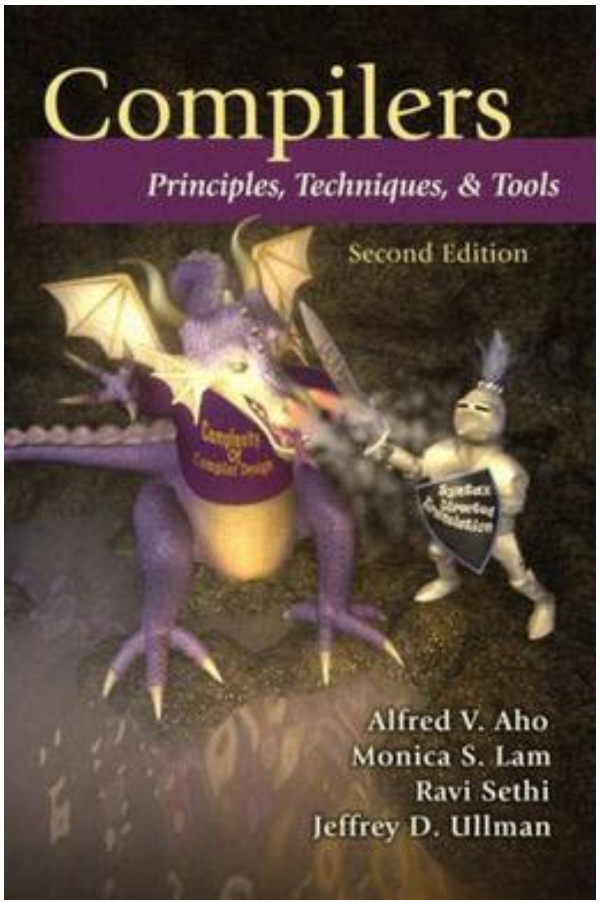
Green Dragon Book (1977)
A.V. Aho, J.D. Ullman
Principles of Compiler Design

Complessità di progetto di un compilatore



Red Dragon Book (1986)
A.V. Aho, R. Sethi, J.D. Ullman
Compilers: Principles, Techniques, and Tools (1st edition)

Complessità di progetto di un compilatore

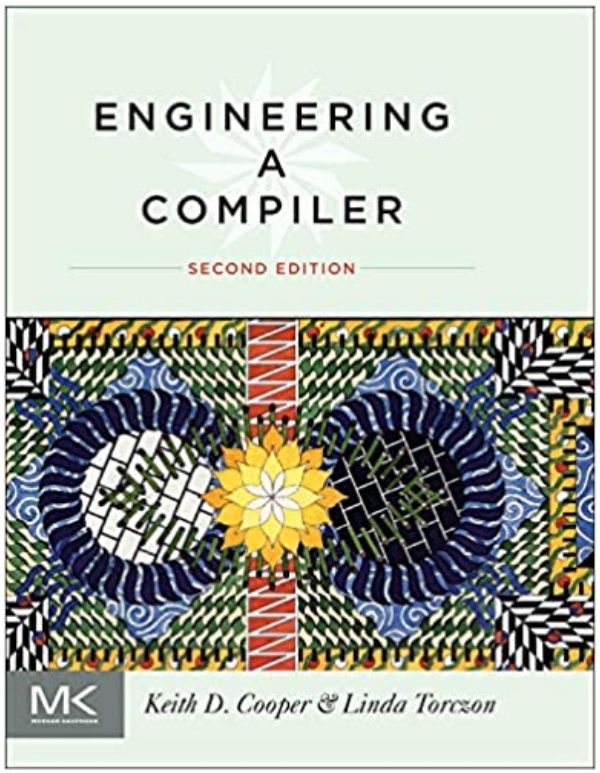


Purple Dragon Book (2006)
A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman
Compilers: Principles, Techniques, and Tools (2nd edition)

Complessità di progetto di un compilatore

More dragons: il logo della LLVM Compiler Infrastructure

Nota bene: il nostro libro di testo



Keith D. Cooper, Linda Torczon
Engineering a Compiler
2nd Edition, Morgan Kaufmann, 2011

Nella prossima lezione

La struttura di un compilatore

- Il classico compilatore a due passi: front-end e back-end
- Il **front-end**
  - Analisi lessicale, sintattica, semantica statica
  - Generazione codice IR
- Il **back-end**
  - Generazione codice target
  - Selezione e schedulazione istruzioni, allocazione registri
- Compilatori a tre passi: il **middle-end**
  - Ottimizzazioni per il codice IR
  - Suddivisione in sottopassi del middle-end