UNIVERSITÀ
DEGLI STUDI
DI TORINO

Linguaggi Formali e Traduttori

a.a. 2018-2019

Riferimenti bibliografici

Compilatori: principi, tecniche e strumenti, A.H. Aho, M. S.Lam, R. Sethi, J. D. Ullman

Introduzione

1.1 Elaboratori di linguaggi

1.2 Struttura di un compilatore

1.2.1 Analisi lessicale

1.2.2 Analisi sintattica

1.2.3 Analisi semantica

1.2.4 Generazione di codice intermedio

1.2.5 Ottimizzazione del codice

1.2.6 Generazione del codice

1.2.7 Gestione della tabella dei simboli

Linguaggi formali e traduttori

- Linguaggi di programmazione = formalismi per descrivere procedimenti di calcolo
- Per poter essere eseguito, un programma deve essere tradotto in una forma che può essere «compresa» da un calcolatore

Compilatore, Interprete

- Il corso si propone di introdurre principi e tecniche utili per costruire compilatori, ma utilizzati anche in molte altre aree dell'informatica: linguaggi di programmazione, architettura, teoria dei linguaggi, algoritmi, ingegneria del software…

Ad esempio, l'invenzione dell'architettura RISC è stata influenzata dalle tecniche di compilazione. I processori CISC prevedevano modalità di indirizzamento articolate per facilitare l'accesso a strutture dati non elementari e complicate istruzioni per la chiamate di procedura. Le ottimizzazioni fatte dai compilatori sono oggi in grado di ridurre tali istruzioni a un numero limitato di operazioni semplici.

Sembra quindi preferibile sviluppare insiemi di istruzioni semplici che i compilatori sono in grado di combinare in modo ottimale per realizzare funzioni più complesse.

Dal programma sorgente al programma eseguibile

- ***Compilatore***: programma che legge un programma scritto in un linguaggio di programmazione (linguaggio sorgente) e lo traduce in un programma equivalente scritto in un altro linguaggio (linguaggio oggetto)
- Un compilatore mostra anche gli errori che trova durante la traduzione
- Il linguaggio oggetto potrebbe già essere eseguibile

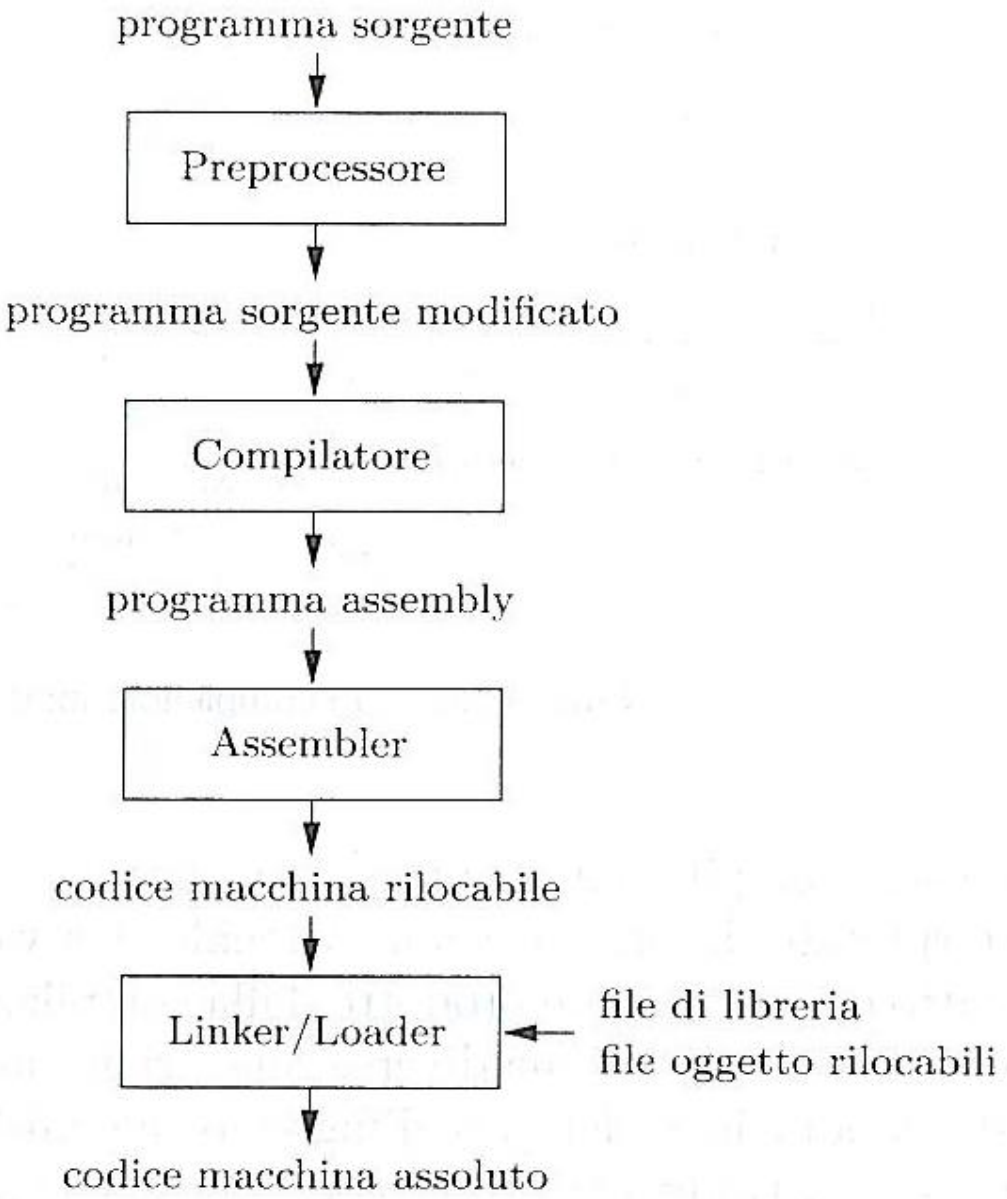
- ***Interprete***: esegue direttamente le operazioni specificate nel programma sorgente (traducendo un'istruzione alla volta ed eseguendo il pezzo di programma oggetto generato)
- Più veloce l'esecuzione di programmi compilati, ma l'interpretazione di solito fornisce una migliore diagnostica degli errori

Java combina interpretazione e compilazione: un programma Java viene prima compilato in una forma intermedia chiamata bytecode, che poi viene interpretata.

Dal programma sorgente al programma eseguibile

- Raccogliere moduli memorizzati in file separati ed espandere le macro è compito di un *preprocessore*
- Il nuovo programma sorgente è fornito in ingresso al *compilatore*
- Spesso il compilatore produce un programma in linguaggio *assembly,* più semplice da generare e più leggibile
- Il programma *assembler* produce il codice macchina rilocabile
- Programmi di grandi dimensioni sono di solito compilati a pezzi in file oggetto separati: il codice rilocabile dei file oggetto e quello di eventuali librerie viene collegato in un unico codice macchina dal *linker* che risolve i riferimenti a locazioni di memoria e simboli definiti in un altro file
- il *loader* carica insieme in memoria i file oggetto eseguibili rendendoli così pronti per l'esecuzione

Dal programma sorgente al programma eseguibile

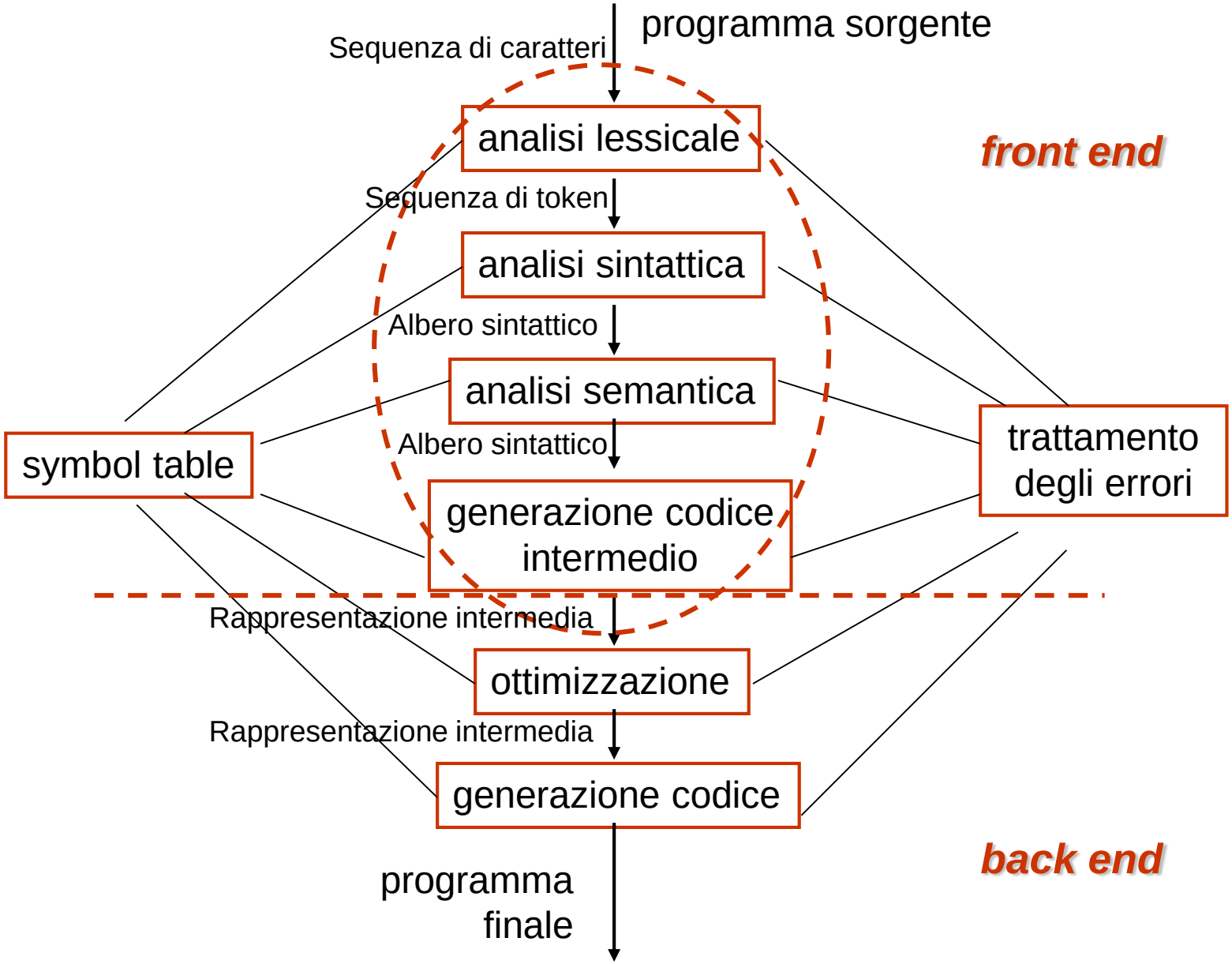


Struttura del compilatore

- Due parti: analisi e sintesi
  - **Analisi** (*front end*): prende un programma e lo divide in parti su cui impone una struttura; basandosi su tale struttura crea una rappresentazione intermedia del programma sorgente; segnala possibili errori; memorizza informazioni sul programma in una tabella, da passare, insieme alla rappresentazione intermedia alla parte di sintesi
  - **Sintesi** (*back end*): costruisce il programma oggetto dalla rappresentazione intermedia e dalle informazioni nella tabella dei simboli

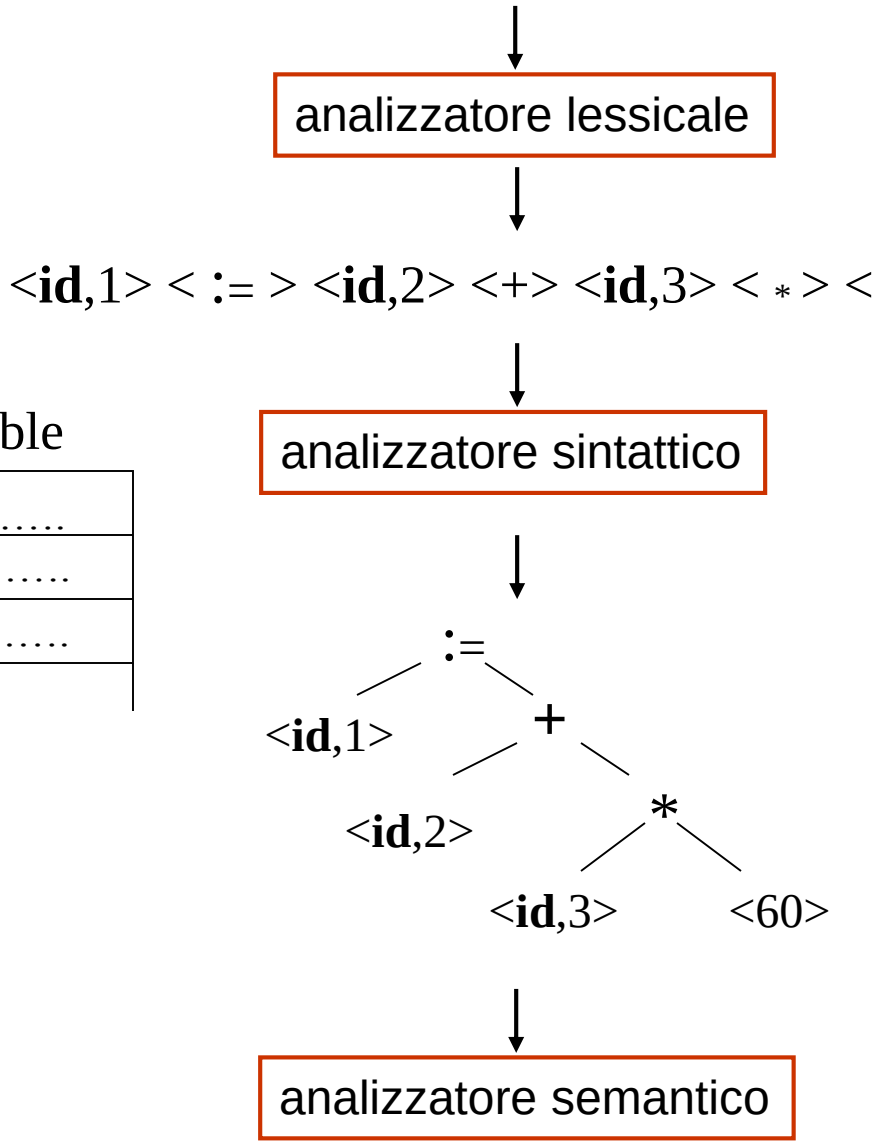
- Sequenza di fasi: ogni fase trasforma una rappresentazione del programma sorgente in un'altra
- Fasi principali: analisi lessicale, analisi sintattica, analisi semantica, generazione del codice intermedio, ottimizzazione del codice, generazione del codice oggetto

Struttura del compilatore



Il processo di compilazione

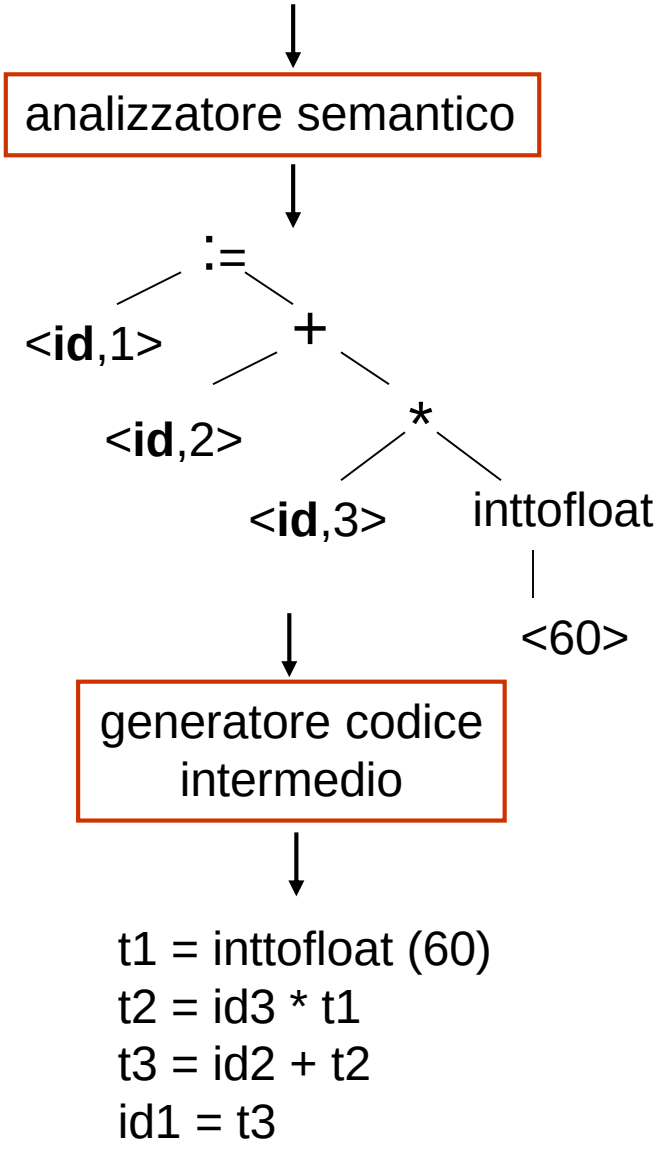
position := initial + rate * 60



Symbol Table

1	position	…….
2	initial	…….
3	rate	…….
4		

Il processo di compilazione



Symbol Table

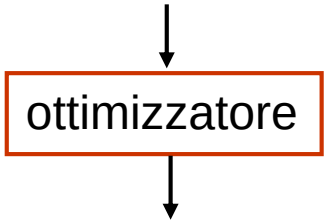
1	position	…….
2	initial	…….
3	rate	…….
4		

Il processo di compilazione

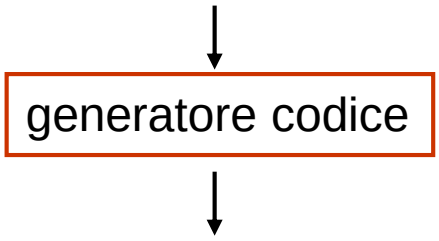
Symbol Table

1	position	…….
2	initial	…….
3	rate	…….
4		

t1 = inttofloat (60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3



t1 = id3 * 60.0
id1 = id2 + t1




```
LDF    R2, id3
MULF   R2, R2, #60.0,
LDF    R1, id2
ADDF   R1, R1, R2
STF    id1, R1
```

Strumenti per la costruzione di compilatori

- *Generatori di scanner*: generano automaticamente un analizzatore lessicale a partire dalla descrizione dei lessemi del linguaggio sorgente tramite espressioni regolari.

- *Generatori di parser*: generano automaticamente un analizzatore sintattico partendo dalla descrizione del linguaggio sorgente (una grammatica).

- *Traduttori guidati dalla sintassi*: producono una collezione di funzioni per la visita degli alberi di parsificazione e la generazione del codice intermedio (in generale di un linguaggio target).

Come descrivere i lessemi e come riconoscerli?