

Traduzione di programmi

Lezione 12 di Fondamenti di informatica

Docente: Giuseppe Scollo

Università di Catania
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica, I livello, AA 2009-10

Indice

1. Traduzione di programmi
2. il processo di traduzione
3. esecuzione di un traduttore
4. oltre la traduzione
5. analisi sintattica
6. diagrammi sintattici
7. costruzione di alberi sintattici
8. ambiguità sintattica
9. analisi semantica
10. generazione del codice
11. ottimizzazione del codice

il processo di traduzione

due tipi principali di traduttori:

compilatori: traduttori "puri": sorgente → oggetto

interpreti: eseguono ciascuna istruzione sorgente subito dopo averla tradotta

fasi principali della traduzione:

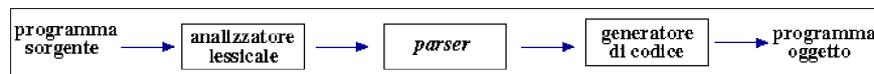
analisi lessicale: riconoscimento dei lessemi (*tokens*) del linguaggio (identificatori, parole riservate, separatori, operatori, etc.)

analisi sintattica (*parsing*):

controllo di **correttezza** del sorgente rispetto alla **grammatica libera** del linguaggio (v. avanti)

costruzione dell'**albero sintattico** del sorgente

generazione del codice: da albero sintattico e **tabella dei simboli**



esecuzione di un traduttore

l'esistenza di una naturale **sequenzialità delle fasi** del processo di traduzione non ne impedisce l'**esecuzione parallela**

l'attività di ciascuna fase trasforma un **flusso (*stream*)** di dati in ingresso in un flusso di dati in uscita

una **porzione** del flusso in ingresso è in genere sufficiente per produrre una corrispondente porzione del flusso in uscita

ciò permette l'esecuzione in **cascata parallela (*pipeline*)** delle fasi della traduzione

oltre la traduzione

compilazione separata:

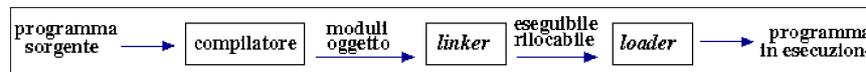
i compilatori sono generalmente in grado di tradurre **moduli** di programma (ad es. procedure, definizioni di funzioni, classi) separatamente e in tempi diversi

linker:

il risultato della traduzione di un **modulo sorgente** non è generalmente un programma eseguibile, bensì un **modulo oggetto**, che va **collegato** ad altri moduli oggetto per costruire un **eseguibile**

loader:

in un programma eseguibile gli indirizzi di memoria non sono generalmente assoluti, bensì **relativi** ad una locazione di base, determinata all'atto del **caricamento** del programma in memoria



analisi sintattica

grammatica libera del linguaggio L : $G_L = (V_T, V_N, P, s_0)$

dove:

V_T : vocabolario terminale

V_N : vocabolario non terminale

$P \subseteq V_N \times (V_T \cup V_N)^*$: produzioni (regole) della grammatica

s_0 : simbolo iniziale della grammatica

linguaggio L generato da G_L : sottoinsieme di V_T^* costituito dalle stringhe terminali **derivabili** da s_0 con le produzioni P

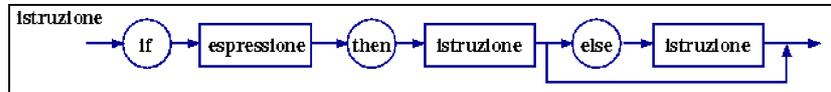
determinare (se esiste) una tale derivazione, per una qualsiasi stringa data di V_T^* , è l'**obiettivo dell'analisi sintattica**

diagrammi sintattici

rappresentazioni grafiche delle **produzioni** (regole) di una **grammatica libera**:

cerchi: simboli **terminali** (lessemi)

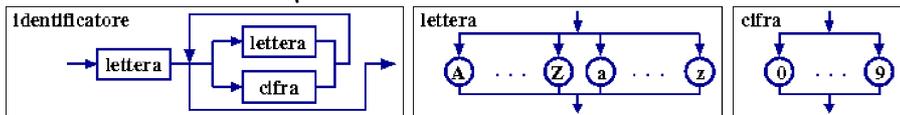
rettangoli: simboli **non terminali** (sintagmi)



un **cammino** dall'ingresso all'uscita del diagramma sintattico rappresenta la stringa R di una produzione (N, R) ; il diagramma stesso è etichettato dal non-terminale N

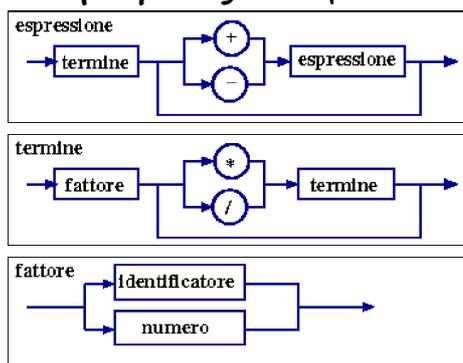
produzione in forma di **Backus-Naur (BNF)**: $N ::= R$

notazione **BNF estesa**: possibilità di **alternative** (diramazioni) e **iterazioni** (cicli)

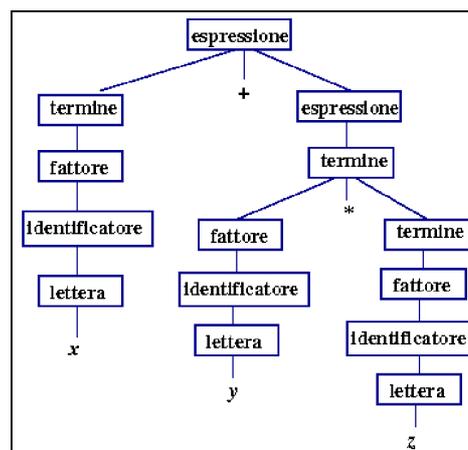


costruzione di alberi sintattici

esempio: parsing dell'espressione $x+y*z$



derivazione dal simbolo iniziale:
top-down o *bottom-up*?



ambiguità sintattica

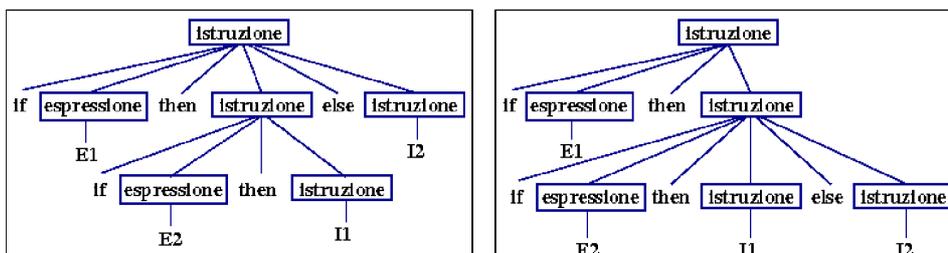
una grammatica è **ambigua** se qualche espressione del linguaggio che essa genera ha **più derivazioni** dal simbolo iniziale

esempio: il precedente diagramma sintattico per istruzioni condizionali

if <espressione> then <istruzione> [else <istruzione>]

rende ambigua la grammatica di simbolo iniziale <istruzione>

infatti "if E1 then if E2 then I1 else I2" ha due derivazioni:



analisi semantica

non tutte le espressioni sintatticamente corrette hanno significato

ad es., la frase seguente ha ben **quattro** derivazioni nella grammatica inglese:

time flies like an arrow

tuttavia, **tre** di esse sono prive di significato

nel processo di traduzione dei programmi, l'**analisi semantica statica** discrimina

gli alberi risultanti dall'analisi sintattica in base ad ulteriori **vincoli**, che devono essere soddisfatti perché l'espressione sia traducibile

ad es., vincoli di concordanza (o compatibilità) di **tipo**

detti vincoli possono essere espressi da **regole semantiche** associate a produzioni di una grammatica libera; le **grammatiche con attributi** permettono di formalizzare tali regole con precisione

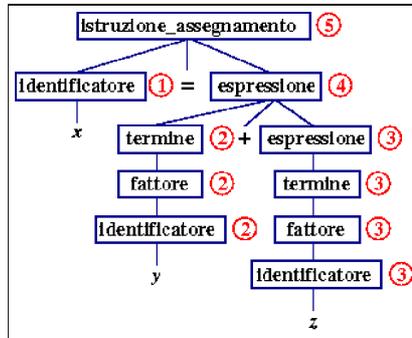
l'analisi semantica genera alberi sintattici **decorati**, ai cui nodi sono associati **record semantici**, con informazione di tipo e riferimenti ad unità lessicali

generazione del codice

input: alberi sintattici decorati, prodotti dall'analisi semantica

carattere essenziale del **processo:** estendere le decorazioni dei nodi dell'albero con opportune sequenze di istruzioni del linguaggio oggetto

in tal modo, la traduzione procede per **composizione bottom-up**



record semantici

#	nome	tipo
1	x	int
2	y	int
3	z	int
4	temp	int
5	x	int

codice generato

```
1 X: .DATA 0
2 Y: .DATA 0
3 Z: .DATA 0
4 TEMP: .DATA 0
...
LOAD Y
ADD Z
STORE TEMP
5 LOAD TEMP
STORE X
```

ottimizzazione del codice

il codice generato nell'esempio precedente presenta **istruzioni superflue**

obiettivo: migliorare l'efficienza del codice generato

ottimizzazione locale:

esamina sequenze brevi delle **istruzioni generate** ed applica semplici trasformazioni, quali ad es.:

valutazione di costanti

eliminazione di trasferimenti memoria-registro superflui

riduzione di complessità, in base al costo computazionale delle istruzioni macchina

ottimizzazione globale:

molto più complessa, esamina le **strutture** del codice, di qualsiasi ampiezza e profondità, per applicarvi trasformazioni ottimizzanti computazionalmente equivalenti (non tutti i compilatori la realizzano)