

linguaggi di programmazione

Matteo

December 21, 2025

ricorda sul libro per modulo 1 capitoli da 1 a 5

1 intro

linguaggi imperativi , linguaggi dichiarativi, funzionali e logici linguaggi imperativi

- basato sulla nozione di stato, le istruzioni sono comandi che cambiano lo stato
- stato= insieme di locazioni di memoria contenente valori
- generalmente di basso livello

linguaggi dichiarativi

- basati sulla nozione di funzioni o relazione
- le istruzioni sono dichiarazioni di nuovi valori

linguaggi funzionali

- basati sulla nozione di funzione: risultato di un programma = valore esplicito di una espressione
- ricorsione
- programmare= costruire una funzione

linguaggi logici

- basati sulla nozione di relazione: risultato di un programma = insieme di valori di variabili determinato da relazioni
- istruzioni = implicazioni logiche fra opportune formule , che possono essere viste come regole di riscrittura
- programmare= definire la relazione che definisce il valore delle variabili di interesse

linguaggi orientati agli oggetti: ora molto usati, sono linguaggi imperativi con alcune metodologie dichiarative. Oggetti (istanze di opportune classi) che contengono i dati (concetti imperativi) e metodi come funzioni per operare su tali oggetti (concetti dichiarativi).

2 Macchine astratte

pag 28 :Ml molto complesso da tradurre in una macchina fisica ad alto livello, quindi vengono costruite macchine fisiche solo per linguaggi di basso livello

altrimenti crei un programma che traduce L e i suoi costrutti in L', un linguaggio già esistente implementabile su una macchina già esistente; minore velocità rispetto al caso precedente

altrimenti anziché programmi usi microprogrammi o firmware che traducono L in un linguaggio di basso livello e usano registri di sola lettura, garantendo buone prestazioni

pag 31 impl. inter. pura : non è vera traduzione, I fa corrispondere a una certa parte di L una certa parte di Lo, poco efficiente, I deve decodificare al momento. più flessibile perché permette di interagire direttamente con l'esecuzione del programma e gli interpreti sono più veloci da creare

impl. comp. pura: vera traduzione dal compilatore che avviene prima dell'esecuzione del programma. alcune informazioni del programma sorgente vanno perse che rende più difficile interagire con il programma in tempo reale

vantaggi e svantaggi interpretazione:

- vantaggio: maggiore flessibilità, permette di interagire con l'interazione del programma.
- vantaggio : più veloce da realizzare
- vantaggio : occupa meno memoria, non generando nuovo codice, problema meno sentito oggi.
- svantaggio : la compilazione interpretativa è meno efficiente perché deve effettuare al momento dell'esecuzione un'interpretazione dei costrutti di L. con diverse occorrenze dello stesso costrutto si richiedono ulteriori decodifiche ogni volta

vantaggi e svantaggi compilazione:

- vantaggio: il compilatore deve compilare una sola volta all'inizio poi i costrutti non devono essere decodificati ogni volta
- vantaggio: maggiore efficienza
- svantaggio: perdita del codice sorgente. se ci fosse un problema sarebbe difficile capire da dove si origina l'errore

siamo a pag 35 di fatto nei linguaggi reali sono presenti entrambe le tecniche. ad esempio abbiamo linguaggio L, compilato in linguaggio intermedio Li, interpretato in Lo se l'interprete della macchina intermedia è molto diverso dal linguaggio finale Lo diremo che è un'implementazione interpretativa, se è quasi uguale è un'implementazione compilativa.

la differenza tra completamente interpretativa e mostly interpretativa è che non tutti i costrutti devono essere simulati

per impl. mostly compilativa alcune funzionalità devono essere simulate perché Li non trova un corrispondente immediato in Lo

ci sono tanti spettri intermedi, di solito si predilige flessibilità

di solito ci sono più livelli di macchine astratte con i loro linguaggi e funzionalità. praticamente tutto nell'informatica è creato attraverso una gerarchia di macchine astratte (programmi e linguaggi).

ogni livello accede a quello inferiore e aggiunge nuove funzionalità, in un certo senso creando un nuovo linguaggio.

ciò permette un certo dominio sulla complessità del sistema e indipendenza tra i livelli

macchina astratta: una formalizzazione astratta di un generico esecutore di algoritmi formalizzato attraverso un linguaggio di programmazione

interprete: un componente essenziale della macchina astratta che ne caratterizza il comportamento, mettendo in relazione "operazionale" il linguaggio della macchina astratta col mondo fisico circostante

3 descrivere i linguaggi di programmazione

secondo Morris i linguaggi hanno tre campi che li descrivono:

- **grammatica:** indica quali frasi sono corrette, il lessico si occupa delle singole parole come sequenze di segni, la sintassi delle frasi come sequenze di parole
- **semantica:** risponde alla domanda "cosa significa una frase corretta", per quanto riguarda i linguaggi artificiali, si tratta del loro scopo
- **pragmatica:** risponde alla domanda "come usare una frase corretta e sensata", si occupa dell'uso della frase rispetto al contesto

per quanto riguarda i linguaggi di programmazione si può parlare anche di implementazione cioè mediante quale processo le frasi "operative" del linguaggio realizzano il loro fine.

3.1 grammatica e sintassi

tecniche generative: espedienti trovati nell'ambito dei linguaggi naturali per limitare nelle grammatiche le ambiguità dei linguaggi naturali non troppo utili per i ling. naturali, ma utilissimi nei ling. di programmazione

3.1.1 grammatiche libere

definito un insieme finito di elementi come **alfabeto**, dato un alfabeto A , e dato A^* = insieme di stringhe finite di A . un linguaggio è un sottoinsieme di A^* , una grammatica formale serve a definire un insieme di stringhe tra quelle possibili su un dato alfabeto

grammatica libera da contesti composta da 4 parti:

- **simboli non terminali:** insieme finito di simboli astratti che non compaiono nel concreto
- **simboli terminali:** insieme finito di simboli concreti che compaiono nelle stringhe del linguaggio
- **produzioni:** le regole di riscrittura che permettono di sostituire un simbolo non terminale con una sequenza di simboli terminali e non terminali

- simbolo iniziale da cui parte tutto

derivazione: sequenza di applicazioni delle produzioni che partendo dal simbolo iniziale porta a una stringa composta solo da simboli terminali

3.1.2 alberi di derivazione

per i linguaggi di programmazione è importante conoscere gli **alberi radicati ordinati**: Un albero radicato e ordinato è una struttura finita con una radice e una sequenza ordinata di sottoalberi, ognuno dei quali è a sua volta un albero. La radice è il simbolo iniziale s , e ogni nodo è un simbolo (terminale o non). ogni nodo ha un solo padre.

3.1.3 ambiguità

Se vogliamo usare gli alberi di derivazione per descrivere la struttura logica di una stringa siamo in una pessima situazione: la grammatica dell'Esempio 2.3 non è in grado di assegnare una struttura univoca alla stringa in questione. A seconda di come è costruita la derivazione, la precedenza tra i due operatori aritmetici varia.

definizione: Una grammatica G è ambigua se esiste almeno una stringa di $C[G]$ che ammette più di un albero di derivazione.

l'ambiguità nasce dal fatto che almeno una stringa abbia più alberi di derivazione ma non dal fatto che la stessa stringa abbia più derivazioni.

una grammatica ambigua non può essere usata per compilare (non sarebbe possibile una traduzione univoca), anche se è possibile modificarla.

lo sforzo di creare una grammatica non ambigua può portare a più complessità e contorcimenti.

3.2 vincoli sintattici contestuali