linguaggi di programmazione

Matteo

December 29, 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 nozionne di funzioni o relazione
- le istruzioni sono dichiarazioni di nuovi valori

linguaggi funzionali

- basati sullla 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 orienatati agli oggetti: ora molto usati, sono linguaggi imperativi con alcune ,etodologie 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

altimenti crei u programma che trduce L e i suoi costrutti in L', un linguaggio gia esistente implementabile su una macchina gia esistente; minore velocita rispetto al casoo precedente

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

pag 31 impl. inter. pura : non e vvera traduzione, I fa corrispondere a una certa parte di L una certa parte di Lo, poco efficente, I deve decodificare al momento. piu flessibile perche permette di interagire direttamnete con l'esecuzione del programma e gli interpreti sono piu 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 piu difficile interagire con il programma in tempo reale

vantaggi e svantaggi interpretazione:

- vantaggio: maggiore flessibilita, permette di interagire con l'interazione del programma.
- vantaggio : piu veloce da realizzare
- vantaggio : occupa meno memorio, non generando nuovo codice, problema meno sentito oggi.
- svantaggio : la compilazione intterpretativa e meno efficente perche 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 al"inizio poi i costrutti non devono essere decodificati ogni volta
- vantaggio: maggiore efficenza
- savntaggio: perdita del codice sorgente. se ci fosse un problema sarebbe difficile capire da dove si origiina 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 intermmedia e molto diverso dal linguaggio finale Lo diremo che e un implementazione interpretativa, se e quasi uguale e un implementazione compilatiiva.

la differnza tr acompletamente interpretattiva e mostly interpretattiva e che non tutti i costrutti devono essere simulati

per impl. mostly compilativa alcune funzionalita devono essere simulate perche Li non trova un corrispondente immediato in molto

ci sono tanti spettri intermedi, di solito si predilige flessibilita

di solito ci sono piu livelli di macchine astratte con i loro linguaggi e funzionalita. praticamente tutto nell'informatica e creato attravero una gerarchi adi macchine astratte(programmi e linguaggi).

ogni livello accede a quello inferiore e aggiunge nuove funzionalita, in un certo senso creando un nuo liguaggioMnLn

cio peremtte un certo dominio sulla complessita del sistemaa e independenza 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, i 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 puo parlare anche di implementazione cio 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 ambiguita 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 alfabetoA, e dato A*= insieme di stringhe finite di A. un linguagggio e un sottoinsieme di A*, una grammatica formaale serve a definire un di stringhe tra quelle possibili su un dato alfabeto

grammatica libera da conteso composta sa 4 parti:

- simbolii non terminali: insieme finito di simboli astratti che non compaionon 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 tuttoù

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 e 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 e il simbolo iniziale s, e ogni nodo e un simbolo(terminale o non).ogni nodo ha un solo padre.

3.1.3 ambiguita

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 11na struttura 11nivoca 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[GJ che ammette più di un albero di derivazione.

ambiguita nasce dal fatto che almeno una stringa abbia piu alberi di derivazione ma non dal fatto che la stessa stringa abbia piu derivazioni.

una grammatica ambigua non puo essere usata per compilare(non sarebbe possibile una traduzione univoca), am e posiibile modificarla.

lo sforzo di creare una grammatica non ambigua puo portare a piu complessita e contorcimenti.

3.2 vincoli sintattici contestuali

alcune volte anche i linguaggi di programazione dipendono dal contesto. ad esempio in c la variabile deve essere dichiarata prima di essere usata. questi sono detti vincoli sintattici e sono impossibili da descrivere in una grammatica libera da conttesto. farlo e invece compito dei linguaggi formali, ed esistono grammatiche dette grammatiche contsetuali.

tuttavia queste sono molto complesse ed e comunque impossibile stabilire metodi di traduzione per questo il contesto e ignorato dalle grammatiche e espresso al megliob tramite linguaggio naturale. ediante strumenti formali quali i sistemi di transizione, che vedremo nel Paragrafo 2.5, dedicato alla semantica.

nel gergo dei linguaggi di programmazione ci si riferisce ad essi come a **vincoli di semantica statica**

Nel gergo, "sintassi" significa in genere "descrivibile con una grammatica libera", "semantica statica" significa "descrivibile con vincoli contestuali verificabili staticamente sul testo del programma", mentre "semantica dinamica" (o "semantica" tout court) si riferisce a quanto attiene a come e quando il programma sarà eseguito.

La distinzione tra sintassi non contestuale e sintassi contestuale (cioè seman- tica statica) è stabilita precisamente dalla potenza espressiva delle grammatiche libere.

3.2.1 sintassi astratta e concreta

La grammatica di un linguaggio di programmazione definisce un linguaggio come insieme di stringhe

non tutti gli alberi di derivazione corrispondono a programmi legali.

l'analisi di semantica statica ha il compito di selezionarne alcuni, quelli che soddisfano 1 vincoli contestuali del linguaggio stesso. L'insieme degli alberi che risultano da questo processo costituisce la sintassi astratta di un linguaggio.

3.3 compilatori

la struttura logica del compilatore e un modello a cascata, in cui il proggramma sorgente viene tradotto iin varie rappresentazioni intermedie fino ad arrivare al linguaggio oggetto

ci sono varie fasi

1. analisi lessicale: il testo del programma sorgente viene suddiviso in token, insiemi di caratteri.(parole chiave, identificatori, operatori, ecc) (equivalente delle parole.)

2. analisi sintattica: i token vengono raggruppati in frasi secondo la grammatica del linguaggio dall'analizzatore sintattico(o parser).

3. analisi semantica: le frasi vengono verificate rispetto ai vincoli di semantica statica. viene costruita la tabella dei simboli.

4. generazione del codice intermedio: viene prodotto un codice intermedio che rappresenta la struttura logica del programma, ci sono ancora ottimizzazioni da fare percio non e opportuna generare il codice oggetto

5. ottimizzazione del codice intermedio: il codice intermedio viene migliorato per efficienza

6. generazione del codice oggetto: il codice intermedio ottimizzato viene tradotto nel linguaggio macchina della piattaforma di destinazione

3.4 semantica

la semantica di un linguaggio di programmazione e il significato delle sue frasi e la sua descrizione e piuttosto complessa. Dovrebbe essere una descrizione sia precisa che flessibile che non ambigua.

esistono metodi formali per la descrizione semantica adatti ma ulcuni fenomeni semantici rendono la decrizione formale molto complessa. percio per lo piu e usato il linguaggio naturale.

Ciò non toglie che metodi formali per la semantica siano molto utilizzati nelle fasi preparatorie del progetto di un linguag- gio, oppure per descriverne alcune caratteristiche particolari, dove la necessità di evitare ambiguità è predominante rispetto alla semplicità. esistono due grandi famiglie di metodi formali per la descrizione della semantica:

- La semantica denotazionale è l' ap- plicazione ai linguaggi di programmazione di tecniche sviluppate per la semantica del linguaggio logico-matematico. Il significato di un programma è dato da una funzione, che esprime il componamento input/output del programma stesso.

- ell'approccio operazionale, invece, non vi sono entità esterne (per esempio funzioni) da associare ai costrutti di un linguaggio. Usando tecniche opportune, una semantica operazionale specifica il comportamento della macchina astratta, ossia ne

definisce formalmente l'interprete, facendo riferimento ad un formali- smo (astratto) di più basso livello. ci possono essere tecniche che usano automi formali e altre che usano sistemi di regole logico-matematiche.

3.5 pragmatica

serve a capire a cosa serve una frase in un certo contesto. si parla anche di stile di programmazione e metodi di progettazione del software

non ci interessa troppo in questo corso

3.6 implementazione

implementare un linguaggio vuol dire scriverne un compilatore e realizzare una macchina astratta per il linguaggio oggetto del compilatore, op- pure scriverne un interprete e realizzare la macchina astratta del linguaggio nel quale l'interprete stesso è scritto oppure, come avviene nella pratica, un mix di entrambe queste cose.

ancora una volta non ci interessa

4 analisi lessicale dei linguaggi regoalari

4.1 intro

los copo dell'analisi lessicale e riconoscere dnella stringa di ingresso gruppi di caratteri che corrispondono a certe caratteristiche sintattiche. cosi la stringa di ingressso e trasformata in una sequenza di token.

Non c'è un motivo teorico per separare l'analisi sintattica da quella lessicale: tutta 1'elaborazione del testo sorgente potrebbe essere condotta mediante le tecniche di analisi sintattica che tratteremo nel Capitolo 4. Tuttavia la separazione tra i due tipi di analisi è molto utile perché le tecnologie per effettuare l'analisi lessicale sono più efficienti di quelle (più generali) usate per l'analisi sintattica.

4.2 token

un token e una coppia:

- il nome del token(il simbolo del token).
- il valore del token(la stringa di caratteri che lo rappresenta)

poi per ogni nome di token si definiscono quali sequenze di caratteri sono riconosciute come istanze(valori di quel token)

Le sequenze di caratteri associate ad un token sono specificate mediante un pattern, cioè una descrizione generale della forma che le sequenze di caratteri possono assumere. Questa descrizione generale è data mediante espressioni regolari, un concetto che introdurremo tra breve.

con tutto ci viene fatto una lista di token che viene passata al parser.

una stringa dell'alfabeto di ingresso che corrisponde ad un pattern si dice **lessema** .ad un token possono corrispondere anche piu lessemi.

4.3 espressioni regolari

4.3.1 alcune definizioni

:

- **alfabeto**: insieme finito di simboli
- **stringa su A**: sequenza finita di simboli di un alfabeto A
- **lunghezza di una stringa**: numero di simboli che la compongono
- **concatenazione**: operazione che consiste nel mettere due stringhe in sequenza
- **linguaggio sulla'alfabeto A**: insieme di stringhe su A

date due stringhe s e t su A, la loro concatenazione e indicata con st
ora possiamo definire alcune operazioni tra linguaggi sullo stesso alfabeto

- **concatenazione**: dato un limguaggio L e un linguaggio M, la loro concatenazione si dice LM o L·M ed e l'insieme di tutte le stringhe ottenute concatenando una stringa di L con una stringa di M. se uno dei due e un singoletto(insieme di 1 elemento),l'elemento puo rappresentare l'insieme.
- **unione**: dato un linguaggio L e un linguaggio M, la loro unione si dice L U M ed e l'insieme di tutte le stringhe che appartengono a L o a M.
- **chiusura di Kleene**: dato un linguaggio L, la sua chiusura di Kleene si dice L* ed e l'insieme di tutte le stringhe ottenute concatenando un numero qualsiasi di stringhe di L, compresa la stringa vuota. in pratica e l'unione di L^0, L^1, L^2 ecc..
- **chiusura positiva di Kleene**: dato un linguaggio L, la sua chiusura positiva di Kleene si dice L+ ed e l'insieme di tutte le stringhe ottenute concatenando una o piu stringhe di L. in pratica e l'unione di L^1, L^2

4.3.2 espressioni regolari

Un'**espressione regolare** descrive un insieme di stringhe. Questo insieme è un linguaggio ben definito e contiene tutti i possibili lessemi che corrispondono a un certo token.

i token sono classi di lessemi(identifier,number ecc...) i lessemi sono i simboli concreti.

dato un linguaggio L e un'espressioone regolare R, esiste un solo linguaggioL(R), cioe un solo insieme di tutte le stringhe che R descrive.

quindi in pratica un'espressione regolare e l'insieme di tutti i lessemi che corrispondono a un certo token(ad esempio se il token e identificatore x,y ecc... appartengono all'esspressione regolare)

tra due espressioni regolari sullo stesso linguaggio ci sono diverse proprieta: es commutativa, associativa ecc..

4.3.3 definizioni regolari

Per scrivere espressioni complesse è utile poterle suddividere in sottoespressioni, ognuna definita separatamente con un nome associato. Fissato un alfabeto A, una definizione regolare su A è costituita da una lista di definizioni

d1 := s1
d2 := s2
...
dn := sn

nella quale tutti 1 dn sono simboli nuovi, distinti tra loro e non appartenenti ad A, mentre ogni sn è un'espressione regolare sull'alfabeto A U d1 ...., di- 1.

4.4 automi finiti

le espressioni regolari specificano la forma deilessemi associati ad un token, ma rimane il problema di identificarli prima e associarli.

un **automa finito**: è un algoritmo con un pool di stati, che prende in input un lessema, consuma uno alla volta i caratteri del lessema e cambia stato in base al carattere letto e allo stato corrente. dopo aver consumato l'ultimo carattare, se l'automa si trova in uno stato di accettazione(tali stati sono detti stati finali), il lessema e riconosciuto come appartenente al token associato all'automa. altrimenti se non si trova in uno stato di accettazione o si e fermato prima rifiuta l'input.

4.4.1 automi finiti non deterministici

in certi casi puo essere utile avere piu scelte su quale stato andare in base al carattere letto e allo stato corrente. in questo caso si parla di **automa finito non deterministico**: ogni coppia (stato corrente,carattere letto) puo portare a piu stati successivi. in questo caso l'automa accetta il lessema se esiste almeno un percorso che porta a uno stato di accettazione.

definizione formale: un automa finito non deterministico è una quintupla(S,Q,d,q0,F) dove

- S è l'alfabeto degli input
- Q è l'insieme finito degli stati
- d è la funzione di transizione che associa ad ogni coppia (stato corrente, carattere letto) un insieme di stati successivi.
- q0 è lo stato iniziale
- F è l'insieme degli stati finali

4.4.2 automi finiti deterministici

un DFA(definite finite automata) e un automa finito non deterministico in cui ogni coppia (stato corrente,carattere letto) porta a un solo stato successivo.

- una sola computazione possibile
- piu semplice da implementare

- nessuna ambiguita
- efficente

a partire da qulasiasi NFA(non deterministic finite automata) e possibile costruire un DFA equivalente che accetta lo stesso linguaggio e viceversa.

in un NFA ci sono 2 cause di non determinismo:

- piu stati successivi per la stessa coppia (stato corrente,carattere letto)
- transizioni epsilon, che permettono di cambiare stato senza leggere alcun carattere

di solito e buona norma trasformare gli NFA in DFA per semplificare l'implementazione.

cio si fa attraverso la **costruzione per sottoinsiemi**: cioe ogni stato del DFA corrisponde a un sottoinsieme di stati dell'NFA. Uno stato dell'DFA e finale se contiene almeno uno stato finale dell'NFA.

4.5 da espressioni regolari ad automi finiti