Linguaggi di Programmazione

DaveRhapsody Jacopo De Angelis DlcGold

30 Settembre 2019

Indice

Capitolo 1

Introduzione al corso

1.1 Programma del corso

Il corso è volto ad insegnare dei paradigmi di programmazione dei seguenti tipi:

1.1.1 Logica Matematica e Linguaggi logici (Prolog)

Termini, fatti(predicati), regole, unificazione, procedura di risoluzione

1.1.2 Linguaggi funzionali e Lisp (et al.)

Atomi, liste, funzioni e ricorsione

1.1.3 Linguaggi imperativi

Memoria, stato, assegnamenti, puntatori

Il concetto è che con questo corso si vanno a studiare paradigmi più evoluti, usati tutt'ora e comunque aventi un ampio approccio logico, oltretutto LISP è usato nelle pagine web (Si userà moltissimo la ricorsione, A I U T O)

1.2 Modalità d'esame

1.2.1 subsection name

- il voto finale sarà una media pesata dei voti conseguiti nell'esame relativo alla parte teorica e nell'esame del progetto
  - Occhio, il peso è a discrezione dei prof

1.2.2 Prove parziali

Le prove d'esame sono costituite da uno scritto di 6-10 domande, e da un progetto da consegnare entro una data prefissata

1.3 Appelli regolari

Gli appelli regolari sono composti da un progetto ed un esame scritto, che può essere seguito da un esame orale a discrezione del docente basato sui temi trattati durante il corso
NON C'E' POSSIBILITA' DI RECUPERI, infatti scritto, orale e progetto vanno sostenuti **NELLO STESSO APPELLO**
Progetto e scritto sono corretti separatamente

NON CI SARANNO ECCEZIONI Lo avete già letto nel passaggio precedente, ma lo ripeto lo stesso perchè deve essere chiaro che N O N S I F A N N O E C C E Z I O N I.

Capitolo 2

Il paradigma

2.1 Cos'è?

E' il metodo di soluzione ad un determinato problema, a seconda dei paradigmi si hanno diversi tipi di linguaggi di programmazione

2.1.1 Storicamente

Il primo paradigma è l'imperativo, cioè il paradigma basato sui tre costrutti di selezione, iterazione e sequenza.
Inoltre si mantiene il concetto di assegnamento di un valore ad una determinata variabile

2.1.2 L'effetto collaterale

Viene definito effetto collaterale quando, a seguito dell'esecuzione di un qualsiasi codice, il contenuto di un'area di memoria viene cambiato; per intenderci, anche solo l'istruzione "x += 1" genera un effetto collaterale, poichè nell'area di memoria di x viene cambiato il valore.
Perchè è importante tutto ciò, direte. Semplice: il paradigma puro funzionale si basa proprio sul fatto che un programma non generi mai, mai, *M A I*, effetti collaterali. Successivamente vedremo che in Prolog ci saranno parecchi problemi se provassimo ad assegnare direttamente un valore ad una variabile

2.2 Logica del primo ordine

Prolog è costituito da una serie di clausole derivanti dalla logica del primo ordine

2.3 Linguaggi funzionali

Questi si basano proprio sui concetti matematici di funzione, ad esempio si ragiona sui domini, sui codomini, sull'insiemistica, solite cose. La loro caratteristica è che ogni funzione, dato sempre lo stesso input, restituisce sempre lo stesso risultato. Cosa vuol dire questo? Che non dipende da variabili esterne (e da qui l'importanza degli effetti collaterali, evitati nei linguaggi funzionali).

2.4 Paradigma imperativo

Le caratteristiche essenziali dei linguaggi imperativi sono legate all'architettura di Von Neumann, costituita dai famosi due componenti **Memoria (componente passiva) e Processore (componente attiva)**
In pratica la principale attività che ha la cpu è quella di eseguire calcoli ed assegnare valori alle variabili, che sono delle celle di memoria.

Va considerato Il concetto di variabile è un'astrazione di una cella di memoria, per dire se giochi su assembly vai a toccare i veri e propri registri, mentre su C o Assembly si ragiona per nome di variabile, non vai di indirizzamento fisico

2.4.1 Il concetto di variabile

In Prolog e LISP cambia completamente il concetto di variabile, ma per come saranno presentati vedremo che non c'entra niente.
In matematica abbiamo il concetto di variabile? Sì, quella che sta dentro una funzione, in informatica è diciamo diverso, non è un'astrazione, ma lo vedremo in seguito

2.5 Modello di Von Neumann

Per manipolare la memoria utilizzo la variabile, simbolo che indica la cella di memoria, nei linguaggi funzionali sarà possibile usare il concetto di variabile matematica.

Alla fine il modello di Neumann è composto da I/O, Memoria e CPU con i suoi cicli di clock

2.6 Stile prescrittivo

Un programma scritto in un linguaggio imperativo prescrive le operazioni che la CPU deve eseguire per modificare lo stato di un sistema

Le istruzioni sono eseguite nell'ordine in cui queste appaiono, ad eccezione delle strutture di controllo

Realizzati sia attraverso interpretazione che compilazione, nati più per manipolazione numerica che simbolica.

2.7 Concetto di programma

Un programma è intendibile come un insieme di algoritmi e di strutture dati ma la struttura di un programma consiste in

- Una parte dichiarativa in cui son presenti le dichiarazioni di tutte le variabili del programma e del loro tipo

- Una parte che descrive l'algoritmo risolutivo utilizzato, mediante istruzioni del linguaggio

2.8 Perchè utilizzare paradigmi diversi?

Per esempio l'intelligenza artificiale si sviluppa su linguaggi di programmazione specifici, bisogna usare linguaggi che operino in un determinato modo, considerati tipo di Altissimo super mega galatticissifantastico livello infatti, utilizzabili pure da non programatori

Infatti son generati per manipolazione simbolica non numerica

2.9 Paradigma logico

Concetto primitivo: Deduzione logica, avente una base di logica formale e un obbiettivo, che è intendibile come formalizzazione del ragionamento

Programmare infatti significa descrivere il problema con frasi (Formule logiche) del linguaggio,
Interrogare il sistema, che effettua deduzioni in base alla "conoscenza rappresentata"

Ai lettori Mi rendo conto che non si capisca un cazzo, voi immaginatevi come mi stia sentendo al momento io mentre prendo appunti.. Perdonatemi

Prolog è un insieme di formule ben formate, ragiona con il linguaggio logico, con una descrizione della realtà di interesse, di fatto è una dimostrazione in un linguaggio logico che costituisce un programma. Più semplicemente ho una frase da dare al mio interprete, Prolog icchè fa? Semplicemente la realizza sotto forma di dimostrazione.

2.10 Esempio di un programma Prolog

Ci sono fondamentalmente:

- Asserzioni incondizionate (fatti) A.
- Asserzioni condizionate (regole) A :- B, C, D, ... , Z.
  - A è la conclusione o conseguente (deve avere una sola clausola)
  - B, C, D, . . . ,Z sono le premesse o antecedenti
- Un'interrogazione ha la forma: :- K, L, M, . . . , P.

Ovviamente A, B, C, *TUTTE LE ALTRE*, sono semplicemente predicati
MI RACCOMANDO MASSIMA ATTENZIONE ALLA SINTASSI, ogni clausola Prolog termina con un punto.

La ',' si legge come AND

2.10.1 Esempio:

Due individui sono colleghi se lavorano per la stessa ditta/azienda Regole Fatti Interrogazione

```
collega(X, Y) :-
        lavora(X, Z),
        lavora(Y, Z),
        diverso(X, Y).

lavora(ciro, ibm)
lavora(ugo, ibm)
lavora(olivia, samsung)
lavora(ernesto, olivetti)
lavora(enrica, samsung)

:- collega(X, Y).
```

Programmare Prolog non è come scrivere in un linguaggio di programmazione, non si scrive un algoritmo, in questo caso abbiamo le famose clausole, (regole e fatti),

ATTENZIONE l'interrogazione non è una clausola, occhio a non confondersi

2.10.2 Esempio dell'ordine di una lista

- **Ordine prescrittivo**: Controlla se la lista è vuota, e dà come risultato la lista vuota stessa, altrimenti calcola una permutazione della lista e controlla se è ordinata, dando come risultato L_1 altrimenti fa una permutazione su L etc.
  Il programmatore deve specificare le istruzioni che generano la sequenza di permutazioni della lista L $\begin{cases} \text{il risultato dell'ordinamento di una lista vuota è la lista vuota} \\ \text{Il risultato dell'ordinamento di una lista L e } L_1 \end{cases}$
  Quindi, lo stile prescrittivo presuppone che OGNI SINGOLO CASO venga considerato e programmato. non esiste il "ma è ovvio che debba fare questo", ogni singolo caso è tua responsabilità. Sì, tua, proprio tu che stai leggendo. Prolog si basa su questo tipo di ordine.

- **Stile Dichiarativo**: L'ambiente si fa carico di generare possibili permutazioni della lista L, secondo deduzione matematica

2.11 Paradigma funzionale

- Si basa sul concetto di funzione matematica, ossia una associazione tra due insiemi che relaziona ad ogni elemento di un insieme (dominio) un solo elemento di un altro insieme (codominio)

- La definizione di una funzione specifica dominio, codominio, e regola di associazione

- ESEMPIO:
  Incr: N $\rightarrow$ N
  Incr(x) - x + 1
- Dopo aver dato definizione, una funzione è aopplicabile ad un elemento del dominio (argomento) per restituire l'elemento del codominio ad esso associato (valutazione)
- incr(3) $\rightarrow$ 4 L'unica operazione utilizzata nel funzionale è l'applicazione di funzioni
- Il ruolo dell'esecutore di un linguaggio funzionale si esaurisce nel calutare l'applicazione di una funzione (il programma) e produrre un valore
- Nel paradigma funzionale puro il valore di una funzione è determinato dagli argomenti che riceve al momento della sua applicazione e non dallo stato del sistema rappresentato dall'insieme complessivo dei valori associati a variabili(e/o locazioni di memoria) in quel momento
- Oggettivamente si ha l'assenza di effetti collaterali

Attenzione Il concetto di variabile che utilizziamo è quello di "costante" matematica, in cui i valori NON sono mutabili, non ho nessun asegnamento

L'essenza della programmazione funzionale consiste nel combinare funzioni mediante composizione e uso della ricorsione

2.11.1 Composizione di funzioni + ricorsione

La struttura di un programma consiste nella definizione di un insieme di funzioni ricorsive mutualmente

L'esecuzione del programma consiste nella VALUTAZIONE dell'APPLICAZIONE di una funzione principale a una serie di argomenti

2.12 LISP

LISt Processing,

Il progretto originale era di creare un linguaggio funzionale puro, infatti nel corso degli anni sono stati sviluppati molti ambienti di sviluppo lisp di cui terremo in considerazione Common Lisp e Scheme, oltre che emacs etc.

2.12.1 Esempio di programma LISP

Controlla un elemento se appartiene ad una lista

```
(defun member (item list))
    (cond ((null list)nil)
        ((equal item(first list))T)
    (T(member item(rest list))))
    (member 42(list 12 34 42))
```

Dopo una parentesi tonda ci va per forza una funzione, è fondamentale, defun definisce una funzione infatti, dopo c'è il nome di tale funzione. In LISP la tabulazione è ESSENZIALE, AUGURI A DISTINGUERE DOVE PORTI LA QUINTA PARENTESI DELLE DODICI CHE HAI SCRITTO PIANGENDO SUL CODICE ALLE 2 DI NOTTE.
Gli elementi si separano con lo spazio NON con la virgola, mi raccomando.
La terza riga è la più ostica e dice: è uguale T al primo elemento della lista? E' molto incastrato ma si riesce a capire, associamo per esempio ad item = 2 (E' un esempio) e list = [1,2]
Noi ci arriviamo con la logica che c'è, ma in realtà cosa faremo? Ragioniamo per gradi

1. null list? false
2. è 2 = al primo elemento della lista? False
3. è 2 = al secondo elemento della lista? True

Se LISP trova T è come se scrivessimo **true**

2.13 Ambienti RunTime di linguaggi logici funzionali e non

- Richiami di nozioni di architettura e programmazione
- Per eseguire un programma di qualsiasi linguaggio il sistema operativo deve mettere a disposizione l'ambiente runtime che dia almeno due funzioni
  - Mantenimento dello stato della computazione(pc, limiti di memoria)
  - Gestione memoria disponibile (fisica e virtuale)
- L'ambiente runtime può essere una vera e propria macchina virtuale tipo la JVM di java
- In particolare la gestione di memoria avviene usando due aree concettualmente ben distinte con funzioni diverse
  - Lo stack, ambiente dell'ambiente runtime che serve per la gestione delle chiamate, a procedure metodi etc
  - L'heap dell'ambiente runtime serve per gestire strutture dinamiche
    * Alberi
    * Liste etc

2.13.1 Activation frame

- La valutazione di procedure avviene mediante la costruzione sullo stack di sistema di activation frames
- i parametri formali di una procedura vengono associati ai valori (si passa tutto per valore, non esistono effetti collaterali)
- E' un altro modo di chiamare i record di attivazione, via
- Il corpo della procedura viene valutato (ricorsivamente) tenendo questi legami in maniera statica

cioè il concetto è che bisogna capire cosa accade con variabili che risultino libere in una sottoespressione

2.14 Activation Frame di una funzione

Contiene:

- Return address
- Registri
- Static / Dynamic link (lo statico punta alle variabili globali)
- Argomenti
- Local definitions (RV)

Se si ha in mente come funziona oggettivamente lo stack (con i record di attivazione) è la stessa cosa
All'esame si potrebbe chiedere cos'è l'activation frame e a icchè serve

2.15 Heap e Garbage Collector

L'heap è l'area di memoria destinata alla memorizzazione delle strutture per i dati dinamiche, mentre invece il garbage collector ha il compito di accumulare lo schifo che si accumula tra variabili non deallocate etc, e le dealloca appunto.

Capitolo 3

Logica e ragionamento

Partiamo con le cose semplici, bisogna passare da quello che è un linguaggio parlato a una stesura di condizioni

Prendiamo un triangolo, vogliamo dimostrare che se due triangoli hanno i due lati uguali allora è isoscele

$$AB = BC \vdash \angle A \angle C$$

1. AB = BC per ipotesi
2. ABH = HBC per (3)
3. Il triangolo HBC è uguale al triangolo HBC ABH per (2)
4. A e C per (1)

3.0.1 Regole di inferenza

1. Introduzione della congiunzione (L'AND)
2. Modus Ponens
3. Eliminazione della congiunzione

Come lavora il **Modus Ponens**:
E' semplice manipolazione sintattica, osservando la formula che ci vien data possiamo riscriverla scrivendo come base di conoscenza il conseguente, cioè in pratica prendo e sostituisco con il conseguente.

Per far si che le mie formule siano vere, se avessi un A or B può esser vera in ben tre casi diversi, non posso eliminare i due casi disgiunti, se voglio mantenere una solidità non posso, e quindi questa regola (disgiunzione) non esiste.

Attenzione in una dimostrazione non si può dare nulla per scontato, tutto cilò che noi diamo per assodato, un pc non lo dà, dobbiamo essere molto precisi nelle indicazioni, bisogna lavorare in un'ottica più precisa

cerchiamo ora di tradurra tutto in un linguaggio più formale SE AB = BC E BH = BH e ABH = HBC allora il triangolo ABH è uguale a HBC ed abbiamo trasformato (1) in

SE triangolo ABH è uguale al triangolo HBC ALLORA AB = BC e BH = BH e AH = HC, E ABH = HBC E AHB = CHB E A = C

Da un punto di vista formale noi partiamo da un'ipotesi, noi vogliamo dimostrare che A = C.

La dimostrazione è un processo sintattico, non ragiono in termini di verità, perchè NON si sta parlando di interpretazione ma manipolazione delle formula

Ogni passo deve corrispondere ad una formula, e subito dopo le etichette

Differenza tra assiomi e ipotesi Gli assiomi sono conoscenza pregressa del dominio, mentre le ipotesi sono solo supposizioni iniziali, uno si specifica, l'altro no

3.1 Dimostrazione

E' una sequenza di passi dove il finale è la formula da dimostrare e abbiamo un insieme di passi intermedi che possono essere presi dalle conoscenze pregresse, oppure applicando regole di inferenza ai passi PRECEDENTI, solo precedenti mi raccomando.

Le regole di inferenza sono applicabili solo ai passi precedenti rispetto ad una formula

3.2 Logica Proposizionale

Nella logica proposizionale ci si occupa delle conclusioni che possiamo trarre da un insieme di proposizioni, abbiamo infatti un insieme P di proposizioni

Si introduce il concetto di **interpretazione** di un insieme di proposizioni, infatti all'insieme P si associa una funzione di verità (True e False)

Questa funzione associa un valore di verità ad ogni elemento di P, ad ogni proposizione. La valutazione è il ponte tra sintassi e semantica di un linguaggio

Chiaramente posso legare tra loro le proposizioni con $\vee\wedge$ e $\neg$ Una formula ben formata è un insieme di espressioni sintatticamente corrette di un linguaggio
In prolog le formule atomiche le chiameremo letterali, che possono esser positivi e negativi, e qui si richiamano i concetti di fondamenti della tabella di verità

Negli esercizi potremo usare

$$\frac{F_1, F_2, ..., F_K}{R}$$

E' la forma generale di una regola, sopra hai l'insieme delle formule vere tra le formule ben formate e R è la formula generata da "inserire" in FBF.

L'esempio di inferenza che si usa di solito è il Modus Ponens:

$$\frac{p \rightarrow q, p}{p}$$

Se il conseguente appare come formula negata, si negherà anche la formula originaria

Esempio:

$\frac{p \vee \neg q}{vero}$ Terzo escluso

$\frac{\neg\neg q}{q}$ eliminazione $\neg$

$\frac{p \wedge vero}{p}$ eliminazione $\vee$

$\frac{p \wedge \neg p}{q}$ contraddizione

3.3 Principio di risoluzione

E' una regola di inferenza generalizzata semplice e facile da utilizzare ed implementare, in pratica opera su fbf trasformate in forme normali congiunte, ed ognuno dei congiunti vien detto **clausola**

L'osservazione fondamentale alla base del principio di risoluzione è un'estensione della nozione di rimozione dell'implicazione su base della contraddizione

In pratica si usa per le dimostrazioni per assurdo.

3.4 Unit Resolution

Da un lato ho una formula ben formata disgiuntiva e dall'altro ho un letterale (o asserito), e una di queste formule è costituita da un solo letterale, per questo si chiama unit, è sintassi, non ci sto capendo più un cazzo pure. io, non so davvero che dirvi..
Eccomi, arrivo da autore esterno a spiegare: prendiamo prima lo schemino semplice semplice
$P <-- A \wedge B \wedge C \wedge D...$
Questo vuol dire che P è vera solo se tutte A, B, C e D sono vere. Ora, prendiamo nuovamente il codice prolog visto all'inizio:

```
collega(X, Y) :-
        lavora(X, Z),
        lavora(Y, Z),
        diverso(X, Y).
```

Cosa cambia dal dire questo o la regola sopra? La corrispondenza è semplicemente:
P è collega(X, Y) :-
        A è lavora(X, Z),

B è lavora(Y, Z),
C è diverso(X, Y).

Questo vuol dire che la nostra unit, P, è vera solo se sono vere le altre. Fine del mio intervento, la linea di nuovo a Dave.

Esempio:

- <non piove>, <piove o c'è il sole>
- <c'è il sole>

La dimostrazione per assurdo di fatto funziona assumendo che la formula negata sia vera, se combinandola con le proposizioni in fbf ottengo una contraddizione , allora si pul concludere con la verità di p