PROLOG

3 DIC 2019

- PROgramming in LOgic. Nasce negli anni ‘70/‘80.
- Il DataLog è un linguaggio usato in basi di dati
- Si basa sulla logica dei predicati e non è un linguaggio procedurale tipo Python bensì dichiarativo e la ritorsione non avviene con cicli for o while ma attraverso meccanismi di backtracking
- I costrutti base non sono comandi o funzioni bensì relazioni
- Si basa sull’unificazione e sul backtracking
- Calcolare in prolog significa costruire una dimostrazione
- A partire da delle formule logiche, prolog cerca di costruire una dimostrazione per una implicazione generica data e restituisce true o false

Idee di base

1) Descrivere la situazione d’interesse
2) Facciamo una domanda (query)
3) Prolog farà delle deduzioni a partire dal programma per rispondere alla query

Un programma in prolog ha significato:

1) Dichiarativo: una serie di fatti, regole, formule che non sono altro che dichiarazioni. E’ costituito da relazioni che sono: 1) fatti 2) regole
2) Operazionale: Vi è un motore di inferenza che permette di rispondere alla query

Fatti:

1) Semplici

   es. soleggiato.
   Chiediamo nel prompt:
   ? - soleggiato
   Il programma lo riconosce nelle dichiarazioni e restituisce:
   True
   es. piove.
   Chiediamo nel prompt:
   ?- piove
   Il programma non lo riconosce nelle dichiarazioni e restituisce:
   False

   - Devono iniziare con una lettera minuscola e terminare con un punto
   - Possono essere costituiti di una qualsiasi combinazione di lettere e numeri, maiuscole e minuscole, e vi può essere l’underscore
   - Si dovrebbe evitare di utilizzare singoli operatori matematici quali somma, differenza, moltiplicazione e divisione

   es. paolo_ha_freddo
   piove.
   paolo_gioca_calcio.

   QUERY: ?-piove
   True

QUERY: ?-nevica
False

QUERY: ?-gianni_ha_freddo
False

es. paoloGiocaCalcio - SI
paoloFreddo1 - SI
PaoloFreddo. - NO

2) Con argomenti
Relazioni che legano elementi
   - relazioni(<arg1>, <arg2>, …,<argn>)
   - Si usano le lettere minuscole

- Commenti: si chiude il commento fra due slash con asterischi
  /* commento */
  Se il commento dura una sola riga si può solamente anteporre il segno di percentuale
  % commento

- L'ordine in cui vengono scritti gli argomenti viene rispettato dal programma
  es. amico(bianca, dario) != amico(dario, bianca)
- Un nome in minuscolo indica un elemento specifico; un nome in maiuscolo indica una variabile.
  es. amico(antonio, Chiunque) == (V Chiunque.amico(antonio, Chiunque))
  QUERY: ?-amico(Antonio,100)
  True

ESEMPI
es.
Mangia(paolo, mele)
Mangia(paolo, arance)
?-mangia(paolo, mele)
True

?-mangia(paolo, fragole)

?-mangia(paolo,What)
(E What.mangia(paolo,What))
What=paolo
——->next (prolog scorre e mostra tutte le possibilità)

What=arance

es.
ama(ugo,maria)
Ama(paolo,calcio)
?-ama(paolo,Chi)
Chi=calcio
——>next
Chi=maria

es.

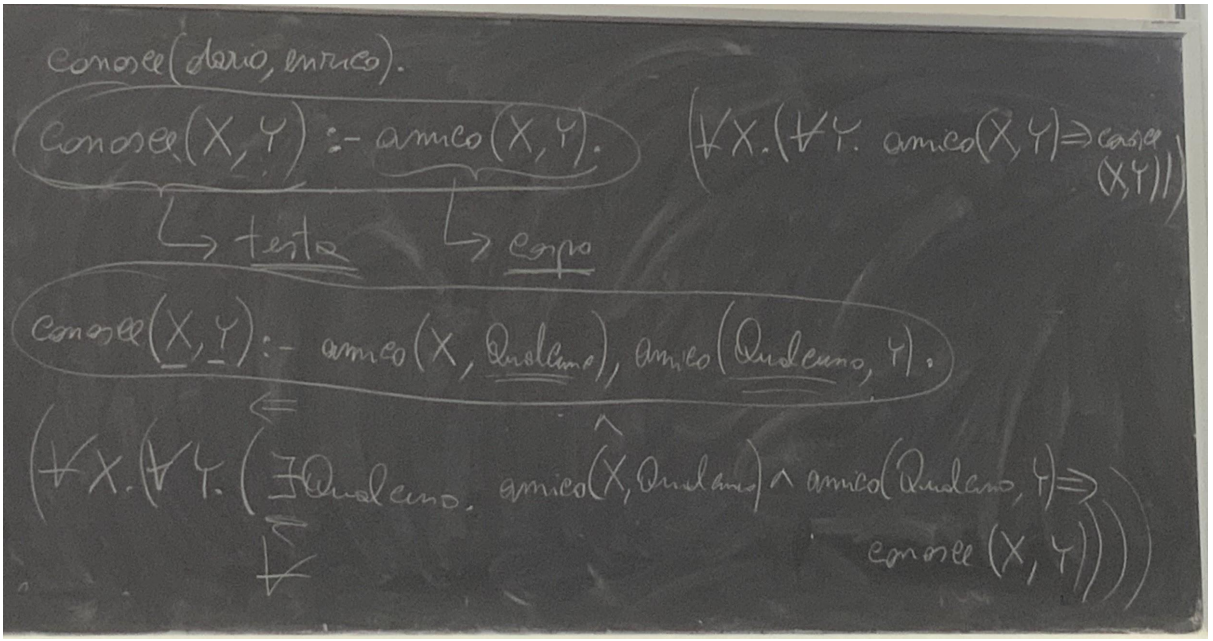
- Per ottenere gli argomenti messi come prompt basta porre una query con delle variabili

cd(1,artista1,album1,canzone1)
cd(2,artista2,album2,canzone2)
cd(3,artista3,album3,canzone3)

?-cd(2,artista2,album2,Cantante2)
Cantante2=canzone3

Regole: formule che ci consentano di fare dichiarazioni condizionali

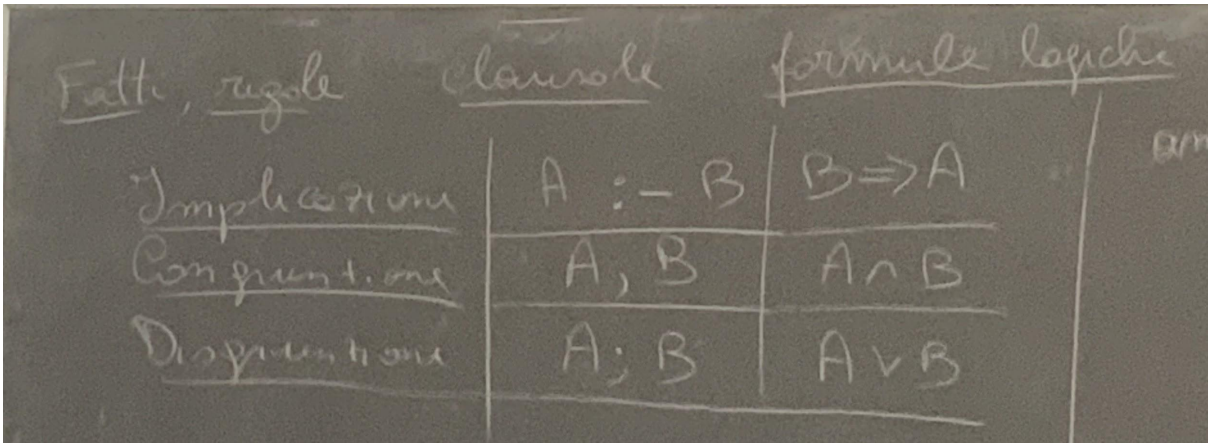
:- == (conseguenza)
amico(antonio,bianca)
amico(antonio,enrico)
amico(bianca,dario)
amico(dario,carla)
amico(enrico,bianca)
amico(antonio,Chiunque)



?-conosce(bianca,carla)
X=bianca Y=carla
amico(bianca,Qualcuno)Aamico(Qualcuno,carla)
Qualcuno=dario
amico(dario,carla)

---

4 DIC 2019



Fatti:
amico(antonio,X) /* Antonio è amico di tutti*/
(VX.amico(antonio,X)
pred(c1,c2,c3…,cn,x1,X2,...,Xn)
(VX1.(VX…(...(Vxmpred(pred(c1,c2,c3…,cn,x1,x2,...,xn))))))

conosce(X,Y):-amico(X,Qualcuno),amico(Qualcuno,Y)

/* per ogni X e Y se esiste Qualcuno tale che X è suo amico e Qualcuno è amico di Y allora X conosce Y */

(VX.(VY.(EQualcuno.amico(X,Qualcuno)andamico(Qualcuno,Y)->conosce(X,Y)))

A(X1,...,Xn):-B(X1,...Xn,Y1,...,Ym).
(VX1.(...(VXn.(EY….(EYm.B(X1,...Xm,Y1...Ym)->A(X1,....,Xn)))...)

- Le variabili nella testa Prolog vengono interpretate come “per ogni”. Quelle nel corpo sono interpretate come un esiste
- Se non interrompiamo il processo, Prolog cerca tutte le possibili soluzioni

QUERY:
?-conosce(bianca,X).
/* esiste X conosciuto da bianca? */
(EX.conosce(bianca,X))

?-goal(X1,...Xn).
(EX1.(E…(EXn.goal(X1...Xn))))?

- La funzione goal non restituisce True o False ma i valori che individua per le variabili della query

- Prolog lavora associando alle variabili dei valori presenti nel prompt e prova tutte le possibili combinazioni logiche

ESEMPIO
genitore(tommaso,francesco).
genitore(francesco,vittorio).
genitore(francesca,linda).
genitore(vittorio,bianca).
nonno(X,Y):-genitore(X,Z),genitore(Z,Y).

?-nonno(tommaso,bianca)
/* prolog scorre tutti i prompt e cerca una testa di nome ‘nonno’ */
X=tommasso Y=bianca Z=vittorio
True

?-nonno(tommaso,Chi)
X=tommaso Y=Chi Z=francesca

Fatti, regole e query sono costruiti sulla base di TERMINI:

1) Termini semplici:
    a) Costanti
    b) Variabili
2) Termini complessi:

a) Strutture
b)Liste

Costanti:

1) atomi:
   A) Un simbolo :- , ; .
   B) Nomi (sequenze di lettere/cifre) che iniziano con una lettera minuscola. Possono iniziare con la prima lettera maiuscola ma il nome deve andare tra apici es. ‘Paolo’
2) numeri: vengono trattati come tali

Variabili: definite tramite la prima lettera maiuscole

Strutture: atomo applicato ad una sequenza di termini -> atomo(termine1,...,termine2) dove ogni termine può a sua volta essere semplice o complesso. Tale atomo è detto funtore

es. libro(‘Le tigri di Mompracem’,autore(emilio,salgari))
nonno(X,maria)
figlio(X,Y):-genitore(Y,X).

Due termini possono essere unificati se:

1) sono lo stesso termine es. vittorio=vittorio tommaso!=vittorio
2) Hanno delle variabili che possono essere assegnate (istanziate) in modo che due termini rappresentino lo stesso termine es. tommaso = X (X=termine) nonno(X,Y) nonno(tommaso,Y) -> X=tommaso Y=Z

   Es. nonno(tommaso,X)=nonno(Y,linda)
   X=linda Y=tommaso
   Nonno(tommaso,X) nonno(vittorio,linda)
   X=linda
   Tommaso!=vittorio
   Nonno(tommaso,X)
   Nonno(x,linda)
   X=tommaso!=linda
   Es. p(X,f(c),X)
   P(f(Y),Y,Z) con X=f(y)

   P(f(Y),f(c),f(Y))
   P(f(Y),Y,Z) con Y=f(c), X=f(f(c))

   P(f(f(c)),f(c),f(f(c)))
   P(f(f(c))),f(c),t) Z=f(f(c))

---

5 DIC 2019

X e padre(X) possono essere unificati?
Prolog lo unifica facendo diventare X=padre(X) e padre(X)=padre(X) che non ha la stessa semantica del senso comune. L'unificazione continua ponendo X=padre(padre(X)) ma non si arriva mai ad una conclusione. Dunque si deve operare l'occur check per verificare come occorre la variabile nei termini da unificare ——> si può operare l'unificazione solo se la variabile X non occorre nel termine che segue es. se in input viene dato ?-X=padre(X), prolog non risponde né True né False ma riscrive X=padre(X)
Esiste però un comando unify_with_occur_check:

```
?_unify_with_occur_check(X,padre(X)).
/* prolog restituisce False */
False
```

- In Prolog non esistono i cicli come in Python dove si usano i cicli for e while
- Ricorsione -> si esprime un funtore in termine del funtore stesso
  es. in_prestito(rom,libro(le_tigri_di_Mompracem,autore(emilio,salgari)))
  Dove:
  rom -> termine semplice
  libro -> termine complesso (struttura a sua volta) dove libro lavora come funtore di altri due termini e dunque:
  le_tigri_di_Mompracem -> termine semplice
  autore -> termine complesso (struttura a sua volta) dove autore lavora come funtore di altri due termini e dunque:
  emilio -> termine semplice
  salgari -> termine complesso
  Se chiedo:
  ?-in_prestito(rossi,libro(X,autore(Y,Z))).
  /* Prolog opera un processo di unificazione con i dati che ha restituendo */
  X=le_tigri_di_Mompracem
  Y=emilio
  Z=salgari
  /* se nel database vi sono più dati relativi al termine rossi, cliccando su next o aggiungendo un punto e virgola si ottengono i vari risultati e le assegnazioni possibili */

  ?-in_prestito(X,libro(Y,autore(emilio,salgari)))
  /* Prolog procede ad assegnare X ed Y rispettivamente alla persona e al libro comprato relativo a quell'autore */
  X=rossi
  Y=le_tigri_di_Mompracem

Strutture con esemplificazione ad albero:
es. studenti(anna,roberto,paolo)
impiegato(rossi,indirizzo(via(pietroBucci),numero(32)))

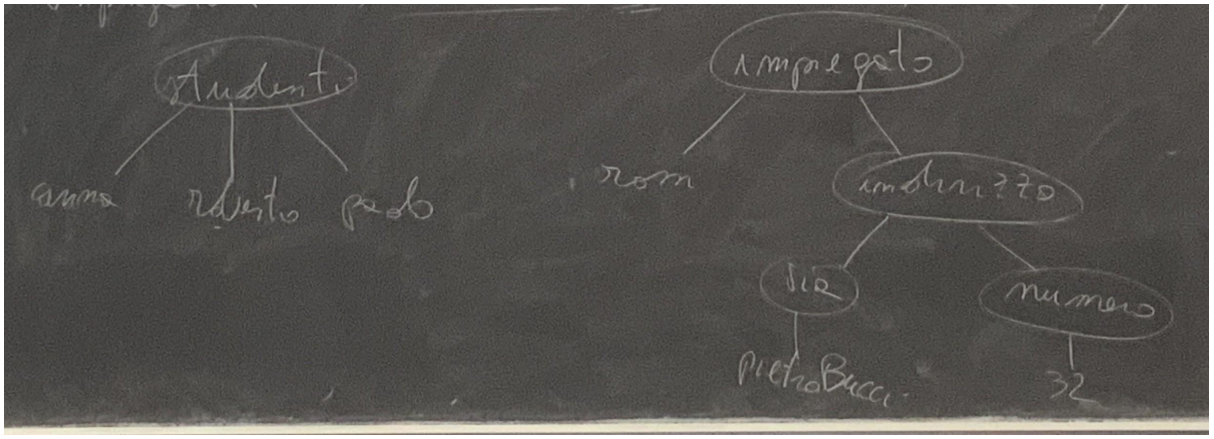
Studenti

——-> anna
——-> roberto
——-> paolo

Impiegato
——-> rossi
——-> indirizzo

Via
——-> pietroBucci

Numero
——-> 32



- Liste: sequenza di un numero finito di elementi separati da virgole e racchiusi tra parentesi quadre [ ]. Ogni elemento a sua volta può essere semplice o complesso e dunque anche una lista a sua volta
  es. [a,b,[c,d]] con a->termine semplice b->termine semplice [c,d]->lista di due termini semplici
  - [ ] -> lista vuota
  - Ogni lista ha una testa e una coda:
    - Testa: primo elemento della lista
    - Coda: sequenza degli elementi dal secondo in poi e cioè ciò che resta della lista tolta la testa

      Testa e coda si indicano attraverso la separazione con una riga verticale: [testa|coda]

es. [genova,milano, roma]
testa -> genova coda -> [milano,roma]

[10,20,[30,40],cinquanta]
testa -> 10 coda -> [20,[30,40],cinquanta]

[a,b,[C,D,[e]]]
testa -> a coda -> [b,[C,D,[e]]]

[]
testa -> - coda -> -

[a]
testa -> a coda -> []

[[a,b],c]
testa -> [a,b] coda -> [c]

La testa può essere un elemento, una lista ecc. La coda è SEMPRE una lista.

Assegnamento delle liste:
Se si chiede a prolog es. ?-[X,Y|Z]=[rossi,verdi] per il principio di unificazione si ha che X=rossi, Y=verdi, Z=[]
es. ?-[X,Y|Z]=[prolog,linp,pascal]
X=prolog
Y=linp
Z=[pascal] /* termine complesso -> lista */

?-[X,Y,Z]=[prolog,linp,pascal]
X=prolog
Y=linp
Z=pascal /* termine semplice */

[studenti,[Y,Z]]=[X,[mario,giovanni]]
X=studenti
Y=mario
Z=giovanni /* opera l'unificazione delle due liste in maniera ordinata e cioè seguendo l'ordine dato */

es. e_una_lista([questa,e,una,lezione,sulle,liste]).
e_una_lista([la,vispa,teresa,[avea,tra,l,erbetta]])
?-e_una_lista([X|Y]).
X=questa
Y=[e,una,lezione,sulle,liste]; Next
X=la
Y=[vispa,teresa,[avea,tra,l,erbetta]].

/*pongo ora una domanda con una variabile muta*/
/* la variabile muta non scrive nulla */
?-e_una_lista([_|X]).
X=[e,una,lezione,sulle,liste]; Next
X=[vispa,teresa,[avea,tra,l,erbetta]]

?-e_una_lista([_,_,_,[X|_]].
X=avea

---

10 DIC 2019

- Controlla se un certo elemento x appartiene ad una lista L:
  member(x,L)
  ?_member(a,[a,b,c]).
  True

  ?_member(d,[a,b,c]).
  False

- Concatenare due liste L1 e L2:
  append(L1,L2,L3)
  ?_append([a,b,c],[d,e,f],L).
  L=[a,b,c,d,e,f]

  ?_append(x,y,[a,b,c]).
  /* la concatenazione avrà successo per tali assegnamenti: */
  x=[ ]
  y=[a,b,c]; Next
  x=[a]
  y=[b,c]; Next
  x=[a,b]
  y=[c]; Next
  x=[a,b,c]
  y=[ ]

  ?_append(x,[1,2,3],[a,b,1,2,3]).
  x=[a,b]

ESERCIZIO n1:

Assumendo tali fatti formulare una query quale “quale corso piace a Simone?”:

- A simone piacciono soltanto i corsi facili.
- I corsi di scienze sono difficili
- Tutti i casi di Intelligenza Artificiale sono facili
- CK300 è un corso di Intelligenza Artificiale

corso(ck300,ia).
facile(X):-corso(X,ia).
/* se X è un corso di ia allora ciò implica che sia facile */

difficile(X):-corso(X,scienze).
piace(simone,X):-facile(X),corso(X,Y).

?_corso(X,Y),piace(simone,X).

ESERCIZIO n2:

Scrivere che collega(X,Y) è vera se X è un collega di Y dati il seguente prompt:

- lavora(X,Z) vera se X lavora nell'azienda Z
- X \== Y vero se X e Y sono diversi
- lavora(imp1,ibm).
- lavora(imp2,ibm).
- lavora(imp3,olivetti).
- lavora(imp4,txt).
- ?_collega(X,Y)
- X=imp1, Y=imp2

collega(X,Y):-lavora(X,Z),lavora(Y,Z), X\==Y.

ESERCIZIO n3:

Siano:

- finanz,univ,ebrown,bill delle costanti
- prof(X), stud(X), rabbia(X) dei predicati unitari
- taglia(X,Y), boccia(X,Y) dei predicati binari
- Se la finanza taglia i finanziamenti all'università, i prof si arrabbiano
- Se i prof si arrabbiano, gli studenti sono bocciati
- La finanza taglia i finanziamenti all'università
- E.Brown è una professoressa e Bill è uno studente
- Query: La prof E.Brown boccia lo studente Bill?

prof(ebrown).
stud(bill).
taglia(finanz,univ).
rabbia(X):-taglia(finanz,univ),prof(X).
boccia(X,Y):-prof(X),stud(Y),rabbia(X).

?_boccia(ebrown,bill).
True

ESERCIZIO n4:

Dati:

- padre(X,Y) X è padre di Y
- madre(X,Y) X è madre di Y

mostrare come ottenere le relazioni e verificare con l'albero genealogico della propria famiglia:

- nonno(X,Y), nonna(X,Y) bisnonno(X,Y) bisnonno(X,Y) nipote(X,Y) pronipote(X,Y) antenato(X,Y)

Svolgimento:
nonno(X,Y):-padre(X,Z),padre(Z,Y)
nonno(X,Y):-padre(X,Z),madre(Z,Y)

nonna(X,Y):-padre(X,Z),padre(Z,Y)
nonna(X,Y):-madre(X,Z),padre(Z,Y)

bisnonno(X,Y):-padre(X,Z),nonno(Z,Y)
bisnonno(X,Y):-padre(X,Z),nonno(Z,Y)

bisnonna(X,Y):-madre(X,Z),nonno(Z,Y)
bisnonna(X,Y):-madre(X,Z),nonna(Z,Y)

nipote(X,Y):-nonno(Y,X)
nipote(X,Y):-nonna(Y,X)