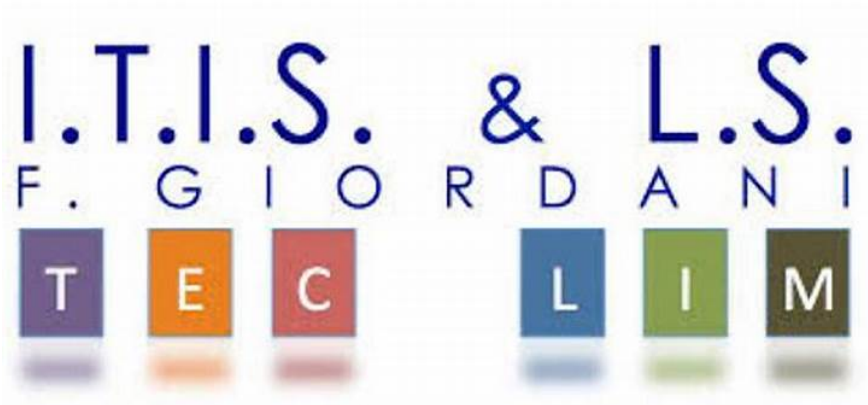


ITIS-LS "Francesco Giordani" Caserta

prof. Ennio Ranucci

a.s. 2019-2020

Semplici esercitazioni Prolog

Turbo Prolog 2.0

Paradigma Logico

- Programmazione dichiarativa (1970s) → enfasi sul cosa invece che sul come
- Domini applicativi
  - Basi di dati → SQL
  - Intelligenza artificiale → **Prolog**
- Paradigma logico: nasce dal bisogno di
  - processing linguaggio naturale → spec della grammatica
  - dimostrazione automatica di teoremi → spec del teorema
- **Prolog** = linguaggio principalmente usato nella programmazione logica
- Proprietà interessanti dei programmi logici
  - **Nondeterminismo** → più soluzioni del pb
  - **Backtracking** → incorporato nel linguaggio

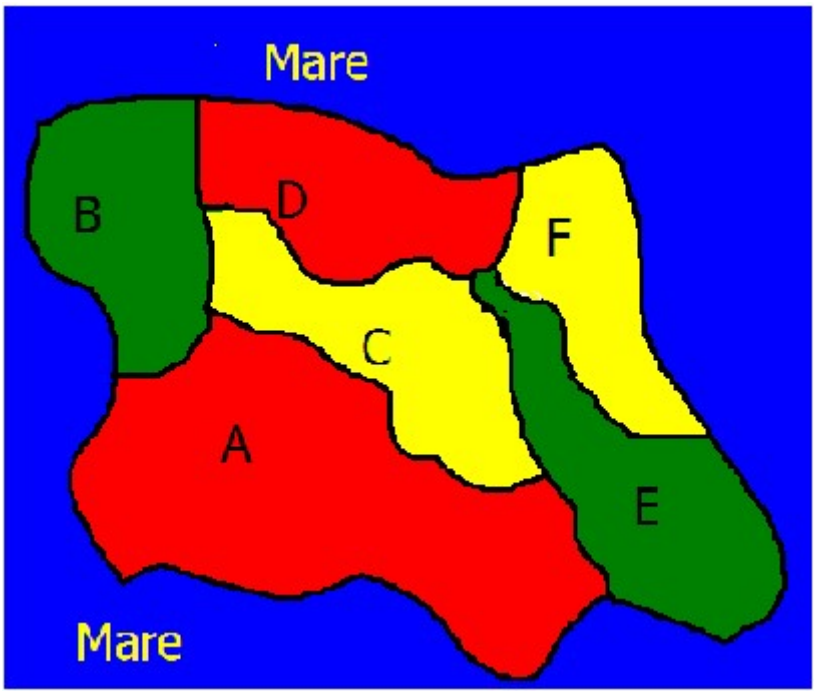
Link utili: *http://www.ce.unipr.it/research/HYPERPROLOG/manuale.html*

Programmazione logica

Il compito di un programmatore logico è idealmente quello di rappresentare tutti e soli i fatti del contesto considerato.

Esempio di base dei fatti elementari finita

Mappa dell'isola



Problema : rappresentare la mappa dell'isola e rispondere ai seguenti quesiti :

1. *Dire se 2 date regioni sono confinanti*
2. *Quali sono le regioni di colore rosso?*
3. *Quali regioni confinano con una data regione?*
4. *Quali regioni confinano con 2 date regioni?*
5. *Esistono regioni confinanti con lo stesso colore?*
6. *Esiste qualche regione che ha delle regioni dello stesso colore ad esso confinanti?*

Definizione della base di conoscenza in linguaggio naturale

A confina con B,C,E e con il mare ed è di colore rosso;

B confina con A,C,D e con il mare ed è di colore verde;

C confina con A,B,D,E ed è di colore giallo;

D confina con B,C,E,F e con il mare ed è di colore rosso;

E confina con A,C,D,F e con il mare ed è di colore verde;

F confina con E,D e con il mare ed è di colore giallo;

Il Mare è di colore blu.

Definizione della base di conoscenza decomposta in fatti elementari

Ogni riga può essere decomposta in fatti elementari .

A confina con B

A confina con C

A confina con E

A confina con il mare

A è di colore rosso

B confina con A

Ecc

Definizione della base di conoscenza in pseudocodifica

Confinanti(A,B);

Confinanti(A,C);

…

Colore(A,rosso);

Colore(B,verde);

…

Definizione della base di conoscenza in L3P

predic

colore(a,rosso).

colore(b,verde).

colore(c,giallo).

colore(d,rosso).

colore(e,verde).

colore(f,giallo).

colore(mare,blu).

confinanti(a,b).

confinanti(a,c).

confinanti(a,e).

confinanti(a,mare).

confinanti(b,a).

confinanti(b,c).

confinanti(b,d).

confinanti(b,mare).

confinanti(c,a).

confinanti(c,d).

confinanti(c,b).

confinanti(c,e).

confinanti(d,b).

confinanti(d,f).

confinanti(d,c).

confinanti(d,e).

confinanti(d,mare).

confinanti(e,a).

confinanti(e,f).

confinanti(e,c).

confinanti(e,mare).

confinanti(f,d).

confinanti(f,e).

confinanti(f,mare).

confinanti(e,d).

endpredic

La base di conoscenza è costituita da proposizioni atomiche .

Nella terminologia della programmazione logica confinanti e colore *sono nomi di* ***predicati*** .

*Nella programmazione logica vale **l'ipotesi di mondo chiuso** , cioè si assume che tutto quello che non è deducibile dalla base di conoscenza sia implicitamente falso.*

Esempio di interrogazione semplice

Dire se 2 date regioni sono confinanti .

?- confinanti(a,b).

Quali sono le regioni di colore rosso?.

?- colore(Regione,rosso).

Quali regioni confinano con una data regione?.

?- confinanti(a,Regione).

Esempio di interrogazione composta

Quali regioni confinano con 2 date regioni ?.

?- confinanti(a,Regione),confinanti(b,Regione).

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es1*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
Scrivere predicati e clausole. Verificare un obiettivo
Obiettivo del programma:
I fatti descrivono animali, vegetali e calcolatori. Verificare se la mucca è un animale.

Nota
Installare Turbo Prolog nella cartella c:\tpro
Turbo Prolog 2.0 è un programma per il sistema operativo Dos. Per utilizzarlo in amiente Win 10 è necessario installare dosBox e configurare il seguente
file:C:\Utenti\Nome_Utente\AppData\Local\DOSBox\dosbox-numero di versione.conf

aprirlo con blocco note, impostare a "true" la proprietà fullscreen ed aggiungere in coda i comandi dos:
mount c c:\dosBox
c:
cd tpro
prolog

```
/* esemp1: verifica di un fatto

 inserire il seguente obiettivo:

 animale(mucca)              */

domains

    cosa = symbol.

 predicates

      animale(cosa).

      vegetale(cosa).

      calcolatore(cosa).

clauses

     animale(mucca).

     animale(pipistrello).

     animale(iguana).

     vegetale(sequoia).

     vegetale(felce).

     calcolatore(pc_ibm).

     calcolatore(apple_macintosh).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es2*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
Scrivere predicati e clausole. Verificare un obiettivo che contiene una variabile
Obiettivo del programma:
I fatti descrivono animali, vegetali e calcolatori. Verificare se X è un animale.

```
/* esemp2:     variabili
 inserire il seguente obiettivo:
 animale(X)              */

/* E' possibile proporre gli obiettivi in
   modo diverso, facendo riferimento non
   a una proprietà precisa, ma a un oggetto
   indefinito, indicato da una variabile.
   Ogni identificatore che inizia con una
   lettera maiuscola viene considerata una
   variabile.
   es. animale(X)                */

/* Il prolog dapprima verifica se il predi-
   cato dell'obiettivo è uguale a quello
   della prima clausola ; se lo è, control-
   la se i due predicati hanno lo stesso
   numero di argomenti. Se anche una sola
   di queste condizioni non è verificata,
   abbandona la prima clausola e prosegue
   con la seconda; se invece l'obiettivo
   e la clausola in esame hanno lo stesso
   numero di argomenti e lo stesso predica-
   to, verifica se gli argomenti sono
```

```
  ordinatamente uguali.
  Nel caso di una variabile, che può stare
  al posto di qualsiasi argomento, l'inter-
  prete assegna temporaneamente alla varia-
  bile il corrispondente argomento della
  clausola in esame. Poi passa alla clausola
  successiva.                    */

domains
    cosa = symbol

predicates
        animale(cosa)
        vegetale(cosa)
        calcolatore(cosa)
clauses
      animale(mucca).
      animale(pipistrello).
      animale(iguana).
      vegetale(sequoia).
      vegetale(felce).
      calcolatore(pc_ibm).
      calcolatore(apple_macintosh).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es3*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
Scrivere predicati, clausole e regole. Verificare un obiettivo
Obiettivo del programma:
I fatti descrivono chi guarda e se rende felice o nervoso l'altro. Verificare chi è felice.

```
/* esemp3:      regole

 la sezione delle clausole, oltre ai fatti può

 contenere anche delle regole.

 Una regola afferma che un obiettivo è

 soddisfatto se determinate condizioni

 sono vere.                    */

 /* goal: felice(Chi)

    goal: nervoso(Chi)

    goal: nervoso(chi)              */

domains
       persona = symbol
predicates
       guarda(persona,persona)
       felice(persona)
       nervoso(persona)
clauses

      guarda(bruno,roberto).

      guarda(gianni,lucia).

      guarda(fabio,carla).

      guarda(massimo,bruno).

      guarda(beatrice,giorgio).

      guarda(roberto,roberto).

      guarda(fabio,giorgio).

      guarda(bruno,filippo).

   felice(paolo) if guarda(beatrice,paolo).

   felice(giorgio) if guarda(beatrice,giorgio).

    nervoso(Chi) if guarda(bruno,Chi).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es4*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
Scrivere predicati, clausole e regole. Verificare un obiettivo
Obiettivo del programma:
I fatti descrivono I fratelli e le sorelle. Verificare fratelli e sorelle.

```
/* esemp4: programma fratello_sorella */

/* trace */

domains

    persona = symbol


predicates

     fratello_sorella(persona,persona,persona)

     uomo(persona,persona)

     donna(persona,persona)



clauses

    uomo(daniele,diana).

    uomo(lorenzo,laura).

    uomo(michele,barbara).

    uomo(pietro,gianna).

    donna(linda,stefania).

    donna(anna,laura).

    fratello_sorella(X,Y,M) if

                uomo(X,M),donna(Y,M).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es5*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
backtracking e tracciamento
Obiettivo del programma:
I fatti descrivono chi guarda e se rende felice o nervoso l'altro. Verificare chi è felice.

```
/* esemp5:   backtracking e tracciamento

   per ottenere una traccia dell'esecuzione

   aggiungere la parola TRACE sopra la riga

   domains.

   CALL:felice(_) significa che il primo

   passo per soddisfare l'obiettivo consiste

   nel cercare un predicato felice con un

   argomento qualsiasi.

   CALL:guarda(beatrice,paolo) significa

   che guarda(beatrice,paolo) è stato

   assunto come sottoobiettivo.

   REDO:guarda(beatrice,paolo) significa

   che non ha potuto unificare il sotto-

   obiettivo con il primo fatto,perchè

   pur essendo uguali il predicato e l'arieta',

   gli argomenti sono diversi;percio tenta con

   la clausola successiva.

   fail:guarda(beatrice,paolo) dichiara che

   la regola felice(paolo) if guarda(beatrice,

   paolo) non permette di soddisfare l'obiet-

   tivo originale felice(Chi).

   REDO:felice(_) l'obiettivo felice(Chi)

   viene ripreso in considerazione

   (backtracking)

   RETURN segnala un successo
```

```
                              */
 /* goal: felice(Chi)
    goal: nervoso(Chi)
    goal: nervoso(chi)                */
domains
     persona = symbol

predicates
        guarda(persona,persona)
        felice(persona)
        nervoso(persona)

clauses
       guarda(bruno,roberto).
       guarda(gianni,lucia).
       guarda(fabio,carla).
       guarda(massimo,bruno).
       guarda(beatrice,giorgio).
       guarda(roberto,roberto).
       guarda(fabio,giorgio).
       guarda(bruno,filippo).

   felice(paolo) if guarda(beatrice,paolo).
   felice(giorgio) if guarda(beatrice,giorgio).
   nervoso(Chi) if guarda(bruno,Chi).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es6*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
variabile anonima
Obiettivo del programma:
I fatti descrivono chi guarda. Verificare se qualcuno guarda.

```
/* esemp6 : variabile anonima

          è rappresentata dallo spazio

          sottolineato, funziona come

          una variabile, ma ad essa non

          viene mai assegnato un valore

  es. goal: guarda(beatrice,_)

          guarda(_,_)

                                  */

domains

     persona = symbol


predicates

       guarda(persona,persona)


clauses

       guarda(bruno,roberto).

       guarda(gianni,lucia).

       guarda(fabio,carla).

       guarda(massimo,bruno).

       guarda(beatrice,giorgio).

       guarda(roberto,roberto).

       guarda(fabio,giorgio).

       guarda(bruno,filippo).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es7*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
goal che contengono connettivi logici
Obiettivo del programma:
I fatti descrivono chi guarda. Verificare un obiettivo composto da due obiettivi.

```
/* esemp7 : obiettivi composti


  goal : guarda(bruno,roberto) and

       guarda(gianni,lucia)

  goal : guarda(X,roberto) and

       guarda(gianni,Y)

                              */


domains


    persona = symbol


predicates

      guarda(persona,persona)


clauses

     guarda(bruno,roberto).

     guarda(gianni,lucia).

     guarda(fabio,carla).

     guarda(massimo,bruno).

     guarda(beatrice,giorgio).

     guarda(roberto,roberto).

     guarda(fabio,giorgio).

     guarda(bruno,filippo).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es8*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
regole composte
Obiettivo del programma:
I fatti descrivono chi miagola, chi fa le fusa, chi graffia. Verificare chi è gatto perchè miagola, fa le fusa e graffia.

```
/* esemp8:                */

domains

    animale = symbol


predicates

      è_un_gatto(animale)

      miagola(animale)

      fa_le_fusa(animale)

      graffia(animale)


clauses

    è_un_gatto(X) if miagola(X),

                fa_le_fusa(X),

                graffia(X).

    miagola(pallino).

    miagola(silvestro).

    fa_le_fusa(pallino).

    fa_le_fusa(silvestro).

    graffia(pallino).

    graffia(silvestro).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es9*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
taglio o cut
Obiettivo del programma:
I fatti descrivono chi miagola, chi fa le fusa, chi graffia. Verificare chi è gatto perchè miagola, fa le fusa e graffia.

```
/* esemp9: taglio

        il taglio inpedisce all'interprete

        di effettuare il backtracking

        per cercare un'altra soluzione    */

domains

    animale = symbol

 predicates

       è_un_gatto(animale)

       miagola(animale)

       fa_le_fusa(animale)

       graffia(animale)

clauses

     è_un_gatto(X) if

                 miagola(X),

                 fa_le_fusa(X),

                 graffia(X),

                 !.

     miagola(pallino).

     miagola(silvestro).

     fa_le_fusa(pallino).

     fa_le_fusa(silvestro).

     graffia(pallino).

     graffia(silvestro).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es10*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
regole annidate
Obiettivo del programma:
I fatti descrivono chi può guidare, chi ha l'auto, chi si fa dare un passaggio, chi ha bevuto, chi ha la patente, chi è amico, chi puo andare a piedi. Verificare chi può andare a casa perchè può guidare ed ha la macchina oppure può farsi dare un passaggio oppure, può andare a piedi.

```
/* esemp10: */

domains

    persona = symbol


predicates

       può_andare_a_casa(persona)

       può_guidare(persona)

       ha_la_macchina(persona)

       può_farsi_dare_un_passaggio(persona)

       può_andare_a_piedi(persona)

       non_ha_bevuto(persona)

       ha_la_patente(persona)

       amici(persona,persona)


clauses

      può_andare_a_casa(Nome1) if

        può_guidare(Nome1),

        ha_la_macchina(Nome1);

        può_farsi_dare_un_passaggio(Nome1);

        può_andare_a_piedi(Nome1).


     può_guidare(Nome1) if

        non_ha_bevuto(Nome1) and
```

```
   ha_la_patente(Nome1).
 può_farsi_dare_un_passaggio(Nome1) if
   amici(Nome1,Nome2) and
   può_guidare(Nome2).

può_andare_a_piedi(paolo).
può_andare_a_piedi(federica).

ha_la_macchina(filippo).
ha_la_macchina(federica).
ha_la_macchina(riccardo).

non_ha_bevuto(federica).
non_ha_bevuto(paolo).

ha_la_patente(federica).
ha_la_patente(riccardo).
ha_la_patente(filippo).

amici(filippo,federica).
amici(riccardo,filippo).
amici(federica,paolo).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es11*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
liste, testa, coda
Obiettivo del programma: *utilizzare una lista*

```
/*esemp11: liste

  goal: vettore([Testa|Coda])

                        */

domains

    lista_di_numeri = integer*

predicates

    vettore(lista_di_numeri)

clauses

    vettore([1,2,3,4,5]).
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 4^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es12*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
ricerca in profondità
Obiettivo del programma: *utilizzare un database*

```
/* esemp12: ricerca in profondit… */

database

   volo(symbol,symbol,integer)

   visitata(symbol)

   sulla_rotta(symbol)


predicates

   intestazione

   trova_rotta

   rotta(symbol,symbol,integer)

   esiste_volo(symbol,symbol,integer)

   aggiungi_alla_rotta(symbol)

   aggiungi_alle_visitate(symbol)

   visualizza_visitate

   visualizza_rotta

   pulisci


goal

   assert(volo(newyork,chicago,1000)),

   assert(volo(newyork,toronto,800)),

   assert(volo(newyork,denver,1900)),

   assert(volo(chicago,denver,1000)),

   assert(volo(toronto,losangeles,1800)),

   assert(volo(toronto,chicago,500)),

   assert(volo(toronto,calgary,1500)),
```

```
   assert(volo(denver,losangeles,1000)),
   assert(volo(denver,houston,1500)),
   assert(volo(denver,urbana,1000)),
   assert(volo(houston,losangeles,1500)),
   intestazione,
   trova_rotta,
   nl,
   nl,
   write("ancora? "),
   readln(R),
   R=n,
   pulisci.

clauses
 /* intestazione video */
 intestazione if
   clearwindow,
   write("----------------------------\n"),
   write("Ricerca in profondit…\n"),
   write("----------------------------\n"),
   nl.

 /* richiesta partenza e destinazione */
 trova_rotta if
   write("Da: "),
   readln(Part),
   write("A: "),
   readln(Dest),
   rotta(Part,Dest,Dist),
nl, write("La distanza Š di ",Dist," miglia"),
nl,
visualizza_visitate,
```

```
nl,
visualizza_rotta,
nl.

/* si prepara a trovare la rotta */
  rotta(Part,Dest,Dist) if
    esiste_volo(Part,Dest,Dist).

/* o a segnalare il fallimento */
  rotta(_,_,_) if
  nl,
  write("Non esistono altre rotte"),
  nl,
  pulisci,
  fail.

/* controlla se esiste una connessione tra
  le due citt… */

  esiste_volo(Part,Dest,Dist) if
     volo(Part,Dest,Dist),
     aggiungi_alle_visitate(Part),
     aggiungi_alle_visitate(Dest),
     aggiungi_alla_rotta(Part),
     aggiungi_alla_rotta(Dest).

  esiste_volo(Part,Dest,Dist) if
     volo(Part,X,Dist2),
     X<>Dest,
     aggiungi_alle_visitate(Part),
     aggiungi_alle_visitate(X),
     aggiungi_alla_rotta(Part),
```

```
    esiste_volo(X,Dest,Dist3),
    Dist=Dist2+Dist3.
  esiste_volo(Part,_,_) if
    write("Vicolo cieco a ", Part),
    nl,
    fail.

/* aggiunge alla lista delle citt… visitate */

  aggiungi_alle_visitate(Citt…) if
    not(visitata(Citt…)),
    assert(visitata(Citt…)),
    !.
  aggiungi_alle_visitate(_).

/* aggiunge alla lista delle citt… sulla rotta */

  aggiungi_alla_rotta(Citt…) if
    assert(sulla_rotta(Citt…)).
  aggiungi_alla_rotta(Citt…) if
    retract(sulla_rotta(Citt…)),
    fail.

/* si prepara per una nuova esecuzione */

  pulisci if
    retract(volo(_,_,_)),
    fail.
  pulisci if
    retract(visitata(_)),
    fail.
  pulisci if
```

```
    retract(sulla_rotta(_)),
    fail.
  pulisci.

/* visualizza le città visitate */

  visualizza_visitate if
    nl,
    write("Le citt… visitate finora sono: "),
    nl,
    visitata(Citt…),
    write(Citt…," "),
    fail.
    visualizza_visitate.

/* visualizza le citt… che si trovano sulla
  rotta */

  visualizza_rotta if
    nl,
    write("La rotta Š: "),
    nl,
    sulla_rotta(Citt…),
    write(Citt…," "),
    fail.
  visualizza_rotta.
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 4^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es13*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Turbo prolog 2.0*
Obiettivo didattico:
funtori
Obiettivo del programma: *utilizzarei funtori*

```
/* il prolog  prevede delle strutture più
   complesse, gli oggetti composti,
   in cui un predicato diventa
   l'argomento di un altro predicato.
   ad esempio:
   predicato1(argomento1,predicato2(argo-
   mento2,argomento3)).
   Quello che in questo caso dovrebbe
   essere il secondo argomento del
   predicato, è in pratica diventato
   un altro predicato che prende il
   nome di FUNTORE.
   Il funtore ed i tipi dei suoi
   argomenti,che possono essere
   a loro volta degli oggetti composti,
   devono essere dichiarati nella sezione
   Domains.
   */
domains
    pers,giorno,materia = symbol
    valore = turno(giorno,materia)
predicates
    studente(pers)
```

```
   interrogato(pers,valore)
clauses
   studente(maria).
   studente(anna).
   studente(paola).
   studente(luigi).
   studente(marco).
   studente(matteo).
   studente(tommaso).
   studente(francesco).
 interrogato(maria,turno(lunedi,storia)).
 interrogato(luigi,turno(lunedi,storia)).
 interrogato(tommaso,turno(lunedi,matematica)).
 interrogato(paola,turno(lunedi,fisica)).
 interrogato(francesco,turno(martedi,italiano)).
 interrogato(anna,turno(martedi,italiano)).
 interrogato(matteo,turno(martedi,inglese)).
 interrogato(luigi,turno(mercoledi,matematica)).
 interrogato(anna,turno(mercoledi,informatica)).
 interrogato(marco,turno(venerdi,italiano)).
 interrogato(tommaso,turno(venerdi,italiano)).

/* goals:
   interrogato(paola,Quando)
   interrogato(Chi,turno(lunedi,_)
   interrogato(Chi,turno(_,italiano)
   */
```

```
        /* LCC1 */

DATABASE
 sulla_riva_sx(symbol)
 sulla_riva_dx(symbol)

PREDICATES
 inizializza
 cancella_riva_sx
 ripeti
 esegui_mossa_lecita
 applica_mossa(symbol)
 sit_legale_a_dx(symbol)
 sit_legale_a_sx
 fiume_attraversato

CLAUSES
 inizializza:-
 cancella_riva_sx,
 assert(sulla_riva_dx(lupo)),
 assert(sulla_riva_dx(capra)),
 assert(sulla_riva_dx(cavolo)).

cancella_riva_sx:-
 retract(sulla_riva_sx(_)),
 fail.
cancella_riva_sx.

fiume_attraversato:-
 ripeti,
 esegui_mossa_lecita,
 sulla_riva_sx(lupo),
```

```
  sulla_riva_sx(capra),
  sulla_riva_sx(cavolo).

ripeti.
  ripeti:-ripeti.

esegui_mossa_lecita:-
  sulla_riva_dx(X),
  sit_legale_a_dx(X),
  applica_mossa(X).

applica_mossa(X):-
  retract(sulla_riva_dx(X)),
  assert(sulla_riva_sx(X)),
  sit_legale_a_sx,
  !.

applica_mossa(X):-
  sulla_riva_sx(Y),
  Y<>X,
  retract(sulla_riva_sx(Y)),
  assert(sulla_riva_dx(Y)),
  !.

sit_legale_a_dx(lupo):-
  sulla_riva_sx(capra);
  sulla_riva_sx(cavolo).
  sit_legale_a_dx(capra).
  sit_legale_a_dx(cavolo):-
  sulla_riva_sx(lupo);
  sulla_riva_sx(capra).
```

```
sit_legale_a_sx:-
  sulla_riva_dx(lupo),
  sulla_riva_dx(cavolo),
  !;
  sulla_riva_sx(lupo),
  sulla_riva_sx(cavolo),
  !;
  sulla_riva_dx(capra).
```

```
        /* LCC2 */
DATABASE
 sulla_riva_sx(symbol)
 sulla_riva_dx(symbol)
 pos_contadino(symbol)

PREDICATES
 fiume_attraversato
 ripeti
 esegui_mossa_lecita
 applica_mossa(symbol)
 sit_legale_a_dx(symbol)
 sit_legale_a_sx
 pulisci
 inizializza
 disegna_fiume
 visu_mossa(symbol,symbol)
 muovi_a_sx(symbol,integer,integer,integer)
 muovi_a_dx(symbol,integer,integer,integer)
 contadino_a_dx
 sposta_contadino(integer)

GOAL
 assert(sulla_riva_dx(lupo)),
 assert(sulla_riva_dx(capra)),
 assert(sulla_riva_dx(cavolo)),
 fiume_attraversato,
 pulisci.

CLAUSES
fiume_attraversato:-
```

```
inizializza,
ripeti,
esegui_mossa_lecita,
sulla_riva_sx(lupo),
sulla_riva_sx(capra),
sulla_riva_sx(cavolo),
cursor(20,3),
write("Sono tutti sulla riva sx sani e salvi"),
 nl,
 nl.

ripeti.
ripeti:-ripeti.

esegui_mossa_lecita:-
sulla_riva_dx(X),
sit_legale_a_dx(X),
applica_mossa(X).

applica_mossa(X):-
contadino_a_dx,
retract(sulla_riva_dx(X)),
visu_mossa(X,sx),
assert(sulla_riva_sx(X)),
sit_legale_a_sx,
!.

applica_mossa(X):-
sulla_riva_sx(Y),
Y<>X,
retract(sulla_riva_sx(Y)),
visu_mossa(Y,dx),
```

```
 assert(sulla_riva_dx(Y)),
 !.

sit_legale_a_dx(lupo):-
 sulla_riva_sx(capra),!;
 sulla_riva_sx(cavolo).
 sit_legale_a_dx(capra).
 sit_legale_a_dx(cavolo):-
 sulla_riva_sx(lupo),!;
 sulla_riva_sx(capra).

sit_legale_a_sx:-
 sulla_riva_dx(lupo),
 sulla_riva_dx(cavolo),
 !;
 sulla_riva_sx(lupo),
 sulla_riva_sx(cavolo),
 !;
 sulla_riva_dx(capra).

pulisci:-
 retract(sulla_riva_sx(_)),
 fail.
pulisci:-
 retract(sulla_riva_dx(_)),
 fail.
pulisci:-
 retract(pos_contadino(_)),
 fail.
pulisci.

inizializza:-
```

```
 graphics(1,1,0),
 disegna_fiume,
 cursor(6,25), write("contadino"),
 cursor(9,25), write("lupo"),
 cursor(12,25), write("capra"),
 cursor(15,25), write("cavolo"),
 cursor(0,4),
 write("contadino, lupo, capra e cavolo").

disegna_fiume:-
 line(5000,12000,25000,12000,2),
 line(5000,18000,25000,18000,2).

visu_mossa(lupo,sx):-
 muovi_a_sx(lupo,9,25,5).
 visu_mossa(capra,sx):-
 muovi_a_sx(capra,12,25,5).
 visu_mossa(cavolo,sx):-
 muovi_a_sx(cavolo,15,25,5).
 visu_mossa(lupo,dx):-
 muovi_a_dx(lupo,9,5,25).
 visu_mossa(capra,dx):-
 muovi_a_dx(capra,12,5,25).
 visu_mossa(cavolo,dx):-
 muovi_a_dx(cavolo,15,5,25).

muovi_a_sx(T,X1,Y1,Y2):-
 Y1=Y2,
 cursor(X1,Y1), write("         "),
 cursor(X1,Y1), write(T),
 cursor(6,Y1),  write("         "),
 cursor(6,Y1), write("contadino"),
```

```
 assert(pos_contadino(sx)),
 !.

muovi_a_sx(T,X1,Y1,Y2):-
 cursor(X1,Y1), write("        "),
 cursor(X1,Y1), write(T),
 cursor(6,Y1),  write("        "),
 cursor(6,Y1), write("contadino"),
 disegna_fiume,
 Temp=Y1-1,
 muovi_a_sx(T,X1,Temp,Y2).

muovi_a_dx(T,X1,Y1,Y2):-
 Y1=Y2,
 Temp=Y1-1,
 cursor(X1,Temp), write("        "),
 cursor(X1,Y1), write(T),
 cursor(6,Temp),  write("        "),
 cursor(6,Y1), write("contadino"),
 retract(pos_contadino(sx)),
 !.
muovi_a_dx(T,X1,Y1,Y2):-
 Temp=Y1-1,
 cursor(X1,Temp), write("        "),
 cursor(X1,Y1), write(T),
 cursor(6,Temp),  write("        "),
 cursor(6,Y1), write("contadino"),
 disegna_fiume,
 Temp2=Y1+1,
 muovi_a_dx(T,X1,Temp2,Y2).

contadino_a_dx:-
```

```
 not(pos_contadino(sx)),
 !.
contadino_a_dx:-
 sposta_contadino(5).
 sposta_contadino(Y):-
 Y=25,
 cursor(6,24), write("          "),
 cursor(6,25), write("contadino"),
 retract(pos_contadino(sx)),
 disegna_fiume,
 !.
sposta_contadino(Y):-
 Temp=Y-1,
 cursor(6,Temp), write("          "),
 cursor(6,Y), write("contadino"),
 disegna_fiume,
 Temp2=Y+1,
 sposta_contadino(Temp2).
```

```
        /* MEC1 */
DOMAINS
 Primopasseg,Secpasseg =string
DATABASE
 sulla_riva_sx(symbol)
 sulla_riva_dx(symbol)
 m_cancellati(symbol)
 c_cancellati(symbol)
PREDICATES
 inizializza
 cancella_riva_sx
 ripeti
 esegui_mossa_lecita
 applica_mossa(symbol,symbol)
 sit_legale_a_dx(symbol,symbol)
 rimetti_c_dx
 rimetti_m_dx
 fiume_attraversato
 contac_dx(integer,integer)
 contam_dx(integer,integer)
 tutto_a_sx

CLAUSES
 inizializza:-
 cancella_riva_sx,
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(c)),
 assert(sulla_riva_dx(c)),
 assert(sulla_riva_dx(c)).
```

```
cancella_riva_sx :-
  retract(sulla_riva_sx(_)),
  fail.
cancella_riva_sx.

contac_dx(Cont,N):-
  retract(sulla_riva_dx(c)),
  assert(c_cancellati(c)),
  Temp=Cont+1,
  contac_dx(Temp,N).
contac_dx(N,N).

rimetti_c_dx :-
  c_cancellati(X),
  X=c,
  retract(c_cancellati(X)),
  assert(sulla_riva_dx(X)),
rimetti_c_dx.
rimetti_c_dx.

contam_dx(Cont,P):-
  retract(sulla_riva_dx(m)),
  assert(m_cancellati(m)),
  Temp=Cont+1,
  contam_dx(Temp,P).
contam_dx(P,P).

rimetti_m_dx :-
  m_cancellati(X),
  X=m,
  retract(m_cancellati(X)),
  assert(sulla_riva_dx(X)),
```

```
rimetti_m_dx.
rimetti_m_dx.

fiume_attraversato:-
  ripeti,
  esegui_mossa_lecita,
  tutto_a_sx.

ripeti.
  ripeti:-ripeti.

tutto_a_sx:-
  contac_dx(0,N),
  contam_dx(0,P),!,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N=3,
  P=3.

esegui_mossa_lecita:-
  write("batti mossa "),
  readln(Primopasseg),
  readln(Secpasseg),
  sit_legale_a_dx(Primopasseg,Secpasseg),!,
  applica_mossa(Primopasseg,Secpasseg),!.
esegui_mossa_lecita.

applica_mossa(X,Y):-
  assert(sulla_riva_sx(X)),
  assert(sulla_riva_sx(Y)),
  retract(sulla_riva_dx(X)),
  retract(sulla_riva_dx(Y)).
```

```
applica_mossa(_,_).
/*  sit_legale_a_sx,
  !.*/

/*applica_mossa(X,Y):-
  retract(sulla_riva_sx(X),
  assert(sulla_riva_dx(X),
  retract(sulla_riva_sx(Y)),
  assert(sulla_riva_dx(Y)),
  !.*/

sit_legale_a_dx(X,Y):-
  X=m,
  Y=m,
  retract(sulla_riva_dx(m)),
  retract(sulla_riva_dx(m)),
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  assert(sulla_riva_dx(m)),!,
  assert(sulla_riva_dx(m)),!,
  N<P.

sit_legale_a_dx(X,Y):-
  X=c,
  Y=c,
  contac_dx(0,N),
  !,
  rimetti_c_dx,!,
  N>=2.
```

```
sit_legale_a_dx(X,Y):-
  X=c,
  Y=m;
  X=m,
  Y=c,
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N>=1,
  P>=1.

sit_legale_a_dx(X,Y):-
  X=c,
  Y=(""),
  contac_dx(0,N),
  !,
  rimetti_c_dx,!,
  N>=1.
```

```
/* MEC2 */
DOMAINS
 Primopasseg,Secpasseg =string
DATABASE
 sulla_riva_sx(symbol)
 sulla_riva_dx(symbol)
 m_cancellati(symbol)
 c_cancellati(symbol)
PREDICATES
 inizializza
 cancella_riva_sx
 ripeti
 esegui_mossa_lecita
 sit_legale_a_dx(symbol,symbol)
 rimetti_c_dx
 rimetti_m_dx
 fiume_attraversato
 contac_dx(integer,integer)
 contam_dx(integer,integer)
 sit_legale_a_sx(symbol,symbol)
 rimetti_c_sx
 rimetti_m_sx
 contac_sx(integer,integer)
 contam_sx(integer,integer)
 tutto_a_sx

CLAUSES
 inizializza:-
 cancella_riva_sx,
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
```

```
  assert(sulla_riva_dx(c)),
  assert(sulla_riva_dx(c)),
  assert(sulla_riva_dx(c)).

cancella_riva_sx :-
  retract(sulla_riva_sx(_)),
  fail.
cancella_riva_sx.

contac_dx(Cont,N):-
  retract(sulla_riva_dx(c)),
  assert(c_cancellati(c)),
  Temp=Cont+1,
  contac_dx(Temp,N).
contac_dx(N,N).

contac_sx(Cont,N):-
  retract(sulla_riva_sx(c)),
  assert(c_cancellati(c)),
  Temp=Cont+1,
  contac_sx(Temp,N).
contac_sx(N,N).

rimetti_c_dx :-
  c_cancellati(X),
  X=c,
  retract(c_cancellati(X)),
  assert(sulla_riva_dx(X)),
rimetti_c_dx.
rimetti_c_dx.

rimetti_c_sx :-
```

```
  c_cancellati(X),
  X=c,
  retract(c_cancellati(X)),
  assert(sulla_riva_sx(X)),
rimetti_c_sx.
rimetti_c_sx.

contam_dx(Cont,P):-
  retract(sulla_riva_dx(m)),
  assert(m_cancellati(m)),
  Temp=Cont+1,
  contam_dx(Temp,P).
contam_dx(P,P).

contam_sx(Cont,P):-
  retract(sulla_riva_sx(m)),
  assert(m_cancellati(m)),
  Temp=Cont+1,
  contam_sx(Temp,P).
contam_sx(P,P).

rimetti_m_dx :-
  m_cancellati(X),
  X=m,
  retract(m_cancellati(X)),
  assert(sulla_riva_dx(X)),
rimetti_m_dx.
rimetti_m_dx.

rimetti_m_sx :-
  m_cancellati(X),
  X=m,
```

```
  retract(m_cancellati(X)),
  assert(sulla_riva_sx(X)),
rimetti_m_sx.
rimetti_m_sx.

fiume_attraversato:-
  ripeti,
  esegui_mossa_lecita,
  tutto_a_sx.

ripeti.
  ripeti:-ripeti.

tutto_a_sx:-
  contac_dx(0,N),!,
  contam_dx(0,P),!,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N=0,!,
  P=0,!.

esegui_mossa_lecita:-
  write("batti mossa "),
  readln(Primopasseg),
  readln(Secpasseg),
  sit_legale_a_dx(Primopasseg,Secpasseg),
  sit_legale_a_sx(Primopasseg,Secpasseg).
esegui_mossa_lecita.

sit_legale_a_dx(X,Y):-
  X=m,
  Y=m,
```

```
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N>=P+2,!,
  assert(sulla_riva_sx(m)),!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(m)),!,
  retract(sulla_riva_dx(m)),!.

sit_legale_a_dx(X,Y):-
  X=c,
  Y=c,
  contac_dx(0,N),
  rimetti_c_dx,!,
  N>=2,!,
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(c)),!,
  retract(sulla_riva_dx(c)),!,
  retract(sulla_riva_dx(c)),!.

sit_legale_a_dx(X,Y):-
  X=c,
  Y=m,
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N>=1,!,
  P>=1,!,
```

```
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(c)),!,
  retract(sulla_riva_dx(m)),!.

sit_legale_a_dx(X,Y):-
  X=c,
  Y=(""),
  contac_dx(0,N),
  !,
  rimetti_c_dx,!,
  N>=1,!,
  assert(sulla_riva_sx(c)),!,
  retract(sulla_riva_dx(c)),!.

sit_legale_a_dx(X,Y):-
  X=m,!,
  Y=(""),!,
  contac_dx(0,N),!,
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N>=P+1,!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(m)),!.

 sit_legale_a_sx(X,Y):-
  X=m,
  Y=m,
  contac_sx(0,N),
  contam_sx(0,P),
```

```
 !,
 rimetti_c_sx,!,
 rimetti_m_sx,!,
 N>P+2,!,
 write("situazione illegale a sx"),!,
 retract(sulla_riva_sx(m)),!,
 retract(sulla_riva_sx(m)),!,
 assert(sulla_riva_dx(m)),!,
 assert(sulla_riva_dx(m)),!.

sit_legale_a_sx(X,Y):-
 X=c,
 Y=c,
 contac_sx(0,N),
 contam_sx(0,P),
 !,
 rimetti_c_sx,!,
 rimetti_m_sx,!,
 P>0,!,
 N>P,!,
 write("situazione illegale a sx"),!,
 retract(sulla_riva_sx(c)),!,
 retract(sulla_riva_sx(c)),!,
 assert(sulla_riva_dx(c)),!,
 assert(sulla_riva_dx(c)),!.

sit_legale_a_sx(c,m).

sit_legale_a_sx(X,Y):-
 X=c,
 Y=(""),
 contac_sx(0,N),
```

```
contam_sx(0,P),
!,
rimetti_c_sx,!,
rimetti_m_sx,!,
P>0,!,
N>P,!,
write("situazione illegale a sx"),!,
retract(sulla_riva_sx(c)),!,
assert(sulla_riva_dx(c)),!.

sit_legale_a_sx(m,"").
```

```
        /* MEC3 */
code=5000
DOMAINS
 Primopasseg,Secpasseg =string

DATABASE
 sulla_riva_sx(symbol)
 sulla_riva_dx(symbol)
 m_cancellati(symbol)
 c_cancellati(symbol)

PREDICATES
 inizializza
 cancella_riva_sx
 ripeti
 esegui_mossa_lecita
 sit_legale_a_dx(symbol,symbol)
 sit_legale_a_dx_2(symbol,symbol)
 rimetti_c_dx
 rimetti_m_dx
 fiume_attraversato
 contac_dx(integer,integer)
 contam_dx(integer,integer)
 sit_legale_a_sx(symbol,symbol)
 sit_legale_a_sx_2(symbol,symbol)
 rimetti_c_sx
 rimetti_m_sx
 contac_sx(integer,integer)
 contam_sx(integer,integer)
 tutto_a_sx
 visualizza_riva_sx
 visualizza_riva_dx
```

```
 riva_sx_senza_m_1
 riva_dx_senza_m_1
 m_non_mangiati_a_sx_1
 m_non_mangiati_a_dx_1
 mossa_legale_a_sx_1
 mossa_legale_a_dx_1
 riva_sx_senza_m_2
 riva_dx_senza_m_2
 m_non_mangiati_a_sx_2
 m_non_mangiati_a_dx_2
 mossa_legale_a_sx_2
 mossa_legale_a_dx_2

CLAUSES
 inizializza:-
 cancella_riva_sx,
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(m)),
 assert(sulla_riva_dx(c)),
 assert(sulla_riva_dx(c)),
 assert(sulla_riva_dx(c)),
 assert(sulla_riva_dx(b)).

cancella_riva_sx :-
 retract(sulla_riva_sx(_)),
 fail.
cancella_riva_sx.

visualizza_riva_sx:-
 nl,
 write("riva_sx : "),
```

```
  sulla_riva_sx(Chi),
  write(Chi),
  fail.
visualizza_riva_sx.

visualizza_riva_dx:-
  nl,
  write("riva_dx : "),
  sulla_riva_dx(Chi),
  write(Chi),
  fail.
visualizza_riva_dx.



contac_dx(Cont,N):-
  retract(sulla_riva_dx(c)),
  assert(c_cancellati(c)),
  Temp=Cont+1,
  contac_dx(Temp,N).
contac_dx(N,N).

contac_sx(Cont,N):-
  retract(sulla_riva_sx(c)),
  assert(c_cancellati(c)),
  Temp=Cont+1,
  contac_sx(Temp,N).
contac_sx(N,N).

rimetti_c_dx :-
  c_cancellati(X),
  X=c,
```

```
  retract(c_cancellati(X)),
  assert(sulla_riva_dx(X)),
rimetti_c_dx.
rimetti_c_dx.

rimetti_c_sx :-
  c_cancellati(X),
  X=c,
  retract(c_cancellati(X)),
  assert(sulla_riva_sx(X)),
rimetti_c_sx.
rimetti_c_sx.

contam_dx(Cont,P):-
  retract(sulla_riva_dx(m)),
  assert(m_cancellati(m)),
  Temp=Cont+1,
  contam_dx(Temp,P).
contam_dx(P,P).

contam_sx(Cont,P):-
  retract(sulla_riva_sx(m)),
  assert(m_cancellati(m)),
  Temp=Cont+1,
  contam_sx(Temp,P).
contam_sx(P,P).

rimetti_m_dx :-
  m_cancellati(X),
  X=m,
  retract(m_cancellati(X)),
  assert(sulla_riva_dx(X)),
```

```
rimetti_m_dx.
rimetti_m_dx.

rimetti_m_sx :-
 m_cancellati(X),
 X=m,
 retract(m_cancellati(X)),
 assert(sulla_riva_sx(X)),
rimetti_m_sx.
rimetti_m_sx.

riva_dx_senza_m_1:-
 contam_dx(0,P),
 !,
 rimetti_m_dx,!,
 P-2=0,!.
riva_dx_senza_m_1.

m_non_mangiati_a_dx_1:-
 contac_dx(0,N),!,
 contam_dx(0,P),
 !,
 rimetti_c_dx,!,
 rimetti_m_dx,!,
 P-2>=N,!.
m_non_mangiati_a_dx_1.

mossa_legale_a_dx_1:-
  riva_dx_senza_m_1;
  m_non_mangiati_a_dx_1,!.

riva_dx_senza_m_2:-
```

```
  contam_dx(0,P),
  !,
  rimetti_m_dx,!,
  P-1=0,!.
riva_dx_senza_m_2.

m_non_mangiati_a_dx_2:-
  contac_dx(0,N),!,
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  P-1>=N,!.
m_non_mangiati_a_dx_2.

mossa_legale_a_dx_2:-
   riva_dx_senza_m_2;
   m_non_mangiati_a_dx_2,!.

fiume_attraversato:-
  ripeti,
  esegui_mossa_lecita,
  tutto_a_sx.

ripeti.
  ripeti:-ripeti.

tutto_a_sx:-
  contac_dx(0,N),!,
  contam_dx(0,P),!,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
```

```
  N=0,!,
  P=0,!.

esegui_mossa_lecita:-
  sulla_riva_dx(X),
  X=b,
  visualizza_riva_sx,
  visualizza_riva_dx,
  nl,
  write("batti mossa "),
  readln(Primopasseg),
  readln(Secpasseg),
  sit_legale_a_dx(Primopasseg,Secpasseg),
  sit_legale_a_sx(Primopasseg,Secpasseg).
esegui_mossa_lecita.

esegui_mossa_lecita:-
  sulla_riva_sx(X),
  X=b,
  visualizza_riva_sx,
  visualizza_riva_dx,
  nl,
  write("batti mossa "),

  readln(Primopasseg),
  readln(Secpasseg),
  sit_legale_a_sx_2(Primopasseg,Secpasseg),
  sit_legale_a_dx_2(Primopasseg,Secpasseg),
esegui_mossa_lecita.


sit_legale_a_dx(X,Y):-
```

```
  X=m,
  Y=m,
  mossa_legale_a_dx_1,!,
  assert(sulla_riva_sx(m)),!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(m)),!,
  retract(sulla_riva_dx(m)),!,
  assert(sulla_riva_sx(b)),!,
  retract(sulla_riva_dx(b)),!.


sit_legale_a_dx(X,Y):-
  X=c,
  Y=c,
  contac_dx(0,N),!,
  rimetti_c_dx,!,
  N>=2,!,
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(c)),!,
  retract(sulla_riva_dx(c)),!,
  retract(sulla_riva_dx(c)),!,
  assert(sulla_riva_sx(b)),!,
  retract(sulla_riva_dx(b)),!.


sit_legale_a_dx(X,Y):-
  X=c,
  Y=m,
  contac_dx(0,N),!,
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
```

```
  rimetti_m_dx,!,
  N>=1,!,
  P>=1,!,
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(c)),!,
  retract(sulla_riva_dx(m)),!,
  assert(sulla_riva_sx(b)),
  retract(sulla_riva_dx(b)).

sit_legale_a_dx(X,Y):-
  X=c,
  Y=(""),
  contac_dx(0,N),
  !,
  rimetti_c_dx,!,
  N>=1,!,
  assert(sulla_riva_sx(c)),!,
  retract(sulla_riva_dx(c)),!,
  assert(sulla_riva_sx(b)),
  retract(sulla_riva_dx(b)).

sit_legale_a_dx(X,Y):-
  X=m,
  Y=(""),
  mossa_legale_a_dx_2,!,
  assert(sulla_riva_sx(m)),!,
  retract(sulla_riva_dx(m)),!,
  assert(sulla_riva_sx(b)),
  retract(sulla_riva_dx(b)).

sit_legale_a_sx(X,Y):-
```

```
  X=m,
  Y=m,
  contac_sx(0,N),!,
  contam_sx(0,P),
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
  N>P,!,
  write("situazione illegale a sx"),!,
  retract(sulla_riva_sx(m)),!,
  retract(sulla_riva_sx(m)),!,
  assert(sulla_riva_dx(m)),!,
  assert(sulla_riva_dx(m)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).

sit_legale_a_sx(X,Y):-
  X=c,
  Y=c,
  contac_sx(0,N),!,
  contam_sx(0,P),
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
  P>0,!,
  N>P,!,
  write("situazione illegale a sx"),!,
  retract(sulla_riva_sx(c)),!,
  retract(sulla_riva_sx(c)),!,
  assert(sulla_riva_dx(c)),!,
  assert(sulla_riva_dx(c)),!,
  assert(sulla_riva_dx(b)),
```

```
  retract(sulla_riva_sx(b)).

sit_legale_a_sx(c,m).

sit_legale_a_sx(X,Y):-
  X=c,
  Y=(""),
  contac_sx(0,N),
  contam_sx(0,P),
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
  P>0,!,
  N>P,!,
  write("situazione illegale a sx"),!,
  retract(sulla_riva_sx(c)),!,
  assert(sulla_riva_dx(c)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).

sit_legale_a_sx(m,"").

riva_sx_senza_m_1:-
  contam_sx(0,P),
  !,
  rimetti_m_sx,!,
  P-2=0,!.
riva_sx_senza_m_1.

m_non_mangiati_a_sx_1:-
  contac_sx(0,N),!,
  contam_sx(0,P),
```

```
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
  P>=2,!,
  P-2>=N,!.
m_non_mangiati_a_sx_1.

mossa_legale_a_sx_1:-
   riva_dx_senza_m_1;
   m_non_mangiati_a_sx_1,!.

riva_sx_senza_m_2:-
  contam_sx(0,P),
  !,
  rimetti_m_sx,!,
  P-1=0,!.
riva_sx_senza_m_2.

m_non_mangiati_a_sx_2:-
  contac_sx(0,N),!,
  contam_sx(0,P),
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
  P-1>=N,!.
m_non_mangiati_a_sx_2.

mossa_legale_a_sx_2:-
   riva_sx_senza_m_2;
   m_non_mangiati_a_sx_2,!.

sit_legale_a_sx_2(X,Y):-
```

```
  X=m,
  Y=m,
  mossa_legale_a_sx_1,!,
  assert(sulla_riva_dx(m)),!,
  assert(sulla_riva_dx(m)),!,
  retract(sulla_riva_sx(m)),!,
  retract(sulla_riva_sx(m)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).

sit_legale_a_sx_2(X,Y):-
  X=c,
  Y=c,
  contac_sx(0,N),
  rimetti_c_sx,!,
  N>=2,!,
  assert(sulla_riva_dx(c)),!,
  assert(sulla_riva_dx(c)),!,
  retract(sulla_riva_sx(c)),!,
  retract(sulla_riva_sx(c)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).


sit_legale_a_sx_2(X,Y):-
  X=c,
  Y=m,
  contac_sx(0,N),
  contam_sx(0,P),
  !,
  rimetti_c_sx,!,
  rimetti_m_sx,!,
```

```
  N>=1,!,
  P>=1,!,
  assert(sulla_riva_dx(c)),!,
  assert(sulla_riva_dx(m)),!,
  retract(sulla_riva_sx(c)),!,
  retract(sulla_riva_sx(m)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).

sit_legale_a_sx_2(X,Y):-
  X=c,
  Y=(""),
  contac_sx(0,N),
  !,
  rimetti_c_sx,!,
  N>=1,!,
  assert(sulla_riva_dx(c)),!,
  retract(sulla_riva_sx(c)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).


sit_legale_a_sx_2(X,Y):-
  X=m,
  Y=(""),
  mossa_legale_a_sx_2,!,
  assert(sulla_riva_dx(m)),!,
  retract(sulla_riva_sx(m)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_sx(b)).

sit_legale_a_dx_2(X,Y):-
```

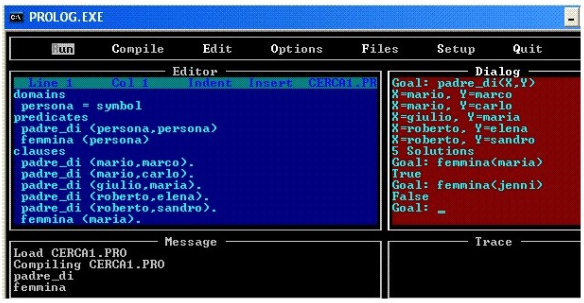
```
  X=m,
  Y=m,
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  N>P,!,
  retract(sulla_riva_dx(m)),!,
  retract(sulla_riva_dx(m)),!,
  assert(sulla_riva_sx(m)),!,
  assert(sulla_riva_sx(m)),!,
  assert(sulla_riva_dx(b)),
  retract(sulla_riva_dx(b)).

sit_legale_a_dx_2(X,Y):-
  X=c,
  Y=c,
  contac_dx(0,N),
  contam_dx(0,P),
  !,
  rimetti_c_dx,!,
  rimetti_m_dx,!,
  P>0,!,
  N>P,!,
  write("situazione illegale a dx"),!,
  retract(sulla_riva_dx(c)),!,
  retract(sulla_riva_dx(c)),!,
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(c)),!,
  assert(sulla_riva_sx(b)),
  retract(sulla_riva_dx(b)).
```

```
sit_legale_a_dx_2(c,m).

sit_legale_a_dx_2(X,Y):-
 X=c,
 Y=(""),
 contac_dx(0,N),
 contam_dx(0,P),
 !,
 rimetti_c_dx,!,
 rimetti_m_dx,!,
 P>0,!,
 N>P,!,
 write("situazione illegale a dx"),!,
 retract(sulla_riva_dx(c)),!,
 assert(sulla_riva_sx(c)),!,
 assert(sulla_riva_sx(b)),
 retract(sulla_riva_dx(b)).

sit_legale_a_dx_2(m,"").
```



Il Prolog è impiegato in molti programmi di intelligenza artificiale, la sua sintassi e la semantica sono molto semplici e chiare (lo scopo primitivo era quello di fornire uno strumento di lavoro a linguisti privi di conoscenze informatiche).

Il Prolog si basa sul calcolo dei predicati (precisamente il calcolo di predicati del primo ordine); tuttavia la sintassi è limitata a formule dette clausole di Horn che sono disgiunzioni di letterali del primo ordine quantificate universalmente con al più un letterale positivo.

L'esecuzione di un programma Prolog è comparabile alla dimostrazione di un teorema mediante la regola di inferenza detta risoluzione (introdotta da Robinson nel 1965). I concetti fondamentali sono l'unificazione, la ricorsione in coda e il backtracking.

backtracking e prolog

Il backtracking (in italiano, ritorno all'indietro) è una tecnica per trovare soluzioni a problemi in cui devono essere soddisfatti dei vincoli. Questa tecnica enumera tutte le possibili soluzioni e scarta quelle che non soddisfano i vincoli.

Una tecnica classica consiste nell'esplorazione di strutture ad albero e tenere traccia di tutti i nodi e i rami visitati in precedenza, in modo da poter tornare indietro al più vicino nodo che conteneva un cammino ancora inesplorato nel caso che la ricerca nel ramo attuale non abbia successo. I nodi a profondità uguale rappresentano i possibili valori di una variabile.
Una applicazione del backtracking è nei programmi per giocare a scacchi, che generano tutte le mosse possibili per una profondità di N mosse a partire da quella attuale e poi esaminano con il backtracking le varie alternative, selezionando alla fine quella migliore.

Tipi di inferenza

Deduttiva: si deriva una conclusione che è latente, implicita, prevista nelle premesse (l'informazione di cui siamo in possesso).

Induttiva: la conclusione aggiunge informazione alle premesse, non è una conseguenza logica delle premesse; caso tipico: la generalizzazione

Inferenze inconsce

• Il padre del mio amico è anche il padre di sua sorella

• Un triangolo ha tre lati

• Il libro che leggo ora è lo stesso che leggevo ieri

• Il mio vicino di casa è anche la persona che si muove al piano di sopra

Il metodo deduttivo o deduzione

Il metodo deduttivo o deduzione è il procedimento razionale che fa derivare una certa conclusione da premesse più generiche, dentro cui quella conclusione è implicita.

L'introduzione del concetto di deduzione si deve ad Aristotele (384 a.C.-322 a.C.), il quale lo identificava sostanzialmente con il sillogismo. Da questa identificazione deriva l'interpretazione tradizionale, accettata fino ai tempi moderni, secondo la quale il procedimento di deduzione consente di partire da una legge universale per giungere a conclusioni particolari. Il procedimento contrario viene chiamato induzione, che viceversa muove dal particolare all'universale.

Un esempio di sillogismo aristotelico è il seguente: «Tutti gli uomini sono mortali; Socrate è un uomo; dunque Socrate è mortale».

La differenza tra ragionamento deduttivo e induttivo non esiste sul piano pratico, essendo solo una speculazione sofistica di tipo filosofico. Tutti i ragionamenti sono induttivi, anche quelli matematici, altrimenti non sono "ragionamenti" ma conclusioni tautologiche, del tutto prive di contenuto.

Ma se il ragionamento che "produce conoscenza" è solo induttivo, si potrebbe in sostanza dire che l'induzione non è altro che una deduzione basata sull'osservazione dei fatti o su un'esperienza personale.