17/03/2019

Contest sul ProLog

Struttura della base di conoscenza e alcune prove di funzionamento

Benfenati Domenico N46003380
Galasso Alessandro N39000729
Banchini Federico N46003517
Aveta Carmine N46003385

Sommario

Introduzione al linguaggio ProLog

Questo elaborato ha lo scopo di descrivere ciò che è stato spiegato in aula, ovvero, la realizzazione di programmi logici mediante l'utilizzo di un linguaggio fortemente dichiarativo come il ProLog.

Il linguaggio **ProLog** (Programming in Logic) è un linguaggio formale del primo ordine basato sul calcolo dei predicati, i quali vengono utilizzati per esprimere in maniera sobria e coincisa ragionamenti logici.
Per determinare la verità o la falsità di un obiettivo, il ProLog utilizza solo fatti e regole presenti nella **base della conoscenza (KB)**.

Esso viene impiegato in molti programmi dell'intelligenza artificiale e in tanti altri rami applicativi per poterne eseguire: query su database, dimostrazione automatica di teoremi, calcoli simbolici, realizzazione di parsing ecc.

Nel nostro caso, il ProLog è stato utilizzato per concepire due programmi logici, in particolare, un quiz e un programma riguardante una linea ferroviaria.
Durante la progettazione, si è voluto dimostrare come anche un semplice problema possa essere riformulato in termini di ragionamento logico e come il ProLog possa essere un linguaggio complesso, potente ed elegante.

Primo Contest: Quiz a domande multiple

Descrizione

Lo scopo principale di questo programma logico è quello di rispondere correttamente alle sette domande che ci vengono poste all'interno del quiz. Per ogni quesito, vengono stampate a video quattro possibili risposte e successivamente l'utente potrà inserire da tastiera la risposta giusta. Verificata la correttezza della risposta , si passa alla successiva, e così via fino alla terminazione del quiz.

Per la realizzazione di quest'ultimo, sono stati innanzitutto introdotti ,all'interno della base delle conoscenze, fatti e regole. Spulciando tra i fatti, possiamo notare quelli che concernono le domande, le risposte e in alcuni casi, per poter semplificare alcune soluzioni, come ad esempio la stampa a video, sono state introdotte delle liste contenenti le probabili risposte. Infatti, ad ogni domanda viene stampata: la stringa del quesito e l'insieme delle probabili quattro risposte.

Per stabilire a primo acchito la correttezza della risposta, entra in gioco il seguente fatto, valido per ogni differente domanda (*corretta(Domanda, Risposta_esatta)*). Arrivati in questo punto, si è cercato di rendere il quiz il più interattivo e semplice possibile, in quanto, l'utente può scrivere semplicemente la risposta in linguaggio naturale e non in linguaggio logico. Per poter utilizzare questa soluzione, è stato introdotto il metodo read, dichiarandolo all'interno di una regola (check).

Il suo funzionamento è molto semplice: prelevata da tastiera una variabile valida come *risposta(risposta_x)*, viene verificata se quest'ultima soddisfa la regola '*rispondoalladomanda_x*' ;tale regola viene soddisfatta se: innanzitutto è vero il fatto '*corretta(domanda_x,risposta_x)*', è vero che la risposta appartenga alla lista delle possibili quattro risposte, e deve essere vero che risposta_x sia una vera è propria risposta.

Nel caso in cui la risposta sia completamente sbagliata, o non appartenga alla lista, o non sia una vera risposta ecc. non ci siamo persi d'animo perché tale eccezione viene gestita con successo, cioè, viene stampato a video un messaggio di risposta sbagliata e vengono ripetuti i passi per poter immettere da tastiera una nuova risposta. Per rendere più semplici e leggibili queste soluzioni, sono state introdotte regole principali che presentano nel lato della precondizione una serie di and logici tra altre regole, al fine di poter eseguire linearmente tutte le domande in ordine.

Documentazione

Il funzionamento del programma logico ,come visto in precedenza è basato soprattutto sul soddisfacimento dei fatti e delle regole per la determinazione del goal. Ci siamo basati principalmente sugli argomenti trattati in classe, quindi abbiamo aggiunto predicati molto semplici:

- il predicato **write(Term)**, che stampa semplicemente a video la variabile contenuta.
- il predicato **read(Terms)**, del quale riportiamo la definizione ufficiale fornita dal manuale del ProLog: *Read the next Prolog term from the current input stream and unify it with Term*.
- il predicato **member(X,List[])**, che essenzialmente ci semplifica il lavoro, ovvero, ci fornisce *True* se X è un elemento della lista. Anche in questo caso ci siamo basati sulle risorse apprese sul manuale ufficiale del ProLog.

Tramite anche l'ausilio di questi predicati sono state generate le regole per una corretta esecuzione del programma logico.

Codice

```
%%% testi delle domande (fatti)
```

```
domanda1('qual è lo stato più piccolo d\'Europa?').
domanda2('quante volte è stato presidente Giulio Andreotti?').
domanda3('quale tra questi mari non bagna l\'Italia?').
domanda4('a quando risale l\'invenzione dello champagne da parte di
Don Perignon?').
domanda5('chi è l\'autore del "cantico delle creature"?').
domanda6('qual è lo strato dell\'atmosfera che si trova più in
basso?').
domanda7('dove si svolgerà il prossimo campionato del mondo di
calcio?').

%%% definizione delle risposte (fatti)

risposta('vaticano').
risposta('litchenstein').
risposta('principatoDiMonaco').

risposta(3).
risposta(5).
risposta(7).
risposta(14).

risposta('tirreno').
risposta('ionio').
risposta('mediterraneo').
risposta('morto').

risposta(1677).
risposta(1452).
risposta(1668).
risposta(1790).

risposta('danteAlighieri').
risposta('sanFrancesco').
risposta('petrarca').
risposta('guittone').

risposta('stratosfera').
risposta('troposfera').
risposta('termosfera').
risposta('mesosfera').

risposta('brasile').
risposta('italia').
```

```
risposta('qatar').
risposta('sudan').

%%% definizione della risposta corretta per le varie domande (fatti)

corretto ('qual è lo stato più piccolo d\'Europa?' , 'vaticano').
corretto ('quante volte è stato presidente Giulio Andreotti?' , 7).
corretto ('quale tra questi mari non bagna l\'Italia?' , 'morto').
corretto ('a quando risale l\'invenzione dello champagne da parte di
Don Perignon?' , 1668).
corretto ('chi è l\'autore del "cantico delle creature"?' ,
'sanFrancesco' ).
corretto ('qual è lo strato dell\'atmosfera che si trova più in basso?
',' troposfera' ).
corretto ('dove si svolgerà il prossimo campionato del mondo di
calcio?' , 'qatar' ).

%%% elenchi di risposte interpretati come liste (fatti)

rispostedomanda1([ 'vaticano','litchenstein','principatoDiMonaco' ]).
rispostedomanda2([ 3 , 5 , 7, 14 ]).
rispostedomanda3([ 'tirreno', 'ionio', 'mediterraneo', 'morto' ]).
rispostedomanda4([ 1677, 1452, 1668, 1790 ]).
rispostedomanda5([ 'danteAlighieri', 'sanFrancesco', 'petrarca',
'guittone' ]).
rispostedomanda6([ 'stratosfera', 'troposfera', 'termosfera',
'mesosfera' ]).
rispostedomanda7([ 'brasile', 'italia', 'qatar', 'sudan' ]).

%%% regole per effettuare una risposta ad una domanda

rispondoalla1 (Y):-corretto( 'qual è lo stato più piccolo d\'Europa?'
, Y), member(Y,['vaticano', 'litchenstein', 'principatoDiMonaco' ]),
risposta(Y), write(corretto), nl.

rispondoalla2 (Y):-corretto('quante volte è stato presidente Giulio
Andreotti?', Y), member(Y,[3,5,7,14]), risposta(Y), write(corretto),
nl.

rispondoalla3 (Y):-corretto('quale tra questi mari non bagna
l\'Italia?', Y), member(Y,['tirreno', 'ionio', 'mediterraneo',
'morto'] ), risposta(Y), write(corretto), nl.
```

```
rispondoalla4 (Y):-corretto('a quando risale l\'invenzione dello champagne da parte di Don Perignon?', Y), member(Y,[1677,1452,1668,1790]), risposta(Y), write(corretto), nl.

rispondoalla5 (Y):-corretto( 'chi è l\'autore del "cantico delle creature"?', Y), member(Y, ['danteAlighieri', 'sanFrancesco', 'petrarca', 'guittone']), risposta(Y), write(corretto), nl.

rispondoalla6 (Y):-corretto( 'qual è lo strato dell\'atmosfera che si trova più in basso?', Y), member(Y,['stratosfera', 'troposfera', 'termosfera', 'mesosfera']), risposta(Y), write(corretto), nl.

rispondoalla7 (Y):-corretto('dove si svolgerà il prossimo campionato del mondo di calcio?', Y), member(Y,['brasile','italia', 'qatar','sudan']),risposta(Y), write(corretto), nl.

%%% funzione per il testing della risposta inserita

test1(A):-rispondoalla1(A).
test1(_):-write('sbagliato, riprova!'), nl, check1.
test2(B):-rispondoalla2(B).
test2(_):-write('sbagliato, riprova!'), nl, check2.
test3(C):-rispondoalla3(C).
test3(_):-write('sbagliato, riprova!'), nl, check3.
test4(D):-rispondoalla4(D).
test4(_):-write('sbagliato, riprova!'), nl, check4.
test5(E):-rispondoalla5(E).
test5(_):-write('sbagliato, riprova!'), nl, check5.
test6(F):-rispondoalla6(F).
test6(_):-write('sbagliato, riprova!'), nl, check6.
test7(O):-rispondoalla7(O).
test7(_):-write('sbagliato, riprova!'), nl, check7.

/* la regola indicata con test(_) senza argomenti sta a definire una regola di default. Quando, durante la chiamata della regola test stessa, l'utente inserisce qualcosa di errato o non inserisce nulla viene mostrata a video la dicitura "sbagliato, riprova" e viene effettuata la chiamata ad una sottoprocedura del programma in ProLog che indichiamo con "check" */

%%% sottoprocedure per la verifica delle risposte

check1 :- read(G), test1(G).
check2 :- read(H), test2(H).
check3 :- read(I), test3(I).
```

```
check4 :- read(L), test4(L).
check5 :- read(M), test5(M).
check6 :- read(N), test6(N).
check7 :- read(P), test7(P).
```

/* la sottoprocedura check ha la stessa struttura di una regola, ma non presenta dei parametri di ingresso. Il corpo della sottoprocedura funziona in maniera differente rispetto al corpo di una regola: infatti, quello che scriviamo nel corpo non sono delle condizioni in AND tra loro, la cui veridicità implica la veridicità della condizione scritta in testa, ma rappresentano le operazioni fatte in sequenza quando vado a richiamare la procedura stessa tramite il suo nome. Ciò mi permette di porzionare l'esecuzione del codice e di renderlo il più leggibile possibile. La funzione utilizzata che mi permette di ottenere delle stringhe inserite manualmente dall'utente è la funzione *read* */
%%% sottoprocedure per la struttura grafica del quiz

```
intro:- nl, nl, write('*****Benvenuto nel nostro test*****'),nl, nl.
quesito1 :- domanda1(X), write(X), nl, rispostedomanda1(Y), write(Y),
nl, check1, nl.
quesito2 :- domanda2(X), write(X), nl, rispostedomanda2(Y), write(Y),
nl, check2, nl.
quesito3 :- domanda3(X), write(X), nl, rispostedomanda3(Y), write(Y),
nl, check3, nl.
quesito4 :- domanda4(X), write(X), nl, rispostedomanda4(Y), write(Y),
nl, check4, nl.
quesito5 :- domanda5(X), write(X), nl, rispostedomanda5(Y), write(Y),
nl, check5, nl.
quesito6 :- domanda6(X), write(X), nl, rispostedomanda6(Y), write(Y),
nl, check6, nl.
quesito7 :- domanda7(X), write(X), nl, rispostedomanda7(Y), write(Y),
nl, check7, nl.
fine:- nl, write('******hai riposto correttamente a tutte le
domande*****'), nl, nl.
```

/*come accadde per le procedure di check, anche le procedure precedenti sono un insieme di operazioni sequenziali tra cui la pesca della domanda e delle relative risposte e la stampa a video di entrambe. Con il classificatore *nl* si indica lo andare a capo a video. Le procedure di *intro* e *fine* sono puramente grafiche e non eseguono controlli sulla base di conoscenza */

/* per avviare il quiz, basta digitare il comando *start_test* dall'interprete, che viene definito come segue */

```
start_test :- intro, quesito1, quesito2, quesito3, quesito4, quesito5,
quesito6, quesito7, fine.
```

Output

Di seguito vengono presentati alcuni esempi di funzionamento del programma logico:

```
?- start_test.

*****Benvenuto nel nostro test*****
qual è lo stato più piccolo d'Europa?
[vaticano, litchenstein, principatoDiMonaco]
|: principatoDiMonaco.
sbagliato, riprova!
|: vaticano.
corretto


quante volte è stato presidente Giulio Andreotti?
[3,5,7,14]
|: 16.
sbagliato, riprova!
|: 7.
corretto


quale tra questi mari non bagna l'Italia?
[tirreno, ionio, mediterraneo, morto]
|: 2.
sbagliato, riprova!
|: ionio.
sbagliato, riprova!
|: morto.
corretto


a quando risale l'invenzione dello champagne da parte di Don Perignon?
[1677,1452,1668,1790]
|: 1668.
corretto


chi è l'autore del "cantico delle creature"?
[danteAlighieri, sanFrancesco, petrarca, guittone]
|: guittone.
sbagliato, riprova!
|: sanFrancesco.
corretto
```

```
qual è lo strato dell'atmosfera che si trova più in basso?
[stratosfera,troposfera,termosfera,mesosfera]
|: terra.
sbagliato, riprova!
|: troposfera.
corretto


dove si svolgerà il prossimo campionato del mondo di calcio?
[brasile,italia,qatar,sudan]
|: sudan.
sbagliato, riprova!
|: italia.
sbagliato, riprova!
|: qatar.
corretto


******hai riposto correttamente a tutte le domande*****

true.
```

Al fine di visualizzare uno ad uno tutti i passi eseguiti dal motore inferenziale per determinare il goal, attiviamo il comando *trace* prendendo come esempio la risposta alla domanda numero 1, considerando casi in cui la risposta fornita è quella corretta e casi in cui vediamo in azione il meccanismo di backtracking del ProLog, fornendo una risposta errata. Per salvare su file le opeazioni che svolge il motore inferente è possibile eseguire il comando *protocol("nomeFile")*. In questo modo, sul file allegato "esempio_trace.txt" verranno salvati i comandi eseguiti durante il tracciamento dell'unificazione della regola "*quesito1.*"

```
[trace]  ?- quesito1.

   Call: (8) quesito1 ? creep

   Call: (9) domanda1(_1148) ? creep

   Exit: (9) domanda1('qual è lo stato più piccolo d\'Europa?') ? creep

   Call: (9) write('qual è lo stato più piccolo d\'Europa?') ? creep

qual è lo stato più piccolo d'Europa?

   Exit: (9) write('qual è lo stato più piccolo d\'Europa?') ? creep

   Call: (9) nl ? creep


   Exit: (9) nl ? creep

   Call: (9) rispostedomanda1(_1148) ? creep
```

```
   Exit: (9) rispostedomanda1([vaticano, litchenstein, principatoDiMonaco]) ?
creep

   Call: (9) write([vaticano, litchenstein, principatoDiMonaco]) ? creep

[vaticano,litchenstein,principatoDiMonaco]

   Exit: (9) write([vaticano, litchenstein, principatoDiMonaco]) ? creep

   Call: (9) nl ? creep

   Exit: (9) nl ? creep

   Call: (9) check1 ? creep

   Call: (10) read(_1166) ? creep

|: principatoDiMonaco.

   Exit: (10) read(principatoDiMonaco) ? creep

   Call: (10) test1(principatoDiMonaco) ? creep

   Call: (11) rispondoalla1(principatoDiMonaco) ? creep

   Call: (12) corretto('qual è lo stato più piccolo d\'Europa?',
principatoDiMonaco) ? creep

   Fail: (12) corretto('qual è lo stato più piccolo d\'Europa?',
principatoDiMonaco) ? creep

   Fail: (11) rispondoalla1(principatoDiMonaco) ? creep

   Redo: (10) test1(principatoDiMonaco) ? creep

   Call: (11) write('sbagliato, riprova!') ? creep

sbagliato, riprova!

   Exit: (11) write('sbagliato, riprova!') ? creep

   Call: (11) nl ? creep

   Exit: (11) nl ? creep

   Call: (11) check1 ? creep

   Call: (12) read(_1166) ? creep

|: vaticano.

   Exit: (12) read(vaticano) ? creep

   Call: (12) test1(vaticano) ? creep
```

```
   Call: (13) rispondoalla1(vaticano) ? creep

   Call: (14) corretto('qual è lo stato più piccolo d\'Europa?', vaticano) ? creep

   Exit: (14) corretto('qual è lo stato più piccolo d\'Europa?', vaticano) ? creep

   Call: (14) lists:member(vaticano, [vaticano, litchenstein, principatoDiMonaco])
? creep

   Exit: (14) lists:member(vaticano, [vaticano, litchenstein, principatoDiMonaco])
? creep

   Call: (14) risposta(vaticano) ? creep

   Exit: (14) risposta(vaticano) ? creep

   Call: (14) write(corretto) ? creep

corretto

   Exit: (14) write(corretto) ? creep

   Call: (14) nl ? creep

   Exit: (14) nl ? creep

   Exit: (13) rispondoalla1(vaticano) ? creep

   Exit: (12) test1(vaticano) ? creep

   Exit: (11) check1 ? creep

   Exit: (10) test1(principatoDiMonaco) ? creep

   Exit: (9) check1 ? creep

   Call: (9) nl ? creep

   Exit: (9) nl ? creep

   Exit: (8) quesito1 ? creep

true.

[trace]  ?- nodebug.

true.

?- noprotocol.
```

Secondo Contest: Linee ferroviarie

Descrizione

La base di conoscenza è volta a rappresentare delle tratte ferroviarie, quali la linea metropolitana 2 che parte da Campi Flegrei e arriva a Garibaldi, la Circumvesuviana nella tratta Napoli Garibaldi-Sorrento e infine la linea regionale Garibaldi-Salerno.

Le tratte vengono rappresentate suddividendole in vari spezzoni, ognuno dei quali rappresenta il percorso che il treno effettua tra una fermata e la sua successiva. Ogni spezzone da una stazione alla successiva viene indicato con un fatto.

Il caso preso in esame si presa molto bene allo sviluppo di un esempio di ricorsione: infatti è possibile schematizzare un viaggio in treno in maniera ricorsiva, sapendo che per andare dalla stazione A alla stazione C farò un viaggio dalla stazione A ad una stazione B intermedia, e un viaggio dalla stazione B alla stazione di destinazione finale C, essendo il viaggio composto da più fermate.

Per aggiungere funzionalità al nostro programma, è stato pensato di aggiungere una scrittura su file per generare una sorta di biglietto temporaneo, valido per un solo viaggio. Purtroppo, essendo il formato Date molto complesso da utilizzare in ProLog, si è convenuto formattare il biglietto con le sole stazioni di partenza e di arrivo, non riuscendo ad implementare quindi una sorta di timer per la scadenza del biglietto stesso. È inoltre possibile tramite interprete dei comandi, pescare informazioni da un file, e potere quindi visualizzare il proprio biglietto nella shell dei comandi di ProLog, oltre che avere il biglietto fisico su file.

Documentazione

Oltre agli strumenti utilizzati nella precedente base di conoscenza, sono state aggiunte delle semplici funzioni descritte in seguito:

- il predicato **open(+SrcDest, +Mode, --Stream, +Options),** di cui si riporta la dicitura del sito ufficiale di SWI-ProLog: *true when SrcDest can be opened in Mode and Stream is an I/O stream to/from the object. SrcDest is normally the name of a file, represented as an atom or string. Mode is one of read, write, append or update.*
- il predicato **get_char**: *read the current input stream and unify Char with the next character as a one-character atom.*

Codice

```
%%% prossima fermata linea 2 (fatti)

      linea2(campi_flegrei, leopardi).
      linea2(leopardi, mergellina).
      linea2(mergellina, amedeo).
      linea2(amedeo, montesanto).
      linea2(montesanto, cavour).
      linea2(cavour, garibaldi).

%%% prossima fermata direzione Salerno (fatti)

      dir_salerno(garibaldi,gianturco).
```

```
dir_salerno(gianturco,san_giovanni_barra).
dir_salerno(san_giovanni_barra,pietrarsa).
dir_salerno(pietrarsa,portici).
dir_salerno(portici,torre_del_greco).
dir_salerno(torre_del_greco,santa_maria_la_bruna).
dir_salerno(santa_maria_la_bruna,torre_annunziata_centrale).
dir_salerno(torre_annunziata_centrale,pompei).
dir_salerno(pompei,scafati).
dir_salerno(scafati,angri).
dir_salerno(angri,pagani).
dir_salerno(pagani,nocera_inferiore).
dir_salerno(nocera_inferiore,nocera_superiore).
dir_salerno(nocera_superiore,cava_dei_tirreni).
dir_salerno(cava_dei_tirreni,vietri_sul_mare).
dir_salerno(vietri_sul_mare,duomo_via_vernieri).
dir_salerno(duomo_via_vernieri,salerno).

%%% prossime fermate linea Circumvesuviana (fatti)

circumvesuviana(garibaldi, ercolano_scavi).
circumvesuviana(ercolano_scavi, torre_del_greco).
circumvesuviana(torre_del_greco, torre_annunziata).
circumvesuviana(torre_annunziata, pompei_scavi).
circumvesuviana(pompei_scavi, via_nocera).
circumvesuviana(via_nocera, castellammare).
circumvesuviana(castellammare, vico_equense).
circumvesuviana(vico_equense, seiano).
circumvesuviana(seiano, meta).
circumvesuviana(meta, piano).
circumvesuviana(piano, s_agnello).
circumvesuviana(s_agnello, sorrento).

%%% definizione delle regole ricorsive per un viaggio nei due sensi

viaggio(Partenza,Arrivo):-linea2(Partenza,Arrivo), write('Prendi la
linea 2 fino a '), write(Arrivo).

viaggio(Partenza,Arrivo):-linea2(Partenza,Cmezzo),
viaggio(Cmezzo,Arrivo).

/*
le prime 2 regole viaggio riguardano il viaggio fino a Garibaldi. La regola
base viene richiamata solo nel caso in cui le fermate siano  consecutive.
Quando invece la partenza e l'arrivo non sono consecutive la regola ricorsiva
```

```
viaggio unifica la fermata successiva alla partenza con la variabile Cmezzo
e poi richiama nuovamente sé stessa con argomenti Cmezzo al posto della
Partenza e con lo stesso Arrivo. Ripete tale procedimento finché la variabile
Cmezzo e l'Arrivo sono consecutive e in tale caso richiama la regola base
che inoltre stampa a video il messaggio "prendi la linea 2 fino a Arrivo".
*/

viaggio(Partenza,Arrivo):- circumvesuviana(Partenza,Arrivo),
    write('Prendi la linea 2 fino a Garibaldi poi prendi la
circumvesuviana fino a '), write(Arrivo).

viaggio(Partenza,Arrivo):- circumvesuviana(Partenza,Cmezzo),
    viaggio(Cmezzo,Arrivo).

/*
le precedenti 2 regole viaggio riguardano invece la tratta Garibaldi-Sorrento
della circumvesuviana. Tale regola viene richiamata quando l'utente desidera
arrivare in una stazione della circumvesuviana e il MI la usa quando Cmezzo
della prima regola viaggio è stato unificato con Garibaldi. Il funzionamento
interno di questa regola è identico a quello illustrato prima.
*/

viaggio(Partenza,Arrivo):- dir_salerno(Partenza,Arrivo),
    write('Prendi la linea 2 fino a Garibaldi poi prendi il treno in
direzione Salerno fino a '), write(Arrivo).

viaggio(Partenza,Arrivo):- dir_salerno(Partenza,Cmezzo),
    viaggio(Cmezzo,Arrivo).

/*
le precedenti 2 regole viaggio riguardano invece la tratta Garibaldi-Salerno
della linea regionale. Tale regola viene richiamata quando l'utente desidera
arrivare in una stazione della circumvesuviana e il MI la usa quando Cmezzo
della prima regola viaggio è stato unificato con Garibaldi. Il funzionamento
interno di questa regola è identico a quello illustrato prima.
*/

%%% procedure per la stampa e per l'input da file

      process_stream(end_of_file,_Z):- !.
      process_stream(Char,Stream):-
            write(Char),
            get_char(Stream,Char2),
            process_stream(Char2,Stream).
```

```
crea_biglietto(File):-
      open(File , write , Stream),
      read(X), read(Y),
      write(Stream , 'Biglietto valido da  '),
      write(Stream , X), write(Stream , ' a '),write(Stream , Y),
      close(Stream).

leggi_biglietto(File):-
    open(File , read , Stream),
    get_char(Stream, Char1),
    process_stream(Char1, Stream),
    close(Stream).
```

```
/* le regole process_stream vengono utilizzate per definire il flusso di dati da o verso un file. La prima regole sta ad indicare che il flusso si interrompe quando il file termina, mentre la seconda funziona un po' come le regole ricorsive, e mi dice che finché lo stream è aperto devo accettare o ricevere dati da esso. La funzionalità di creazione del biglietto comprende l'apertura del file, in modalità di scrittura, usando il flusso Stream tramite la funzione di sistema open. Successivamente, acquisisco da tastiera due variabili che saranno le stazioni di partenza e arrivo del viaggio, e le stampo sul file. La funzionalità di lettura del biglietto è la medesima, ma utilizza la funzione di sistema get_char per la lettura del file stesso */
```

Output

Riportiamo ora alcuni esempi di funzionamento del programma logico:

```
?- viaggio(campi_flegrei,garibaldi).

Prendi la linea 2 fino a garibaldi

true

?- viaggio(mergellina,cavour).

Prendi la linea 2 fino a cavour

true

?- viaggio(campi_flegrei,san_giovanni_barra).

Prendi la linea 2 fino a Garibaldi poi prendi il treno in direzione Salerno fino a san_giovanni_barra

True

?- viaggio(campi_flegrei,salerno).

Prendi la linea 2 fino a Garibaldi poi prendi il treno in direzione Salerno fino a salerno
```

```
True
?- viaggio(campi_flegrei, seiano).
Prendi la linea 2 fino a Garibaldi poi prendi la circumvesuviana fino a seiano
True
?- viaggio(campi_flegrei, sorrento).
Prendi la linea 2 fino a Garibaldi poi prendi la circumvesuviana fino a sorrento
True
?- viaggio(campi_flegrei, torre_del_greco).
```

Il caso della fermata di torre del greco risulta particolare: come si può osservare dalla base di conoscenza, Torre del Greco è una stazione raggiungibile sia attraverso la circumvesuviana sia attraverso la linea regionale per Salerno, questo fa sì che il MI nella ricerca della risposta giusta verifichi prima la regola viaggio riguardante la circumvesuviana perché posta per prima dai progettisti nella KB e poi la regola viaggio riguardante la direzione Salerno. Per ottenere la seconda risposta basta inserire il “;” in seguito alla prima risposta.

Riferimenti

Predicati per la scrittura a video *:* *http://www.swi-prolog.org/pldoc/doc_for?object=write/1*

Funzionalità sui sottoprogrammi : *http://www.swi-prolog.org/man/threads.html*

Predicati sulla lettura da tastiera : *http://www.swi-prolog.org/pldoc/man?predicate=read/1*

Predicati di scrittura su file : *http://www.swi-prolog.org/pldoc/doc_for?object=write/2*

Predicati di lettura da file : *http://www.swi-prolog.org/pldoc/doc_for?object=read/2*

Appartenenza di elementi in una lista : *http://www.swi-prolog.org/pldoc/doc_for?object=member/2*

Stampa del file trace : *http://www.swi-prolog.org/pldoc/doc_for?object=protocol/1*

Predicato di get di carattteri : *http://www.swi-prolog.org/pldoc/doc_for?object=get_char/2*