Deadly Med

A.Moggio R.Mele R.Pane

IA – CONTEST 1, 24-05-2020

1 Obiettivo

Si vuole creare un applicativo che, dato in ingresso un codice utente ed un medicinale, restituisca una compatibilità (vero) o una incompatibilità (falso) tra i due, attraverso un controllo tra gli ingredienti e le allergie.

2 Possibili Applicazioni

Una possibile applicazione è quella del contesto farmaceutico.
Ovvero implementare l'applicativo all'interno del sistema informatico della farmacia ed effettuare un controllo automatico all'acquisto di un generico prodotto.

Più in generale si potrebbe valutare un database ed una rete nazionale, in cui i vari utenti sono identificati tramite il loro Codice Fiscale.

3 Let's have a Look Inside

```
14 username(giacvall12).
15 username(raffverdi57).
16 username(angrossi69).
17 username(nicorazz99).
18
19 medicinal(froben).
20 medicinal(bronchenolo).
21 medicinal(tachipirina).
22 medicinal(acetamol).
```

Nella prima porzione di codice abbiamo i cosiddetti 'Fatti', quindi gli username e i medicinali della Knowledge Base.

Ogni utente è identificato da un username.

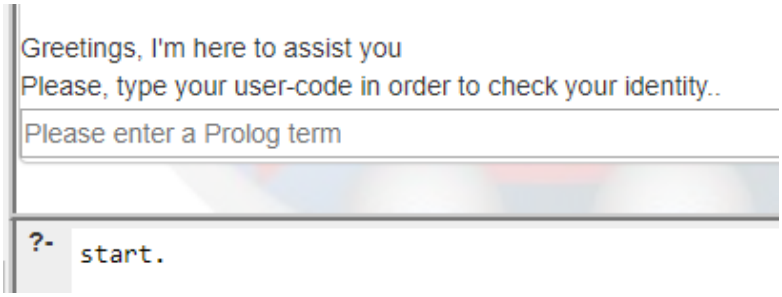
La sintassi per un generico utente è del tipo:
short-name + short-surname + two generic numbers

```
related(giacvall12,[paracetamolo,fibrinogeno]).
related(raffverdi57,[benzidamina, penicillina]).
related(angrossi69,[bilastina, zinco]).
related(nicorazz99,[zinco,fibrinogeno]).


related(froben, [paracetamolo, bilastina]).
related(bronchenolo, [solfiti, zinco]).
related(tachipirina, [fibrinogeno, benzidamina]).
related(acetamol, [zinco,paracetamolo]).
```

Successivamente troviamo le relazioni che intercorrono tra i vari utenti e le allergie di cui soffrono.

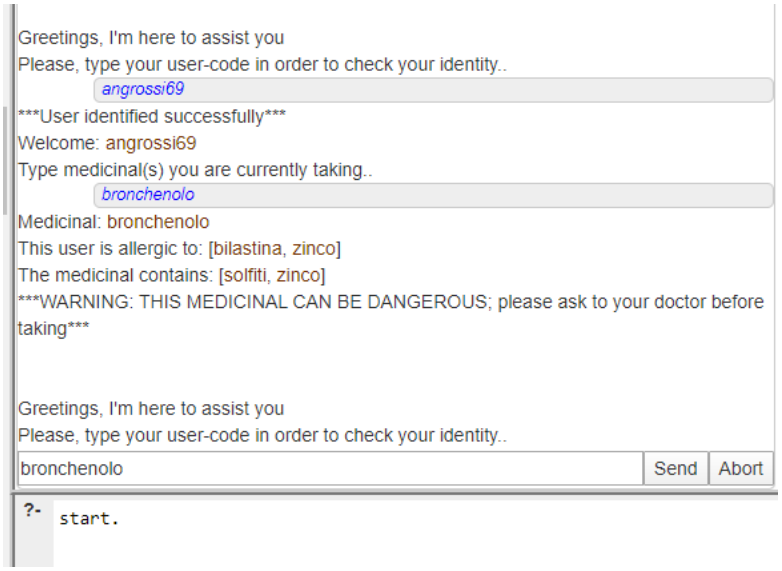
Nella seconda i medicinali con i corrispettivi ingredienti.



Il programma viene avviato tramite il comando "start", che inizializza una chat interattiva in cui viene chiesto di inserire prima il codice identificativo e poi il medicinale da mettere in esame.

Ogni passaggio viene accompagnato da alcune frasi che "guidano" l'utente nella corretta interrogazione.

Vengono stampati a video, rispettivamente per l'utente e il medicinale selezionati, la lista degli allergeni e quella degli ingredienti.



<- --[messaggio in caso di incompatibilità]

Il risultato finale sarà quindi un messaggio a video che mostrerà se il medicinale è pericoloso o meno per la salute dell'utente.

Al termine dell'operazione il sistema offrirà in automatico la possibilità di una nuova interazione.

```
ask_med:-
    write("Type medicinal(s) you are currently taking..").

ok_med:-
    write("You can take this medicinal without consequences"), nl.

bad_med:-
    write("***WARNING: THIS MEDICINAL CAN BE DANGEROUS; please ask to your doctor before taking***"), nl.
```

<- --[alcuni esempi di predicati atti a simulare un' interazione più "umana"]

```
41 deadly_med(U,M):-
42
43     custom_list(U,L1),
44     flatten(L1,Q1),
45     %write(Q1), nl,
46     custom_list(M,L2),
47     flatten(L2,Q2),
48     write("The medicinal contains: "),
49     write(Q2), nl,
50     match_lists(Q1,Q2).
```

Il cuore del programma è la regola **deadly_med**.
Essa prende in ingresso la lista U (utente) e quella M (medicinale) restituendo un risultato logico True se uno degli ingredienti presenti nel medicinale è presente anche tra gli allergeni dell'utente.

Ma scendiamo nel dettaglio.

```
53 %matching list: if L1 and L2 (lists) have
54 match_lists(L1,L2):-
55     member(E,L1),
56     member(E,L2).
57
58 %Check if a X user is present of the KB
59 is_id(X):-
60     username(X).
61
62 %Builds lists from facts
63 custom_list(U,L) :-
64     findall([X,Y],(related(U,[X,Y])),L).
65
66 %Check if X is member of a given list
67 member(X,[X|_]).
68 member(X,[_|T]):-
69     member(X,T).
70
```

La **custom_list** costruisce una lista da dei fatti.

Si crea quindi una Lista di Liste che viene poi elaborata da **flatten**, che la trasforma in una lista.

Attraverso questo processo si vengono a formare quindi Q1 e Q2, contenenti l'una gli allergeni dell'utente e l'altra gli ingredienti del medicinale.

match_lists controlla se Q1 e Q2 hanno elementi in comune.

Se risulterà Vero allora lo sarà anche per **deadly_med.**

```
121 start:-
122     nl,
123     hello,
124     read(R),
125     ok_id(R)->(
126     ask_med,
127     get_med([X]),
128     write("Medicinal: "),
129     write(X),nl,
130     custom_list(R,L),
131     flatten(L,Q),
132     write("This user is allergic to: "),
133     write(Q), nl,
134     deadly_med(R,X)->
135         (
136         bad_med, nl
137         );
138          (
139         ok_med, nl
140          )
141          ),
142     repeat, start.
143
```

Start è stata realizzata per simulare un'interazione tra il sistema ed un utente.

ok_id effettua un controllo sul corretta digitazione del codice utente, e quindi sulla presenza dello stesso nella Knowledge Base. In caso negativo ripropone l'immissione del codice.

get_med invece costruisce una lista del medicinale immesso in input.

Il tutto viene poi elaborato e dato alla **deadly_med**, di cui si è già ampiamente parlato.

Sono inoltre presenti predicati utili alla corretta interpretazione dei segnali in output.

Ad esempio **bad_med** e **ok_med** mostrano a video una spiegazione, rispettivamente, per il risultato True e quello False della query.

In fondo, è presente un **repeat** per non far terminare la chat dopo una singola interazione.

```
106 get_med([Med|List]):-
107     read(Med),
108     dif(Med,stop),
109     get_med(List).
110
111 get_med([]).
112
113 endme:-!,fail.
```

```
72 %Utility predicate: they are used to simulate a kind but effective chat between the machine and the user
73 % 'nl' stays for 'new line'
74
75 hello:-
76     write("Greetings, I'm here to assist you"), nl,
77     write("Please, type your user-code in order to check your identity.."), nl.
78
79 ok_id(X):-
80     is_id(X)->(
81     write("***User identified successfully***"), nl,
82     write("Welcome: "), write(X), nl
83     );
84     bad_id(),
85     endme.
86
87 %If the user is not registrated in the system, the user can't be identified
88 bad_id():-
89     %not(is_id(X)),
90     write("***GENERIC ERROR: ~bad_login_user_not_identifed***").
```

4 So, What's Next?

È possibile espandere il progetto, aggiungendo features attualmente non sviluppate:

- Possibilità di modifica della Knowledge Base tramite query;
- Controllo sulle possibili incompatibilità tra i vari medicinali;
- Implementazione di un'interfaccia più User-Friendly;
- Ampliamento di scala tramite l'utilizzo di Codice Fiscale ed un database centrale a cui potersi collegare in remoto per l'acquisizione dei dati.

5 Strumenti Utilizzati

- Swi-Prolog
- Overleaf

6 References

- Slides del Corso