German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming Languages, Spring term 2018
Project Description
"Your Personal Assistant"
Deadline: 9/4/2018

A chatbot is a computer program that is able to interact with users. Interaction is carried out using natural language (e.g. English, Arabic, ... etc) [1]. In addition to being used for entertaining users, chatbots have been applied in different fields including information retrieval and e-commerce [1]. You can currently find a lot of Facebook based businesses using chatbots for handling and interacting with their customers.

A chatbot could be simply pre-programmed to choose a random answer from a big pool of possible responses. However, with chatbots being used in many interesting and complicated tasks, they should be intelligent to actually become helpful.

The aim of this project is to explore the use of Prolog as a general purpose language for building a chatbot.

Chatbot Main Functionalities

Our Prolog personal assistant is specifically built to help its users regarding what they could eat. The assistant should give information about food ingredients, calories, and meals.

- The assistant receives questions from the user and answers them.
- The user gets back those answers and asks more questions, if he/she wants to.
- It can also receive information about the user, for example, what he/she had for a specific meal.
- In order to have a more intelligent agent, the chatbot should keep track of the previously asked questions and responses.
- At the end, the assistant should print a report for the user about the food he/she ate during the day.

Chatbot Knowledge Base

The program will know information about the following:

- Total Number of Calories to consume in one day
- Food Names
- Food Ingredients
- Food Categories
- Food Calories
- Meals, and which Food Names can be eaten at them

All of these information will be given to you in a file `foods.pl`. The file contains the information as facts in the following form (**YOU ARE NOT ALLOWED TO MODIFY THE INFORMATION IN THIS FILE**):

```
%% Information about food ingredients
%% and their associated categories

prop(tomato,is,vegetable).
prop(onion,is,vegetable).

prop(strawberry,is,fruit).
prop(mango,is,fruit).

%% Information about food names and their ingredients

prop(green_salad,contain,tomato).
prop(green_salad,contain,onion).

%% Information about food ingredients
%% and how many calories they contain

prop(tomato,contain,15,cal).
prop(onion,contain,25,cal).

%% Information about meals and which food should not
%% be eaten at them

prop(yogurt,not,lunch).
prop(pizza,not,breakfast).
```

Language of Communication

The program is capable of understanding restricted form of statements those will be described in details in subsection 'User Language'. Moreover, it can reply with restricted form of answers 'Assistant Language'.

User Language

The user is allowed to ask one of the following questions:

a) How many calories does *[a food type / a food ingredient]* contain?

   Example: `How many calories does cheese contain?`

b) What does *[a food type]* contain?

   Example: `What does green_salad contain?`

c) Can I have *[a food type]* for *[a meal type]*?

   Example: `Can I have croissant for breakfast?`

d) What is *[a food ingredient]*?

   Example: `What is fig?`

e) How many calories do I have left?

f) What kind of *[a food category]* does *[a food type]* contain?

Example: `What kind of protein does pizza contain?`

g) Is *[a food ingredient]* a *[a food category]* in *[a food type]*?

Example: `Is minced_meat a protein in lasagne?`

h) What can I have for *[a meal type]* that contains *[a food ingredient]*?

Example: `What can I have for lunch that contains pasta?`

In addition to that, the user can state one of the following facts:

a) I ate *[a food type]* for *[a meal type]*.

Example: `I ate pizza for lunch.`

b) I do not eat *[a food ingredient]*.

Example: `I do not eat fish.`

If the user wants to end the communication session with the assistant, the user should write the following command:

```
quit.
```

Assistant Language

The assistant begins the communication with the user by printing the following prompt:

```
Welcome to your personal assistant
```

The assistant can reply to any *correctly formatted* user question with one of the following statements:

a) *[a food type]*

Example: `pizza`

b) *[a food ingredient]*

Example: `hotdog`

c) *[a food category]*

Example: `fruit`

d) *[Number]* Calories

Example: `187 Calories`

e) You can have *[a food type]* for *[a meal type]*

Example: `You can have yogurt for dinner`

f) *[a food type]* is not suitable for *[a meal type]*

Example: `fried_chicken is not suitable for breakfast`

g) I do not know

h) I told you that before

i) Nothing from what I know

j) Yes

k) No

If the user asks or states something which is not *correctly formatted* or not in the language described above, the assistant should reply with the following statement:

```
I can not understand you
```

In case the user stated a fact in one of the two possible forms, the assistant will only confirm reading the statement correctly by saying:

```
Ok
```

If the user commanded to end the session by writing `quit.`, the assistant should print a report for the user mentioning what the user ate in each meal. The report should have the following form:

You had *[a food type / -]* for breakfast

You had *[a food type / -]* for lunch

You had *[a food type / -]* for dinner

Bye

Example:

```
You had spanish_egg for breakfast
You had hamburger for lunch
You had yogurt for dinner
Bye
```

In case the user skipped one of the meals. For example, he/she did not have breakfast.

Example:

```
You had - for breakfast
You had hamburger for lunch
You had yogurt for dinner
Bye
```

Checking the Syntax Validity of User Statements

Prolog supports building grammar rules using what is called definite clause grammars; or DCG for short. Such rules could be used for parsing purposes or restricting the formats of possible inputs [2]. One possible approach to check the validity of the user input with respect to our restrictions is to build a grammar to parse that input.

Regarding this issue, you are free to choose the way you check for the statements' validity. Either by using DCG or by implementing your own predicates.

Valid Combinations of User and Assistant Messages

Question Number	Possible Replies	Condition for Reply
Q(a)	d	In case you know that food type or ingredient.
Q(b)	b	In case you know that food type.
Q(c)	e	In case you know that food type. Moreover, this food type should be suitable for that meal. In addition to that, if the user ate that food type the user will not exceed the calories limit. When the assistant replies with that answer, you assume that the user will eat what the assistant told him/her. So this reply affects the calories remaining.
	f	In case you know that food type. However, this food type is not suitable for that meal.
	k	In case you know that food type. Moreover, this food type should be suitable for that meal. However, if the user ate that food type the user will exceed the calories limit.
Q(d)	c	In case you know that food ingredient.
Q(e)	d	In case you know all the food types eaten by the user so far.
Q(f)	b	In case you know that food type and category. Moreover, it should have at least one ingredient from that food category.
	i	In case you know that food type and category. However, that food type has no ingredient from that food category.
Q(g)	j	In case you know that food type, ingredient, and category. Moreover, that food ingredient falls under that category, and it is found as an ingredient in that food type.
	k	In case you know that food type, ingredient, and category. However, that food ingredient either does not fall under that category, or it is not an ingredient in that food type.
Q(h)	a	In case you know that food ingredient. Moreover, this food type should be suitable for that meal, contains that ingredient, and does not contain anything from the ingredients that the user hates. In addition to that, if the user ate that food type the user will not exceed the calories limit.
	i	In case you know that food ingredient. Moreover, this food type should be suitable for that meal, contains that ingredient, and does not contain anything from the ingredients that the user hates. In addition to that, if the user ate that food type the user will not exceed the calories limit.
All	g	This will be the reply for any question that you can't answer. For example, if the user is asking about the remaining calories and the user previously stated that he/she ate something that you do not know. Thus, you do not have the information of how many calories it contains. As a result you should not be able to calculate the remaining calories. Anything that you are not sure of, should be answered with that reply.
	h	In case the answer of the question will be a repetition of something you replied with before to the same question. Which means that you can't generate a new reply because all what you can reply with, you already said it before.

Tabelle 1: Questions and Answers Combinations

Knowing Total Amount of Calories

The total amount of calories to be consumed in one day, should be given as a fact by the predicate `totalCal(X)`, where `X` is the total amount of calories. Whenever you need to know the calories, you should use this predicate. You **SHOULD NOT** hard code it in your implementation. So if we changed the total amount, we should get different replies in some cases. You can assume for now that it is 1800. Anyways, it shouldn't affect your implementation by any means.

Predicates to be Implemented

readInputTillQuit/0

This predicate should be true prompting for input as long as the user enters any sentence other than "quit.". In case the user enters "quit." it succeeds and stops. Every time the user is asked for input they see the character `>`.
Example:

```
?- readInputTillQuit.
> Welcome to your personal assistant
> How many calories do I have left?
> 1800
> quit.
> You had - for breakfast
You had - for lunch
You had - for dinner
Bye
true
```

isValid/1

`isValid(Q)` holds in case the question `Q` is in one of the valid formats.

filterProp/2

`filterProp(Relation,Result)` holds if Result is a list of pairs of terms connected through the property `Relation`.

matchFirst/3

`matchFirst(T1,LF,LM)` holds if `LM` is a list with the same length as list of pairs `LF`. Each Element in `LM` is in the format (`E-Occ`) where `Occ` is 1 iff `T1` matches the first element of the corresponding pair in `LF` and `E` matches the second, and is 0 otherwise.

matchSecond/3

`matchSecond(T1,LF,LM)` holds if if `LM` is a list with the same length as list of pairs `LF`. Each Element in `LM` is in the format (`E-Occ`) where `Occ` is 1 iff `T1` matches the second element of the corresponding pair in `LF` and `E` matches the first, and is 0 otherwise.

mergeMatchLists/3

`mergeMatchLists(ML1,ML2,R)` holds if `R` is the result of merging the mathcing lisrs `ML1` and `ML2`.

bestMatches/2

`bestMatches(ML,BL)` holds if `BL` is a list of matching with the same score whoch represent the highest matching score in `ML`.

foodCal/2

Let `F` be a food type or ingredient, then `foodCal(F,C)` holds if `C` is the Calories contained in `F`.

foodCalList/2

Let `FL` be list a food type or ingredient, then `foodCalList(FL,C)` holds if `C` is the Calories contained in the list of foods `FL`.

calcCalories/4

`calcCalories(F,PQ,PR,C)` holds if the the remaining calories after having `F` given the previous questions `PQ` and previous responses `PR` is `C`.

getDiffAnswer/5

`getDiffAnswer(Q,PQ,PR,CR,R)` holds if `R` is the a new response from the list containing the candidate answers `CR` for the the question `Q`. You can retrieve the previous answers for `Q` form the list of previous questions `PQ`, and the list for previous replies `PR`.

responseO/4

`responseO(Q,PQ,PR,LR)` that holds if `LR` is a list possible response-distance pairs to the question `Q` given the previous questions `PQ` and the previous responses `PR`. For example the list:
`[pizza-2, hamburger-1,salad-3, pasta-0]` means that the response pizza has 2 matchings with the terms in `Q` which hamburger has 1 matching , .. etc.

response/4

`response(Q,PQ,PR,R)` that holds if `R` is the response to the question `Q` given the previous questions `PQ` and the previous responses `PR`.

listOrderDesc/2

`listOrderDesc(LP,OLP)` holds if `OLP` is a list containing the same elements in list of pairs `LP` ordered descendingly according to the second value of the pair

foodFromHistory/2

`foodFromHistory(HL,FL)` holds if `FL` is the list of foods eaten by the user mentioned in the statements of `HL`.

getUnlikedIngredients/2

`getUnlikedIngredients(PQ,FL)` holds if `FL` is the list of food ingredients hated by the user mentioned in the previous questions `PQ`.

Example of Running Script

```
?- readInputTillQuit.
> Welcome to your personal assistant
> weird sentence.
> I can not understand you
> How many calories do I have left?
> 1800 Calories
> How many calories does cheese contain?
> 431 Calories
> I ate spanish_omelette for breakfast.
> Ok
> How many calories does spanish_omelette contain?
> 481 Calories
> how many calories do I have left?
> 1319 Calories
> What kind of vegetable does green_salad contain?
> tomato
> what kind of vegetable does green_salad contain?
> onion
> what kind of vegetable does green_salad contain?
> lettuce
> Is tomato a vegetable in pizza?
> Yes
> Is tomato a vegetable in pizza?
> I told you before
> What kind of fat does pizza contain?
> oil
> I ate pizza for lunch.
> Ok
> how many calories do i have left?
> 273 Calories
> can I have fried_chicken for dinner?
> fried_chicken is not suitable for dinner
> How many calories does yogurt contain?
> 218 Calories
> can I have yogurt for dinner?
> You can have yogurt for dinner
> quit.
> You had spanish_omelette for breakfast
You had pizza for lunch
You had yogurt for dinner
Bye
true
```

Literatur

[1] Bayan Abu Shawar and Eric Atwell. Chatbots: Are they really useful? *LDV Forum*, 22(1):29–49, 2007.

[2] DCG Grammar rules. `http://www.swi-prolog.org/pldoc/man?section=DCG`.