Artificial Intelligence Lab 2

Assignment/Report

By Aaron Gangemi – N1903373E

Course: Artificial Intelligence – CZ3005

Lab Number: TSR1 – Thursday 12.30pm -2.30pm

Contents

[Overview of Assignment: 3](#_Toc23794110)

[Instructions to run the program: 3](#_Toc23794111)

[Structure of Program: 3](#_Toc23794112)

[Implementation Explained: 4](#_Toc23794113)

[Implementation: 5](#_Toc23794114)

[Output of Program: 9](#_Toc23794115)

[Conclusion: 9](#_Toc23794116)

# Overview of Assignment:

For the Lab 2 Assignment, I decided to choose assignment 2. The task involved writing a prolog script that acts as a parent talking to its kid (the user) about their day at school. The user can only respond back to the parent with yes or no. For the purposes of exiting the program, the user can also enter ‘q’ to exit. The program begins by asking a question. If the user enters ‘y’ which denotes yes, then the user should see a question of similar context displayed. However, if the user enters ‘n’, which denotes no, then the user will have a random question asked that has no relation to the previous question. If the user does not enter either ‘y’, ‘n’ or ‘q’, then the program assumes the user’s response is no. The parent can take these responses and converse intelligently with the kid. The program is implemented and executed in prolog. Instead of implementing a GUI, I have used the prolog console to incorporate input and output functionality.

# Instructions to run the program:

To run the program, the following must be done:

1. Install a working version of Prolog.
2. Change the directory to the directory containing “ParentTalkingToKid.pl” using the following command. I have my file in Lab2\_Assignment.
3. A close up of a mans face

   Description automatically generated
4. Load the file into prolog using:



1. Run the program using:

A picture containing furniture

Description automatically generated

1. The program will then run and the user must enter either “y/n/q” followed by a terminating full stop to answer.

# Structure of Program:

The assignment only contains one Prolog file containing various functions. The “ask(X)” function will be typed into the prolog console to run the program. In this function, the getNextQuestion(X, L) rule/function is called upon. This rule/function begins by getting a random question from any list. If the user enters yes later in the program, this function will call on relatedQuestion(X, Y) to find a similar question to the one that was just asked. This function implements findall/3 to get the associated result. From here, the rule/function “processOption(L)” is run to process both the yes list and no list to correspond appropriately to the response that will be given by the user. This function ensures that the question is not asked twice by converting the list to a set which can only contain unique elements. The user is then asked to input a response by calling getResponse(Y) and after the user has entered their input, if the response is not ‘q’ then ask(Y) is called to loop back to ask the user the next question. Furthermore, the program below contains 5 lists. Each list contains associated questions regarding that topic. These questions are displayed to the user.

# Implementation Explained:

The first 2 statements “: - dynamic yes/1” and “: - dynamic no/1.” Both statements inform the prolog interpreter that the predicates “yes” and “no” may change during execution. This makes sense because when the user enters either ‘y’ or ‘n’, the question is appended to one of these lists during runtime. The following is a overview of each function:

* **ask(X):** is run from the prolog console and calls getNextQuestion(X, L) and processOptions(L).
* **getNextQuestion(X, L):** is called upon after ask(X) is run. The first time it is run, this function retrieves a random question from any list incorporated in the program. It does this by calling upon Findall/3. Findall/3 creates an instantiation list, storing the result/s of the **randomQuestion(Y, X):** This result is then displayed to the user as the first question. From here, if the user answers yes to a question, then the predicate “yes(X)” is updated and this function calls the Findall/3, passing in a call to the relatedQuestion(X, Y) function. This will retrieve a list storing any related question and instantiate a list to store the results.
* **processOptions(L):** The processOptions(L) function is used to process the lists so that no element in the current list can be repeated. This is first done by retrieving all questions that the user has responded to. It does this by using the findall/3 function. After this, both lists are appended to a “ResponseList”. The “ResponseList” is then converted into set ‘R’ using Prolog’s in-built “list\_to\_set/2” function. Properties of a set ensures that the elements are unique. In addition, the list passed into processOptions(L) is also converted into a set called “ListSet”. Proceeding this, the program then looks for the differences between “ListSet” and ‘R’. Following this, the member function then uses the just found difference to determine the next question. After this is found, getResponse(Y) is called.
* **getResponse(Y):** The function starts by printing the question that was found using the member function in processOptions(L). The algorithm then waits for the user to input either “y/n/q/other.” This function then takes the users inputted data and uses conditional statements to determine how to process the response. If the user enters ‘y’, then the program asserts the question into “yes(Y)” using Prolog’s “assert/1 function. If the user enters ‘n’, then the program will assert the question to “no(Y)”. If the user enters ‘q’, then the program aborts, and if the response is neither, then the program assumes the response is no and continues to assert “no(Y).” Finally, if the response given is anything but ‘q’, then the program loops to ask the user the next question.
* If the user enters ‘y’ upon being asked by the program, the **relatedQuestion(X, Y)** function is called. This function is implemented 5 times throughout my program. One for each list in the program. These functions determine which list the previous question asked by the program was a member of and then gets the next member from that list. This data is then returned to the findall/3 function which is called in the processOption(L) function.
* On the other hand, if the user enters ‘n’, which denotes no on user input, the program calls the **randomQuestion(X, Y)** function. This function first initialises a variable to each list.

**Play = A, Eat = B, Learn = C, Sports = D, Friends = E.**

The algorithm then works out which list the previous question was a member of. It then appends the alternative 4 lists together into one larger list and finds a random item from that list. By appending the alternative 4 lists, the program ensures that a question from the same list as the previous question cannot be asked. For example, if the user is asked "Did you learn math today?", which is a question from the learn list, and the users responds no. The program will append the lists “play”, “eat”, “sports” and “friends” together and the next question will be a random question from that list.

* The final 2 statements of the program display:



This initialises both yes() and no() to begin with no elements before the program begins.

# Implementation:

|  |  |
| --- | --- |
|  | :- dynamic yes/1. |
|  | :- dynamic no/1. |
|  | /\*yes and no are set to dynamic as they can change at run time. |
|  | This is due to options being added based on the response |
|  | provided by the user\*/ |
|  | ask(X):- getNextQuestion(X,L), processOptions(L). |
|  | /\*ask(X) will be user to run the program. It calls the below functions\*/ |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* getNextQuestion: |
|  | \* Purpose: runs the relatedQuestion(X,Y) function if the response is |
|  | yes. If the response is no, then finds a random question to run. Uses |
|  | the find all to get all possible questions and then gets chooses only 1 |
|  | in each function. |
|  | \* Parameters: Y = Question, L = List |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | getNextQuestion(Y,L):- |
|  | yes(Y), findall(X,relatedQuestion(X,Y),L); /\*Get related Question\*/ |
|  | findall(X,randomQuestion(Y,X),L). /\*Get random question.\*/ |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* processOptions: |
|  | \* Purpose: Gets all values in the yesList and noList. Appends all found |
|  | values to the responseList. By converting the list passed in to a |
|  | set, this ensures each value is unique so it is not asked twice. The |
|  | same principle is applied for the responseList. They are then |
|  | subtracted from each other and a member of the subtracted List is |
|  | found. This method then calls getResponse(Y) to ask for user input. |
|  | \*Parameters: L = List |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | processOptions(L):- |
|  | findall(X,yes(X),YesList), %find everything in yes list |
|  | findall(X,no(X),NoList), % find everything in no list |
|  | append(YesList,NoList, ResponseList), |
|  | list\_to\_set(L,ListSet), %Ensure all elements are unique |
|  | list\_to\_set(ResponseList,R), %Ensure all elements are unique. |
|  | subtract(ListSet,R,ValidOption), %find differences |
|  | member(Y,ValidOption), getResponse(Y). %ask for response of question Y |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* getResponse(Y): |
|  | \* Purpose: Takes in a question, and asks the user in prolog to |
|  | enter a valid response (y/n/q). If neither of these is found, then the |
|  | program assumes the response is no. The next question is then asked. |
|  | \* Parameters: Y = Question |
|  | \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | getResponse(Y):- |
|  | print(Y), nl, read(Response), %Print question and get response |
|  | ((Response == y -> assert(yes(Y)); %if response yes, add to yesList |
|  | Response == n -> assert(no(Y)); %if response no, add to noList |
|  | ((Response \= n, Response \= y, Response \= q) -> print("Assuming that's a no"),nl, assert(no(Y))); %if response isn't yes/no/quit, then assume no and inform user |
|  | Response == q -> print("exiting program..."),abort)), %if response if q,  abort program. |
|  | ask(Y). %ask next question |
|  | /\*ask(Y). \*/ |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | play[] = a list of question that are asked when the user responds yes to |
|  | a previous play question or selects no to a different category question |
|  | and play is the new question |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | play(["Did you play with the slides?", |
|  | "Did you play in the sandbox?", |
|  | "Did you play with the toys?", |
|  | "Did you play with the trains?", |
|  | "Did you play with the cars?", |
|  | "Did you use the playmat?", |
|  | "Did you play with the bears?", |
|  | "Did you play with the animals?", |
|  | "Did you play on the swings?", |
|  | "Did you use the soft\_toys?", |
|  | "Did you play with the alphabet?", |
|  | "Did you play with the numbers?"]). |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | eat[] = a list of question that are asked when the user responds yes to |
|  | a previous eat question or selects no to a different category question |
|  | and eat is the new question |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | eat(["Did you eat the cake I packed for you?", |
|  | "Did you eat the toffee I packed for you?", |
|  | "Did you eat any candy today?", |
|  | "Did you eat the sandwich I packed for lunch?", |
|  | "Did you eat any pizza today?", |
|  | "Did you eat any cheerios today?", |
|  | "Did you eat any veggies today?", |
|  | "Did you eat any fries today?"]). |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | learn[] = a list of question that are asked when the user responds yes |
|  | to a previous learn question or selects no to a different category |
|  | question and learn is the new question |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | learn(["Did you learn math today?", |
|  | "Did you learn english today?", |
|  | "Did you read any books today?", |
|  | "Did you learn science today?", |
|  | "Did you get any homework?", |
|  | "Did you learn humanities today?", |
|  | "Did you use the computers today?"]). |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | sports[] = a list of question that are asked when the user responds yes |
|  | to a previous sports question or selects no to a different category |
|  | question and sports is the new question |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | sports(["Did you play any sports today?", |
|  | "Did you play soccer today?", |
|  | "Did you play cricket today?", |
|  | "Did you run much today?", |
|  | "Did you sprint much today?", |
|  | "Did you play any football today"]). |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | friends[] = a list of question that are asked when the user responds yes |
|  | to a previous friends question or selects no to a different category |
|  | question and friends is the new question |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | friends(["Did you see your friends today?", |
|  | "Was Billy at school today?", |
|  | "Was Jimmy at school today?", |
|  | "Did you play with your friends on the playground?"]). |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* relatedQuestion(X,Y): |
|  | \* Purpose: There are 5 related question functions, one for each list. |
|  | \* If the question passed in is a member of the associated list, then |
|  | \* this will ask the next question in the list. |
|  | \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | relatedQuestion(X,Y):- play(L), member(X,L), member(Y,L).  %get next question in play list |
|  | relatedQuestion(X,Y):- eat(L), member(X,L), member(Y,L).  %get next question in eat list |
|  | relatedQuestion(X,Y):- learn(L), member(X,L), member(Y,L).  %get next question in learn list |
|  | relatedQuestion(X,Y):- sports(L), member(X,L), member(Y,L).  %get next question in sport list |
|  | relatedQuestion(X,Y):- friends(L), member(X,L), member(Y,L).  %get next question in friends list |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* randomQuestion(X,Y): |
|  | \* Purpose: Checks which list the previous question was a member of. |
|  | It then appends all other lists that the previous question is not apart |
|  | of and asks a question from this list. |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | randomQuestion(X,Y):- |
|  | play(A),eat(B),learn(C),sports(D),friends(E), |
|  | ((member(X,A)-> append(B,C,BC),append(BC,D,BCD),append(BCD,E,BCDE),  random\_member(Y,BCDE));  %Check if a member of play, then append other lists |
|  | (member(X,B)-> append(A,C,AC),append(AC,D,ACD),append(ACD,E,ACDE),  random\_member(Y,ACDE)); %Check if a member of eat, then append other lists |
|  | (member(X,C)-> append(A,B,AB),append(AB,D,ABD),append(ABD,E,ABDE),  random\_member(Y,ABDE));  %Check if a member of learn, then append other lists |
|  | (member(X,D)-> append(A,B,AB),append(AB,C,ABC),append(ABC,E,ABCE),  random\_member(Y,ABCE));  %Check if a member of sports, then append other lists |
|  | (member(X,E)->append(A,B,AB),append(AB,C,ABC),append(ABC,D,ABCD),  random\_member(Y,ABCD))). %Check if a member of friends, then append other lists |
|  | yes(nothing). %Set yes list initially to have no elements |
|  | no(nothing). %Set no list initially to have no elements |

# Output of Program:

The below images are output of my program:

A close up of text on a black background

Description automatically generated

The tree diagram also shown below is a graphical representation of how the above output is traced:

A close up of a map

Description automatically generated

# Conclusion:

In conclusion, I have developed a basic prolog program that utilizes predicates, rules and facts to analyse the kid’s response to the parent. Using such data develops a knowledge base in which the program/parent uses to create a conversation with the kid and enable the parent to converse intelligently with the kid.