Assignment Artificial Intelligence CZ3005

Subway sandwich interactor

Koch Philipp Frederik Edward, N1903454H

2019
November

1 Task

The goal of the task was the implementation of a subway sandwich interactor. This was to guide the customer through the selection. It had to be taken into account that some options could be chosen more often, such as the chosen vegetables. Furthermore, a restriction must be made by previously selected options, so that no meat-containing ingredients are permitted in the vegetarian menu for example. Hints were offered for the task, which were also used. An *options/1* rule and a *selected/2* rule were proposed in the notes. The *options/1* rule should offer the possible options for the respective sandwich part and the *selected/2* rule should assign an option for the respective sandwich part. *selected(0)* should trigger a jump on the list of sandwich parts and initiate the next assignment. If X = 1, a *done/1* rule should display the options already selected.

2 Implementation

The hints described above are very suitable for a command line based dialog program and were therefore chosen as the basis for the following implementation. For this type of program, where a rule is called again each time, an abstracted state is indispensable. The selected properties of the sandwich, the state of the selection and the number of possible reusable options must be considered. Therefore, three predicates were chosen, which should manage these states during runtime (*collection/1*, *state/1* and *counter/1*). Both predicates *state/1* and *counter/1* where only used with one Variable, so that the state is first retracted and then newly asserted.

Since the list, from which the user may select, is to be changed with *selected(0)*, the central state management was implemented here. The respective state is changed by every call and at the end is set again to the initial state and for the two calls of state transition, the rule *switchState/2* was implemented. What is special about this method is the treatment of the last state, where the *collection/1* is reset, and the treatment in state veggie, where multiple selection is allowed. The more detailed treatment of a multi-selection is defined in the rule *multipleSelection/1*. The *counter/1* predicate is also used by limiting the multiple selection. Finally, *collection/1*, *counter/1* is being reset, while *state/1* is changed to the next (initial) state.

In the *selected/2* rule, the chosen option is added to the *collection/1* predicate. For this purpose, the corresponding list in the knowledge base is determined via *call/2* and then the list is compared with the current state. If the state is correct, a list of possible options is created,using the *suggested/2* rule which is then used as a reference. If the selected item is in the created list, the chosen option is added to the *collection/1* via *addToSelection/1*. The *suggested/2* rule filters the results based on the previous choice. First a list out of the *collection/1* predicate is created with the *findnsols/4* rule and then checked if a certain choice has been made. If a choice was detected which is connected to a certain track, the options of this track are returned here as the variable *Output*. Otherwise, the entered list is returned without modification. Furthermore the different tracks are needed, as well as the rules for the specific tracks. For the specific tracks, a list of allowed options is stated in the knowledge base. With the previous rules an endless operation is possible. To view all chosen options, the *done/1* rule was implemented. This rule can be used to print the current selected list. First, *collection/1* is printed as a list to the command line via the function *findnsols/4*. *Options_/1* then outputs the elements in several lines. The rule *printhelpnote()* outputs a hint to the help rule. Furthermore the rule *options/1* was implemented, which outputs a list in several points. In each call the head of the list is printed and the *options_/1* rule is called again with the tail of the list. In order to simplify the interaction, the rules *printhelpnote/0* and *helpsubway/0* were also implemented, but they only print information as text.

Code

```
:- (dynamic collection/1).
:- (dynamic state/1).
:- (dynamic counter/1).


% listed with all selected items
collection(nothing).
% state for asking the reight questions
state(breads).
% state for toppings that can be choosen multiple times
counter(0).

% User Experience
printhelpnote():- print("Type␣helpsubway().␣for␣help!"), put(10).

helpsubway():-
    print("Use␣options(<parts-of-your-sandwich>).␣to␣get␣the␣information␣about␣all␣items."),put(10),
    print("parts-of-your-sandwich:␣breads,␣main,␣veggies,␣sauce,␣sides"), put(10),
    print("Use␣selected(<option>,<parts-of-your-sandwich>).␣to␣choose␣your␣items.").


% compute suggested Options
suggested(L, Output) :-
    findnsols(100, X, collection(X), Z),                  % get a list of the previous selection
    (   member(healthy, Z)                                % check wether healthy is part of the
          previous selection
    ->  findnsols(100, Y, healthytrack(L, Y), Output)     % Assign the list of the allowed options to
          the Output Var
    ;   member(veggie, Z)                                 % check wether veggie is part of the previous
          selection
    ->  findnsols(100, Y, veggietrack(L, Y), Output)      % Assign the list of the allowed options to
          the Output Var
    ;   append([], L, Output)                             % Output has to be L
    ).


% display options
options(Name) :-
    call(Name, L),                                        % get List of predicate with the name 'Name'
    suggested(L, Lst),                                    % get the allowed suggestions for this list
    print("The␣following␣options␣are␣available␣for␣your␣order:"),
    options_(Lst).                                        % print the possible options

% helper function to display the items in multiple lines
options_([]).                                             % termination condition
options_([Head|Tail]) :-
    print(Head),                                          % print the first element of the list
    put(10),                                              % newline
    options_(Tail).                                       % recursive call of the function with the
          rest / tail of the list


% switch state -> next selection
selected(0) :-
    state(X),                                             % get current state
    (   X==breads                                         % check for specific state (1)
    ->  switchState(breads, main),                        % change to the new state  (2)
        print("Choose␣the␣main␣topping␣now!"),
        put(10), printhelpnote()
    ;   X==main                                           % analogous to (1)
    ->  switchState(main, veggies),                       % analogous to (2)
        print("Choose␣the␣vegetables␣now!"),
        put(10)
    ;
    % specific case for veggies. There can be more than one item selected
    X==veggies                                            % analogous to (1)
    ->
        (
            mulitpleSelection(maxVeggies);                % check for multiple selection
            % continue with the next case, set new state
            switchState(veggies, sauce),                  % switch to next state
            print("Choose␣the␣sauce␣now!"),
            put(10)
          )
     ;
    X==sauce                                                    % analogous to (1)
     ->  switchState(sauce, sides),                             % analogous to (2)
         print("Choose␣the␣sides␣now!"),
         put(10)
     ;   X==sides                                               % analogous to (1)
     ->  switchState(sides, breads),                            % analogous to (2)
         done(1),                                               % show results

         % reset states

         abolish(collection/1),                                 % clear collection predicate
         assert(collection(nothing)),                           % reassert collection predicate with nohing
         counter(Y),                                            % get actual Variable from counter
         retract(counter(Y)),                                   % remove Variable from counter
         assert(counter(0)),                                    % reassert counter predicate with 0
         print("Thanks␣for␣eating␣at␣Subway"),
         put(10)
     ).

% change state
switchState(X,Y):- retract(state(X)), assert(state(Y)).        % retract old state and assert new state

% Rule for multiple selection
mulitpleSelection(MaxPred):-                                    % Variable MaxPred can be any maximum for a multi selection
     call(MaxPred, MAX),                                        % get Variable of predicate MaxPred
     counter(Number),                                           % get Counter Variable
     (
     % Ask for more toppings
     Number < MAX->                                             % check if the maximum is reached
         print("Do␣you␣want␣to␣choose␣more?␣[y/n]"),            % askk the user for more toppings
         read(Like),
         Like==y
         ->  retract(counter(Number)),                          % update the counter state, retract actual number
             assert(counter(Number + 1))                        % assert new updated number
     ).


% add order
selected(X, L) :-
     call(L, Lst),                                              % get List from predicate name
     state(Y),                                                  % get state
     (   Y==L                                                   % check if selected is in the correct state
     ->  suggested(Lst, SuggLst),                               % get the suggested list
         (   member(X, SuggLst)                                 % check if the option is member of the suggested list, so that the options will stay on track
         ->  addToSelection(X),                                 % add to selection
             print("Good␣choice."),
             put(10),
             selected(0)                                        % go to next list
         ;   print("I␣am␣sorry.␣This␣item␣is␣unfortunately␣not␣available.")
         )                                                      % error messae if the option is not member of the suggested list
     ;   print("Something␣went␣wrong.␣You␣have␣to␣choose"),     % error message if the state is not correct
         print(Y),
         put(10)
     ).

addToSelection(X) :-
     (   collection(Y),                                         % get the collection
Y==nothing                                                      % check if it's nothing to (Beginning of the process)
     ->  retract(collection(Y)),                                % retract nothing
         assert(collection(X))                                  % assert the choosen option
     ;   assert(collection(X))                                  % assert the choosen option
     ).


% show options
done(1) :-
     print("You␣selected:"),
     put(10),
     findnsols(100, Y, collection(Y), History),                 % get the collected options as a list
     options_(History),                                         % print the list
     put(10),
     printhelpnote().                                           % print a help note for the user


% specific tracks
veggietrack(Lst, X) :-                                          % check if element of Lst is also part of the veggietrack
     veggiemember(Vl),                                          % get all veggie options
     member(X, Lst),                                            % check if the Variable is in Lst and in Vl
     member(X, Vl).
healthytrack(Lst, X) :-                                         % analogous to veggietrack
     healthymember(Vl),
     member(X, Lst),
     member(X, Vl).


% Knowledge base

% Max for Veggie selection
maxVeggies(3).                                                  % const for the maximum of veggie selections


% everything that is allowed in a specific track
veggiemember([lettuce, tomato, mustard, chipotle, bbq, mayonaise, chilli, soda, cookie, apple]).
healthymember([lettuce, tomato, chipotle, bbq, chilli, soda, apple]).


% offers
```

```
breads([parmesan, honeywheat, italian, cheddar, flatbread, honeyoat]).
main([chicken, tuna, veggie, italian_bmt, healthy]).
veggies([cucumber, lettuce, tomato, jalapeno, spinach]).
sauce([mustard, chipotle, bbq, mayonaise, chilli, cesarsauce]).
sides([soup, soda, cookie, apple]).
```

3 Using the program

To use the program, you have to navigate to the respective directory and execute swi-prolog. Via ['name of the program ']. the program is loaded. The rules *selected/2*, *helpsubway/0*, *options/1* and *done(1)* are intended for use. Due to the architecture, a certain sequence must first be processed. They have to be processed one after the other:

1. Bread (*breads*),
2. Main (*main*),
3. Vegetables (*veggies*),
4. Sauces (*sauce*),
5. Sides (*sides*)

can be selected. In between, the state can be queried again and again with *done(1)* and *options/1* can display the possible options for a step. The selection takes place with *selected/2*.

4 Conclusion

With this architecture the program can be easily extended. No new rules have to be implemented, only the states have to be adapted and optionally a multiple selection has to be considered. Further tracks can also be added, which also require only minor changes during implementation.