NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

School of Computer Science and Engineering

CZ3005: Artificial Intelligence

Semester 2 2020/2021

Assignment 3: Talking Box with Prolog

Subway Sandwich Interactor (WicheryBot)

Submitted by:

Chew Kit Waye Andrel

U1920276H

Group: BCG3

Date: 12 Apr 2021

Contents Page

Subway Sandwich Interactor

Overview

With the help of a Knowledge-Based System(KBS), decision making and problem solving abilities of programs are improved. In this assignment, a telegram bot (@wicherybot) is configured to support the prolog codes, while bringing an interactive experience. Several decision factors play a part for a sandwich order, with different users with various dietary preferences. With the help of the KBS, logic of all orders are processed based on meal preference.

Prolog Logic

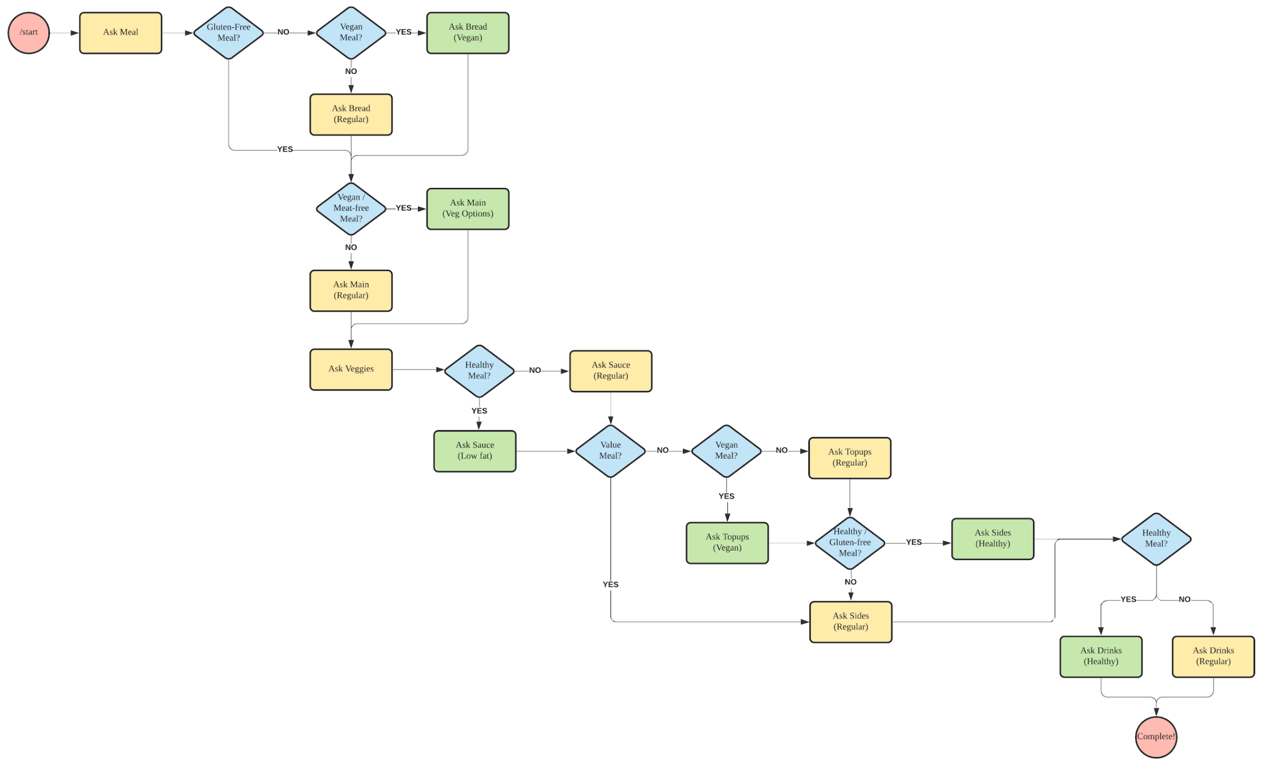


Figure 1: Decision Flow of a Sandwich order

With the use of prolog, the decision flow above was designed, with different meal preferences in mind. A vegan and meat-free meal will show meatless options, with vegan meals having no cheese top-ups and vegan friendly bread options. When a healthy meal is chosen, only healthy sauces and drink options will be given. A value meal will show no top-up option, while a gluten-free meal will not have bread options available. When a regular meal is selected, meal choices for all dietary preferences are shown.

Prolog Codes

Initialisation of the append method

```
% for appending options into a list
append([], Y, Y).
append([H|X], Y, [H|Z]):-
   append(X, Y, Z).
```

True/False Condition Check for Meals

```
% for ensuring that meals are correct
healthy_meal(healthy).
value_meal(value).
vegan_meal(vegan).
veg_meal(meat_free).
gluten_free_meal(gluten_free).
```

Definition of Category-based Choices

: The following shows the possible range of options by category, including dietary specific choices and regular choices. For example, if a vegan meal is selected, only the `vegan_breads` list will be displayed. However, if a regular meal is selected, the lists will be combined and displayed to the user.

```
meals([regular, healthy, value, vegan, meat_free, gluten_free]).
reg_breads([parmesan_oregano, honey_oat]).
vegan_breads([italian_wheat, hearty_italian, multigrain, flatbread, tortilla]).

reg_mains([cold_cut_trio, tuna, egg_mayo, steak, chicken_breast, blt, meatballs]).
veg_mains([mushroom_galore, veggie_patty]).

veggies([cucumber, capsicum, lettuce, onions, tomatoes, olives, jalapenos, pickles]).
reg_sauces([chipotle_southwest, bbq, ranch, mayonnaise, mustard]).
lf_sauces([honey_mustard, sweet_onion, ketchup]).

reg_topups([mozerella_cheese, cheddar_cheese, egg_mayo]).
vegan_topups([avocado, mushrooms]).

reg_sides([chips, cookie]).
lf_sides([banana, granola_bar, mushroom_soup, tomato_soup]).

reg_drinks([soda]).
lf_drinks([bottled_water, orange_juice, green_tea, black_tea, americano]).
```

Combination of Lists

: The following code snippet shows the joining of lists using the append method defined above. The purpose of this is to display the full suite of options for users with regular meals. They are allowed to have selections from any category by default.

```
breads(X):- vegan_breads(A1), reg_breads(A2), append(A1, A2, X).
mains(X):- veg_mains(B1), reg_mains(B2), append(B1, B2, X).
sauces(X):- lf_sauces(C1), reg_sauces(C2), append(C1, C2, X).
topups(X):- vegan_topups(D1), reg_topups(D2), append(D1, D2, X).
sides(X):- lf_sides(E1), reg_sides(E2), append(E1, E2, X).
drinks(X):- lf_drinks(F1), reg_drinks(F2), append(F1, F2, X).
```

Logic of Options Displayed

: The following code snippet from the prolog script shows the dietary logic as depicted in the decision flow above. All decisions will be based on the meal chosen at the start of the ordering process. Gluten free meals will not display the bread category. If vegan is chosen, only vegan breads, mains and top-ups will be displayed. Healthy meals will only show low-fat sauces, sides and drinks.

```
display_meals(X):- meals(X).

display_breads(X):-
   chosen_meals(Y), vegan_meal(Y) -> vegan_breads(X);
   chosen_meals(Y), \+ gluten_free_meal(Y), breads(X).

display_mains(X):-
   chosen_meals(Y), vegan_meal(Y) -> veg_mains(X);
   chosen_meals(Y), veg_meal(Y) -> veg_mains(X);
   chosen_meals(Y), value_meal(Y) -> reg_mains(X);
   mains(X).

display_veggies(X):- veggies(X).

display_sauces(X):-
   chosen_meals(Y), healthy_meal(Y) -> lf_sauces(X);
   sauces(X).

display_topups(X):-
   chosen_meals(Y), vegan_meal(Y) -> vegan_topups(X);
   chosen_meals(Y), \+ value_meal(Y), topups(X).

display_sides(X):-
   chosen_meals(Y), healthy_meal(Y) -> lf_sides(X);
   chosen_meals(Y), gluten_free_meal(Y) -> lf_sides(X);
   sides(X).

display_drinks(X):-
   chosen_meals(Y), healthy_meal(Y) -> lf_drinks(X);
   drinks(X).
```

Logic of Lists

: The following code snippet from the prolog script shows the logic of the lists for display. `default_options(Opt, X)` shows default list of options. `available_options(Opt, X)` displays based on the logic of options based on `chosen_meals(X)` while `chosen_options(Opt, X)` is used for printing the complete list of selected options at the end of every order.

```prolog
% display default list of options
default_options(Opt, X):-
   Opt == meals -> meals(X);
   Opt == breads -> breads(X);
   Opt == mains -> mains(X);
   Opt == veggies -> veggies(X);
   Opt == sauces -> sauces(X);
   Opt == topups -> topups(X);
   Opt == sides -> sides(X);
   Opt == drinks -> drinks(X).

% display available options based on selected meal
available_options(Opt, X):-
   Opt == meals -> display_meals(X);
   Opt == breads -> display_breads(X);
   Opt == mains -> display_mains(X);
   Opt == veggies -> display_veggies(X);
   Opt == sauces -> display_sauces(X);
   Opt == topups -> display_topups(X);
   Opt == sides -> display_sides(X);
   Opt == drinks -> display_drinks(X).

% display list of final selection
chosen_options(Opt, X):-
   Opt == meals -> findall(X, chosen_meals(X), X);
   Opt == breads -> findall(X, chosen_breads(X), X);
   Opt == mains -> findall(X, chosen_mains(X), X);
   Opt == veggies -> findall(X, chosen_veggies(X), X);
   Opt == sauces -> findall(X, chosen_sauces(X), X);
   Opt == topups -> findall(X, chosen_topups(X), X);
   Opt == sides -> findall(X, chosen_sides(X), X);
   Opt == drinks -> findall(X, chosen_drinks(X), X).
```

(Prolog Codes can be found in ChewKitWayeAndrel_qn_3.pl)

Interaction with Prolog (Command Line)

: The prolog statement `display_meals(X)` is used to display all available meal options. `assert(chosen_meals(X))` is then used to append the meal choice to the chosen_meals list. When `chosen_meals(vegan)` is selected, bread options `parmesan_oregano & honey_oat` are not shown. Similarly, only meat-free options are seen for the mains. This is due to the predefined vegan options.

```
?- display_meals(X).
X = [regular, healthy, value, vegan, meat_free, gluten_free].

?- assert(chosen_meals(vegan)).
true.

?- chosen_meals(X).
X = vegan.

?- display_breads(X).
X = [italian_wheat, hearty_italian, multigrain, flatbread, tortilla].

?- assert(chosen_breads(flatbread)).
true.

?- chosen_breads(X).
X = flatbread.

?- display_mains(X).
X = [mushroom_galore, veggie_patty].

?- assert(chosen_mains(mushroom_galore)).
true.

?- chosen_mains(X).
X = mushroom_galore.
```

(Video demo can be found in demo/videos/terminal.mov)

User Manual of Bot

Requirements:
1. Download python3
2. Telegram app

Install Requirements:

```
pip install -r requirements.txt
cd subway-wichery
```

-- activate virtual environment

```
virtualenv venv
source venv/bin/activate
```

-- install requirements

```
pip install -r requirements.txt
```

-- deactivate virtual environment

```
deactivate
```

Telegram:
1. Upon launching Telegram, search for @wicherybot (https://t.me/wicherybot)
2. The bot will be available when the program is up and running.

Run Program:
1. cd to project folder
2. `python3 tele.py`
3. enter /start in telegram bot to begin

Telegram Bot

The WicheryBot has been developed using two python files: `tele.py` and `convert.py`. tele.py acts as the main file for customised telegram messages, while convert.py acts as a bridge between the bot and prolog script.

The following shows some decision rules of the bot as outlined in the prolog script:

- Vegan meals have no cheese or animal products.
- Veggie and Vegan meals will only have meat-free options.
- Healthy meals will not have high fat sauces.
- Value meals does not include top-ups.
- Gluten-free meals include no bread or wheat options.

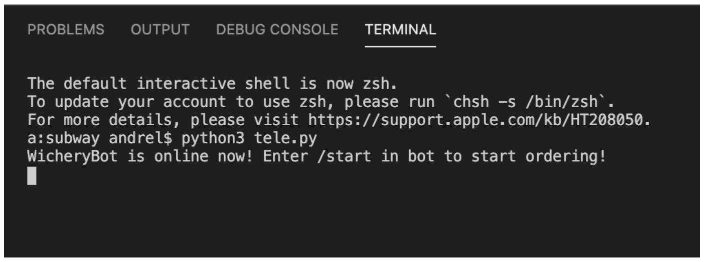
Conversion from Prolog

The python script `convert.py` converts the prolog code to be used in the main program `tele.py`. Using the prolog function `retractall` , the previously selected options are cleared to ensure that the program creates a brand new sandwich order. Throughout the order, user choices are added into the prolog lists through the prolog function `assertz` , seen from the code snippet below. It acts as the same function with the above Interaction with Prolog segment above, using the prolog statement eg. `assert(chosen_breads(flatbread))` , used to append choice of bread to the list.

```
def addChoice(self, X, op):
   if op == 'meal':
     self.prolog.assertz("chosen_meals({})".format(X))
   elif op == 'bread':
     self.prolog.assertz("chosen_breads({})".format(X))
   elif op == 'main':
     self.prolog.assertz("chosen_mains({})".format(X))
   elif op == 'veg':
     self.prolog.assertz("chosen_veggies({})".format(X))
   elif op == 'sauce':
     self.prolog.assertz("chosen_sauces({})".format(X))
   elif op == 'topup':
     self.prolog.assertz("chosen_topups({})".format(X))
   elif op == 'side':
     self.prolog.assertz("chosen_sides({})".format(X))
   else:
     self.prolog.assertz("chosen_drinks({})".format(X))
```

(Codes can be found within convert.py)

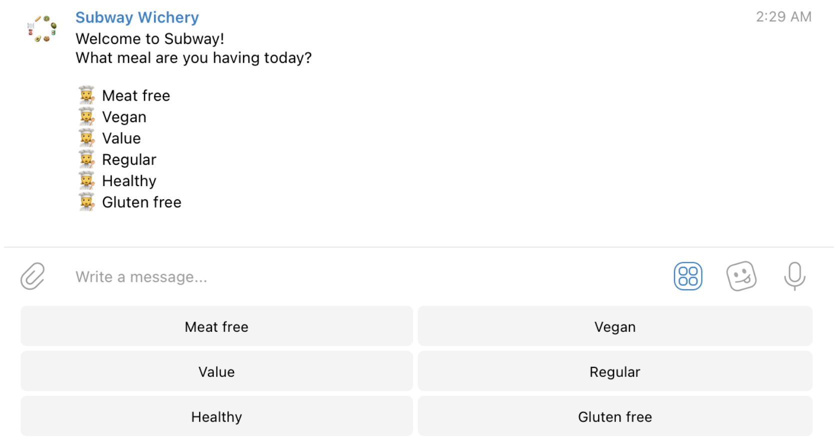
Starting the Bot
The python script `tele.py` is used as the main program for initiating the telegram bot. Upon running the program, the bot will be available at @wicherybot (https://t.me/wicherybot). Instructions on running the program is available as part of the user manual above.



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
a:subway andrel$ python3 tele.py
WicheryBot is online now! Enter /start in bot to start ordering!
```

Figure 2: Running `tele.py` with command line

Using the Bot
In order to amplify user experience, the prolog logic is implemented as a bot on telegram, with massive buttons for a simplified and convenient input. Users will be first greeted upon meals with vegan and gluten free options, to accommodate to varying dietary needs.

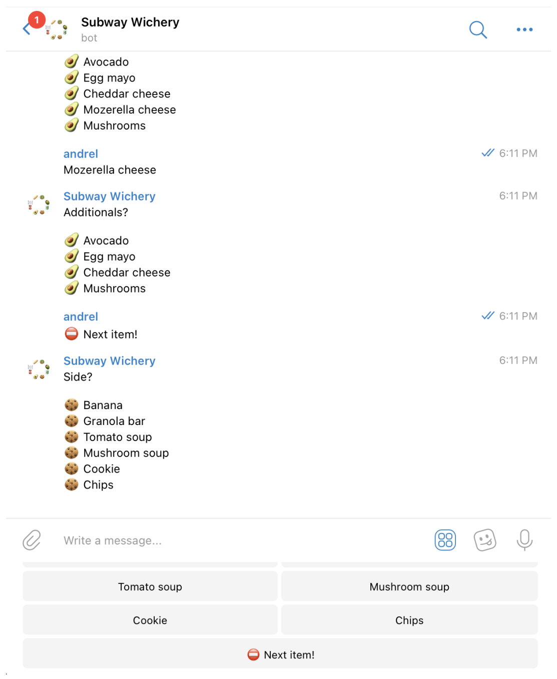


Subway Wichery 2:29 AM
Welcome to Subway!
What meal are you having today?

🧑‍🍳 Meat free
🧑‍🍳 Vegan
🧑‍🍳 Value
🧑‍🍳 Regular
🧑‍🍳 Healthy
🧑‍🍳 Gluten free

Figure 3: Asking of Meal Preference

Through the course of the ordering process, users will be asked based on the defined decision flow from meal to bread, main filling, vegetables, sauces, additional top-ups, sides and drinks. Categories apart from meal, bread, mains and drinks will be iterative, where users can choose more than one from each category. For example when choosing vegetables, they may want to include lettuce, tomato, cucumbers and jalapenos. After selecting lettuce as the first option, the question will reappear with the updated list of unselected veggies.



Subway Wichery
bot

Avocado
Egg mayo
Cheddar cheese
Mozerella cheese
Mushrooms

andrel 6:11 PM
Mozerella cheese

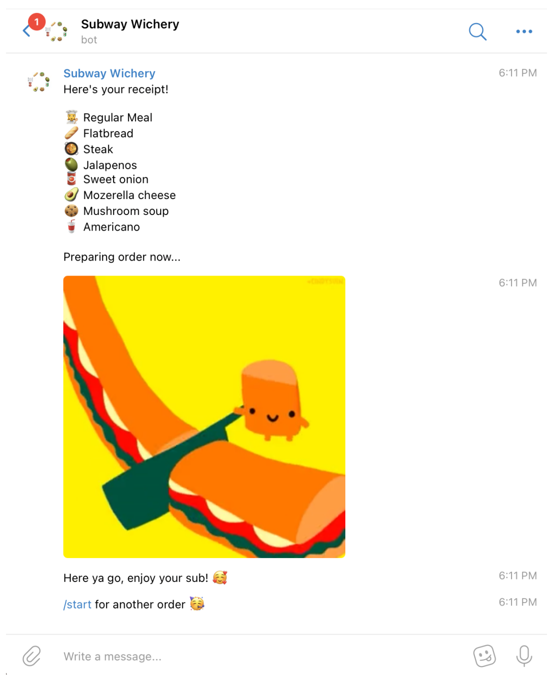
Subway Wichery 6:11 PM
Additionals?

Avocado
Egg mayo
Cheddar cheese
Mushrooms

andrel 6:11 PM
Next item!

Subway Wichery 6:11 PM
Side?

Banana
Granola bar
Tomato soup
Mushroom soup
Cookie
Chips



Subway Wichery
bot

Subway Wichery 6:11 PM
Here's your receipt!

Regular Meal
Flatbread
Steak
Jalapenos
Sweet onion
Mozerella cheese
Mushroom soup
Americano

Preparing order now...

6:11 PM

Here ya go, enjoy your sub! 6:11 PM

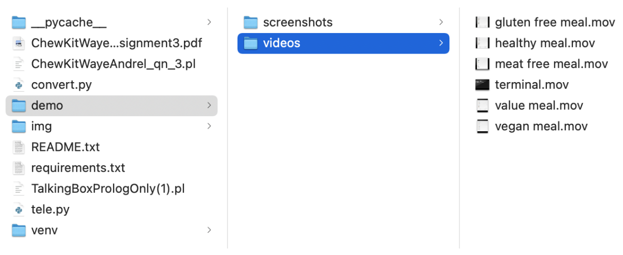
/start for another order 6:11 PM

Figure 4: Option of 'Next item!'

Figure 5: Displaying order summary

Similarly, some categories will allow users to skip directly to the next category, in the event that they choose not to opt for that category. This is to give users better flexibility in their choices. For example, a user may choose not to have any vegetables. They will be able to skip directly to the sauces category.

At the end of the order, a list of the user's choices will be listed in order, with the prolog function `findall` under `chosen_options(Opt, X)` defined in the prolog script. Empty categories and lists will be ignored, where only the selected options will be printed. Users can choose to stop the bot, or alternatively, enter /start to create a new order. Upon the selection of a new order, counters and prolog will be re-initialised to start anew.



__pycache__
ChewKitWaye...signment3.pdf
ChewKitWayeAndrel_qn_3.pl
convert.py
demo
img
README.txt
requirements.txt
TalkingBoxPrologOnly(1).pl
tele.py
venv

screenshots
videos

gluten free meal.mov
healthy meal.mov
meat free meal.mov
terminal.mov
value meal.mov
vegan meal.mov

Figure 6: File Organisation

Recordings of diet specific orders can be found in the demo/videos, including vegan and gluten free meals, and including a command line version using prolog assert functions. Screenshots of the telegram bot shown above can be found in demo/screenshots.