

# LBA: Restaurant Selection Expert System

---

Isaac Schaal, Gelana Tostaeva, Yoav Rabinovich  
December 19, 2018

## Report

In this assignment we designed an expert system for restaurant selection based on constraints by user input. This is achieved by unifying the constraints with rules based on the restaurants we surveyed around our neighborhood in Berlin. We have also implemented an extra feature that lets groups of people choose a restaurant together, by aggregating the constraints of several people, and also catching some fail cases automatically before sending the input to Prolog for processing.

## 1 Restaurants

We surveyed 12 restaurants: Azzam, Brachvogel, Burgermeister, Chaparro, Haci Baba, Kori Fay, Maroush, Mawal, Mimatoha, Nest, Peperoncino, Shezan. **Here's** a link to the spreadsheet with all the information.

## 2 Rules & Askables

For each restaurant we have the following askables:

- the type of cuisine (by using the `what_cuisine` rule);
- the maximum price in euros (by using the `smaller_than` rule);
- the distance in km (by using the `smaller_than` rule);
- the payment method (by using the `what_payment` rule).

Each one of multiple users is asked for their preferences in each one of these areas, providing a list of acceptable cuisines, limits for price and distance, and whether they want to pay by card and/or cash. The preferences are then aggregated before being fed to the system.

We have the following rules:

- `what_cuisine`: Makes use of Prolog lists to find at least one common item between the restaurant's cuisine list and the input cuisine list;
- `what_payment`: Similar to `what_cuisine`;

- `smaller_than`: Determines if the price or distance of the restaurant is smaller than the user input;
- `pick_restaurant`: A different rule for each restaurant in the database, adding it to the list of solutions if the input constraint parameters are successfully unified with the restaurant properties.

These can be seen in our KB:

```

1 KB = """
2 pick_restaurant(nest) :- cuisine(Cuisine), what_cuisine(Cuisine, [international, vegetarian
    ]), price(Price), smaller_than(12, Price), distance(Distance), smaller_than(1.6,
    Distance), payment(Method), what_payment(Method, [card, cash]).
3 pick_restaurant(burgermeister) :- cuisine(Cuisine), what_cuisine(Cuisine, [american, burger
    ]), price(Price), smaller_than(7, Price), distance(Distance), smaller_than(0.9, Distance
    ), payment(Method), what_payment(Method, [cash]).
4 pick_restaurant(mimatoha) :- cuisine(Cuisine), what_cuisine(Cuisine, [chinese, asian]),
    price(Price), smaller_than(9, Price), distance(Distance), smaller_than(0.8, Distance),
    payment(Method), what_payment(Method, [cash]).
5 pick_restaurant(maroush) :- cuisine(Cuisine), what_cuisine(Cuisine, [middle-eastern, doner])
    , price(Price), smaller_than(5, Price), distance(Distance), smaller_than(0.7, Distance),
    payment(Method), what_payment(Method, [cash]).
6 pick_restaurant(chaparro) :- cuisine(Cuisine), what_cuisine(Cuisine, [mexican]), price(Price
    ), smaller_than(10, Price), distance(Distance), smaller_than(1.1, Distance), payment(
    Method), what_payment(Method, [cash]).
7 pick_restaurant(azzam) :- cuisine(Cuisine), what_cuisine(Cuisine, [middle-eastern]), price(
    Price), smaller_than(6, Price), distance(Distance), smaller_than(2.7, Distance), payment
    (Method), what_payment(Method, [cash]).
8 pick_restaurant(kori_and_fay) :- cuisine(Cuisine), what_cuisine(Cuisine, [thai, asian]),
    price(Price), smaller_than(16, Price), distance(Distance), smaller_than(2.1, Distance),
    payment(Method), what_payment(Method, [card, cash]).
9 pick_restaurant(brachvogel) :- cuisine(Cuisine), what_cuisine(Cuisine, [german]), price(
    Price), smaller_than(15, Price), distance(Distance), smaller_than(1.9, Distance),
    payment(Method), what_payment(Method, [card, cash]).
10 pick_restaurant(peperoncino) :- cuisine(Cuisine), what_cuisine(Cuisine, [italian]), price(
    Price), smaller_than(14, Price), distance(Distance), smaller_than(1.8, Distance),
    payment(Method), what_payment(Method, [card, cash]).
11 pick_restaurant(shezan) :- cuisine(Cuisine), what_cuisine(Cuisine, [indian]), price(Price),
    smaller_than(13, Price), distance(Distance), smaller_than(1.3, Distance), payment(Method
    ), what_payment(Method, [card, cash]).
12 pick_restaurant(haci.baba) :- cuisine(Cuisine), what_cuisine(Cuisine, [middle-eastern, doner
    ]), price(Price), smaller_than(4, Price), distance(Distance), smaller_than(0.6, Distance
    ), payment(Method), what_payment(Method, [cash]).
13 pick_restaurant(mawal) :- cuisine(Cuisine), what_cuisine(Cuisine, [middle-eastern, doner]),
    price(Price), smaller_than(5, Price), distance(Distance), smaller_than(0.7, Distance),
    payment(Method), what_payment(Method, [cash]).
14
15 what_cuisine([Xh|Xt], Y) :- member(Xh,Y); what_cuisine(Xt, Y).
16 what_payment(H, P) :- member(H,P). % Payment should be card if only card, cash otherwise
17 smaller_than(X,Y) :- X <= Y.
18
19 """
20 with open("KB_A.pl", "w") as text_file:
21     text_file.write(KB)

```

### 3 Test Cases

1. To start off, we tested our system for 1 person who wanted some American or Doner or Burgers or Asian food. They were willing to pay a maximum of 9 Euros and walk a maximum of 1.7 km. They did not require card payments.

Our system suggested several restaurants, as seen in Figure 1.

**Yes or No : NoWe wholeheartedly suggest:**

```
haci_baba
mawal
maroush
mimatoha
burgermeister
Thank you for using our service.
Have fun, and please come back!
```

Figure 1: Output for test 1.

2. We also found a case for which our system did not find an appropriate restaurant. In this case, two people wanted to eat. One person's preferences were: International cuisine, limits of 10 Euros and 10 km, and no need for card payment. The other person wanted American, had the same limits of 10 Euros and 10 km, and no preferences for payment methods. Our system could not find a restaurant since there are none in our database that are considered to be both International and American (Figure 2).

---

**We're sorry, but someone has to consider more cuisines!**

Figure 2: Output for test 2.

3. Finally, we tested our own preferences.
- **Isaac:** International, Middle Eastern, limits of 10 Euros and 2 km, no preferences for payment methods;
  - **Gelana:** Middle Eastern, Asian, limits of 8 Euros and 3 km, no preferences for payment methods;
  - **Yoav:** German, Middle Eastern, Doner, limits of 12 Euros and 1.8 km, no preferences for payment methods.

Our system suggested us three restaurants to choose from (Figure 3). We happily decided to check out Maroush, as seen in Figure 4. Our system did great - highly recommend!

**We wholeheartedly suggest:**

```
haci_baba
mawal
maroush

Thank you for using our service.
Have fun, and please come back!
```

Figure 3: Output for test 3.



Figure 4: Isaac, Gelana, and Yoav testing at the Maroush restaurant. Yoav is clearly pleased.

## 4 Contributions

*Isaac*: implemented the input routine including extensions, pySwip troubleshooting, debugging.

*Gelana*: surveyed the restaurants, implemented the KB, drafted the report.

*Yoav*: consolidation system and pySwip implementation including extensions, debugged the KB.

## 5 References

We consulted the example implementation considered in class.

## Code

GitHub Gist

```

1 def cuisine_conversion(cuisine):
2     """ Converts the input list of numbers to a set of strings for use with Prolog
3
4     output_dic = {1 : 'international', 2: 'american', 3: 'chinese', 4: 'mexican', 5: 'german',
5                   6: 'italian', 7: 'indian', 8: 'thai', 9: 'middle-eastern', 10: 'asian',
6                   11: 'vegetarian', 12: 'burger', 13: 'doner'}
7
8     output = set()
9     for i in range(len(cuisine)):
10         output.add(output_dic[cuisine[i]])
11     return output
12
13 def consolidate(votes):
14     """ Consolidates multiple user inputs into one input to Prolog
15     # Takes into account all users requests and returns an

```

```

16 # error if nothing can be agreed upon
17
18 cuisines=set(['indian',"international","vegetarian","american","burger","middle-eastern"
19 ,"doner","mexican","thai","german","italian"])
20 price=float("inf")
21 distance=float("inf")
22 card='cash'
23 error = 0
24 for vote in votes:
25     # Limit cuisines
26     for cuisine in vote[0]:
27         cuisines = set.intersection(cuisines ,vote[0])
28     # Error 1: No cuisines agreed upon
29     if not len(cuisines):
30         error = 1
31     # Limit price
32     if vote[1] < price:
33         price = vote[1]
34     # Error 2: Someone's too cheap
35     if price < 2:
36         error = 2
37     if vote[2] < distance:
38         distance = vote[2]
39     # Error 3: Someone's too lazy
40     if distance < 0.5:
41         error = 3
42     if vote[3]=='card':
43         card='card'
44     # Return cuisines as a list
45     return [list(cuisines),price,distance,card], error
46
47 def one_user():
48     ### Gets the Input for one user
49
50     # Used for checking inputs
51     yes_options = ['Yes', 'yes', 'ye', 'Ye', 'y', 'Y', 'YES']
52     no_options = ['No', 'no', 'n', 'N', 'NO']
53     number_options = [i+1 for i in range(13)]
54
55     ## First get the cuisines
56     cuisine = []
57     print ("Hello,\nWhat would you like to eat today? ")
58     print ( '''\n1 for International, 2 for American, 3 for Chinese, 4 for Mexican, 5 for
59     German, 6 for Italian, 7 for Indian, 8 for Thai,
60     9 for Middle Eastern, 10 for Asian, 11 for Vegetarian, 12 for Burgers, 13 for D ner ''')
61     done = False
62     while done == False:
63         food_type = raw_input("1-13 :")
64         try :
65             food_type = int(food_type)
66             if food_type >= 1 and food_type <=13:
67                 done = True
68                 cuisine.append(food_type)
69             else:
70                 print "Please enter a number 1 through 13"
71         except ValueError:
72             print "Please enter a number 1 through 13"
73
74     # ask if they want another food type
75     done = False
76     while done == False:
77         print ("Would you consider eating another cuisine?")
78         print ( '''\n1 for International, 2 for American, 3 for Chinese, 4 for Mexican, 5
79         for German, 6 for Italian, 7 for Indian, 8 for Thai,

```

```

78 9 for Middle Eastern, 10 for Asian, 11 for Vegetarian, 12 for Burgers, 13 for Diner''')
79     another = raw_input('1-13 or No :')
80     if another in no_options:
81         done = True
82         pass
83     else :
84         try :
85             food_type = int(another)
86             if food_type >= 1 and food_type <=13:
87                 cuisine.append(food_type)
88             else:
89                 print "Please enter a number 1 through 13"
90         except ValueError:
91             print "Please enter a number 1 through 13 or No"
92
93 #convert the type list
94 cuisine = cuisine_conversion(cuisine)
95
96 ## Get the maximum price
97 done = False
98 while done == False:
99     print ("What is your maximum price? :")
100    max_price = raw_input('Price in Euro')
101    try:
102        max_price = float(max_price)
103        done = True
104    except ValueError:
105        print 'Please enter a number'
106
107 # Get the maximum distance
108 done = False
109 while done == False:
110     print ("What is your maximum distance?")
111     max_distance = raw_input('Distance in Km :')
112     try:
113         max_distance = float(max_distance)
114         done = True
115     except ValueError:
116         print 'Please enter a number'
117
118 # Is card required ?
119 done = False
120 while done == False:
121     print ('Do you require that they accept card?')
122     card = raw_input(' Yes or No :')
123     if card in yes_options:
124         card = True
125         done = True
126     elif card in no_options:
127         card = False
128         done = True
129     else:
130         print('Please answer Yes or No')
131 if card == True:
132     payment = 'card'
133 elif card == False:
134     payment = 'cash'
135     # Return cuisine as list
136 return [list(cuisine), max_price, max_distance, payment]
137
138 def input_routine():
139     global global_answer
140     global global_error
141     # due to cocalc breaking
142     done = False

```

```

143 while done == False:
144     print ('Hello, \nHow many people want to choose a restuarant?')
145     number = raw_input("")
146     try:
147         number = int(number)
148         if number == 0:
149             print ("Cool")
150             # Dont change the global vars
151             return
152         else:
153             done = True
154     except ValueError:
155         print ('Please enter an integer')
156 if number == 1:
157
158     answer = one_user()
159     error = None
160
161
162 elif number >= 1:
163     output_list = []
164     for i in range(number):
165         output_list.append(one_user())
166         if i != number-1:
167             print ("Please hand me over to the next user")
168     answer, error = consolidate(output_list)
169
170 global_answer = answer
171 global_error = error

```

```

1 global_answer = None
2 global_error = None
3
4 done = False
5 while done == False:
6     input_routine()
7
8     if global_error:
9         if global_error == 1:
10             print("We're sorry, but someone has to consider more cuisines! \nPlease try
again:")
11             input_routine()
12         elif global_error == 2:
13             print("Oh no! Someone's finances are too tight! Consider paying for them? \
nPlease try again:")
14             input_routine()
15         elif global_error == 3:
16             print("Come on people, move your butts! No restaurants are close enough! \
nPlease try again:")
17             input_routine()
18     done = True
19
20 from pyswip.prolog import Prolog
21 from pyswip.easy import *
22 prolog = Prolog() # Global handle to interpreter
23
24 prolog.consult("KB.A.pl") # open the KB
25
26 prolog.dynamic('cuisine/1')
27 prolog.dynamic('price/1')
28 prolog.dynamic('distance/1')
29 prolog.dynamic('payment/1')
30
31 a = global_answer
32

```

```

33 prolog.asserta("cuisine(" + str(a[0]) + ")")
34 prolog.asserta("price(" + str(a[1]) + ")")
35 prolog.asserta("distance(" + str(a[2]) + ")")
36 prolog.asserta("payment(" + str(a[3]) + ")")
37
38
39 solution_set = set()
40 for soln in prolog.query("pick_restaurant(X)."):
41     solution_set.add(soln['X'])
42
43 if not solution_set:
44     print("No restaurant's meet your criteria :(")
45 else:
46     print("We wholeheartedly suggest: \n")
47
48     for i in range(len(solution_set)):
49         print(list(solution_set)[i])
50     print("\nThank you for using our service. \nHave fun, and please come back!")
51
52 # Retract answers after obtaining results
53 prolog.retractall("cuisine(" + str(a[0]) + ")")
54 prolog.retractall("price(" + str(a[1]) + ")")
55 prolog.retractall("distance(" + str(a[2]) + ")")
56 prolog.retractall("payment(" + str(a[3]) + ")")

```

## Note

When running the code in a CoCalc Notebook, the kernel must be restarted (and all other cells rerun) each time the main cell runs. We understand that this is undesirable and have gone to great lengths to fix the problem, but it appears to be a problem in CoCalc.

Firstly, we were unable to run the code in anything other than CoCalc, as the modified PySwip package works only in CoCalc. The problem arises with the `raw_input()` function. The first problem was that when the `input_routine()` function was called and assigned to a variable in one line (like `a,err = input_routine()`), the `raw_input` would “freeze” after it was inputted, and the kernel would break. This problem would not arise when running the function in a normal Jupyter notebook, but would in CoCalc (we are unsure why). However, when `input_routine()` was run in a line by itself (not assigning the output to a variable), the `raw_input` worked. We thus changed the `input_routine()` function to use global variables instead of returning values.

However, it didn't fully fix the problem. The `input_routine()` (and other functions that included a `raw_input()` that we tested) would work only before PySwip was imported, but would break after. We are unsure if the problem is with PySwip, CoCalc or both. We believe the code would allow for multiple user interactions without restarting the kernel if run in a Jupyter notebook (or as a .py document), but we are unable to use PySwip outside of CoCalc.