Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2021-2022

Programming Exam 9

Due date: 18 June, 2022, Saturday, 23:59

1 Problem Definition

In this exam, you are going to implement predicates that work on an order preparation system for a salad bar. This salad bar has several vegan and vegetarian options that you have to take into consideration. You do not have to know which food item belongs to which category, these limitations will be defined for you. Still, dietary restrictions can be summarized as follows:

Vegan diet: No food products *derived from* animals.
Vegetarian diet: No animal meat.
Omnivorous diet: No limitation.

By definition, anything vegan is both vegetarian and omnivorous. Similarly, anything vegetarian is also omnivorous. Hence, we can think of the relation between the types as `vegan` $\subseteq$ `vegetarian` $\subseteq$ `omnivorous`.

1.1 Knowledge Base

The example knowledge base given to you in kb.pl file can be seen below, with descriptions:

```
% vegan(Name), vegetarian(Name), and omnivorous(Name) predicates,
% defining the diets of customers
    vegan(chidi).
    vegetarian(tahani).
    omnivorous(eleanor).
    omnivorous(jason).

% base(Name) predicate, options for salad base
    base(lettuce).

% protein(Name) predicate, options for proteins
    protein(smoked_turkey).
    protein(hellim_cheese).
    protein(smoked_tofu).

% topping(Name) predicate, options for toppings
    topping(cucumber).
    topping(tomato).
```

```
    topping(pickled_onion).
    topping(olive).

% condiment(Name) predicate, options for condiments
    condiment(mustard).
    condiment(mayonnaise).
    condiment(ranch).

% sauce(Name) predicate, options for sauces
    sauce(olive_oil).
    sauce(lemon_juice).
    sauce(balsamic_vinegar).

% not_vegan(Item) and not_vegetarian(Item) predicates,
% defining dietary restrictions
    not_vegetarian(Item) :-
        member(Item, [smoked_turkey]), !.
    not_vegan(Item) :-
        not_vegetarian(Item), !;
        member(Item, [hellim_cheese,mayonnaise,ranch]), !.
```

2 Specifications

In this exam, you are expected to implement 4 different predicates with varying difficulties. All predicates will be queried with different arguments given as variables, but not all arguments will be queried for all predicates. Querying details for each predicate can be seen in the example queries for that predicate.

2.1 `salad_type/2` - 15 points

Two argument predicate where `ItemList` is the list of food items in the salad, `Type` is the dietary type of the salad. `Type` can be `vegan`, `vegetarian` or `omnivorous`. Predicate has the form:

```
salad_type(ItemList, Type).
```

Example queries:

```
?- salad_type([lettuce,smoked_tofu,pickled_onion,olive_oil],Type).
Type = vegan ;
Type = vegetarian ;
Type = omnivorous.

?- salad_type([lettuce,smoked_tofu,pickled_onion,olive_oil], omnivorous).
true.

?- salad_type([lettuce,smoked_turkey,cucumber,olive_oil,ranch], vegan).
false.
```

- In the case of several possible answers for the `Type` variable for a given `ItemList`, possible answers should be listed in the order of more defining to less defining, as seen in first query.
- This predicate **will not be queried with `ItemList` argument as a variable**, so you do not need to worry about the behavior of the queries of the form `salad_type(ItemList, vegan)`.

2.2 person_can_eat/3 - 20 points

Three argument predicate where `Person` is the name of the person, and `SaladList` is a list of salads given as item lists, and `Salad` is a salad from `SaladList` that `Person` can eat. Predicate has the form:

```
person_can_eat(Person, SaladList, Salad).
```

Example queries:

```
?- person_can_eat(jason, [[lettuce,tomato,smoked_turkey],
    [lettuce,hellim_cheese,ranch],
    [lettuce,smoked_tofu,balsamic_vinegar]], Salad).
Salad = [lettuce, tomato, smoked_turkey] ;
Salad = [lettuce, hellim_cheese, ranch] ;
Salad = [lettuce, smoked_tofu, balsamic_vinegar] ;
false.

?- person_can_eat(chidi, [[lettuce,tomato,smoked_turkey],[lettuce,hellim_cheese,↩
   ranch]], Salad).
false.
```

- This predicate **will only be queried with `Salad` argument as a variable**, so you do not need to worry about the behavior of the queries with `SaladList` or `Person` arguments as variables.

2.3 possible_order_options/3 - 30 points

Three argument predicate where `Person` is the name of the customer, `OrderString` is a string that defines customer's order, and `ItemList` is the list of possible items that can be included in the order. Predicate has the form:

```
possible_order_options(Person, OrderString, ItemList).
```

`OrderString` is a string consisting of letters b, p, t, c, and s, meaning base, protein, topping, condiment and sauce respectively. The letters can repeat, and an order string does not have to contain every letter. Example queries:

```
?- possible_order_options(chidi, bps, ItemList).
ItemList = [lettuce, smoked_tofu, olive_oil] ;
ItemList = [lettuce, smoked_tofu, lemon_juice] ;
ItemList = [lettuce, smoked_tofu, balsamic_vinegar] ;
false.

?- possible_order_options(eleanor, ppppbcts, ItemList).
false.

?- possible_order_options(tahani, psp, ItemList).
ItemList = [hellim_cheese, olive_oil, smoked_tofu] ;
ItemList = [hellim_cheese, lemon_juice, smoked_tofu] ;
ItemList = [hellim_cheese, balsamic_vinegar, smoked_tofu] ;
ItemList = [smoked_tofu, olive_oil, hellim_cheese] ;
ItemList = [smoked_tofu, lemon_juice, hellim_cheese] ;
ItemList = [smoked_tofu, balsamic_vinegar, hellim_cheese] ;
false.
```

- An order can at most have 9 items, so `OrderString` cannot be longer than 9 characters. (There will be no input of such form.)

- `ItemList` cannot contain any food items that does not comply with the dietary restrictions of the customer `Person`.
- `ItemList` cannot contain any repeating food items. If there is no such order possible, query must fail. As shown in the second sample query, since there are only 3 protein options and there are 4 "p"s in the `OrderString`, query fails.
- Ordering of the food items in `OrderString` is important. Created `ItemList` should follow this order.
- Different orderings of the food items of same types in `ItemList` counts as different orders. For example, for the an `OrderString` of the form `pp`, following two `ItemList`s are considered different possible orders: `[smoked_tofu, hellim_cheese]` and `[hellim_cheese, smoked_tofu]`. One other example is shown in the third query.
- Ordering for the different possible orders (i.e. ordering after pressing ';') is not important.
- This predicate **will not be queried with `Person` or `OrderString` arguments as a variables**, so you do not need to worry about the behavior of such queries.

2.4 `count_per_type/2` - 35 points

Two argument predicate where `SaladList` is a list of salads given as item lists, and `TypeList` is the list of `[Type, Count]` pair-lists for each type. Predicate has the form:

```
count_per_type(SaladList, TypeList).
```

Example queries:

```
?- count_per_type([[lettuce,tomato,smoked_turkey],
    [lettuce,hellim_cheese,ranch],
    [lettuce,smoked_tofu,balsamic_vinegar]], TypeList).
TypeList = [[omnivorous, 1], [vegan, 1], [vegetarian, 1]].

?- count_per_type([[lettuce,hellim_cheese,ranch],
    [lettuce,smoked_tofu,balsamic_vinegar],
    [lettuce,tomato,cucumber,lemon_juice,olive_oil]], TypeList).
TypeList = [[vegetarian, 1], [vegan, 2]].

?- count_per_type([], TypeList).
TypeList = [].
```

- If a type of salad does not exist in the `SaladList`, it should not be included in the `TypeList`.
- For a salad in the `SaladList`, type should be selected as the most defining type, similar to the order in the `salad_type/2` predicate. (In other words, the first answer returned from `salad_type(ItemList, Type)` should be used).
- Salads should not be count more than once. For example, if a salad is vegan, it should **not** also be counted as vegetarian or omnivorous.
- Ordering of the `TypeList` is not important. For example, for the second sample query, `[[vegan, 2], [vegetarian, 1]]` is also correct, but returning just one of them is enough.
- This predicate **will not be queried with `SaladList` argument as a variable**, so you do not need to worry about the behavior of the queries of the form `count_per_type(SaladList, [[vegan, 1], [vegetarian, 2]])`.

3 Hints

- You may want to use `not` and `!(cut)`.
- When dealing with the `OrderString` in `possible_order_options/3` predicate, the following predicate may be helpful to obtain the list of characters:
  https://www.swi-prolog.org/pldoc/man?predicate=atom_chars/2
- To eliminate repetition, delete/3 or is_set/1 predicates may be helpful depending on your approach.

4 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "PE9" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.
- **Programming Language:** You must code your program in Prolog. Your submission will be run with swipl on ODTUCLASS. You are expected make sure your code runs successfully with swipl on ODTUCLASS.
- **Modules:** You are not allowed to import any modules. However, you are allowed to use the base predicates present in Prolog or define your own predicates.
- **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.
- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases.

  Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.