Início	quarta, 18 de janeiro de 2023 às 09:13
Estado	Prova submetida
Data de submissão:	quarta, 18 de janeiro de 2023 às 11:13
Tempo gasto	2 horas
Nota	**15,18** de um máximo de 20,00 (**76**%)

Informação

Read these instructions carefully before you start the test.

- This is an open-book test; you can use the materials available on the local computer and Moodle's course page, but no printed materials are allowed.
- You can use the resources available on the computer, such as SICStus Prolog.
- **The presence of electronic and/or communication devices is strictly forbidden.**
- The maximum duration of the test is stated below.
- The score of each question is stated in the test, totaling 20 points.
- A wrong answer in a multiple-choice question with five options implies a penalty of 25% of the question's value.
- A wrong answer in a True/False question implies a penalty of 50% of the question's value.
- Fraud attempts will be punished by the annulment of the test for all intervenients.

- **You can use predicates requested in previous questions even if you haven't implemented them.**
- **If you implement auxiliary predicates, you must include them in the answers to all questions where they are used.**
- **Do not include the answer to previous questions in the current one.**
- **Do not copy any code provided in the questions into the answer text box.**
- **Document your code and use intuitive variable names so as to clarify code interpretation.**
- **Pay attention to syntactic errors (such as forgetting the '.' after each rule/fact).**

Informação

Consider the following knowledge base about the *Pasta & Flour Lounge* (PFL) restaurant.

Each *dish/3* fact contains a dish's name, the price for which it is sold in the restaurant, and a list with the quantity of each ingredient needed to produce it (each ingredient appears only once on the list). For instance, to produce a pizza, which is sold for 2200 cents, 300g of cheese and 350g of tomato are needed.

Each *ingredient/2* fact contains an ingredient's name and unit cost (i.e., how many cents are required to buy 1 gram).

```
% dish(Name, Price, IngredientGrams).
dish(pizza,         2200, [cheese-300, tomato-350]).
dish(ratatouille,   2200, [tomato-70, eggplant-150, garlic-50]).
dish(garlic_bread,  1600, [cheese-50, garlic-200]).

:- dynamic ingredient/2.

% ingredient(Name, CostPerGram).
ingredient(cheese,   4).
ingredient(tomato,   2).
ingredient(eggplant, 7).
ingredient(garlic,   6).
```

Answer questions 1 to 7 **WITHOUT** using multiple solution predicates (findall, setof, and bagof), and **WITHOUT** using any SICStus library.

Pergunta 1

Respondida Pontuou 1,000 de 1,000

Implement *count_ingredients(?Dish, ?NumIngredients)*, which determines how many different ingredients are needed to produce a dish.

```
count_ingredients(Dish,NumIngredients) :- dish(Dish,_,Lista),
                                          length(Lista,NumIngredients).
```

Pergunta 2

Respondida Pontuou 1,000 de 1,000

Implement *ingredient_amount_cost(?Ingredient, +Grams, ?TotalCost)*, which determines the total cost (in cents) of buying a certain amount (in grams) of an ingredient.

```
ingredient_amount_cost(Ingredient,Grams,TotalCost) :- ingredient(Ingredient,CostGram), TotalCost is C
```

Pergunta 3

Respondida Pontuou 1,250 de 1,250

Implement *dish_profit(?Dish, ?Profit)*, which determines the profit of selling a dish in the restaurant. A dish's profit is the difference between its price and the combined cost of its ingredients.

```
somar([],0).
somar([Nome-Quantidade | T],Res) :- somar(T,R2),
                                   ingredient_amount_cost(Nome,Quantidade,R3),
                                   Res is R2 + R3.

dish_profit(Dish,Profit) :- dish(Dish,Price,Ingredients),
                            somar(Ingredients,Custo),
                            Profit is Price - Custo.
```

Pergunta 4

Respondida Pontuou 1,000 de 1,000

Implement *update_unit_cost(+Ingredient, +NewUnitCost)*, which modifies the knowledge base by updating the unit cost of an ingredient. If the ingredient does not exist, it should be added to the knowledge base. The predicate must always succeed.

```
update_unit_cost(Ingredient,NewUnitCost) :- ingredient(Ingredient,FirstCost),
                                            retract(ingredient(Ingredient,FirstCost)),
                                            assertz(ingredient(Ingredient,NewUnitCost)).

update_unit_cost(Ingredient,NewUnitCost) :- assertz(ingredient(Ingredient,NewUnitCost)).
```

Pergunta 5

Respondida Pontuou 1,250 de 1,250

Implement *most_expensive_dish(?Dish, ?Price)*, which determines the most expensive dish one can eat at the restaurant and its price. In case of a tie, the predicate must return, via backtracking, each of the most expensive dishes.

```
most_expensive_dish(Dish,Price) :- dish(Dish,Price,_),
                   \+ (dish(DiffName,DiffPrice,_),DiffPrice > Price).
```

Pergunta 6

Respondida Pontuou 0,113 de 1,500

Implement *consume_ingredient(+IngredientStocks, +Ingredient, +Grams, ?NewIngredientStocks)*, which receives a list of ingredient stocks (as pairs of Ingredient-Amount), an ingredient, and an amount (in grams) and computes a new list obtained from removing the given amount of ingredient from the original stock. The predicate must only succeed if there is enough ingredient in stock.

Constraint: In this question, and in this question only, you are **not** allowed to use recursion. Solutions using recursion will only receive up to 25% of the maximum score.

Examples:

```
| ?- consume_ingredient( [garlic-600, tomato-800, cheese-750], tomato, 150, L).
L = [garlic-600, tomato-650, cheese-750] ? ;
no
| ?- consume_ingredient( [garlic-600, tomato-800, cheese-750], tomato, 1000, L).
no
```

```
consume_ingredient([],X,Y,[]).
consume_ingredient([Ingredient-Amount | Stock],Ingredient,Grams,[Ingredient-New | T]) :- New is Amoun
consume_ingredient([Ingredient-Amount | Stock],X,Y,[Ingredient-Amount | T]) :- consume_ingredient(Sto
```

Pergunta 7

Respondida Pontuou 1,500 de 1,500

Implement *count_dishes_with_ingredient(+Ingredient, ?N)*, which determines how many dishes use the given ingredient.

```
hasIngredient([Name-Price | T],Name).
hasIngredient([Name-Price | T], X) :- hasIngredient(T,X).

getAllDishesWithIngredient(Acc,Res,Ingredient) :- dish(Name,Price,Lista),
                                        hasIngredient(Lista,Ingredient),
                                        \+member(Name-Price,Acc),
                                        !,
                                        New = [Name-Price | Acc],
                                        getAllDishesWithIngredient(New,Res,Ingredient).

getAllDishesWithIngredient(Res,Res,Ingredient).

count_dishes_with_ingredient(Ingredient,N) :- getAllDishesWithIngredient([],Res,Ingredient),
                                              length(Res,N).
```

Informação

In the following questions, you can use multiple solution predicates (findall, setof, and bagof) as well as any SICStus library.

Pergunta 8

Respondida Pontuou 1,250 de 1,250

Implement *list_dishes(?DishIngredients)*, which returns a list of pairs Dish-ListOfIngredients.

Example:

```
| ?- list_dishes(L).
L = [pizza-[cheese, tomato], ratatouille-[tomato, eggplant, garlic], garlic_bread-[cheese, garlic]] ? ;
no
```

```
get_ingredients(Dish,[],[]).
get_ingredients(Dish,[Name-Price | T],[Name | W]) :- get_ingredients(Dish,T,W).

list_dishes(DishIngredients) :- findall(Name-Ingredients,(dish(Name,Price,Todos),get_ingredients(Name
```

Pergunta 9

Respondida Pontuou 1,000 de 1,250

Implement *most_lucrative_dishes(?Dishes)*, which returns the restaurant's dishes, sorted by decreasing amount of profit. In case of a tie, any order of the tied dishes will be accepted.

Example:

```
| ?- most_lucrative_dishes(L).
L = [ratatouille,pizza,garlic_bread] ? ;
no
```

```
get_name([],[]).
get_name([Name-Price | T],[Name | W]) :- get_name(T,W).

most_lucrative_dishes(Dishes) :- setof(Dish-Profit,dish_profit(Dish,Profit),Lista),
                                 reverse(Lista,Sorted),
                                 get_name(Sorted,Dishes).
```

Informação

Consider the following predicates.

```
predX(S, D, NewS):-
       dish(D, _, L),
       predY(L, S, NewS).

predY([], L, L).
predY([I-Gr|Is], S, NewS):-
       !,
       consume_ingredient(S, I, Gr, S1),
       predY(Is, S1, NewS).
```

Pergunta 10

Correta Pontuou 1,250 de 1,250

Complete the text below (in each hole to fill, only one option is correct):

predX/3 receives as argument (in order) a list of ingredient-quantity pairs ✔ and a dish name ✔ to return a new list of ingredient-quantity pairs ✔ after using the ingredients in stock to produce the dish ✔ .

predX/3 fails if there aren't enough ingredients in stock ✔ .

Pergunta 11

Correta Pontuou 0,250 de 0,250

The cut present in the recursive clause of *predY/3* is green. True or false?

Selecione uma opção:

- Verdadeiro ✔
- Falso

Informação

Consider the following predicate.

```
predZ:-
        read(X),
        X =.. [_|B],
        length(B, N),
        write(N), nl.
```

Pergunta 12

Respondida Pontuou 1,000 de 1,000

Explain concisely (in one sentence) what *predZ/0* does.

O predicado predZ recebe como input qualquer elemento, escrevendo no terminal a aridade do mesmo.

Pergunta 13

Correta Pontuou 0,500 de 0,500

Which of the following statements about tail recursion is correct?

- a. *predY/3* uses tail recursion due to the usage of a cut in the recursive clause.
- b. By using an extra argument, one can rewrite certain recursive predicates so that they are tail-recursive. ✔
- c. *predZ/0* uses tail recursion.
- d. Tail recursion occurs not only in rules but also in facts.
- e. All other statements are incorrect.

Pergunta 14

Incorreta Pontuou -0,125 de 0,500

Consider the following statements about difference lists.

A - The [a,b|T]\T difference list is equivalent to [a,b].
B - Difference lists provide O(1) access to an uninstantiated prefix of a list.
C - Using difference lists, we can compute the length of a list's prefix in constant time (O(1)).

Which statements are correct?

- a. B and C
- b. Only C
- c. A, B, and C
- d. A and C ✖
- e. Only A

Informação

Using operators, the goal is to write facts in the following format:

```
garlic_bread requires cheese and garlic.
garlic_bread requires cheese and some garlic.
ratatouille requires tomato and eggplant and garlic.
```

Note: "some" can be used once before each ingredient.

Consider "*requires*" to be defined as follows:

```
:- op(590, xfx, requires).
```

Pergunta 15

Correta Pontuou 0,250 de 0,250

Which is the most correct way of defining "*some*" in terms of syntax and semantics?

a. `:- op(570, fx, some).` ✔

b. `:- op(600, fy, some).`

c. `:- op(570, fy, some).`

d. `:- op(570, xf, some).`

e. `:- op(600, xf, some).`

Pergunta 16

Correta Pontuou 0,250 de 0,250

Which is the most correct way of defining "*and*" in terms of syntax and semantics?

a. `:- op(580, yxfx, and).`

b. `:- op(580, yf, and).`

c. `:- op(580, xf, and).`

d. `:- op(580, xfy, and).` ✔

e. `:- op(580, xfx, and).`

Informação

Consider the following query:

```
| ?- member(X, [a,b,c,d,e]), !, member(Y, [1,2,3,4]).
```

Pergunta 17

Incorreta Pontuou 0,000 de 0,250

The cut present in the query is green. True or false?

Selecione uma opção:

Verdadeiro ✖

Falso

Pergunta 18

Correta Pontuou 0,250 de 0,250

How many children does the root of the search tree of the query have?

- a. 1 ✔
- b. 2
- c. 5
- d. 0
- e. 3

Pergunta 19

Incorreta Pontuou -0,063 de 0,250

How many unifications are performed, in total, when executing the query and asking for all the solutions via backtracking?

- a. 4 ✖
- b. 9
- c. 20
- d. 1
- e. 5

Pergunta 20

Correta Pontuou 0,250 de 0,250

Moving the cut to the beginning of the query decreases the number of unifications performed when executing the query. True or false?

Selecione uma opção:

- Verdadeiro
- Falso ✔

Informação

When several drugs/medicaments have the effect of healing a patient with some disease, a smart step is to determine a molecular substructure common to them all. This substructure might be a starting point for understanding the adequate treatment for that disease.

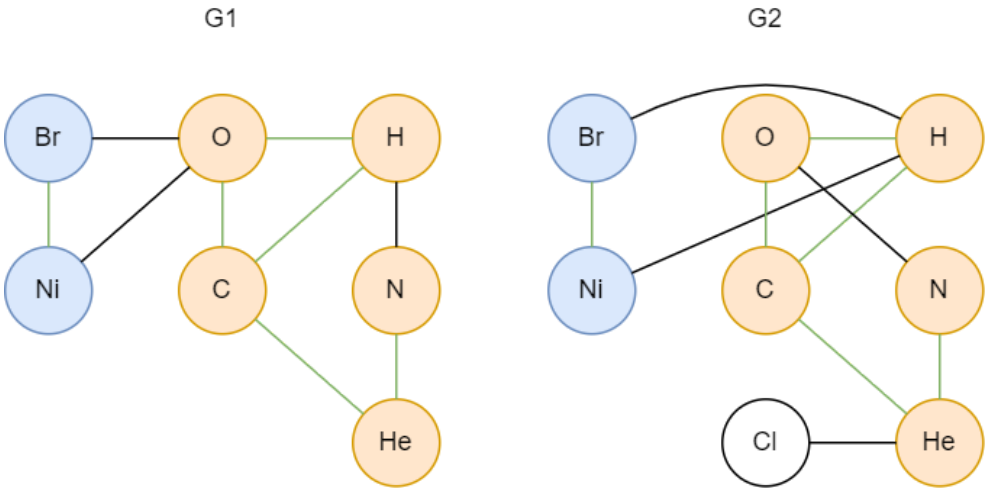
Consider that molecules are represented as undirected graphs, where atoms are the graph's vertices, and bonds are the graph's edges. One relevant procedure in this type of analysis would be determining the largest subgraph common to a given pair of molecules. Note: In each graph, the vertex names (i.e. atom names) are unique.

Consider the following examples of graphs:

```
%G1
edge(g1, br, o).
edge(g1, br, ni).
edge(g1, o, ni).
edge(g1, o, c).
edge(g1, o, h).
edge(g1, h, c).
edge(g1, h, n).
edge(g1, n, he).
edge(g1, c, he).

% G2
edge(g2, br, h).
edge(g2, br, ni).
edge(g2, h, ni).
edge(g2, h, o).
edge(g2, h, c).
edge(g2, o, c).
edge(g2, o, n).
edge(g2, n, he).
edge(g2, c, he).
edge(g2, cl, he).
```

The figure below depicts both graphs. The edges in green are the common edges of both graphs. The colored vertices denote the two common subgraphs.



Pergunta 21

Respondida Pontuou 2,000 de 2,000

Implement *common_edges(+G1, +G2, ?L)*, which, given the identifiers of two graphs (G1 and G2), computes their list of common edges.

Example:

```
| ?- common_edges(g1,g2,L).
L = [br-ni,o-c,o-h,h-c,n-he,c-he] ? ;
no
```

```
common_edges(G1,G2,L) :-
                        findall(I-F, (edge(G1,I,F), (edge(G2,I,F); edge(G2,F,I))),L).
```

Pergunta 22

Não respondida Pontuação 2,000

Implement *common_subgraphs(+G1, +G2, ?Subgraphs)*, which determines the list of vertices of each common subgraph of both input graphs. Any order of the subgraphs and of the vertices will be accepted.

Example:

```
| ?- common_subgraphs(g1,g2,L).
L = [[br,ni],[c,h,he,n,o]] ? ;
no
```