CPS721: Assignment 1

Due: September 26, 2022, 9pm
Total Marks: 100 (worth 4% of course mark)
You MUST work in groups of 2 or 3

Late Policy: The penalty for submitting even one minute late is 10%. Assignments are not accepted more than 24 hours late.

Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions as they come up. These will be monitored regularly. Clarifications will be made there as needed. A Frequently Asked Questions Page will also be created. You may also email your questions to your instructor, but please check the D2L forum and frequently asked questions first.

Collaboration Policy: You can only discuss this assignment with your group partners or with your CPS721 instructor. By submitting this assignment, you acknowledge that you have read and understood the course policy on collaboration as stated in the CPS721 course management form.

PROLOG Instructions: When you write your rules in PROLOG, you are not allowed to use ";" (disjunction), "!" (cut), and "->" (if-then). You are only allowed to use ";" to get additional responses when interacting with PROLOG from the command line. Note that this is equivalent to using the "More" button in the ECLiPSe GUI.

We will be using ECLiPSE Prolog release 6 to mark the assignments. If you run any other version of PROLOG, it is your responsibility to check that it also runs in ECLiPSE Prolog release 6.

Submission Instructions: You should submit ONE zip file containing 9 files:

- `question1_books.pl`, `question1_queries.txt`, `question1_interaction.txt`
- `question2_grocery_bill.pl`, `question2_queries.txt`, `question2_interaction.txt`
- `question3_intersection.pl`, `question3_queries.txt`, `question3_interaction.txt`

Your submission should not include any other files. If you submit a `.rar`, `.tar`, `.7zip`, or other compression format aside from `.zip`, you will lose marks. The name of the zip file should be `yourLoginName.zip` where `yourLoginName` is only the group member who made the submission. Note that only one student in the group should make submissions, as D2L can present confusing information if more than one student in the group submits.

All submissions should be made on D2L. Submissions by email will not be accepted.

You are allowed to make as many submissions as you want, and you do not have to inform anyone about this. The new copy will override the old one. The time stamp of the last submission will be used to determine the submission time. However, the same group member should make all the submissions for the reasons explained above.

If you write your code on a Windows machine, make sure the files are saved on plain text and are readable on Linux machines. Ensure your PROLOG code does not contain any extra binary symbols and that they can be compiled by ECLiPSE Prolog release 6.

1 A PROLOG Knowledge Base [30 marks]

For this problem, you will be making a knowledge base about books and then perform queries on it. The knowledge base and rules will be stored in the given file `question1_books.pl`. Your test queries should be added to `question1_queries.txt`. Finally, a log of your interaction when applying those queries to the knowledge base should be put in a file `question1_interaction.txt`. All three of these files, which can be found in the provided zip file, need to be submitted.

a. [10 marks] Create the knowledge base by adding atomic propositions to the file `question1_books.pl`. The atomic propositions should ONLY use the following predicates:

- `authorOf(AuthorName, BookTitle)` - the author of the book
- `publishedBy(Publisher, BookTitle)` - the publisher of the book
- `publicationYear(BookTitle,Year)` - the year the book was released

You should add 10-15 sentences per predicate about books (not necessarily real ones). However, the facts should mean that at least 8 of the queries in part b below succeed. Note, you can use a site like goodreads.com if you are interested in using real book information.

All atomic propositions should be added to the file `question1_books.pl` in the correct sections. You may lose marks if you do not put propositions in the correct place.

b. [20 marks] Create queries for each of the 10 statements below and add them to the file `question1_queries.pl`. Ensure that you put each query under the correct comment as otherwise you may lose marks. You can only use the predicates listed above with variables or constants as arguments, conjunction, and "not" (*ie.* negation) in your queries. You may also use $<, >, <=$, or $>=$, which PROLOG uses for less than, more than, etc. Keep in mind that when using a predicate $X < Y$, both X and Y should be instantiated before the comparison.

1. Did Chimamanda Ngozi Adichie write "Half of a Yellow Sun"?
2. Who published "Station Eleven"?
3. Did the same author write both "The Apprenticeship of Duddy Kravitz" and "Barney's Version", and if so, who was it?
4. Was there a book published in 2019 that was not published by Random House or Harper Collins? In your interaction, list all such books using the ";" command if you are using the command line or the "More" button if you are using the GUI.
5. Did any author write at least two different books, each with a different publisher?
6. Did the author of "A Wild Sheep Chase" write any books before that one? In your interaction, list all such books using the ";" command if you are using the command line or the "More" button if you are using the GUI.
7. Was there a year from 1977 to 1992 that Stephen King did not publish a book?
8. Was there an author that published a book in 3 consecutive years starting after 2005?
9. Did any publisher only publish exactly one book in 1995?
10. What was the earliest year that Macmillan Publishers published a book in?

You should also submit your interaction with the knowledge base in a file called `question1_interaction.txt`.

2 Arithmetic in PROLOG [30 marks]

This questions involves numerical calculations in PROLOG. The knowledge base and rules will be stored in the given file `question2_grocery_bill.pl`. Your test queries should be stored in `question2_queries.txt`. Finally, a log of your interaction when applying those queries to the knowledge base should be put in a file `question2_interaction.txt`. All three of these files, which can be found in the provided zip file, need to be submitted.

a. [**10 marks**] You will be calcuating the bill at a grocery store on five items: `bread`, `lettuce`, `apple`, `chocolate_bar`, and `ginger_ale`. For this step, you should create a knowledge base that contains atomic statements that use the following predicates:

- `cost(Item, Cost)` - defines the cost of the item
- `twoForOneSale(Item)` - indicates that the item is on a two-for-one sale. Note that this means that if 5 of an item that is on a two-for-one sale are bought, then the price will be equivalent to buying 3 of the item.
- `taxable(Item)` - whether the item is taxable or not.
- `numPurchased(Item, Count)` - indicates the number of the corresponding item that have been purchased.
- `taxRate(Rate)` - the tax rate

The atomic propositions should set only `chocolate_bar`, and `ginger_ale` as taxable. Put any two of the items of your choosing on a `two_for_one_sale`. You may also set the prices for the items and the number of each item purchased to anything of your choosing, as long as they are greater than 0. Finally, you may set the tax rate to anything of your choosing provided it is greater than 0 but less than 1 (*ie.* the tax rate of 13% would be 0.13).

All atomic propositions should be added to the file `question2_grocery_bill.pl` in the correct sections. You may lose marks if you do not put propositions in the correct place.

b. [**15 marks**] Now add rules that accomplish the following:

- Add rule `costAfterTax(Item, AfterTax)` which calculates the price per unit of the given `item` after tax has been added. If the item is not taxable, then the after tax value is the same as the original cost. If it is taxable, then the after tax value is given by the original value increased by the tax rate. For example, if the tax rate is 13%, you will get the after tax value by multiplying the pre-tax value by 1.13. You should not need 5 such rules, but you may need more than 1.
- Add a rule, `costAfterTaxAndSale(Item, AfterSaleAndTax)` that calculates the total for the number purchased of the given item after the sale is included. The value should include the tax if the item is taxable. It may be useful to use the "X mod Y" operator and the "X // Y" operator (integer division) in PROLOG for this purpose.
- Add a rule `totalCost(Cost)` that calculates the total cost for the five items purchased.

All rules should be added to the file `question2_grocery_bill.pl` in the correct sections. You may lose marks if you do not put them in the correct place.

c. [**5 marks**] Create 5 total queries that test the `costAfterTax`, `costAfterTaxAndSale`, and `totalCost` rules. You should not use the ";" command in these queries. Any queries of your choosing on these three rules is allowed, as long as each is tested at least once. You should also add a comment above each query stating what the query checks in plain English.

The queries should be written in the file `question2_queries.txt` . In addition, you should submit your interaction with the knowledge base in a file called `question2_interaction.txt`.

3 Recursive Rules [40 marks]

In this question, we will develop recursive rules to define the rules of driving at an intersection in Ontario. This sort of reasoning would be necessary for an autonomous driving system. For simplicity, assume there is one intersection, with a simple streetlight in each direction. We will refer to these directions as north, east, south, and west. Note, do not add any predicates aside from those described below.

The knowledge base and rules will be stored in the given file `question3_intersection.pl`. Your test queries should be stored in `question3_queries.txt`. Finally, a log of your interaction when applying those queries to the knowledge base should be put in a file `question3_interaction.txt`. All three of these files, which can be found in the provided zip file, need to be submitted.

a. [**5 marks**] For this part, you will add atomic statements about the relative location of directions:

- `clockwise(Direction1,Direction2)` - indicates that direction1 is clockwise of direction2 (*ie.* east is clockwise of north).
- `counterclockwise(Direction1,Direction2)` - indicates that direction1 is counterclockwise of direction2 (*ie.* north is counter clockwise of east)
- `reverseDirection(Direction1,Direction2)` - direction1 is the reverse direction of direction2 (*ie.* east is the reverse of west)

The constants you should use for the directions are `north`, `east`, `south`, and `west`. Do not introduce any other constants for this question.

These atomic propositions should be added to `question3_intersection.pl`. Ensure you put them in the correct sections. You may lose marks for putting them in the incorrect section.

b. [**5 marks**] Let us now set up a scenario that we will use for testing later. To do so, we will use the following predicates:

- `facing(Entity, Direction)` - indicates entity is facing the given direction
- `lightColour(Direction,Colour)` - indicates that the cars facing the given direction see a light of the given colour.

Add atomic propositions that capture the following scenario:

"There is a toyota facing south, a nissan facing north, and a chevrolet facing east. The lights are currently green for both north and south, and red for the other directions."

For the cars, use the constants `toyota`, `nissan`, and `chevrolet`. For the colours, use the constants `red`, `green`, and `yellow`.

These atomic propositions should be added to `question3_intersection.pl`. Ensure you put them in the correct sections. You may lose marks for putting them in the incorrect section.

c. [**20 marks**] Add rules for the `canGo(Car, Direction)` predicate, which indicates that it is legal to drive in the given direction for the current scenario. Your rules should take into account the following rules of the road:

- A car can go straight through an intersection or turn right on a green light.
- A car can turn left on a green light if there is no car that is coming in the opposite direction. Note that for this assignment, we do not distinguish between if the car coming in the opposite direction is turning or not.
- A car can turn right on a yellow light.
- A car can turn left on a yellow light. Note, that we are assuming traffic coming in the opposite direction has stopped when the light is yellow.

- A car can turn right on a red light, if there is no car coming in on the left that can travel straight (*ie.* you must yield to oncoming traffic).

These rules should be added to the appropriate sections in `question3_intersection.pl`. You may lose marks for putting them in the incorrect section.

Remember, you should not add any additional predicates beyond those mentioned above. In particular, you do not need to represent the fact that some other car is turning, only what options are available given where the cars are.

Note, that the situation you have encoded in part b may not fully test your rules for all cases. You are responsible for making sure the rules work even in these other cases and we may test them as such.

d. [**10marks**] Create 5 total queries that test the `canGo` rules. You should not use the ";" command in these queries. The queries should be written in the file `question3_queries.txt` . The queries may be anything you chose that you think will help demonstrate that your rules work. You should also add a comment above each query stating what the query checks in plain English.

In addition, you should submit your interaction with the knowledge base in a file called `question3_interaction.txt`. You may use ";" in the interaction.