cps721: Assignment 1 (100 points).
Due date: Electronic file - Thursday, September 26, 2019, 21:00 (sharp).

YOU SHOULD NOT USE ";" , "!" AND "–>" IN YOUR PROLOG RULES.

You have to work in groups of TWO, or THREE, you cannot work alone. You can discuss this assignment only with your CPS721 group partners or with the CPS721 instructor. By submitting this assignment you acknowledge that you read and understood the course Policy on Collaboration in homework assignments stated in the CPS721 course management form.

1 (40 points). This problem is designed to get you started in Prolog. Use only named variables.

Almost all information that is currently available on the WWW is intended exclusively for humans, and there are few programs that can do limited processing of this information. The long-term goal of the Semantic Web project is to transform the current World Wide Web so that the information and services are in a machine-processable form. The Semantic Web will create an environment where software agents (sometimes, these computer programs are called "soft-bots" or services) can autonomously perform sophisticated tasks and help humans find, understand, integrate, and use information. The key distinguishing feature of the Semantic Web will be representation of information in specialized formats and logical rules for reasoning about information. In this assignment, you are asked to design a simple prototype of an on-line store that keeps information about electronic equipment in a specified format, so that a "soft-bot" can automatically query store's KB to make a purchase.

More specifically, you have to do the following. Make up a simple knowledge base (KB) of atomic sentences about products available for purchase (not necessarily real ones), manufacturers and prices, using the following predicates.

inStore(ItemID,ProductType,Quantity) – *Quantity* of a *ProductType* (tablet, laptop, printer, etc) with an unique *ItemID* (it can be UPC, or any other unique item identifier) is available in the store, where *Quantity* is the number of distinct copies of the product that remain in stock.

manufacturer(ItemID,Company,Year) – *ItemID* has been manufactured by *Company* in *Year*.

price(ItemID,P) – the cost of *ItemID* is *P*.

We assume that all prices are represented as integers. Make up about 10-15 atomic statements for each predicate (they do not need to be real ones). Try to design your facts in a way that most of your queries will retrieve information successfully from your knowledge base. Examples:

```
inStore(abc123,dslrCamera,12).
manufacturer(abc123,nikon,2019).
price(abc123,456).
```

Now pose the following queries to Prolog using these predicates only, and obtain Prolog's answers. You cannot introduce any other predicates to formulate queries. In particular, you cannot use any predicates from Prolog's standard library.

1. Is there an item (apart from flash drives) such that the store has more than 4 copies of this item? Note that $X < Y$ (and $X > Y$) is Prolog's `less than` (`greater than`, respectively) predicate: $X < Y$ succeeds if the number X is less than the number Y. Hint: When formulating in Prolog a question like "Is there an item such and such", check whether quantity of this item is greater than 0.

2. Is there an Acer laptop manufactured in 2018 or later with the price less than 500?

3. Is there a Asus phone that is less expensive than an Apple phone?

4. Does the store carry any item manufactured by more than one company? (Usually, each item is manufactured by one company only; so, we would expect the answer "No" to this query.)

5. What are the prices of all items manufactured by Apple available in the store? List all prices (by clicking "more" in a GUI version, or by using the "**;**" command in a command-line version).

6. Does the store have distinct products manufactured by the same company?

7. What is the total price of a purchase that includes an Acer laptop and a Canon printer? If your KB includes several different combinations of these products, then use "**;**" (or "more" button in GUI) to retrive all answers.

8. Out of those printers which are not manufactured by HP, are there printers with the same price manufactured by distinct companies?

9. Which company is the only one that manufactured more than 2 distinct product types?

10. Find the least expensive Asus phone in the store. (Note that KB can include arbitrary many items.)

Handing in solutions: (a) An electronic copy of your program with all atomic statements (the name of this file must be **store.pl**); (b) An electronic copy of your session with Prolog that shows clearly the queries you submitted to Prolog, and the answers that were returned (the name of this file must be **store.txt**);

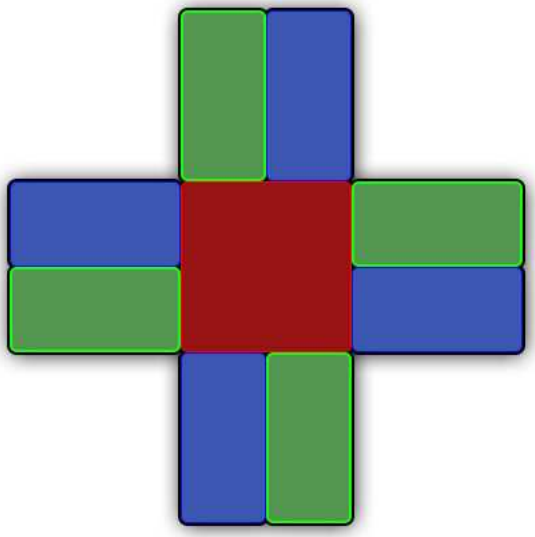

Figure 1: Incoming lanes are green, outbound – are blue. Traffic light is visible outside the red square.

2 (35 points). This problem will exercise what you have learned about rules in Prolog, by having you implement a small *knowledge base* with both *atomic* statements and logical *rules*.

The driverless car projects involve developing technology for autonomous cars. All autonomous vehicles have to interact with both manned and unmanned vehicle traffic in an urban environment. In addition, robotics vehicles are judged also on their ability to comply with local highway traffic rules. See `https://www.avinhub.ca/` and `http://en.wikipedia.org/wiki/Google_driverless_car`

for video and additional information. The long term goal of this effort is an exciting task of developing cars that drive themselves on highways and in cities.

In this question, you are asked to express in Prolog a few simple traffic rules that a driver-less car might need to follow in a city. For simplicity, let us consider an intersection with a single 4-directional traffic light located at the point **(0,0)** and one lane only in each direction. Let x-axis go from West to East and separate eastward and westward lanes, and y-axis go from South to North on the border between northward and southward lanes. Assume that each lane has width 20 units, and the length of each lane is 200 units (i.e., we disregard everything that is more than 200 units away from the traffic light). As usual in North America, according to right-hand traffic regulation, all traffic is required to keep to the right side of the road. We consider a static scene only because taking motion into account would require knowledge that is not tested by this assignment.

You have to use the following predicates; you are **not** allowed to introduce your own predicates. The predicate `at(Car,X,Y)` holds if Car is located at the point (X, Y). The predicate `light(State,X,Y)` means that a traffic light is in $State$ (one of: green, red, yellow) when you look at the light from the location (X, Y). For simplicity, we assume that the traffic light is not visible at the intersection itself, i.e., we never need to define its state for points (X, Y) inside the quadrant $-20 < X < 20$ and $-20 < Y < 20$. The predicate `distance(X1,Y1,X2,Y2,D)` is true if D is the Euclidean distance between the points $(X1, Y1)$ and $(X2, Y2)$. The predicate `canDriveStraight(Car,X,Y)` holds if Car can keep driving straight. Finally, the predicate `canTurnRight(Car,X,Y)` (the predicate `canTurnLeft(Car,X,Y)`, respectively) means that Car can turn right (turn left, respectively) from the location (X, Y). You are asked to express a few Ontario traffic laws using these predicates and other auxiliary predicates introduced below. Your task is to implement in Prolog the following English specifications (in the order shown).

- Write one rule that implements the predicate `distance(X1,Y1,X2,Y2,D)`.
- Write 4 rules implementing directions of travel. Use the following predicates: `eastbound(X,Y)`, `southbound(X,Y)`, `northbound(X,Y)`, `westbound(X,Y)`. For example, the predicate `eastbound(X,Y)` holds if a point (X, Y) is on the incoming lane from which traffic goes from West to East. In the same vein, the predicate `southbound(X,Y)` is true at (X, Y) if this point belongs to the lane from which cars incoming from North travel to South. Other predicates have very similar meaning. Note that according to the assumptions stated above about the coordinate system, `southbound(X,Y)` is true if $-20 < X < 0$, and $20 < Y < 200$. Notice that we define "sounthbound" in a bit restricted way: only points on the incoming segment (before the intersection) are considered southbound. The reason is that we do not use the points from the outgoing segments to define any traffic rules. So, we do not introduce any of the predicates for points on the outbound segments.
- Let the predicate `oppDir(X1,Y1,X2,Y2)` mean that points $(X1, Y1)$ and $(X2, Y2)$ belong to the lanes from which traffic goes in the opposite directions. For example, southbound traffic and northbound traffic have opposite directions. Write all the rules that are required to implement this predicate completely.
- Let the predicate `perpendicDir(X1,Y1,X2,Y2)` mean that points $(X1, Y1)$ and $(X2, Y2)$ belong to the incoming lanes from which vehicles travel in the perpendicular directions (i.e., directions at right angle to each other). Because this predicate will be subsequently used to characterize

which right turns are permitted, only lanes relevant to making right turns on the red light should be mentioned in its implementation. For example, on a red light, one can make a right turn from a point (X', Y') on the eastbound lane to an outbound segment that is continuation of a southbound segment only if there is no nearby car at a point (X'', Y'') driving on the southbound lane. Consequently, to specify geometry of the intersection, we would like to say that for these points `perpendicDir(X',Y',X'',Y'')` is true. Notice that in this example the point (X'', Y'') should be characterized using the predicate `southbound(X'',Y'')` even though the vehicle makes turn into continuation of a southbound segment, but not into the southbound segment itself. The main reason for this wording is that only cars from the incoming southbound segment are relevant to deciding whether a right turn is safe from the eastbound lane. As for an opposite example, one cannot make a right turn from a point (X'', Y'') on the southbound lane into an outbound segment that can be reached by straight motion from the point (X', Y') on the eastbound lane since that would be a left turn. Consequently, for this second pair of points `perpendicDir(X'',Y'',X',Y')` is false. Write all the rules that are required to implement this predicate completely.

- Implement the predicate `canDriveStraight(Car,X,Y)`: it is true if Car is located at (X, Y) and the traffic light is seen as green from this location. This rule means that a car facing a direction can keep going in that direction if the light is green.

- Write 2 rules implementing the predicate `canTurnRight(Car,X,Y)`. First, write rule saying that a car located at (X, Y) can turn right on the green light. Second, write rule saying that a car located at (X, Y) can turn right on the red light, if there is no other incoming car that travels straight in the direction of a turn and such that the distance to that car is less than 45 units.

- Write a rule implementing the predicate `canTurnLeft(Car,X,Y)`. A car can turn left on the green light if it is not the case that there is another car that can travel in the opposite direction and such that the distance to that car is less than 80 units.

Before you test your program, make sure that rules with the same predicate in the head are consecutive. Or you may include as the first line the following compiler directive

```
:- discontiguous(myPred/2).  % 2 means number of arguments
```

to say that the rules for `myPred(X,Y)` alternate with other rules. Otherwise, a compiler may complain that some of your predicates (heads of rules and atomic statements) are not consecutive. Test your program using the following atomic statements:

```
at(honda,11,-26).  at(subaru,-3,34).  at(bmw,-22,-19).  at(ford,21,14).
light(green,11,-26).  light(green,-3,34).  light(red,-22,-19).  light(red,21,14).
```

More specifically, ask the following queries (use `Tools/Tracer` in ECLiPSe Prolog to visualize)

```
?- canDriveStraight(Car,X,Y).
?- canTurnLeft(Car,X,Y).
?- canTurnRight(Car,X,Y).
```

What answers you are getting? Save them in your file **traffic.txt** Explain briefly your results in this file. Argue if they are correct or not. You can lose marks if you do not explain.

Handing in solutions. An electronic copy of: (a) your program (the name of the file must be **traffic.pl**) that includes all your rules (and given atomic statements). (b) An electronic copy of your session with Prolog that shows clearly the queries you submitted to Prolog, and the answers that were returned (the name of this file must be **traffic.txt**);

3 (25 points). This problem will exercise what you have learned about rules in Prolog, by having you implement a toy expert system that provides simple financial advises.[1]

The function of the financial advisor tool is to help a user who got some extra cash decide whether to save portion or all cash in a savings account or invest in the stock market. Some investors may wish to split their money between the two depending on their circumstances. The financial decision that will be recommended for individual investors depends on their income, number of dependents and the current amount they saved according to the following criteria:

- Individuals with an inadequate savings account should always save the whole amount of extra cash, regardless of their income.
- Individuals with an adequate savings account and an adequate income should consider splitting in half their surplus cash between savings and stocks investment to increase the cushion in savings while attempting to increase their income through stocks.
- Individuals with a lower income who already have an adequate savings account should prefer a riskier but potentially more profitable investment in the stock market by investing 80% of their extra cash in stocks and allocating the remaining money into savings.

The adequacy of both savings and income is determined by the number of dependents an individual must support. In this assignment, we assume one has to keep \$9000 in savings for each dependent. We introduce the predicate $minSavings(NumberDependents, MinSav)$ to characterize the amount $MinSav$ as just stated. The predicate $minIncome(NumberDependents, MinInc)$ is used to calculate the minimal income amount as \$25000 per year plus an additional \$8000 for each dependent. The current financial description of an individuals is given using the predicates $saved(S)$, which is true if the individual saved the amount S, the predicate $earnings(E)$, which is true if the individual earns E per year from employment, the predicate $cash(C)$, which is true if C is surplus cash, and the predicate $dependents(D)$, which is true if the individual has D dependents in total. The expert system is provided with 4 atomic statements describing an individual: (1) how much the individual saved, (2) what are the earnings, (3) how much cash is available, (4) how many dependents the individual has.

To implement our toy expert system, you have to translate the guidelines given above into PROLOG. Use only the following predicates in your implementation: you cannot introduce any other predicates.

1. Write rules implementing the predicates $minSavings(D, M)$ and $minIncome(D, M)$.
2. Implement the predicate $savingsAdequate(Amount, D, Min)$ with the single rule that makes this predicate true if the saved $Amount$ is greater (or equal) than the minimal amount Min in savings required for an individual with D dependents.
3. Write the single rule implementing the predicate $incomeAdequate(Amount, D, Min)$ that is true if $Amount$ of earnings exceeds or equal to the minimal income amount Min for an individual with D dependents.

[1] *Disclaimer*: the following rules may not be realistic and should not be considered reliable when making any financial decisions. They do not provide offer, or recommendation to acquire or dispose of any investment or to engage in any other financial transaction.

4. Let the predicate $save(What)$ be true if $What$ is portion of extra cash that should be saved given the current financial circumstances. Implement 3 rules calculating what amount should be saved depending on whether savings and income are adequate or not. Each rule should implement one of the guidelines given above.

5. Let the predicate $invest(What)$ be true if $What$ is part of surplus that should be invested. Write 3 rules in PROLOG calculating what amount should be invested depending on whether savings and income are adequate or not. In your implementation, use the guidelines given above.

Once you have implemented all the predicates, make up atomic statements with the predicates $saved$, $earnings$, $cash$ and $dependents$ to describe one individual. Add these 4 atomic statements to your rules. Ask the queries with variables $save(X)$ and $invest(Y)$ and save the answers in your file. You have to submit both your PROLOG program and the file with queries and answers.

Handing in solutions. An electronic copy of: (a) your program (the name of the file must be **advisor.pl**) that includes all your rules. (b) Make up your own atomic statements and run queries with variables: `save(S)` and `invest(I)`. Submit an electronic copy of your session with Prolog that shows clearly the queries you submitted to Prolog, and the returned answers (name this file as **advisor.txt**)

How to submit this assignment. No printouts. Read regularly *Frequently Answered Questions* and replies to them that are linked from the Assignments Web page at

`http://www.scs.ryerson.ca/mes/courses/cps721/assignments.html`

If you write your code on a Windows machine, make sure you save your files as plain text that one can easily read on Linux machines. Before you submit your Prolog code electronically make sure that your files do not contain any extra binary symbols: it should be possible to load `book.pl` or `rules.pl` into a recent release 6 of ECLiPSe Prolog, compile your program and ask testing queries. TA will mark your assignment using ECLiPSe Prolog. If you run any other version of Prolog on your home computer, it is your responsibility to make sure that your program will run on ECLiPSe Prolog (release 6 or any more recent release), as required. For example, you can run a command-line version of */opt/ECLiPSe/bin/x86_64_linux/eclipse* on moon remotely from your home computer to test your program (read handout about running *ECLiPSe Prolog*). Read the *handout* posted on D2L about running *ECLiPSe Prolog*. To submit files electronically do the following. First, create a **zip** archive:

```
zip yourLoginName.zip store.pl store.txt traffic.pl traffic.txt
                      advisor.pl advisor.txt
```

where `yourLoginName` is the login name of the person who submits this assignment from a group. Remember to mention at the beginning of each file *student numbers* and *names* of all people who participated in discussions (see the course management form). You may be penalized for not doing so. Second, upload to D2L into "Assignment 1" folder **all** individual files and your ZIP file **yourLoginName.zip**

Improperly submitted assignments will **not** be marked. In particular, you are **not** allowed to submit your assignment by email to a TA or to the instructor.

Revisions: If you would like to submit a revised copy of your assignment, then upload your files again. The same contact person must upload the files from a group. A new copy of your assignment will override the old copy. You can submit new versions as many times as you like, and you do not need to inform anyone about this. Don't ask your team members to upload your files, because TA will be confused which version to mark: only one person from a group should submit different revisions of the assignment. The time stamp of the last file you submit will determine whether you have submitted your assignment on time.