cps721: Assignment 4 (100 points).
Due date: Electronic file - November 16, Monday, 2020, before 21:00. Read Page 6.

You must work in groups of TWO, or THREE. You cannot work alone.

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

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.

This assignment will exercise what you have learned about natural language processing. In this assignment, we imagine a computerized banking assistant. Ultimately, we would like to be able to tell our assistant what to do using ordinary English imperative sentences like "Transfer $500 from my Bank of Montreal account to the small account of my partner in the Metro Credit Union."[1] As in class, to interpret English sentences like these, we need to construct a *database* of facts about our world, a *lexicon*, and a *parser/interpreter*. As in class, we will restrict our attention here to the English noun phrases like "the large account of a woman from Scarborough in a Canadian bank".

1. Before we start using any English words, build in Prolog a simple database of facts (e.g., 10 facts of each type) about imaginary accounts, their owners and banks with the following 4 predicates:

- $account(Number, Name, Bank, Balance)$ where $Number$ is an account number, $Name$ is a name of a person who holds this account, $Bank$ is a name of a bank where this account is located, and $Balance$ is the current amount stored in the account. For example:
```
account(12,ann,metro_credit_union,2505).
account(13,robert,royal_bank_of_canada,1001).
```
- $created(Number, Name, Bank, Month, Year)$ where $Number$ is an account number as before, $Name$ is the name of the owner, $Month, Year$ are the month and the year when the account was opened. For example:
```
created(12,ann,metro_credit_union,8,2018).
created(13,robert,royal_bank_of_canada,6,2019).
```
- $lives(P, City)$, where P is a person name and $City$ is a city. For example:
```
lives(philip,richmondHill).  lives(ann,markham).
```
- $location(X, C)$, where either X is a city and C is a country, or X is bank and C is a city (include about 10 facts for both alternatives). The cities and banks should be same that you mentioned before. For example:
```
location(scarborough,canada).  location(markham,canada).
location(sanFrancisco,usa).  location(royal_bank_of_canada,toronto).
```

Keep your database in the file **nlu.pl** Later, when you will be developing your lexicon, you can add to your database *new rules* defining concepts that represent meaning of the words that you will need in the lexicon. The bodies of the rules can *use only the 4 predicates given above*. Show that your database works properly by formulating the following queries in Prolog (similar to what you did in the first assignment), and obtaining suitable answers:

(a) Is there an account in the Royal Bank of a man from Richmond Hill ?

(b) Is there a Canadian who has more than one account in CIBC?

(c) What are the banks in Toronto?

(d) What is a balance of an account in the Bank of Montreal of a person from Scarborough?

(e) What bank keeps accounts of two distinct local persons ? (Note: for purposes of this assignment, a person is *local* if it is a person who lives in Canada, and *foreign* otherwise.)

[1] Of course, executing these commands will require proper authorization and verification of access privileges to an account. This might be a topic for another course.

(f) What are the cities in the USA?

(These queries should *not* use English noun phrases!) You can use more queries, but at least all 6 queries mentioned above. It is up to you to formulate a range of queries that demonstrates that your program works properly. Keep your queries and answers computed by Prolog in the file **nlu.txt**

Once you have this working, you are ready to consider English noun phrases and the accounts they refer to. Here are some examples of the kinds of queries your system should be able to answer:

1. `what([a, city, in, canada], X).`
2. `what([the, canadian, man, with, a, large, account, in, a, local, bank], X).`
3. `what([any, foreign, male, person, with, a, small, account, in, a, canadian, bank], X).`
4. `what([a, foreign, male, person, from, losAngeles, with, a, small, account, in, rbc], X).`
   (note: for simplicity, we use "rbc" or "losAngeles" as if they were single English words.)
5. `what([a, balance, of, a, large, account, in, a, local, bank], X).`
6. `what([any, local, bank, with, an, account, of, a, man, from, usa], X).`
7. `what([an, owner, from, canada, of, a, large, account], X).`
8. `what([a, woman, from, markham, with, a, medium, account], X).`
9. `what([a, bank, in, canada, with, a, small, account, of, a, foreign, person], X).`
10. `what([a, medium, account, in, a, canadian, bank, with, a, small, account, of, an, american], X).`
11. `what([the, balance, of, the, medium, account, in, metro_credit_union, of, a, woman, from, markham], X).`
12. `what([a,balance,of,an,account,of,an,american,with,a,small,account,in, a,local,bank,with,a,large,account],X).`

2. Build a lexicon, as we did in class, of articles, adjectives, proper nouns, common nouns, and prepositions. Keep all lexicon-related rules in the same file **nlu.pl** where you keep your database. To cover the above examples, you will need at least the following words:
articles: `a, an`
common nouns: `bank, city, country, man, woman, owner, person, account, balance,`...
prepositions: `of, from, in, with, ...`
proper nouns: `toronto, losAngeles, canada, rbc, metro_credit_union, ...`
adjectives: `american, female, local, foreign, small, medium, large, old, recent, ...`
(Actually, an adjective "small" is vague when it is used to characterize a balance of an account, but for simplicity, in this assignment, we assume that any amount less than \$1000 is small, any amount greater than \$10000 is large, and any amount in between is medium.) Any account opened this year is considered to be "recent", all other accounts are "old".

At this stage, you might wish to elaborate your database by adding there several new rules defining other predicates such as $city(X)$, $bank(B)$, $person(P)$ and so on in terms of the predicates given to you in Part 1. These predicates will be new concepts linked to English words described in the lexicon. You should **not** introduce any new atomic statements, i.e., your atomic statements must use only the 4 predicates mentioned above in Part 1. Your lexicon should include all the words mentioned above (in total, **25 words or more**, apart from articles). The word "any" should be treated as an article. Notice that some words are ambiguous, and for this reason, you need several rules for them in your lexicon: one rule per possible meaning. (For comparison, consider the 4 rules for the preposition "with" discussed in class.) Remember that it is easy to defeat a language understanding program by using a word that it does not know about. Vocabulary is important in these systems. Make yours as smart as you can. Keep both the database and your lexicon in the file **nlu.pl**

3. Copy the Prolog parser/interpreter for noun phrases given in class (or write your own), and define the `what` predicate used above. The parser must be also in the same file **nlu.pl**

4. Test the `what` predicate on a variety of noun phrases, like those above, showing that it is capable of identifying the entities being referred to by your noun phrases. It is up to you to choose noun phrases for testing, but you must convincingly demonstrate that your program works properly. Try at least **10 new** different noun phrases (in addition to the phrases given to you). Remember that testing your program is very important part of the software development cycle. You lose marks if you do not test your program as required. Copy all results of your tests into **nlu.txt** Copy all results of your tests into **nlu.txt**

5. Do this work only when the previous parts of your assignment are complete. The English noun phrases described above are quite limited. Generalize your program to handle some additional features of English:

- Handle the article "the". The trick here is that a noun phrase like "the balance of the medium account in metro_credit_union of a woman from markham" should succeed in naming a balance if there is a *unique* account of the appropriate kind.
- Handle prepositional phrases of the form "between x and y" applied to balances, e.g. `what([a, balance, between, 100, and, 25000], X)`. *Hint*: you can do this by modifying the parser, e.g., by adding there new rules that handle the words "between", "and". Also, you can use the built-in predicate `number(X)` that is true if X is a number.

Handing in solutions:
(a) An electronic copy of your database, lexicon, and parser in one file **nlu.pl** must be included in your **zip** archive;
(b) a copy of your session(s) with Prolog, showing the queries you submitted and the answers returned: name this file **nlu.txt** and include it in your **zip** archive. It is up to you to formulate a range of queries that demonstrate that your program is working properly. You must try all queries mentioned above and at least 10 new noun phrases.

6. Bonus work (**up to 40 points**):

To make up for a grade on another assignment or test that was not what you had hoped for, or simply because you find this area of Artificial Intelligence interesting, you may choose to do extra work on this assignment. *Do **not** attempt any bonus work until the regular part of your assignment is complete.* This Part 6 is much harder than the other parts of this assignment. If your assignment is submitted from a group, write whether this bonus part was implemented by one person only (in this case only this student will get all bonus marks). Otherwise, it is assumed this bonus question was implemented by all team memebrs; in this case bonus marks will be evenly divided between students. Wrong or incomplete solutions may be graded as 0. Note that the students who submit this bonus work can be invited for an interview with the instructor where they can be asked to explain their work or solve similar exercises. Consult the Policy on Collaboration in the cps721 Course Managment Form for details.

This part of the assignment is related to learning decision trees. It is continuation of the last part of the 1st assignment and the 3rd part of the 2nd assignment. Recall, that in the 1st assignment we considered 14 records representing applications for a loan. We also considered 5 attributes (properties): *risk, history, debt, collateral, income*, out of which four last attributes are used to process the application, and the first attribute (risk) represents a decision that should be made after evaluating each application according to other 4 attributes. Recall Figure 2 of the 1st assignment that provides an example of a decision tree. In a decision tree, each internal node represents a test on some property, such as *credit history* or *debt*; each possible value of that property corresponds to a branch of the tree. Leaf nodes represent goal class labels, such as *low risk* or *moderate risk*. We observed that all given 14 records can be correctly classified by this tree, i.e., for each record we can determine whether it belongs to the class label *low risk* or *moderate risk*. Now, if we encounter a new record (new loan application) of unknown type, it may be correctly labeled by traversing this tree starting from the root node. At each internal node, we evaluate the record using the test condition for that node and take the appropriate branch. This continues until reaching a leaf node providing the class label for the record.

Classifying a record is straightforward once a decision tree has been constructed. In this part of the assignment, we are going to write a recursive program that constructs (learns) decision trees. As an input, this program takes a finite list of attributes to be used in internal nodes, a goal attribute that determines class labels for leaf nodes, and a finite set of records (training instances). In principle, there are many different decision trees that can be constructed from a given lists of attributes and records. While some of these trees are larger than others, and some trees are more accurate than others (in a sense that they classify correctly more records), finding the optimal tree for large lists of records is computationally infeasible because of the huge number of alternative decision trees that should be searched and compared with each other. Nevertheless, practically efficient algorithms have been developed to induce a reasonable accurate, albeit suboptimal, decision tree in a reasonable amount of time. For this reason, these algorithms are widely used for a variety of applications. In this assignment, to simplify the matters, we consider subsequently only binary attributes (i.e., the properties with two outcomes only). In the 1st assignment, the attribute *income* was not binary because it had 3 possible values, and similarly, the attribute *credit history* had 3 values. However, we always can reduce attributes with multiple values to binary attributes. This is what we did in the 2nd assignment, where all attributes that we considered had only 2 possible values: $Bool$ or $notBool$. For this reason, this simplification does not restrict applicability of our algorithm.

The algorithm that you have to implement works top-down: it induces the decision tree from the root down to leaves. On each recursive call, the algorithm divides the remaining list of records into two halves (they can have different sizes). Each half has same value according to one of the attributes that is selected to divide the records. Selecting an optimal attribute to partition the records is a non-trivial task, that we discuss below. But once the best (the most informative) attribute has been selected, the algorithm is straightforward application of divide-and-concur strategy. This best attribute represents a new node in the tree and the two branches going out of this node correspond to partitioning the records into two halves. For each half of the records, the algorithm again selects the next most informative attribute (out of the remaining unused attributes), evaluates the records according to this attribute, partitions them again into two halves, and continues until it will reach the leaf node. Each leaf node represents some records that have same value of the goal class. There is no need to partition them anymore: they can be uniquely labeled by one of the two goal classes. Since the ultimate purpose of the decision tree is ability to assign goal labels to all records, once a sub-list of records has been assigned the same goal label, there is no need for the algorithm to continue recursion and it can terminate.

The only part in this algorithm that remains unspecified is a locally optimal decision about which attribute to use to partition data. In machine learning, a sub-area of AI related to inducing decision trees, there are several well-known criteria for selecting "the best" attribute. Basically, these criteria measure so-called impurity of records with respect to a goal class label. A good attribute should split the examples in subsets as pure as possible. The decision tree in Figure 2 from the 1st assignment selects *credit history* as the first attribute to divide the records. However, this is not the best attribute to select because among the records with *unknown* credit history we find 2 records with *high risk*, 2 records with *low risk*, 1 record with *moderate risk*, among the records with *good* credit history we find 3 records with *low risk*, 1 record with *moderate risk*, and 1 record with *high risk*, among the records with *bad* credit history we find 3 records with *high risk* and 1 record with *moderate risk*. In other words, the attribute *credit history* is not particularly informative in terms of the final decision that the tree is supposed to make whether the risk of a loan application is high, low or moderate. Notice that the attribute *income* is more informative, and indeed, we can construct a smaller decision tree, if we evaluate records first according to income of the applicant, and then evaluate credit history. Most of the records can be correctly classified just after doing these two tests in contrast to the tree in Figure 2 that requires 4 tests in the worst case before decision about the credit risk can be made.

The measure used to determine the best way to split the records is defined in terms of class distribution of the records before and after splitting. Let $p_G(i)$ denote the fraction of records belonging to class i, where i can have only one out of the two values:

$$p_G(i) = \frac{\mid Instance \in Records: \ value(G, Instance, i) \mid}{\mid Records \mid},$$

where $|Instance \in Records: \ value(G, Instance, i)|$ is the number of records with value i of the goal attribute *G*, and the total number of examples is $|Records|$. The two numbers denoted by $(p_G(Bool), p_G(notBool))$ represent the distribution of records and, by the definition, $p_G(Bool) + p_G(notBool) = 1$. Suppose that the class distribution of records before splitting is (0.5, 0.5) meaning that there are an equal number of records from each class. Then, the best split is based on the degree of impurity of the child nodes. The smaller the degree of impurity, the more skewed the class distribution, the more informative it is in the sense that we get closer to assigning one of the two class labels. For example, a node with class distribution (0,1) has zero impurity, whereas a node with uniform class distribution (0.5, 0.5) has the highest impurity. In this assignment we use the GINI index to measure impurity. This index is represented in the program by the predicate *gini(G,Records,Index)* meaning that for the list *Records*, and the goal attribute *G* the GINI index is $1 - (p_G(Bool)^2 + p_G(notBool)^2)$. Recall that in the 2nd assignment we had the predicate *count(Records,A,Bool,T,F)* that is true if there are T records in the list $Records$ with attribute A having value $Bool$ and F records with attribute A having the other value. Using this notation, we can define the GINI index as $Index = 1 - ((\frac{T}{T+F})^2 + (\frac{F}{T+F})^2)$. Here and subsequently, for simplicity, we always assume that there are no records that do not have one of the two values, i.e., $T + F$ is the total number of training instances in the list $Records$. (A side remark: the instances from $Records$ are called training instances because they are used to train a decision tree, i.e., without them our algorithm would not be able to learn a decision tree.)

In this part of assignment, you have to do the following (part of the program is given to you and can be downloaded from the assignments folder at D2L).

- Write a recursive program to implement the predicate *induce(Goal, Records, Attributes, DT)* that is true if the term DT is a decision tree induced from the list of $Records$ using the list of $Attributes$. The purpose of DT is to label each record with one of the two possible values that the attribute $Goal$ can have. The term DT can simply be a constant representing one of these values, if all records have same value of $Goal$. This happens when the algorithm inducing the decision tree terminates. Otherwise, when algorithm creates an internal node using the best attribute Att out of the list of $Attributes$ to divide the records, DT can be a tree represented by the term *tree(iF(Att,Bool),YesTree,NoTree)*, where each of the branches *YesTree, NoTree* is a decision tree itself. (We write "**iF**" above to make it different from the reserved keyword "**if**".) Note that you are given the Prolog rule implementing the predicate *bestAttribute(Goal, Records, Attributes, Att, RemainListAtts)* that is true if Att is the best attribute in the list of $Attributes$, and *RemainListAtts* is the list of remaining never used attributes (some of them might be used later to create new internal nodes, if necessary). This rule uses the predicate *impurity(Att,Goal,Records,Impurity)*, where $Impurity$ is the sum of GINI indexes computed for the lists Pos and Neg that sub-divide the list $Records$ according to the two possible values of the selected attribute Att. In other words, since each atrribute has only 2 values, and since Pos and Neg will be mutually exclusive lists, after computing a GINI Index for each of these two lists, we have to sum them to determine $Impurity$ for Att with respect to $Goal$ attribute. Note that both the GINI index for Pos and the GINI index for Neg are computed with respect to the $Goal$ attribute. By computing impurity for each of the remaining attributes Att with respect to the $Goal$ attribute, we can measure quality of each attribute. Recall that the attribute with the minimal impurity is the best one. This rule also uses the predicate

  minImpurity(Goal,Records,Attributes,CurrAtt,CurrImp,BestAtt,RejectAtts,RemainAtts)

  that is true if $BestAtt$ is the best attribute in the list of all available $Attributes$ in comparison to the currently considered attribute $CurrAtt$ that has the total impurity $CurrImp$, and $RemainAtts$ are the remaining attributes. Initially, when this predicate is called, an attribute is selected, its impurity is computed, and the list of rejected attributes $RejectAtts$ is empty. But when the recursive program (given to you) for this predicate terminates, the variable $RemainAtts$ gets same value as the variable $RejectAtts$ and contains the list of all unused attributes from the list $Attributes$ except of the best one that is selected to split the tree. You are given the rule that implements this predicate.

- Write a Prolog program that implements the predicate *gini(A,Records,GiniIndex)* using the formula given above and use this predicate to implement the predicate *impurity(Att,Goal,Records,Impurity)*. Read explanations above carefully for a hint on how to implement this predicate. (Another hint: both predicates can be implemented by short programs).
- Write all rules mentioned above in the file **dl.pl** given to you. Note that it should include all predicates from the 2nd assignment related to decision trees. You do not need any other programs to complete your program. Once you have completed the program, run it on a simple example given to you (download from the assignments folder on D2L). Copy the decision tree that your program computes into your report file **dl.txt** Note that the TA who will mark this assignment will use another test to find if your program works correctly.
- Finally, write your own testing example. The best (e.g., most funny) examples can be awarded extra marks at the discretion of the TA who marks this assignment. Include any number of atomic statements representing records from which you would like to induce a decision tree. When you select attributes remember that they should be binary (i.e., have only 2 values). Your test can be about anything. Run your program on your test. Include the results in your report **dl.txt**

Handing in solutions. (a) your working program (**dl.pl**) with all defined predicates (you must provide brief comments in the program); (b) your session with Prolog, showing the queries you submitted and the answers returned (the name of the file must be **dl.txt**). Include both dl.pl and dl.txt into your ZIP archive: read instructions below.

How to submit this assignment. 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 `nlu.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 *eclipse* on moon remotely from your home computer to test your program (read handout about running *ECLiPSe Prolog*). To submit files electronically do the following. First, create a **zip** archive:

```
zip yourLoginName.zip nlu.pl nlu.txt [dl.pl dl.txt if you solved Bonus Part 6]
```

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, section 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 your zip file

yourLoginName.zip

to D2L into "Assignment 4" folder. Make sure it includes **all** your files. 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 upload a revised copy of your assignment, then simply upload it again. (The same person must upload.) Do not ask your team members to upload your assignment, because TA will be confused which version to mark: only one person from a group should submit different revisions of the assignment. The groups that submit more than one copy of their solutions will be penalized. The time stamp of the last file you upload will determine whether you have submitted your assignment on time.