EECS 839
Spring 2018

Programming Project

due: April 20, 2018

Implement the following rule-checking system. **Your project should be implemented on one of the departmental computers, e.g., cycle 1, under Linux**. Your program should accept two input files: the first input file will contain rules, the second input file will contain data in the LERS format. Both input files may contain comments (anything that starts from "!" to the end of the line is a comment).

In the text that follows, spaces should be understood not only as ordinary spaces but also as white space characters, such as the end of line, tab, etc. Any line of both input files may start from one or more spaces and one or more spaces may end it. Additionally, there may be empty lines in both files. You may assume that the i**nput data file does not contain errors**.

Rules will be preceded by three numbers, separated by commas and zero or more spaces: specificity, strength, and the total number of matching training cases. A rule contains one or more conditions and one action. Conditions are separated by some symbols, e.g., "&", "^". The conditions and the action may be preceded or followed by any number of spaces. Rules are separated by one or more spaces and/or by empty lines. A condition starts from "(", then zero or more spaces, the name of attribute, zero or more spaces, comma, zero or more spaces, the attribute value, such as "high", "3", or "1.2..3.5", zero or more spaces, finally, ")". The rule's last condition and the action are separated not only by possible spaces but also by a string of symbols (e.g., "->", "-->", etc.) that starts from "-". An action is described the same way as the condition, except that the role of "attribute" is played by "decision".

Input data files may start from the list of variable declarations. This list starts form "<", then comes a sequence of some symbols, the last symbol of the list is ">". This list should be ignored. The second part of the input data file starts from "["; then a space, then comes a list of attributes and a decision name, separated by spaces; then another space, and then "]". The decision is the last name. The following part contains values of the attributes and decisions. Value "?", "*", and "-" mean that the actual value of attribute or decision is missing. During matching a case *x* with the rule set, if the value of an attribute *a* for the case *x* is ?, all conditions starting from "*a*" should not be matched. If the value of *a* is * or -, all conditions starting from "(*a*" should be matched. Attribute values and decision values are either symbolic or numerical. Cases of symbolic values are: *medium*, 12..14, 1.25..2.37, etc. Examples of numerical values are: 42, 12.97, –0.1234, +2.3, etc.

Any case should be processed first by *complete matching* and, if it is impossible, by *partial matching*. Your program should **match a numerical value of a case with an interval of the corresponding condition of a rule**. Intervals should be considered to be closed from both sides.

First your program should ask the user for the name of the input rule file. The expected response of the user is the name followed by pressing the <RETURN> key. If the specified rule file does not contain three numbers preceding every rule, your program should inform the user that the rule file is not in the proper format. Then the program should ask for the name of the input data file. The expected response of the user is the name followed by pressing the <RETURN> key.

Then the program should ask four questions related to the choice of the best decision. The first question should be whether the user wishes to use the matching_factor. The expected response should be either "y" followed by pressing the <RETURN> key or—if not—pressing the <RETURN> key. The next question is whether *strength* or *conditional probability* should be used as the strength_factor. The expected response should be either "s" followed by pressing the <RETURN> key or "p" followed by pressing the <RETURN> key. If "p" is selected, your program should check whether in any rule the total number of matching cases is equal to zero . If so, the user should be informed that the conditional probability option cannot be executed. The third question is whether the user wishes to use the factor associated with specificity or not. The expected response should be either "y" followed by pressing the <RETURN> key or—if not—pressing the <RETURN> key. The fourth question is whether the user wishes to use the support of other rules or not. The expected response should be either "y" followed by pressing the <RETURN> key or—if not—pressing the <RETURN> key.

Then the program should ask the user whether the user wishes to know concept statistics. The expected response should be either "y" followed by pressing the <RETURN> key or—if not—pressing the <RETURN> key. Then the program should ask the user whether the user wishes to know how cases associated with concepts were classified. The expected response should be either "y" followed by pressing the <RETURN> key or—if not—pressing the <RETURN> key. After all of that, the program should provide the first part of the report called "General Statistics", supplying the following information on the screen:

This report was created from: <input rule file> and from: <input data file>.
The total number of cases: <integer>
The total number of attributes: <integer>
The total number of rules: <integer>
The total number of conditions: <integer>
The total number of cases that are not classified: <integer>
PARTIAL MATCHING:
The total number of cases that are incorrectly classified: <integer>
The total number of cases that are correctly classified: <integer>
COMPLETE MATCHING:
The total number of cases that are incorrectly classified: <integer>
The total number of cases that are correctly classified: <integer>
PARTIAL AND COMPLETE MATCHING:
The total number of cases that are not classified or incorrectly classified: <integer>
Error rate: <percentage> *precision: two digits after the decimal point*.

When *concept statistics* was required, the program should report on the screen for each "decision_value" the following:

Concept ("decision_name", "decision_value"):
The total number of cases that are not classified: <integer>
PARTIAL MATCHING:
The total number of cases that are incorrectly classified: <integer>
The total number of cases that are correctly classified: <integer>
COMPLETE MATCHING:
The total number of cases that are incorrectly classified: <integer>
The total number of cases that are correctly classified: <integer>
The total number of cases in the concept: <integer>.

When information *how cases associated with concepts were classified* was required, the program should report on the screen for each "decision_value" the following:

Concept ("decision_name", "decision_value"):
List of cases that are not classified: <list of cases>
PARTIAL MATCHING:
List of cases that are incorrectly classified: <list of cases>
List of cases that are correctly classified: <list of cases>
COMPLETE MATCHING:
List of cases that are incorrectly classified: <list of cases
List of cases that are correctly classified: <list of cases>

<list of cases> contains cases, one case per line; each line contains the list of all attribute values and then decision value, separated by commas. <list of cases with justification> contains first a case, i.e., the list of all attribute values and then decision value, separated by commas; then, in the case where support was used, ";" and the total score, then lines representing all matching rules (describing all concepts). Every such line contains a list of conditions, separated by "&", then "->" and an action, where a condition starts from "(", then attribute name, comma, attribute value, and ")", similarly action, and then ";" and the rule score.

Finally, the program should ask the user whether the user wishes to exit the program or to start the program all over again.

General Remarks. Your program should be able to deal with unexpected answers of the user and not crush but rather repeat the question. Use of recursion is not encouraged. You should expect input data files of any size, more than a thousand cases and more than a hundred attributes.

Include all comments, including instructions about compiling and linking and the name of a computer on which you implemented your program, in a single file called **read.me**. Do not forget to include your name and KUID#. When you are ready to submit the project, send ALL necessary source files, makefile (if any), and the read.me file by e-mail to the TA. Do not send object files, executable files, and test data files. Late projects will be accepted with 10% penalty per day up to five days.