**Project for 5392**

Write

* 1. class framework for a CTL formula representation as a parse tree
  2. a corresponding parser that can parse a string with a CTL formula into a parse tree using your class framework
  3. a translator of a parse tree into a canonical representation that uses only those CTL operators for which the “Logic in Computer Science” textbook provides algorithms (p. 227: EX, AF, EU)
  4. testcases that would take a CTL formula string, parse it, report problems if any and would output a string with a canonical representation of that formula

The “Logic in Computer Science” textbook describes the CTL equivalences on p. 216 and in the section about the CTL algorithms pseudo-code starting from p. 225.

The system should produce meaningful warning messages to console if an input string is not a well formed formula in CTL and/or no reasonable output is possible.

The parsing contingencies should be implemented via an exception mechanism.

You should submit your archived (zipped) Java project directory with source files to a TRACS Dropbox.

Class hierarchy for Kripke structure

“ for CTL formula

Implement 3 algorithms from the book

Build model checker from scratch for CTL model checker for Kripke structure

Scanner class in Java

Input: formula and Kripke structure

**CTL model checker project for 5392 (Formal Methods)**

Develop a Java standalone application that implements a model checking analysis tool for verification of properties defined in the CTL temporal logic. An application should be able to take as input the name of a file that contains definition of the Kripke structure to be analyzed, a state ID for which the property should be checked and a CTL formula that defines the property. The output should notify the user if the property is held or fails in the given state.  
The name of the input file that contains a Kripke structure definition should be entered via a GUI or a command line. The CTL formula should be defined either in another file or a GUI text field that does proper CTL syntax checking. The application must perform syntax checking and provide meaningful error messages (line number and error description)  if a Kripke structure definition cannot be parsed. The result of the analysis can also be supplied either via a GUI or a standard output console.  
The language for a Kripke structure definition can be very simple, it can just use tables. For instance, a definition starts from an enumeration of states, next - enumeration of transitions with a source and destination state for each transition, next a list of states with enumeration of propositional atoms true in each state. Reasonable choice of delimiters can be used to separate table entries.

The following is an example content of a file that defines a Kripke structure.

s1, s2, s3, s4;  
t1 : s1 - s2,           (transition t1 is from state s1 to state s2)  
t2 : s1 - s3,  
t3 : s3 – s4,  
t4 : s4 – s2,  
t5 : s2 – s3;  
s1 : p q,                 (propositional atom names are separated by a space; a name consists of letters, it is case-sensitive)  
s2 : q t r,  
s3 : ,                       (i.e. set of propositional atoms for state s3 is empty)  
s4 : t;

The system's GUI should have a text field for entry of the CTL formula. It can also contain a textfield for entry of the state ID for which the property should be verified, otherwise the analysis output should enumerate states in which the given property holds.

The definition language for a CTL formula should follow the CTL syntax. Example representations for the operators: not, and, or, ->, EX, AX, EG, AG, E[ p U q ] etc.

The application must check that the given state ID does exist in the input Kripke structure.  
Example result output (either in a corresponding GUI textfield or console): “Property  {the given CTL formula} does not hold in state s4” (assuming that property was to be checked for state s4).

**Submission**

Turn in the assignment electronically to the TRACS drop box.  
The files of the problem should be archived into one archive file named modelCheckCTL<your initials>. The archive should preserve the directory structure starting from the root directory of the software system (directory named modelCheckCTL ).  
Classes should be in packages modelCheckCTL.model, modelCheckCTL.view, modelCheckCTL.controller (placed according to the Model View Controller architecture). If needed, there can be a package modelCheckCTL.util in addition to the ones already mentioned.  
The archive file should contain:

1. Description of acceptance testcases
2. Description of execution of acceptance testcases illustrated with screenshots of all the windows and pop-up windows of the system and console output along an acceptance testcase
3. UML class diagram for the software system
4. Source code (archive of directory structure starting from modelCheckCTL dir)