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, EF, AF, EG, AG, E[p U q], A[p U q]].

The pseudocode for CTL checking is on p.227 of "Logic in Computer Science", a scan of which is on TRACS in Homeworks folder.
An example implementation in C# is on TRACS in ExampleProjects/ModelCheckCTL_IGS_C#.zip.
In it, the example implementation of CTL model checking algos is in
...\ModelCheckCTL_IGS_\ModelCheckCTL\ModelCheckCTL\ModelCheckCTL\Objects\CtlFormula.cs file.

In the same "Objects" directory there are classes for Kripke structure implementation: State.cs, Transition.cs, KripkeStructure.cs.
This code is given as an example of implementation very closely following the pseudocode from p. 227 of the "Logic in Computer Science" textbook.
You can implement it in Python or Java or C# itself (as long as you do not directly copy the C# code).

You should not use already existing libraries for model checking. The point is for you to understand the algorithms.
The executable for that C# implementation is in
...\ModelCheckCTL_IGS_\ModelCheckCTL\ModelCheckCTL\ModelCheckCTL\bin\Debug
\ModelCheckCTL.exe

For testing, use the testcases in Test Files.zip in ExampleProjects on TRACS AND specify the microwave example for the given property in mchPeled.zip in the same folder on TRACS. During the presentation I will ask you to run at least the critical section testcase (Model 4 in Test Files.zip) and the microwave example.

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)