COL703: Logic for Computer Science
Assignment2

Submission Deadline: Tue 24 Nov 2020, 23:59
Submission Deadline with Late Penalty: Fri 27 Nov 2020, 23:59

Problem Statement

In this assignment you task is to implement ROBDD. You may refer to An Introduction to Binary Decision Diagrams by Anderson for reference.

What you have to do

1. You have to write this assignment in OCaml.
2. Write all of your code in `A2.ml` and `bdd.ml` files provided along with this document.
3. The file `formula.ml` contains the datatype for boolean formula. We will use the same type during the evaluations. Do not modify it.
4. Implement function `sat_count` that gives number of satisfying truth assignments to any given problem and function `all_sat` that prints all possible satisfying truth assignments.
5. The names of variables in input will be strings that may contain lowercase English alphabets (a-z) and digits (0-9).
6. The type `robdd` in `bdd.mli` is used to represent the ROBDD in the signature. You can define the type according to your implementation. Note that this type is used as input and output in some functions. You need to be sure that these functions use the type `robdd` consistently.
7. Implement the functions `bddFromExpr` in `bdd.ml`. The first input to this function is a boolean formula. The second input is a list of variables, representing the ordering to be considered while making ROBDD. This function returns the ROBDD as type `robdd`.
8. The input to function `sat_count: robdd -> int` is a ROBDD and the output should be an integer, that gives number of satisfying truth assignments of the input ROBDD node.
9. The input to function `all_sat: robdd -> sat_assignment list` is a ROBDD and output should be a list of satisfying truth assignments of the input ROBDD node.
10. The type `sat_assignment` corresponds to a satisfying assignment. Each satisfying truth assignment is a list of variables that are assigned true; rest of the variables are considered false.

11. The function `any_sat` takes a ROBDD of type `robdd` and returns any possible satisfying assignment of that ROBDD.

12. The function `to_bdd` generates the dot file for input `robdd` and write it to file named `bdd.dot`. You may use `ocamlgraph` library.

13. Solve either N-Queen or Knight's tour problem using your BDD implementation. Leave the function that you are not implementing as it is in `A2.ml`, i.e., it should raise the exception `Not_implemented`.

14. If you choose to solve N-Queen problem, implement function `n_queen: int -> string list` in `A2.ml`. The input to this function is an integer n, i.e., the size of the board. The function must return one possible solution of N-Queen problem. The solution is represented as list of strings, where each string denotes the position of a queen in the solution. The position is a string `c` (lower case c) appended with two single digit integers i and j, where i and j are row and column numbers respectively, starting from 0. For example, the string for cell in row 0 and column 4 should be `c04`.

15. If you choose to solve Knight's tour problem, implement function `knight: int -> int -> int -> string list` in `A2.ml`. The input should be an integer n, defining the size of the board, and integers i and j, denoting row and column number (starting from 0) respectively of initial position of the knight on the board. If the size of board is 4x4 and initially the knight is standing in row 1, column 3 (both starting from 0), the function will be called as `knight 4 1 3`. The output to this function should be a sequence of strings. The first element in the sequence should be cell name (as defined in point above) corresponding to the initial position of the knight. Each subsequent string in the sequence should represent the next cell visited by the knight. All the string should be formatted as defined in the point above.

16. You are not supposed to handle the exception in your program. Our script depends on you raising the exception from your code.

Submission Instruction

Submit one .zip file to the programming assignment named "A2: BDD" on `gradescope.com`. The name of the file should be < *yourEntryNumber* >.zip. On unzipping the file, it should produce the files `formula.ml, bdd.mli, bdd.ml` and `A2.ml` Do not change the interface of BDD module. It should also contain any other the source file used in your program. You may provide a README.md that contains any instruction for the evaluator.

Important Notes

1. Read the instruction in `Instructions.md` file.

2. Your code should follow the functional programming paradigm.
3. Do not change any of the names given in the signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.
4. You may define any new functions you like besides those mentioned in the signature.
5. Follow the input output specification as given. We will be using automated scripts to execute the code for evaluation. In case of mismatch, you will be awarded zero marks.
6. Implement the interface `bdd.mli` in `bdd.ml` file only. You can defined your own functions in `bdd.ml` as required.
7. Make sure your structure matches with the signature provided. We will not entertain any requests regarding minor change in the signature.