Homework 4

Due Thursday, 02/18/2016, by 11:55 p.m.

Your programs should be in a single file that can be loaded into ghci and tested. Problem numbers and any discussion of the programs should be set off as comments (using –) to make the Haskell compiler ignore them. Your program file should be turned in on Sakai like last week.

I should be able to load your file into Haskell (ghci) and have it compile without error. If there are programs with compilation errors, please comment them out so they don't get in the way of your other programs.

Please include good comments and use good variable names with your programs. Otherwise it is very difficult to read your code because Haskell is so concise.

Important: Your programs should have the exact names and types specified in the problem statement as your code will be automatically tested. Code that uses the incorrect name or expects different types (i.e., won't correctly execute my tests) will be counted as incorrect. Notice that all the functions to be written are in curried form.

Use comments to explain what each function does, especially helper functions.

Problems that are not programs should be written using LaTeX. When completed, please put the program file and the pdf generated from LaTeX into a folder, zip it up, and then turn it in as usual on Sakai.

1. (8 points) Use the functions in `FsemF.hs` (e.g., `eval`, `tautology`, `satisfiable`, `contradiction`, `implies`) to answer the following by writing Haskell programs that return the correct answer. Set the answers to identifiers named `ans1ai`, `ans1aii`, etc.

   (a) Which of the following are tautologies? If they are not tautologies, provide an assignment that does not satisfy the formula.

      i. $p \rightarrow (q \rightarrow p)$
      ii. $((p \wedge (q \vee r)) \rightarrow ((p \wedge q) \vee (p \wedge r))$
      iii. $((p \wedge (q \vee r)) \rightarrow ((p \wedge q) \vee r))$
      iv. $((p \wedge q) \vee r)) \rightarrow ((p \wedge (q \vee r))$

   (b) Show $\neg(p \wedge q) \models (\neg p) \vee (\neg q)$

2. A formula F is in disjunctive normal form if it can be written in the form $G_1 \vee ... \vee G_n$ where each G_i is of the form $A_1 \wedge ...A_k$ and each of these A_js is either a proposition letter or the negation of a proposition letter. Thus $(p \wedge q) \vee (p \wedge r \wedge \neg q) \vee \neg r$ is in disjunctive normal form. There is a theorem stating that any formula of propositional logic is equivalent to a formula in disjunctive normal form. Two formulas are equivalent if each logically implies the other, or, equivalently, each has the exact same truth table (or, equivalently, for each valuation of their propositional variables, they have the same values).

(a) (10 points) Please describe an algorithm that, given a formula of the propositional logic, returns an equivalent formula in disjunctive normal form. Illustrate your algorithm with the formula $(p \wedge (q \vee r))$.

Hint: Given a formula, write out a truth table for that formula. Now look only at the lines that make the formula true. For each of those lines, write a conjunction of proposition letters and negations of proposition letters that is true only for the values of proposition letters given in that line. From these, create a formula in disjunctive normal form that is true at exactly those lines of the truth table that make the formula true.

(b) (10 points) Use the `update` function to implement this algorithm as a function in Haskell. Your function should be named `toCNF`. Here are some useful intermediate functions that should help you get there:

i. `valToCForm :: [(String, Bool)] -> Form`
This function takes a valuation (i.e., row of a truth table) and forms a conjunction of propositional letters and negations of propositional letters that is only true for the given valuation.

ii. `valToForm:: [[(String, Bool)]] -> Form`
Takes a list of valuations (e.g. all the valuations that make a formula true) and returns a formula of propositional logic that is in disjunctive normal form and is true only for those valuations.

Using these helper functions, define `toCNF :: Form -> Form` that takes a formula, computes the list of valuations that make it true (using `allVals` and `update` from FSemF.hs to compute an equivalent formula in disjunctive normal form

3. (12 points) Given the sentences below, translate into (untyped) predicate logic. Introduce (and explain) relations and constants as needed:

(a) No student fails CS181.

(b) If someone is happy, then Mary is happy.

(c) All professors who fail their students are evil.

(d) Every student takes some computer science course.

(e) John likes every person who likes him.

Please represent the fourth sentence as an item of type Formula Term in Haskell. Please name the term `good`.