CSCI 239—Discrete Structures of Computer Science

Lab 2—Verifying the Truth of Propositions

This lab is an exercise using Haskell to evaluate propositional expressions.

Objectives:

- to learn to use Haskell to evaluate propositional expressions
- to learn to use and write Haskell functions that handle functions
- to gain experience with propositional logic

You will write several Haskell functions in this lab. You must include Haskell type signatures for each function. (See the given code for examples of how the type signatures are included.) However, you don't have to figure out the type signature yourself; let Haskell do it for you. Implement and test each function without the signature, then use the `:type` command to get the type signature and add it to your function definition.

Part 1: Using Haskell to verify truth tables

Make a truth table for each of the following propositional expressions using pencil and paper:

1. $\neg p \wedge q \vee p \wedge \neg q$
2. $p \vee \neg q \wedge r$
3. $(p \vee q) \wedge (\neg p \vee r)$
4. $(p \vee \neg q) \wedge (r \vee \neg s)$

Write a Haskell function equivalent to each of the expressions above. Recall from Lab 1 that Haskell uses the *or else* operator `||` to implement the mathematical operator ∨, it uses the *and also* operator `&&` to implement the mathematical operator ∧, and it uses the *not* function to implement ¬. Now use the appropriate supplied Haskell function *mapTruth2*, *mapTruth3*, or *mapTruth4* to determine the truth values for each of your functions. (The number of the *mapTruth* function indicates how many arguments the corresponding function takes. For example, if you call your first function *f1*, which has two parameters *p* and *q*, you can type "`mapTruth2 f1`" to get a list of truth-values for the function in truth table order. It is not necessary for you to understand how the *mapTruth* functions work at this stage.)

Compare the values your function returned to your truth tables. If the values are different for any function, determine why, revising the table and/or function until they agree. Does this agreement guarantee that you have done the table and function correctly? Why or why not?

Part 2: Adding → and ↔ to the mix

Haskell does not have built-in operators or functions for the *implies* (→) or *if and only if* (↔) operations. Determine propositional expressions equivalent to these two logical operations, using only the ∧, ∨, and ¬ operations; simplify your expressions, if possible.

Write Haskell functions *implies* and *ifAndOnlyIf* that implement your propositional expressions. Use the *mapTruth2* function to verify that your functions produce the correct truth values.

Now write a Haskell function that implements the expression (p → q) ↔ (r → s) from the pre-lab exercise. Since *implies* and *ifAndOnlyIf* are functions, not operators, you'll need to either use the function name before its two parameters or enclose it in single back-quote characters (`) to convert it into an operator. For example, the expression $p \rightarrow q$ could be written as "`implies p q`" or as "`p `implies` q`". Test the function using *mapTruth4*, and compare the results to your truth table. Again, if the results don't agree, determine why, and make any necessary correction to your truth table and/or function.

Part 3: Using propositional logic to solve a programming problem

Consider the problem of determining what you can see in the sky. Here are a few hypotheses that help define the problem:

- If it's cloudy, you can't see anything in the sky.
- It's always either day or night, but never both.
- It's always either clear or cloudy, but never both (to keep it simple).
- If it's clear in the daytime, you can see the sun.
- If it's clear at night, you can see the stars.
- You can't see the stars during the day.
- You can't see the sun at night.

Assign variables as needed to entities in these propositions, using only one variable for any two mutually exclusive entities, as you did in part 1 of the pre-lab. Then restate the conditions as logical propositions using the variables and the appropriate operators; simplify the expressions, if possible. In particular, you can potentially use propositions that represent exclusive or conditions to eliminate variable. (If $a \leftrightarrow b$ is a hypothesis, then you can replace the variable b with $\neg a$.) Make a truth table or Haskell function for any of the expressions that contain more than one operator.

Design a Haskell function that correctly returns a string representing one the following four values, based on the current time and sky conditions and the propositions above, using the smallest number of *Bool* parameters possible:

1. You can see neither the sun nor any stars
2. You can see the sun, but not any stars
3. You can't see the sun, but you can see the stars
4. You can see both the sun and the stars

To reduce the number of parameters, make sure that you have only one variable for any mutually exclusive conditions. Also, recall that Haskell strings are written like Java strings using double quote characters.

Use the appropriate *mapTruth* function to test your function. (Note that the *mapTruth* functions work, even when the final return value is not *Bool*, as long as all the parameters are *Bool*.) Do you get the results you expected? Are all four return values present in the results? Did your logical analysis help you program the function correctly?