Part I: More on Propositional Logic

The problems in Part I build on the propositional logic package developed in class. For the problem of this part, add your code to a copy of: http://www.cis.syr.edu/courses/cis352/coursework/prop.hs.

Problem 1: (10 points).

Background: A propositional formula is in negation normal form when:

(i) the only binary connectives it uses are & (and) and ∨ (or) and
(ii) the only place a ¬ (negation) appears is before a variable.

So for example, (P ∨ ¬ Q) & R and (((P ∨ Q) & (¬ P) & (¬ Q))) are in negation normal form, but P → Q, ¬ (¬ R), ¬ (P & Q), and ¬ T are not.

Your task: Write a function

isNNF :: Prop -> Bool

that tests whether a proposition is in negation normal form.

Problem 2: (20 points).

Write a function

toNNF :: Prop -> Prop

that translates a proposition to an equivalent proposition in negation normal form. In doing this translation, the following equivalences will be useful:

¬ (P & Q) ↔ (¬ P) ∨ (¬ Q)
¬ (P ∨ Q) ↔ (¬ P) & (¬ Q)
(P → Q) ↔ (¬ P) ∨ Q
(P ↔ Q) ↔ (P → Q) & (Q → P)
¬ (¬ P) ↔ P

My answer for this has 15 cases, including 8 of the form

toNNF (Not stuff) = other-stuff

Your answer may have a few more or few less cases depending on how you do things.

Problem 3: (10 points).

Background: For this homework, a proposition p is called tidy when p = F or p = T or p contains no constants (i.e., T or F).

Your task: Write a function

isTidy :: Prop -> Bool

that tests whether a proposition is tidy. (Hint: A helper function may turn out to be handy.)

Problem 4: (20 points).

Every propositional expression is equivalent to a tidy one. Write a function

toTidy :: Prop -> Prop

that translates a proposition to an equivalent tidy one. To make life simpler, first translate the proposition to a negation normal form, then translate the negation normal form expression by making use of the following equivalences:

¬ T ↔ F
¬ F ↔ T
(P & T) ↔ (T & P) ↔ P
(P & F) ↔ (F & P) ↔ F
(P ∨ T) ↔ (T ∨ P) ↔ T
(P ∨ F) ↔ (F ∨ P) ↔ P
Part II: More on Higher-Types

Problem 5: (12 points).
Write a function

\[
same :: [\text{Int}] \rightarrow \text{Bool}
\]
that returns \text{True} if and only if all of the elements of the list are equal. Your
definition should be of the form

\[
same \text{ xs } = \text{ and (zipWith } ____ \text{ xs (tail xs))}
\]
To get an idea of what would be in the above blank, compute

\[
\text{zip } [1..5] (\text{tail } [1..5])
\]

Problem 6: (16 points).
Write two functions

\[
squash, \text{squash'} :: (a \rightarrow a \rightarrow b) \rightarrow [a] \rightarrow [b]
\]
both of which apply a given function to adjacent elements of a list. E.g.

\[
squash \text{ f } [x_1, x_2, x_3, x_4]
\]
\[
\sim [f \ x_1 \ x_2, f \ x_2 \ x_3, f \ x_3 \ x_4]
\]
(a) (8 points) Fill in the blank in Implement \text{squash} using explicit recursion
and pattern matching.

(b) (8 points) Implement \text{squash'} using \text{zipWith}. (Hint: Use the trick of the
previous problem.)

Problem 7: (12 points).

Background:

\[
data \text{BTree } = \text{Empty } \mid \text{Branch } \text{Int } \text{BTree } \text{BTree}
\]
\[
\text{deriving (Show)}
\]
\[
\text{foldT } :: a \rightarrow (\text{Int } \rightarrow a \rightarrow a \rightarrow a) \rightarrow \text{BTree } \rightarrow a
\]
\[
\text{foldT emp brch Empty } = \text{ emp}
\]
\[
\text{foldT emp brch (Branch n tl tr)} = \text{ brch n vl vr}
\]
\[
\text{ where vl } = \text{ foldT emp brch tl}
\]
\[
vr = \text{ foldT emp brch tr}
\]
The function \text{foldT} amounts to a version of \text{foldr} for the \text{BTree} data type.
For example, here is a version of the \text{bcount} function from Homework 2 page
2 defined via \text{foldT}:

\[
\text{bcount t } = \text{foldT } 0 \text{ tally t}
\]
\[
\text{ where tally } _\text{numl numr } = 1 + \text{numl numr}
\]
Your task:

(a) Fill in the blank in:

\[
\text{bsum t } = \text{foldT } 0 \text{ f t}
\]
\[
\text{ where f n vl vr } = \text{______________}
\]
to define a function that sums all the \text{Int}-values in a \text{BTree}.

(b) Fill in the blanks in:

\[
\text{bmaxDepth'} t = \text{foldT } ____ g t
\]
\[
\text{ where g n vl vr } = \text{______________}
\]
to define a version of \text{bmaxDepth} from Problem 1 of Homework 2.

Side question: What is the problem with trying to define a version of \text{bleaves}
(Homework 2, problem 3) using \text{foldT}?
Testing

- **For isNNF and toNNF:**
  Run (quickCheck nnf1_prop) and (quickCheck nnf2_prop) and add a few tests of your own. Also check that you get the following translations:
  \[
  \begin{align*}
  \text{toNNF(Not } T) & \sim F \\
  \text{toNNF(Not } F) & \sim T \\
  \text{toNNF(Not (Not (Not (Var "P")))}) & \sim \sim P \\
  \text{toNNF(Not (Not (Not (Var "P")))}) & \sim P \\
  \text{toNNF((Not (Var "P")) ;\sim\sim; (Not (Var "Q"))}) & \sim ((P|\sim Q)&(Q|\sim P))
  \end{align*}
  \]

- **For isTidy and toTidy:**
  Run (quickCheck tidy1_prop) and (quickCheck tidy2_prop) and add a few tests of your own.

- **For squash and squash':**
  To test squash and squash' compute the same function, define the property squash_prop:
  \[
  \text{squash_pred xs = (squash (+) ys == squash' (+) ys) where ys = map ('mod' 1000) xs}
  \]
  Then run (quickCheck squash_prop). Also add a few tests of your own.

- **For same, bsum, and bmaxDepth':**
  Devise your own tests.

Administrivia

- **Use** [http://www.cis.syr.edu/courses/cis352/coursework/prop.hs](http://www.cis.syr.edu/courses/cis352/coursework/prop.hs) as a starter file for Part I.

- **For Part II, use a fresh file. You may want to copy some of import-statements from prop.hs to your fresh file.**

- **This assignment is a solo effort. If you trade ideas with another student, document it** in your coversheet file.

- **Let me know if any of my QuickCheck tests seem dodgy.**

- **To turn in the assignment (via Blackboard), include**
  \[(i)\] your coversheet,
  \[(ii)\] the source files,
  \[(iii)\] the transcripts of test runs.

Grading criteria

- **The homework is out of 100 points.**
- **Each problem is, roughly, 70% correctness and 30% testing.**
- **You get 5 points taken off for omitting your name in the source code.**

Credits:  Part I is based in part on a similar problem set by Willem Heijltjes and Phil Wadler. Problems 5 and 6 of Part II are based in part on a similar problem set by Tony Field.