Functional Programming	WS 2019/2020	LVA 703024+703025
Exercises 13		Deadline: January 29, 2020, 6am

- Please write all your Haskell functions from this exercise sheet into a single .hs-file and upload it in OLAT.
- You can use a template .hs-file that is provided on the proseminar page.
- The file should compile with ghci.
- Once the file has been uploaded, it cannot be changed or resubmitted!

Exercise 13.1 Bonus: *Formula Explorer* **5 p.**

In this exercise we want to build a small, interactive program for working with propositional logic formulas[1]. Since this is a bonus exercise sheet, we will provide slightly less guidance, in particular there are no test functions. For some subtasks you *might* need to slightly adjust the given type signature, and you are allowed to modify the `Formula` datatype to some extent (e.g. you may add a **`deriving`** statement if needed, but you may not remove any constructors). In this case explain why. Writing additional auxiliary functions to complete the task is also fine.

1. We want to be able to let the user simplify a formula, so given `Formula a` from the template file, write a function `simp :: Formula a -> Formula a` which simplifies a formula recursively according to (at least) the following rules:

$$\begin{array}{ccccc} \top \wedge x \longrightarrow x & x \wedge \top \longrightarrow x & \top \vee x \longrightarrow \top & x \vee \top \longrightarrow \top & \neg\top \longrightarrow \bot \\ \bot \wedge x \longrightarrow \bot & x \wedge \bot \longrightarrow \bot & \bot \vee x \longrightarrow x & x \vee \bot \longrightarrow x & \neg\bot \longrightarrow \top \\ \neg\neg x \longrightarrow x & \neg(x \wedge y) \longrightarrow \neg x \vee \neg y & \neg(x \vee y) \longrightarrow \neg x \wedge \neg y & & \end{array}$$

   (1 point)

2. The user should be able to fix the assignment[2] of a specific variable (that is, make it `True` or `False`). Write a function `substitute :: Formula a -> a -> Bool -> Formula a` which takes a formula, a variable identifier and a `Bool` and replaces all occurences of the variable in the formula with $\top$ or $\bot$ respectively. (1 point)

3. In order to let the user supply a formula themselves, write a function `parseF :: IO (Formula a)` which reads a formula from `IO`. You may choose whichever input format you want.

   Hint: There is a solution that does not require you to write a parser. (1 point)

4. We now want to get a user interface, and some interaction. For this purpose write a function `loop` following this specification:
   - Print the current formula.
   - Ask the user what to do next. At least provide the following options:
     - Simplify the formula.
     - Add a substitution by setting a specific variable to `True` or `False`.

[1] see *Aussagenlogik* as explained in the Introduction to Theoretical Computer Science course `http://cl-informatik.uibk.ac.at/teaching/ws19/eti/ohp/2-beamer.pdf` or for instance `https://en.wikipedia.org/wiki/Propositional_calculus`

[2] see *Wahrheitswertbelegung* in the previously linked lecture slides or for instance `https://en.wikipedia.org/wiki/Valuation_(logic)`

  - Print the current substitution.
  - Reset the formula and substitution to their initial values.
  - Quit the interactive loop.
- Repeat.

The user interface might look like this:

```
Formula:  Con (Dis Bot (Con Top (Atom "x"))) (Con Top (Dis Bot Bot))
How to proceed?
(s)implify formula  (r)eset formula
fix (a)ssignment  (p)rint assignment
(q)uit
a
Give an assignment (e.g.: (x,True)), where x denotes the atom:
("x",False)

Formula:  Con (Dis Bot (Con Top Bot)) (Con Top (Dis Bot Bot))
How to proceed?
(s)implify formula  (r)eset formula
fix (a)ssignment  (p)rint assignment
(q)uit
s

Formula:  Bot
How to proceed?
(s)implify formula  (r)eset formula
fix (a)ssignment  (p)rint assignment
(q)uit
```

Make sure to check validity of user input for operations. (2 points)