CMPT 383 Comparative Programming Languages

Programming Assignment 1

This assignment is due by 11:59pm PT on Wednesday Feb 16, 2022. Please submit it to Canvas.

Requirements:

- This assignment must be your own work. No collaboration is permitted.
- You can only use library functions from the following modules: `Prelude`, `System.IO`, `System.Environment`, `Data.Map.Strict`. Detailed information of modules can be found on `https://hoogle.haskell.org`

Late policy:
Suppose you can get n (out of 10) points based on your code and report

- If you submit before the deadline, you can get all n points.
- If you submit between 11:59pm PT Feb 16 and 11:59pm PT Feb 17, you get $n-1$ points.
- If you submit between 11:59pm PT Feb 17 and 11:59pm PT Feb 18, you get $n-2$ points.
- If you submit after 11:59pm PT Feb 18, you get 0 points.

(10 points) A formula in propositional logic can be a boolean constant (`Const`) with value `True` or `False`, a boolean variable (`Var`) such as $x_1, x_2, \ldots$, or the composition of formulas using logic connectives $\neg$ (`Not`), $\wedge$ (`And`), $\vee$ (`Or`), $\rightarrow$ (`Imply`), and $\leftrightarrow$ (`Iff`).

For a formula ϕ, a variable assignment is a mapping that maps each variable in ϕ to a truth value in {`True`, `False`}. Given a formula ϕ and a variable assignment, the formula ϕ evaluates to a truth value based on the following truth tables (where T stands for `True` and F stands for `False`).

ϕ_1	$\neg\phi_1$
T	F
F	T

(a) `Not`

ϕ_1	ϕ_2	$\phi_1 \wedge \phi_2$
T	T	T
T	F	F
F	T	F
F	F	F

(b) `And`

ϕ_1	ϕ_2	$\phi_1 \vee \phi_2$
T	T	T
T	F	T
F	T	T
F	F	F

(c) `Or`

ϕ_1	ϕ_2	$\phi_1 \rightarrow \phi_2$
T	T	T
T	F	F
F	T	T
F	F	T

(d) `Imply`

ϕ_1	ϕ_2	$\phi_1 \leftrightarrow \phi_2$
T	T	T
T	F	F
F	T	F
F	F	T

(e) `Iff`

For example, consider a concrete formula ϕ being $x_1 \wedge \neg x_2$. ϕ evaluates to `True` if the variable assignment is $x_1 =$ `True` and $x_2 =$ `False`. Also, ϕ evaluates to `False` if the variable assignment is $x_1 =$ `True` and $x_2 =$ `True`.

A formula ϕ is said to be *satisfiable* if there exists a variable assignment under which ϕ evaluates to `True`. Otherwise, the formula is said to be *unsatisfiable*. In general, the satisfiability of a formula can be checked using the truth table method. Specifically, we can list all possible variable assignments of a formula, and then check if any variable assignment can make the formula evaluate to `True`.

For example, check the satisfiability of $x_1 \wedge \neg x_2$.

x_1	x_2	$\neg x_2$	$x_1 \wedge \neg x_2$
T	T	F	F
T	F	T	T
F	T	F	F
F	F	T	F

Here, $x_1 \wedge \neg x_2$ is satisfiable because there exists a variable assignment $x_1 = T$ and $x_2 = F$ under which the formula evaluates to T.

As another example, check the satisfiability of $\neg(x_1 \rightarrow (x_2 \rightarrow x_1))$.

x_1	x_2	$x_2 \rightarrow x_1$	$x_1 \rightarrow (x_2 \rightarrow x_1)$	$\neg(x_1 \rightarrow (x_2 \rightarrow x_1))$
T	T	T	T	F
T	F	T	T	F
F	T	F	T	F
F	F	T	T	F

Here, $\neg(x_1 \rightarrow (x_2 \rightarrow x_1))$ is unsatisfiable, because there is no variable assignment that can make the formula evaluate to T.

You need to write a Haskell program to check the satisfiability of formulas in propositional logic. The program must be in a form that GHC can compile (i.e., you need a `main`). It needs to take one command-line argument that denotes the path to the formula file. You can assume each line of the file contains a formula to check, and the program needs to print to the console telling whether each formula is satisfiable (print `SAT`) or not (print `UNSAT`).

Sample Input and Output

Suppose we have a formula file called `formulas.txt` that contains the following two lines:

```
(And (Var "x1") (Not (Var "x2")))
(Not (Imply (Var "x1") (Imply (Var "x2") (Var "x1"))))
```

After compiling, we can run the executable and get

```
$ ./p1_x_x formulas.txt
SAT
UNSAT
```

Deliverable

A zip file called `p1_firstname_lastname.zip` that contains at least the followings:

- A file called `p1_firstname_lastname.hs` that contains the source code of your Haskell program. You can have multiple source files if you want, but this file must contain the `main`.
- A report called `p1_firstname_lastname.pdf` that explains the design choices, features, issues (if any), and anything else that you want to explain of your program.