Advanced Programming Languages and Techniques

May 23, 2024

Guidelines

- Submit a single compressed file `tar.gz` file containing all your solutions.
- Inside the compressed file include a separate file for each question of the assignment. Name your Haskell files with the name of the function in question.
- You can use Haskell's standard library if you want.

Assignment 2

1. Define the function `cnf :: LogicExpr -> LogicExpr` that converts any logical expression to its equivalent conjunctive normal form. A logical expression is a data type defined as:

```haskell
data LogicExpr = Var String
               | Or LogicExpr LogicExpr
               | And LogicExpr LogicExpr
               | Not LogicExpr
               deriving Show
```

   For example, the expression $a \wedge (\neg a \vee b)$ is represented as

```haskell
(And (Var "a") (Or (Not (Var "a")) (Var "b")))
```

   An expression is in conjunctive normal form if it is a conjunction of disjunctions of literals, where literals are either variables or negated variables. For example, the expression $\neg(a \wedge b)$ is not in conjunctive normal form since a conjunction is nested within a negation. However, every expression can be converted into an equivalent expression that is in conjunctive normal form. The algorithm is simple:

   - push all negations close to variables using De-Morgan laws; i.e., $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ and $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$,
   - then, use distributive laws of $\wedge$ and $\vee$; i.e., $(A \wedge B) \vee C) \equiv (A \vee C) \wedge (A \vee B)$ and $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$.

   For example, the following call to `cnf` should produce:

```
ghci> cnf (Not (And (Var "a") (Var "b"))
(Or (Not (Var "a")) (Not (Var "b")))
```

2. Define a function, `truthTable` which returns the truth table of a given logical expression. For example, the call

   ```
   truthTable (And (Var "a") (Or (Var "a") (Var "b")))
   ```

   must compute the value of $a \wedge (a \vee b)$ for all possible truth assignments (true or false) of a and b. The type of the function `truthTable` is

   ```
   truthTable :: LogicExpr -> TruthTable
   ```

   where

   ```
   type Binding    = [(String,Bool)]
   type TruthTable = [(Binding,Bool)]
   ```

   The aforementioned example should return:

   ```
   ghci> truthTable (And (Var "a") (Or (Var "a") (Var "b")))
   [([("a", True),  ("b", True) ], True),
    ([("a", True),  ("b", False)], True),
    ([("a", False), ("b", True) ], False),
    ([("a", False), ("b", False)], False)]
   ```