Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH08-320101
Date: 2018-10-18
Due: 2018-10-25

ICS 2018 Problem Sheet #6

Problem 6.1: *completeness of $\rightarrow$ and $\neg$* (2 points)

Proof that the two elementary boolean functions $\rightarrow$ (implication) and $\neg$ (negation) are universal, i.e., they are sufficient to express all possible boolean functions.

Problem 6.2: *conjunctive and disjunctive normal form* (2+1+3 = 6 points)

Consider the following boolean formula:

$$\varphi(P, Q, R, S) = (\neg P \lor Q) \land (\neg Q \lor R) \land (\neg R \lor S) \land (\neg S \lor P)$$

a) How many interpretations of the variables P, Q, R, S satisfy φ? Provide a proof for your answer.

b) Given the interpretations that satisfy φ, write the formula for φ in disjunctive normal form (DNF).

c) Using the equivalence laws for boolean expressions, derive the DNF representation of φ algebraically from the CNF representation. Write the derivation down step wise.

Problem 6.3: *boolean expressions (haskell)* (2 points)

Boolean expressions can be represented in Haskell as shown below:

```haskell
{- |
   Module: p6-boolexpr/boolexpr.hs

-}

module BoolExpr (Variable, BoolExpr(..), evaluate) where

type Variable = Char

data BoolExpr
   = T
   | F
   | Var Variable
   | Not BoolExpr
   | And BoolExpr BoolExpr
   | Or  BoolExpr BoolExpr
   deriving (Eq, Ord, Show)

-- evaluates an expression
evaluate :: BoolExpr -> [Variable] -> Bool
evaluate T ts            = True
evaluate F ts            = False
evaluate (Var v) ts      = elem v ts
evaluate (Not e) ts      = not (evaluate e ts)
evaluate (And e1 e2) ts = evaluate e1 ts && evaluate e2 ts
evaluate (Or  e1 e2) ts = evaluate e1 ts || evaluate e2 ts
```

You can evaluate a boolean expression as follows:

```
> evaluate (And (Var 'a') (Var 'b')) "ab"
True
> evaluate (And (Var 'a') (Var 'b')) "a"
False
```

The first argument is the boolean expression and the second argument is the set of variables that are true. (Variables that do not exist are assumed to be false.)

a) Implement a function `variables :: BoolExpr -> [Variable]`, which returns the list of variables that appear in a boolean expression. Feel free to use the Haskell `union` function to ensure that there are no duplicates in the list and the Haskell `sort` function (defind in `Data.List`) to ensure the variables are returned in a defined order.

```
> variables T
""
> variables (Or T F)
""
> variables (Var 'a')
"a"
> variables (And (Var 'a') (Or (Var 'c') (Var 'b')))
"abc"
> variables (And (Var 'a') (Or (Var 'a') (Var 'a')))
"a"
```

b) Implement a function `subsets :: [Variable] -> [[Variable]]`, which returns all subsets of the set of variables passed to the function. Use this function to implement `truthtable :: BoolExpr -> [([Variable], Bool)]`, which returns the entire truth table.

```
> subsets "abc"
["","c","b","bc","a","ac","ab","abc"]
> truthtable (And (Var 'a') (Or (Var 'c') (Var 'b')))
[("",False),("c",False),("b",False),("bc",False),("a",False),("ac",True),("ab",True),("abc",Tru
```

Submit your Haskell code plus an explanation (in Haskell comments) as a plain text file.