Assignment #6

Student name: *Ivan Kabadzhov*

Course: *Introduction to Computer Science*
Date: *October, 2018*

Problem 6.1: completeness of $\rightarrow$ and $\neg$

Claim: implication and negation are universal

Proof: *Proof by Construction*
It is enough to express an universal elementary boolean function(NOR/NAND gate) with negation and implication to guarantee that they are universal. Manually check that:

p	q	$p \rightarrow q$	$\neg p$	$\neg q$	$p \downarrow q$	$p \uparrow q$
F	F	T	T	T	T	T
F	T	T	T	F	F	T
T	F	F	F	T	F	T
T	T	T	F	F	F	F

Approach 1: From the table deduce that $(p \downarrow q) \equiv (\neg((\neg p) \rightarrow q))$.
Approach 2: From the table deduce that $(p \uparrow q) \equiv (p \rightarrow (\neg q))$.

■

Problem 6.2: conjunctive and disjunctive normal form

a) $\varphi(P, Q, R, S) = (\neg P \vee Q) \wedge (\neg Q \vee R) \wedge (\neg R \vee S) \wedge (\neg S \vee P)$

P	Q	R	S	$\neg P \vee Q$	$\neg Q \vee R$	$\neg R \vee S$	$\neg S \vee P$	φ
F	F	F	F	T	T	T	T	T
F	F	F	T	T	T	T	F	F
F	F	T	F	T	T	F	T	F
F	F	T	T	T	T	T	F	F
F	T	F	F	T	F	T	T	F
F	T	F	T	T	F	T	F	F
F	T	T	F	T	T	F	T	F
F	T	T	T	T	T	T	F	F
T	F	F	F	F	T	T	T	F
T	F	F	T	F	T	T	T	F
T	F	T	F	F	T	F	T	F
T	F	T	T	F	T	T	T	F
T	T	F	F	T	F	T	T	F
T	T	F	T	T	F	T	T	F
T	T	T	F	T	T	F	T	F
T	T	T	T	T	T	T	T	T

$\Rightarrow$ Only 2 interpretations satisfy φ.

b) DNF $= (P \land Q \land R \land S) \lor (\neg P \land \neg Q \land \neg R \land \neg S)$

c) CNF $= (\neg P \lor Q) \land (\neg Q \lor R) \land (\neg R \lor S) \land (\neg S \lor P)$

$$
\begin{aligned}
&\xRightarrow{\text{distributive law}} ((\neg P \land \neg Q) \lor (\neg P \land R) \lor (\neg Q \land Q) \lor (Q \land \neg R)) \land \\
&\qquad \land ((\neg R \land \neg S) \lor (\neg R \land P) \lor (S \land \neg S) \lor (S \land \neg P)) \\
&\xRightarrow{\text{contradiction}} ((\neg P \land \neg Q) \lor (\neg P \land R) \lor (Q \land \neg R)) \land \\
&\qquad \land ((\neg R \land \neg S) \lor (\neg R \land P) \lor (S \land \neg P)) \\
&\xRightarrow{\text{distributive law}} ((\neg P \land \neg Q) \land (\neg R \land \neg S)) \lor ((\neg P \land R) \land (\neg R \land \neg S)) \lor \\
&\qquad \lor ((Q \land R) \land (\neg R \land S)) \lor ((\neg P \land \neg Q) \land (\neg R \land P)) \lor \\
&\qquad \lor ((\neg P \land R) \land (\neg R \land P)) \lor ((Q \land R) \land (\neg R \land P)) \lor \\
&\qquad \lor ((\neg P \land \neg Q) \land (S \land P)) \lor ((\neg P \land R) \land (S \land P)) \lor \\
&\qquad \lor ((Q \land R) \land (S \land P)) \\
&\xRightarrow{\text{associativity}} (\neg P \land \neg Q \land \neg R \land \neg S) \lor (\neg P \land R \land \neg R \land \neg S) \lor (Q \land R \land \neg R \land S) \lor \\
&\qquad \lor (\neg P \land \neg Q \land \neg R \land P) \lor (\neg P \land R \land \neg R \land P) \lor (Q \land R \land \neg R \land P) \lor \\
&\qquad \lor (\neg P \land \neg Q \land S \land P) \lor (\neg P \land R \land S \land P) \lor (Q \land R \land S \land P) \\
&\xRightarrow{\text{contradiction}} (\neg P \land \neg Q \land \neg R \land \neg S) \lor \mathrm{F} \lor \mathrm{F} \lor \mathrm{F} \lor \mathrm{F} \lor \mathrm{F} \lor \mathrm{F} \lor \mathrm{F} \lor (Q \land R \land S \land P) \\
&\xRightarrow{\text{identity}} (\neg P \land \neg Q \land \neg R \land \neg S) \lor (Q \land R \land S \land P) \\
&\xRightarrow{\text{commutativity}} (P \land Q \land R \land S) \lor (\neg P \land \neg Q \land \neg R \land \neg S)
\end{aligned}
$$

Problem 6.3: boolean expressions (haskell)

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

-}

module BoolExpr (Variable, BoolExpr(..), evaluate) where
import Data.List
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


-- 6.3.a.
variables :: BoolExpr -> [Variable]
variables T = "" --following the example on the sheet
variables F = "" --following the example on the sheet
variables (Var x) = [x]
variables (Not y) = variables y --declare the expression condition
variables (And a b) = sort(variables a `union` variables b)
variables (Or a b) = sort(variables a `union` variables b)

-- 6.3.b.
subsets :: [Variable] -> [[Variable]]
subsets [] = [[]]--similiar to the prefixes and suffixes
subsets (head:tail) = subsets(tail) ++ map (head:)(subsets(tail))

truthtable :: BoolExpr -> [([Variable], Bool)]
truthtable n = zip (subsets(variables n))(map (evaluate(n)) (subsets(variables n)))
-- the zip relates tuples with elements in the same position for the 2 lists
```