Introduction to Computer Science
Constructor University
Dr. Jürgen Schönwälder

Module: CH-232
Date: 2023-10-20
Due: 2023-10-27

Problem Sheet #7

Problem 7.1: *not-or is a universal boolean function* (3 points)

Prove that not-or ($\triangledown$) is a universal boolean function by showing how $\triangledown$ functions can implement the classic universal Boolean functions $\wedge, \vee, \neg$.

Problem 7.2: *simplify a boolean expression using algebraic equivalence laws* (4 points)

During our class meeting, we discussed the following boolean function:

$$F(X,Y,Z) = (((X \wedge Y) \vee (X \wedge \neg Z)) \vee (Z \wedge \neg 0))$$

Using a truth table, we found that F is equivalent to G:

$$G(X,Y,Z) = (X \vee Z)$$

By applying boolean equivalence laws, show that the boolean expression defining F can be transformed into the boolean expression defining G. In each step of your derivation, identify which boolean equivalence law you apply.

Problem 7.3: *munged passwords (haskell)* (1+1+1 = 3 points)

Some people try to create stronger passwords through character substitutions. The substitutions can be anything the user finds easy to remember. We use the following substitution:

character	a	b	c	d	e	f	g	h	i	l	o	q	s	x	y
substitution	@	8	(	6	3	{	9	#	1	!	0	2	$	%	?

Using this table, the string `hello world` is munged into the string `#3!!0 w0r!6`.

a) Write a collection of unit tests for the functions described below using the HUnit unit testing framework for Haskell.

b) Implement a function `encChar :: Char -> Char` receiving a character and returning either the character itself or a substitution of it. Implement another function `decChar :: Char -> Char` implementing the inverse of `encChar`.

c) Implement a function `enc :: [Char] -> [Char]` receiving a string and returning a string with all character substitutions applied. Implement another function `dec :: [Char] -> [Char]` implementing the inverse of `enc`.

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