Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CH-232
Date: 2021-10-15
Due: 2021-10-22

ICS 2021 Problem Sheet #6

Problem 6.1: *long life diet rules* (1+3+1+3 = 8 points)

Uwe Schönig published in his book "Logic for Computer Scientists" the following little story:

> A 100 year old was asked "What is the secret of having a long life?" and he did respond to stick to three diet rules:
>
> 1. If you don't take beer, you must have fish.
> 2. If you have both beer and fish, don't have ice cream.
> 3. If you have ice cream or don't have beer, then don't have fish.
>
> The questioner found this answer quite confusing. Can you help to simplify the answer?

We introduce three boolean variables: The variable B is true if we have beer, the variable F is true if we have fish, and the variable I is true if we have ice cream.

a) Provide a boolean formula for the function $D(B, F, I)$ that captures the three rules.

b) Construct a truth table that shows all interpretations of $D(B, F, I)$. Break things into meaningful steps so that we can award partial points in case things go wrong somewhere.

c) Out of the truth table, derive a simpler boolean formula defining $D(B, F, I)$.

d) Take the boolean formula from a) and algebraically derive the simpler boolean formula from c). Annotate each step of your derivation with the equivalence law that you apply so that we can follow along.

Solution:

a) A direct translation yields:

$$D(B, F, I) = (\neg B \to F) \land ((B \land F) \to \neg I) \land ((I \lor \neg B) \to \neg F)$$

Translating the implications yields ($X \to Y$:

$$D(B, F, I) = (B \lor F) \land (\neg(B \land F) \lor \neg I) \land (\neg(I \lor \neg B) \lor \neg F)$$

b) Construction of the truth table:

B	F	I	$\neg B \to F$	$B \land F$	$(B \land F) \to \neg I$	$I \lor \neg B$	$(I \lor \neg B) \to \neg F$	$D(B, F, I)$
0	0	0	0	0	1	1	1	0
0	0	1	0	0	1	1	1	0
0	1	0	1	0	1	1	0	0
0	1	1	1	0	1	1	0	0
1	0	0	1	0	1	0	1	1
1	0	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1
1	1	1	1	1	0	1	0	0

c) There are several options for simpler Boolean formulas:

$$\begin{aligned} D(B, F, I) &= (B \land \neg F \land \neg I) \lor (B \land \neg F \land I) \lor (B \land F \land \neg I) \qquad \text{DNF} \\ &= B \land \neg(I \land F) \\ &= (B \land \neg I) \lor (B \land \neg F) \end{aligned}$$

d) Derivation:

$$\begin{aligned}
D(B,F,I) &= (\neg B \rightarrow F) \wedge ((B \wedge F) \rightarrow \neg I) \wedge ((I \vee \neg B) \rightarrow \neg F) && \text{apply } X \rightarrow Y = \neg X \vee Y \\
&= (B \vee F) \wedge (\neg(B \wedge F) \vee \neg I) \wedge (\neg(I \vee \neg B) \vee \neg F) && \text{de Morgan (twice)} \\
&= (B \vee F) \wedge ((\neg B \vee \neg F) \vee \neg I) \wedge (\neg I \wedge \neg\neg B) \vee \neg F) && \text{double negation} \\
&= (B \vee F) \wedge ((\neg B \vee \neg F) \vee \neg I) \wedge (\neg I \wedge B) \vee \neg F) && \text{distributivity} \\
&= (B \vee F) \wedge ((\neg B \vee \neg F) \vee \neg I) \wedge (\neg I \vee \neg F) \wedge (B \vee \neg F) && \text{associativity} \\
&= (B \vee F) \wedge (\neg B \vee (\neg F \vee \neg I)) \wedge (\neg I \vee \neg F) \wedge (B \vee \neg F) && \text{absorption} \\
&= (B \vee F) \wedge (\neg I \vee \neg F) \wedge (B \vee \neg F) && \text{commutativity} \\
&= (B \vee F) \wedge (B \vee \neg F) \wedge (\neg I \vee \neg F) && \text{distributivity} \\
&= (B \vee (F \wedge \neg F)) \wedge (\neg I \vee \neg F) && \text{complementation} \\
&= (B \vee 0) \wedge (\neg I \vee \neg F) && \text{identity} \\
&= B \wedge (\neg I \vee \neg F) && \text{de Morgan} \\
&= B \wedge \neg(I \wedge F)
\end{aligned}$$

Marking:

a) - *1pt for a proper translation of the text into a boolean formula*
b) - *1pt for a column defining D correctly*
- *0.5pt for a column defining each of the implications*
- *0.5pt for setting up the table correctly*

c) - *1pt for a simpler correct boolean formula*
d) - *3pt for a complete deviation, subtract points for errors or incomplete solutions*

Problem 6.2: *long life diet rules* (1+1 = 2 points)

Uwe Schönig published in his book “Logic for Computer Scientists” the following little story:

> A 100 year old was asked “What is the secret of having a long life?” and he did respond to stick to three diet rules:
>
> 1. If you don’t take beer, you must have fish.
> 2. If you have both beer and fish, don’t have ice cream.
> 3. If you have ice cream or don’t have beer, then don’t have fish.
>
> The questioner found this answer quite confusing. Can you help to simplify the answer?

We introduce three boolean variables: The variable B is true if we have beer, the variable F is true if we have fish, and the variable I is true if we have ice cream.

a) Define a function

```
diet :: Bool -> Bool -> Bool -> Bool
```

that implements the three diet rules directly following the description given above and a function

```
diet’ :: Bool -> Bool -> Bool -> Bool
```

that implements a simplified diet formula.

b) Define a function

```
truthTable :: (Bool -> Bool -> Bool -> Bool) -> [(Bool, Bool, Bool, Bool)]
```

takes a (diet) function as an argument and returns a list where each element is a 4-tuple representing three input values passed to the (diet) function followed by the function’s result.

Submit your Haskell code as a plain text file.

Below is a unit test template that you can use to fill in your code.

```haskell
module Main (main) where

import Test.HUnit

-- The function diet implements the three diet rules directly.
diet :: Bool -> Bool -> Bool -> Bool
diet b f i = undefined

-- The function diet' implements the simplified diet formula.
diet' :: Bool -> Bool -> Bool -> Bool
diet' b f i = undefined

-- The function truthTable takes a function as an argument and returns
-- a list where each element is a 4-tuple representing three input
-- value passed to the function followed by the function's result.
truthTable :: (Bool -> Bool -> Bool -> Bool) -> [(Bool, Bool, Bool, Bool)]
truthTable f = undefined

-- Test whether the two truth tables returned are the same (which is
-- not a very sharp test but I do not want to reveal too many details).
-- You may want to add your own test cases...
dietTests = TestList [ truthTable diet ~?=  truthTable diet' ]

main = runTestTT dietTests
```

Solution:

A possible solution for both parts:

```haskell
module Main (main) where

import Test.HUnit

-- The function diet implements the three diet rules directly.
diet :: Bool -> Bool -> Bool -> Bool
diet b f i = r1 && r2 && r3
    where r1 = not b --> f
          r2 = (b && f) --> not i
          r3 = (i || not b) --> not f
          (-->) x y = not x || y

-- The function diet' implements the simplified diet formula.
diet' :: Bool -> Bool -> Bool -> Bool
diet' b f i = b && not (i && f)

-- The function truthTable takes a function as an argument and returns
-- a list where each element is a 4-tuple representing three input
-- values passed to the function followed by the function's result.
truthTable :: (Bool -> Bool -> Bool -> Bool) -> [(Bool, Bool, Bool, Bool)]
truthTable f = [ (x, y, z, f x y z) | x <- t, y <- t, z <- t ]
    where t = [False, True]

-- Test whether the two truth tables returned are the same (which is
-- not a very sharp test but I do not want to reveal too many details).
-- You may want to add your own test cases...
dietTests = TestList [ truthTable diet ~?= truthTable diet' ]
```

```haskell

main = runTestTT dietTests
```

Marking:

a)	- *0.5pt for a proper `diet` function*
	- *0.5pt for a proper `diet'` function*
b)	- *0.5pt for creating all possible truthtable inputs*
	- *0.5pt for creating the proper function outputs*