School of Computing and Information Systems
COMP30026 Models of Computation Problem Set 2

Content: types, Haskell, propositional logic, truth tables, validity & satisfiability

P2.1 In Haskell, the product of two types `a` and `b` is a type written `(a,b)`, and its inhabitants are pairs of the form `(x,y)`, where `x :: a` and `y :: b`. Given any function `f :: (a,b) -> c`, there is an "equivalent" *curried* function `cf :: a -> (b -> c)`, with the property that

```
(cf x) y == f (x,y)
```

for any `x :: a` and `y :: b`. Notice the different ways these functions accept their arguments. Write a function which takes a function like `f` as input and produces its *curried* version. Then write a function that *un-curries* a function like `cf`.

P2.2 What is the type of `f` defined below? Is it well-typed? Did somebody forget the square brackets in the last equation? Explain the function's behaviour in English.

```
f []  = 0
f [x] = x
f y   = 42
```


P2.3 For each of the followin e the same truth table.

(a) $\neg P \Rightarrow Q$ a $Q \Rightarrow (P \Rightarrow R)$

(b) $\neg P \Rightarrow Q$ a $P \Rightarrow Q) \Rightarrow R$

(c) $\neg P \Rightarrow Q$ and $\neg Q \Rightarrow P$ (g $(Q \Rightarrow R)$

(d) $(P \Rightarrow Q) \Rightarrow P$ and P (h $\Rightarrow R) \wedge (Q \Rightarrow R)$

P2.4 Define your own binary connective $\Box$ by writing o $P \Box Q$ (fill in the middle column however you like). Can you write a formula with the same truth table using only $P, Q, \neg, \wedge, \vee, \Rightarrow$? Repeat this exercise a few times.

P2.5 How many distinct truth tables are there involving two fixed propositional letters? In other words, how many meaningfully distinct connectives could we have defined in the previous question?

P2.6 Find a formula that is equivalent to $(P \wedge \neg Q) \vee P$ but simpler, that is, using fewer symbols.

P2.7 Find a formula that is equivalent to $P \Leftrightarrow (P \wedge Q)$ but simpler, that is, using fewer symbols.

P2.8 Find a formula that is equivalent to $(\neg P \vee Q) \wedge R$ using only $\Rightarrow$ and $\neg$ as logical connectives.

P2.9 Recall that $\oplus$ is the "exclusive or" connective. Show that $(P \oplus Q) \oplus Q$ is logically equivalent to P.

P2.10 Consider the formula $P \Rightarrow \neg P$. Is that a contradiction (is it *unsatisfiable*)? Can a proposition imply its own negation?

P2.11 By negating a satisfiable proposition, can you get a tautology? A satisfiable proposition? A contradiction? Illustrate your affirmative answers.

P2.12 For each of the following propositional formulas, determine whether it is satisfiable, and if it is, whether it is a tautology:

(a) $P \Leftrightarrow ((P \Rightarrow Q) \Rightarrow P)$

(b) $(P \Rightarrow \neg Q) \wedge ((P \vee Q) \Rightarrow P)$

(c) $((P \Rightarrow Q) \Rightarrow Q) \wedge (Q \oplus (P \Rightarrow Q))$

P2.13 Complete the following sentences, using the words "satisfiable, valid, non-valid, unsatisfiable".

(a) F is satisfiable iff F is not ______

(b) F is valid iff F is not ______

(c) F is non-valid iff F is not ______

(d) F is unsatisfiable iff F is not ______

(e) F is satisfiable iff $\neg F$ is ______

(f) F is valid iff $\neg F$ is ______

(g) F is non-valid iff $\neg F$ is ______

(h) F is unsatisfiable iff $\neg F$ is ______

P2.14 Show that $P \Leftrightarrow (Q \Leftrightarrow R) \equiv (P \Leftrightarrow Q) \Leftrightarrow R$. This tells us that we could instead write

$$P \Leftrightarrow Q \Leftrightarrow R \tag{1}$$

without introducing any ambiguity. Mind you, that may not be such a good idea, because many people (incorrectly) tend to read "$P \Leftrightarrow Q \Leftrightarrow R$" as

$$P, Q, \text{ and } R \text{ all have the same truth value} \tag{2}$$

Show that (1) and (2) are incomparable, that is, neither is a logical consequence of the other.

P2.15 Let F and G be p ... $F \equiv G$' and '$F \Leftrightarrow G$'? Show that $F \Leftrightarrow$ ...

P2.16 Is this claim correct: $(P \wedge Q) \Leftrightarrow P$ is logically equiv ... hat is, do we have

$$((P \wedge Q) \Leftrightarrow P) \equiv (\ldots$$

P2.17 (*Challenge*) We can encode a matrix as a list of lists, where each list represents a row of the matrix. For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

would be represented by the list `[[1,2,3],[4,5,6],[7,8,9]] :: [[Int]]`. Write a function `mytranspose :: [[a]] -> [[a]]`, which *transposes* the matrix, i.e. swaps the rows and columns. For example, the transpose of the above matrix would be.

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

If you are familiar with matrix multiplication, use `mytranspose` to write a function

```
mmult :: [[Int]] -> [[Int]] -> [[Int]]
```

which multiplies two matrices (assume the inputs are sensible).