99 questions/46 to 50

From HaskellWiki

< 99 questions

This is part of Ninety-Nine Haskell Problems, based on Ninety-Nine Prolog Problems (https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/) .

1 Logic and Codes

2 Problem 46

(**) Define predicates and/2, or/2, nand/2, nor/2, xor/2, impl/2 and equ/2 (for logical equivalence) which succeed or fail according to the result of their respective operations; e.g. and(A,B) will succeed, if and only if both A and B succeed.

A logical expression in two variables can then be written as in the following example: and(or(A,B),nand(A,B)).

Now, write a predicate table/3 which prints the truth table of a given logical expression in two variables.

Example:

```
(table A B (and A (or A B)))
true true true
true fail true
fail true fail
fail fail fail
```

Example in Haskell:

```
> table (\a b -> (and' a (or' a b)))
True True True
True False True
False True False
False False False
```

Solutions

3 Problem 47

(*) Truth tables for logical expressions (2).

Continue problem P46 by defining and/2, or/2, etc as being operators. This allows to write the logical expression in the more natural way, as in the example: A and (A or not B). Define operator precedence as usual; i.e. as in Java.

Example:

```
* (table A B (A and (A or not B)))
true true true
true fail true
fail true fail
fail fail fail
```

Example in Haskell:

```
> table2 (\a b -> a `and'` (a `or'` not b))
True True True
True False True
False True False
False False False
```

Solutions

4 Problem 48

(**) Truth tables for logical expressions (3).

Generalize problem P47 in such a way that the logical expression may contain any number of logical variables. Define table/2 in a way that table(List,Expr) prints the truth table for the expression Expr, which contains the logical variables enumerated in List.

Example:

```
* (table (A,B,C) (A and (B or C) equ A and B or A and C))
true true true true
true true fail true
true fail true true
true fail fail true
fail true true true
fail true fail true
fail fail true true
fail fail fail true
```

Example in Haskell:

```
> tablen 3 (\[a,b,c] -> a `and'` (b `or'` c) `equ'` a `and'` b `or'` a `and'` c)
-- infixl 3 `equ'`
True  True  True  True
True  True  False True
```

```
True  False True  True
True  False False True
False True  True  True
False True  False True
False False True  True
False False False True

-- infixl 7 `equ'`
True  True  True  True
True  True  False True
True  False True  True
True  False False False
False True  True  False
False True  False False
False False True  False
False False False False
```

Solutions

5 Problem 49

(**) Gray codes.

An n-bit Gray code is a sequence of n-bit strings constructed according to certain rules. For example,

```
n = 1: C(1) = ['0','1'].
n = 2: C(2) = ['00','01','11','10'].
n = 3: C(3) = ['000','001','011','010',´110´,´111´,´101´,´100´].
```

Find out the construction rules and write a predicate with the following specification:

```
% gray(N,C) :- C is the N-bit Gray code
```

Can you apply the method of "result caching" in order to make the predicate more efficient, when it is to be used repeatedly?

Example in Haskell:

```
P49> gray 3
["000","001","011","010","110","111","101","100"]
```

Solutions

6 Problem 50

(***) Huffman codes.

We suppose a set of symbols with their frequencies, given as a list of fr(S,F) terms. Example: [fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)]. Our objective is to construct a list hc(S,C) terms, where C is the Huffman code word for the symbol S. In our example, the result could be Hs = [hc(a,'0'), hc(b,'101'), hc(c,'100'), hc(d,'111'), hc(e,'1101'), hc(f,'1100')] [hc(a,'01'),...etc.]. The task shall be performed by the predicate huffman/2 defined as follows:

```
% huffman(Fs,Hs) :- Hs is the Huffman code table for the frequency table Fs
```

Example in Haskell:

```
*Exercises> huffman [('a',45),('b',13),('c',12),('d',16),('e',9),('f',5)]
[('a',"0"),('b',"101"),('c',"100"),('d',"111"),('e',"1101"),('f',"1100")]
```

Solutions

Category:

- Tutorials
