Combinatorics in Haskell

Data and Function

```
import Test.QuickCheck
  (quickCheck, quickCheckWith, stdArgs, maxSize,
   (==>), Property)
import Data.List (sort, (\\))

-- QuickCheck at a given size

sizeCheck n = quickCheckWith (stdArgs {maxSize = n})
```

Nub, distinct, ascending

```
nub :: Eq a => [a] -> [a]
nub []      =  []
nub (x:xs)  =  x : nub [ y | y <- xs, x /= y ]

-- *Main> nub "avocado"
-- "avocd"
-- *Main> nub "peach"
-- "peach"

distinct :: Eq a => [a] -> Bool
distinct xs  =  xs == nub xs

-- *Main> distinct "avocado"
-- False
-- *Main> distinct "peach"
-- True

ascending :: Ord a => [a] -> Bool
ascending xs = and (zipWith (<=) xs (drop 1 xs))
```

All sublists of a list

```
sub :: Eq a => [a] -> [a] -> Bool
xs `sub` ys  =  and [ x `elem` ys | x <- xs ]

-- *Main> "pea" `sub` "apple"
-- True
-- *Main> "peach" `sub` "apple"
-- False

subs :: [a] -> [[a]]
subs []      =  [[]]
subs (x:xs)  =  subs xs ++ map (x:) (subs xs)

-- *Main> subs [0,1]
-- [[],[1],[0],[0,1]]
-- *Main> subs "abc"
-- ["","c","b","bc","a","ac","ab","abc"]

prop_subs :: [Int] -> Property
prop_subs xs =
  distinct xs ==>
    and [ ys `sub` xs | ys <- subs xs ]
    && distinct (subs xs)
    && all distinct (subs xs)
    && length (subs xs) == 2 ^ length xs

-- *Main> sizeCheck 10 prop_subs
-- +++ OK, passed 100 tests; 30 discarded.
-- (0.77 secs, 6,895,808 bytes)
```

Cartesian product

```
cp :: [[a]] -> [[a]]
cp []        =  [[]]
cp (xs:xss)  =  [ y:ys | y <- xs,
                         ys <- cp xss ]

-- *Main> cp ["ab","cd","efg"]
-- ["ace","acf","acg","ade","adf","adg",
--  "bce","bcf","bcg","bde","bdf","bdg"]

prop_cp :: [[Int]] -> Property
prop_cp xss =
  distinct (concat xss) ==>
    and [ and (zipWith elem ys xss) | ys <- cp xss ]
    && distinct (cp xss)
    && all distinct (cp xss)
    && all (\ys -> length ys == length xss) (cp xss)
    && length (cp xss) == product (map length xss)

-- *Main> sizeCheck 10 prop_cp
-- +++ OK, passed 100 tests; 130 discarded.
-- (1.35 secs, 6,627,416 bytes)
```

Permutations of a list

```haskell
permscp :: Eq a => [a] -> [[a]]
permscp xs | distinct xs =
  [ ys | ys <- cp (replicate (length xs) xs),
         distinct ys ]

-- *Main> permscp "abc"
-- ["abc","acb","bac","bca","cab","cba"]

prop_permscp :: [Int] -> Property
prop_permscp xs  =
  distinct xs ==>
    and [ sort ys == sort xs | ys <- permscp xs ]
    && distinct (permscp xs)
    && all distinct (permscp xs)
    && length (permscp xs) == fac (length xs)

-- *Main> sizeCheck 10 prop_permscp
-- +++ OK, passed 100 tests; 29 discarded.
-- (22.95 secs, 15,303,451,928 bytes)
```

Splitting a list

```haskell
splits :: [a] -> [(a, [a])]
splits xs  =
  [ (xs!!k, take k xs ++ drop (k+1) xs) | k <- [0..n-1] ]
  where
  n = length xs

-- *Main> splits "abc"
-- [('a',"bc"),('b',"ac"),('c',"ab")]

splits' :: [a] -> [(a, [a])]
splits' []      =  []
splits' (x:xs)  =  (x,xs) : [ (y,x:ys) | (y,ys) <- splits' xs ]

-- *Main> splits' "abc"
-- [('a',"bc"),('b',"ac"),('c',"ab")]

prop_splits :: [Int] -> Property
prop_splits xs =
  distinct xs ==>
    and [ sort (y:ys) == sort xs | (y,ys) <- splits xs ]
    && and [ 1 + length ys == length xs | (y,ys) <- splits xs ]
    && distinct (map snd (splits xs))
    && all distinct (map snd (splits xs))
    && length (splits xs) == length xs

-- *Main> quickCheck prop_splits
-- +++ OK, passed 100 tests; 234 discarded.
-- (1.98 secs, 45,225,448 bytes)

prop_splits_splits' :: [Int] -> Bool
prop_splits_splits' xs  =  splits xs == splits' xs

-- *Main> sizeCheck 10 prop_splits_splits'
-- +++ OK, passed 100 tests.
-- (0.26 secs, 1,905,952 bytes)
```

Permutations of a list

```
perms :: [a] -> [[a]]
perms []      =  [[]]
perms (x:xs)  =  [ y:zs | (y,ys) <- splits (x:xs),
                          zs <- perms ys ]

-- *Main> perms "abc"
-- ["abc","acb","bac","bca","cab","cba"]

fac :: Int -> Int
fac n | n >= 0  =  product [1..n]

prop_perms :: [Int] -> Property
prop_perms xs  =
  distinct xs ==>
    and [ sort ys == sort xs | ys <- perms xs ]
    && distinct (perms xs)
    && all distinct (perms xs)
    && length (perms xs) == fac (length xs)

-- *Main> sizeCheck 10 prop_perms
-- +++ OK, passed 100 tests; 36 discarded.
-- (10.86 secs, 7,609,507,616 bytes)

prop_perms_permscp :: [Int] -> Property
prop_perms_permscp xs =
  distinct xs ==> perms xs == permscp xs

-- *Main> sizeCheck 10 prop_perms_permscp
-- +++ OK, passed 100 tests; 26 discarded.
-- (86.36 secs, 64,316,219,480 bytes)
```

World's worst sorting algorithm

```
permsort :: Ord a => [a] -> [a]
permsort xs  =  head [ ys | ys <- perms xs, ascending ys ]

-- *Main> permsort [3,1,2,4]
-- [1,2,3,4]

prop_sort :: [Int] -> Bool
prop_sort xs  =  sort xs == permsort xs

-- *Main> sizeCheck 10 prop_sort
-- +++ OK, passed 100 tests.
-- (2.58 secs, 1,696,127,032 bytes)
```

Choose k elements from a list

Be Brave

here, k can be anything, but for [[ ]], k must be 0

K > n error!

```
choose :: Int -> [a] -> [[a]]
choose 0 []          =  [[]]
choose k (x:xs)
  | k == 0           =  [[]]
  | k == n           =  [x:xs]
  | 0 < k && k < n   =  choose k xs ++
                        map (x:) (choose (k-1) xs)
  where
  n = length (x:xs)
```

diffierance, shorter?

tail

(k-1) is that safe? Yes, k>0.

k is less than n, less or equal to (n-1) is worked!

```
-- *Main> choose 3 "abcde"
--["cde","bde","bce","bcd","ade","ace","acd","abe","abd","abc"]

prop_choose :: Int -> [Int] -> Property
prop_choose k xs =
  0 <= k && k <= n && distinct xs ==>
    and [ ys `sub` xs && length ys == k | ys <- choose k xs ]
    && distinct (choose k xs)
    && all distinct (choose k xs)
    && length (choose k xs) == fac n `div` (fac k * fac (n-k))
    where
    n = length xs

-- *Main> sizeCheck 10 prop_choose
-- +++ OK, passed 100 tests; 431 discarded.
-- (1.84 secs, 18,373,648 bytes)

prop_choose_subs :: [Int] -> Bool
prop_choose_subs xs  =
  sort (subs xs) ==
    sort [ ys | k <- [0..n], ys <- choose k xs ]
  where
  n = length xs

-- *Main> sizeCheck 10 prop_choose_subs
-- +++ OK, passed 100 tests.
-- (0.26 secs, 6,852,984 bytes)
```

All partitions of a given list

```
parts :: [a] -> [[[a]]]
parts []      =  [[]]
parts (x:xs)  =  [ [x]:yss | yss <- parts xs ] ++
                 [ (x:ys):yss | (ys:yss) <- parts xs ]

-- *Main> partitions "abcd"
-- [["a","b","c","d"],["a","b","cd"],
--  ["a","bc","d"],["a","bcd"],
--  ["ab","c","d"],["ab","cd"],
--  ["abc","d"],["abcd"]]

prop_parts :: [Int] -> Property
prop_parts xs =
  distinct xs ==>
    and [ concat yss == xs | yss <- parts xs ]
    && distinct (parts xs)
    && all distinct (parts xs)
    && all (all distinct) (parts xs)
    && length (parts xs) == 2 ^ ((length xs - 1) `max` 0)

-- *Main> sizeCheck 10 prop_parts
-- +++ OK, passed 100 tests; 25 discarded.
```

All partitions of a number

everything is positive!, Can u show me negtive ?

(all of the ∅)

True (also is negative!)

```
partitions :: Int -> [[Int]]
partitions 0           = [[]]
partitions n | n > 0   =  [ k : xs | k <- [1..n],
                                     xs <- partitions (n-k),
                                     all (k <=) xs ]

-- *Main> partitions 5
-- [[1,1,1,1,1],[1,1,1,2],[1,1,3],[1,2,2],[1,4],[2,3],[5]]

prop_partitions :: Int -> Property
prop_partitions n  =
  n >= 0 ==> all ((== n) . sum) (partitions n)
```

k ≤ 0 true !

```
-- *Main> sizeCheck 10 prop_partitions
-- +++ OK, passed 100 tests; 70 discarded.
-- (0.71 secs, 4,511,688 bytes)

prop_partitions' :: [Int] -> Property
prop_partitions' xs  =
  all (> 0) xs ==> sort xs `elem` partitions (sum xs)   ? order

-- *Main> sizeCheck 8 prop_partitions'
-- +++ OK, passed 100 tests; 131 discarded.
-- (2.51 secs, 30,097,560 bytes)
```

All ways to make change for a given amount

```
type Coin = Int
type Total = Int

change :: Total -> [Coin] -> [[Coin]]
change n xs  =  change' n (sort xs)
  where
  change' 0 xs          =  [[]]                 guard
  change' n xs | n > 0  =
    [ y : zs | (y, ys) <- nub (splits xs),
               y <= n,
               zs <- change' (n-y) (filter (y <=) ys) ]

-- *Main> change 30 [5,5,10,10,20]             DFA ?
-- [[5,5,10,10],[5,5,20],[10,20]]

prop_change :: Total -> [Coin] -> Property
prop_change n xs =
  0 <= n && all (0 <) xs ==>
    all ((== n) . sum) (change n xs)

-- *Main> sizeCheck 10 prop_change
-- +++ OK, passed 100 tests; 486 discarded.
-- (2.06 secs, 14,140,144 bytes)
```

Eight queens

```
type Row    =  Int
type Col    =  Int
type Coord  =  (Col, Row)
type Board  =  [Row]
```

```
queens :: Board -> [Board]
queens []  =  [[]]
queens ps | length ps > 0  =
  [ q:rs | (q,qs) <- splits ps,
           rs <- queens qs,
           and [ not (check (1,q) (x,y))
               | (x,y) <- zip [2..] rs ] ]

check :: Coord -> Coord -> Bool
check (x,y) (x',y')  =  abs (x-x') == abs (y-y')

-- *Main> head (queens [1..8])
-- [1,5,8,6,3,7,2,4]
-- (0.04 secs, 20,484,048 bytes)
-- *Main> length (queens [1..8])
-- 92
-- (0.47 secs, 306,502,280 bytes)
```

Eight queens, first attempt

回溯

```
queens' :: Col -> [Board]
queens' 0  =  [[]]
queens' n | n > 0  =
  [ q:qs | q <- [1..8],
           qs <- queens' (n-1),
           and [ not (check' (1,q) (x,y))
               | (x,y) <- zip [2..n] qs ] ]

check' :: Coord -> Coord -> Bool
check' (x,y) (x',y') =
  x == x' || y == y' || x+y == x'+y' || x-y == x'-y'

-- *Main> head (queens 8)
-- [1,5,8,6,3,7,2,4]
-- (9.00 secs, 6,382,153,528 bytes)
-- *Main> length (queens 8)
-- 92
-- (116.21 secs, 82,368,465,712 bytes)
```