Informatics 1

Introduction to Computation

Lectures 17–18

Combinatorial Algorithms

Don Sannella

University of Edinburgh

Part I

Preliminaries

Nub

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

-- > nub "avocado"
-- "avocd"
-- > nub "peach"
-- "peach"
```

Distinct

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

-- > distinct "avocado"
-- False
-- > distinct "peach"
-- True
```

QuickCheck with a bound on size

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

Part II

Sublists

Is a list a sublist of another list?

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

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

All sublists of a list

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

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

QuickCheck for sublists

```
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

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

Part III

Permutations

Select one element from a list

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

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

All permutations of a list

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

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

QuickCheck for permutations

```
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)

-- > sizeCheck 8 prop_perms
-- +++ OK, passed 100 tests; 21 discarded.
-- (2.41 secs, 235,561,416 bytes)
```

Part IV

Choose

Choose k elements from a list

```
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)

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

QuickCheck for choose

```
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

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

QuickCheck relating choose and subs

```
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

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

Part V

Partitions

All partitions of a given number

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

-- > partitions 5
-- [[1,1,1,1,1],[1,1,1,2],[1,1,3],[1,2,2],[1,4],[2,3],[5]]
```

QuickCheck for partitions

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

-- > 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)

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

Part VI

Change

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

-- > change 30 [5,5,10,10,20]
-- [[5,5,10,10],[5,5,20],[10,20]]
```

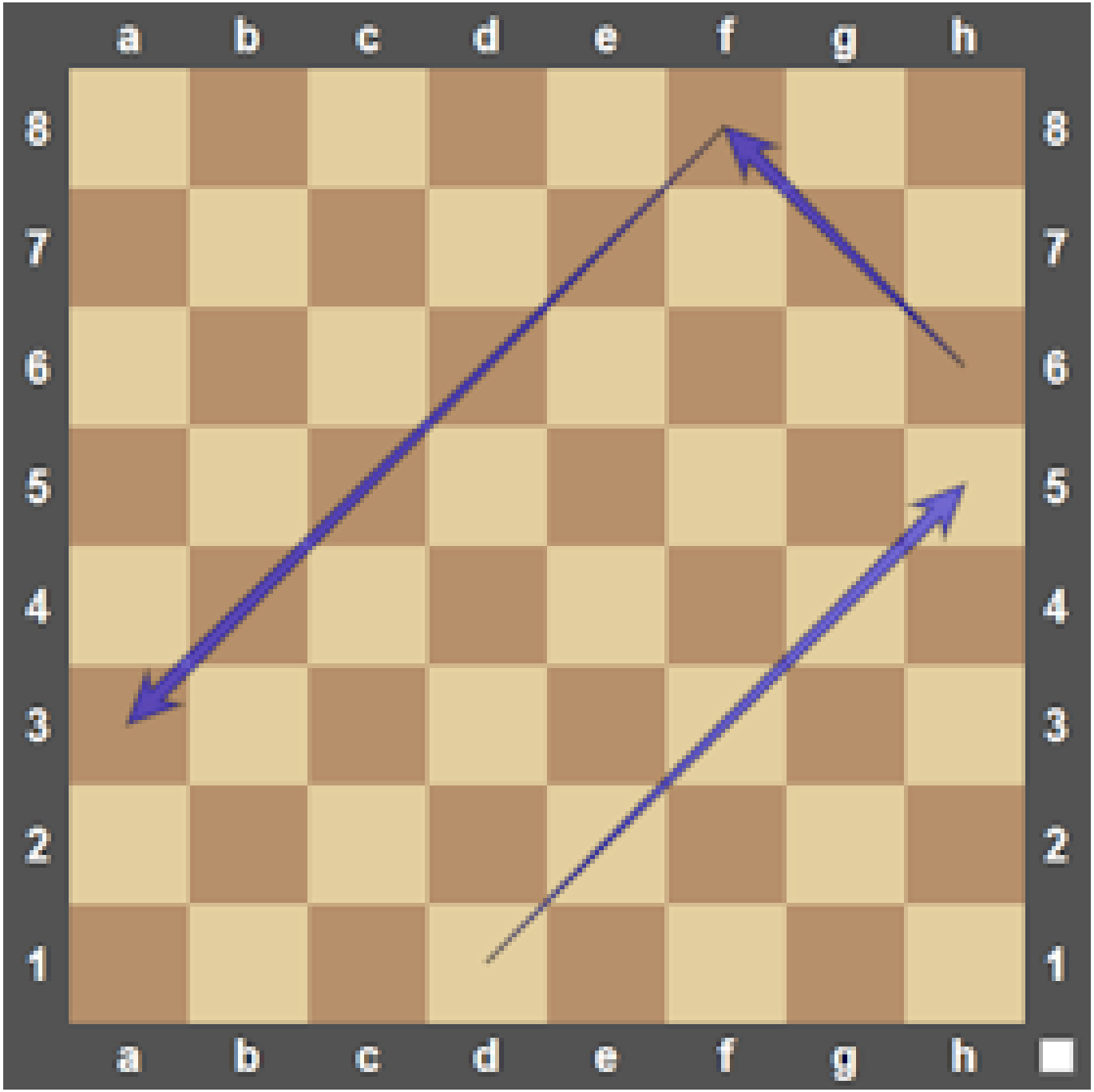
QuickCheck for change

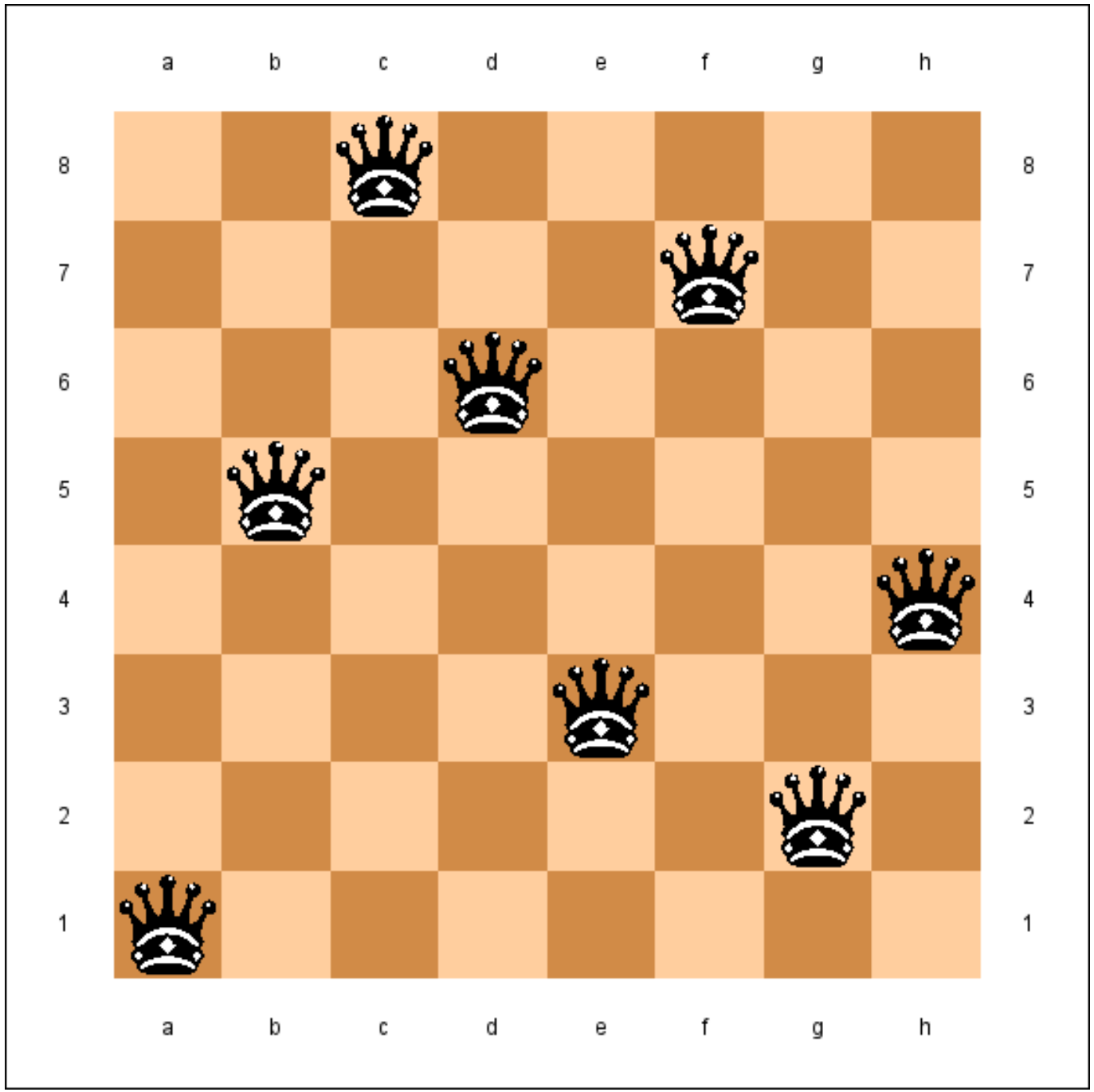
```
prop_change :: Total -> [Coin] -> Property
prop_change n xs =
  0 <= n && all (0 <) xs ==>
    all ((== n) . sum) (change n xs)

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

Part VII

Eight Queens


a b c d e f g h
8 7 6 5 4 3 2 1


a b c d e f g h
8 7 6 5 4 3 2 1
a b c d e f g h

Eight queens

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

queens :: [Board]
queens = filter ok (perms [1..8])

ok :: Board -> Bool
ok qs = and [ not (check p p')
            | [p,p'] <- choose 2 (coords qs) ]

coords :: Board -> [Coord]
coords qs = zip [1..] qs

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

Running eight queens

```
-- > head queens
-- [1,5,8,6,3,7,2,4]
-- (0.13 secs, 46,514,288 bytes)

-- > length queens
-- 92
-- (1.15 secs, 645,843,960 bytes)
```