

```
1  -- PROPERTY TEST -----S
2
3
4  -- Fold L
5  genStack :: [a] -> Stack a
6  genStack [] = Empty
7  genStack (x:xs) = Stack (genStack xs) x
8
9  stackToL :: Stack a -> [a]
10 stackToL Empty = []
11 stackToL (Stack s a) = a : stackToL s
12
13 -- test with + - * div
14 qcFoldl :: Eq b => (b -> a -> b) -> b -> [a] -> Bool
15 qcFoldl f z ls =
16     l == c
17     where
18         s = genStack ls
19         c = foldl f z ls
20         l = stackFoldl f z s
21
22 qcFoldlDiv :: Int -> [Int] -> Property
23 qcFoldlDiv z ls =
24     z /= 0 && all (\a -> a /= 0) ls ==> qcFoldl (div) z ls
25
26 {-
27
28 *Stack> quickCheck $ qcFoldl (+)
29 +++ OK, passed 100 tests.
30 *Stack> quickCheck $ qcFoldl (-)
31 +++ OK, passed 100 tests.
32 *Stack> quickCheck $ qcFoldl (*)
33 +++ OK, passed 100 tests.
34 *Stack> quickCheck $ qcFoldlDiv
35 +++ OK, passed 100 tests; 40 discarded.
36
37 -}
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

```
56 -- Requires Commutative Opeartions +, *
57 qcFoldl1 :: Eq a => (a -> a -> a) -> a -> [a] -> Bool
58 qcFoldl1 f z ls =
59     l' == r' && c == l'
60     where
61         s = genStack ls
62         l' = stackFoldl f z s
63         r' = stackFoldr f z s
64         c = foldl f z ls
65
66 {-
67
68 *Stack> quickCheck $ qcFoldl1 (*)
69 +++ OK, passed 100 tests.
70 *Stack> quickCheck $ qcFoldl1 (+)
71 +++ OK, passed 100 tests.
72 *Stack> quickCheck $ qcFoldl1 (-)
73 *** Failed! Falsified (after 4 tests and 6 shrinks):
74 0
75 [1]
76
77 -}
78
79 -- Fold R
80 -- test with + - *
81 qcFoldr :: Eq b => (a -> b -> b) -> b -> [a] -> Bool
82 qcFoldr f z ls =
83     l == c
84     where
85         s = genStack ls
86         c = foldr f z ls
87         l = stackFoldr f z s
88
89 {-
90
91 *Stack> quickCheck $ qcFoldr (+)
92 +++ OK, passed 100 tests.
93 *Stack> quickCheck $ qcFoldr (-)
94 +++ OK, passed 100 tests.
95 *Stack> quickCheck $ qcFoldr (*)
96 +++ OK, passed 100 tests.
97
98 -}
99
100
101
102
103
104
105
106
107
108
109
110
```

```
111 --commutative operations +, *
112 qcFoldr1 :: Eq b => (a -> b -> b) -> b -> [a] -> Bool
113 qcFoldr1 f z ls = l' ==
114     l && c == l
115     where
116         s' = genStack $ reverse ls
117         s = genStack ls
118         c = foldr f z ls
119         l = stackFoldr f z s
120         l' = stackFoldr f z s'
121
122 {-
123
124 Ok, one module loaded.
125 *Stack> quickCheck $ qcFoldr1 (+)
126 +++ OK, passed 100 tests.
127 *Stack> quickCheck $ qcFoldr1 (*)
128 +++ OK, passed 100 tests.
129 *Stack> quickCheck $ qcFoldr1 (-)
130 *** Failed! Falsified (after 4 tests and 6 shrinks):
131 0
132 [0,1]
133
134 -}
135
136 -- Zip
137 qcZip :: [a] -> [b] -> Bool
138 qcZip ps qs =
139     s == spq && s == sqp
140     where
141         p = genStack ps
142         q = genStack qs
143         pq = stackZip p q
144         spq = stackLen pq 0
145         qp = stackZip q p
146         sqp = stackLen qp 0
147         s = minimum [length ps, length qs]
148
149 {-
150
151 *Stack> quickCheck $ qcZip
152 +++ OK, passed 100 tests.
153
154 -}
155
156
157
158
159
160
161
162
163
164
165
```

```
166 qcZipUnzip :: (Eq a1, Eq a2) => [a1] -> [a2] -> Bool
167 qcZipUnzip ls ms =
168   gsl == l' && m' == gsm && m'' == gsm && l'' == gsl
169   where
170     sx = if length ls <= length ms
171         then length ls
172         else length ms
173     sl = take sx ls
174     sm = take sx ms
175     gsl = genStack sl
176     gsm = genStack sm
177     l = genStack ls
178     m = genStack ms
179     lm = stackZip l m
180     ml = stackZip m l
181     (m'', l'') = stackUnzip ml
182     (l', m') = stackUnzip lm
183     stackUnzip Empty = (Empty, Empty)
184     stackUnzip (Stack s (u,v)) = let (u',v') = stackUnzip s
185                                   in (Stack u' u, Stack v' v)
186
187 -- > quickCheck $ qcZipUnzip
188 -- +++ OK, passed 100 tests.
189
190
191 -- Map
192 qcMap :: Eq b => (a -> b) -> [a] -> Bool
193 qcMap f ls =
194   genStack (map f ls) == stackMap f (genStack ls)
195
196 qcMapDiv :: Int -> [Int] -> Property
197 qcMapDiv d ls =
198   d /= 0 ==> qcMap (`div` d) ls
199
200 {-
201 > quickCheck $ \n -> qcMap (n +)
202 +++ OK, passed 100 tests.
203 > quickCheck $ \n -> qcMap (n -)
204 +++ OK, passed 100 tests.
205 > quickCheck $ \n -> qcMap (n *)
206 +++ OK, passed 100 tests.
207 *Stack> quickCheck $ qcMapDiv
208 +++ OK, passed 100 tests; 13 discarded.
209 -}
210
211
212
213
214
215
216
217
218
219
220
```

```
221 -- length of stack remains same after map, zip
222 stackLen :: Stack a -> Int -> Int
223 stackLen Empty z      = z
224 stackLen (Stack s a) z = stackLen s (z+1)
225
226 -- inverse and length
227 qcMap1 :: Eq a => (a -> b) -> (b -> a) -> [a] -> Bool
228 qcMap1 f f' ls =
229     length ls == m' && m' == lm && mx == s
230     where
231         s = genStack ls
232         m = stackMap f s
233         mx = stackMap f' m
234         lm = stackLen s 0
235         m' = stackLen m 0
236
237 {- apply f then apply inverse of f i.e. f'
238
239 *Stack> quickCheck $ \n -> qcMap1 (n +) ((-n) +)
240 +++ OK, passed 100 tests.
241
242 -}
243
244 -- more ideas
245 -- mapping with (+ 0) or (* 1) results in the same stack (identity)
246
247 -- PROPERTY TEST -----E
```