// Computes an indexs vector to optimize right\_or() function on specific iterations.

Def right\_or\_aux([w1, .., wn]):

len\_w := len(w1)

//Empty initialized indexs vector

indexs

for i in range(1, n, 1):

// Left parse to find first index j where

// wi [j] != ‘s’

if (wi [len\_w-1] == ‘s’):

j := len\_w-1

for j in range(len\_w-2, 0, -1):

if (wi [j] != ‘s’):

break

indexs.append(j)

return indexs

//Recursive algorithm for regex union that returns an equivalent disjoint-union of regexs.

// As a result, algorithm returns a set with NO repeat regexs and NO redundant regexs.

// Must initially call as right\_or([w1, .., wn], indexs, 0), where indexs := right\_or\_aux ([w1, .., wn])

def right\_or([w1, .., wn], indexs, iteration):

len\_w = len(w1)

// Optimization with indexs vector.

if iteration in indexs:

s1 := ‘s’ \* len\_w

if (s1 in [w1, .., wn]):

return s1

// Base case:

if (len\_w == 1):

return or(w1, w2, .. wn)

// Inductive case:

w1 = s1 c1, … wn = sn cn

// Empty initialzed vectors

end\_zero, end\_one

// Sort s1, .., sn in vectors based on ending character

for i in range(1, n, 1):

if (ci == 0):

end\_zero.append(si)

if (ci == 1):

end\_one.append(si)

if (ci == s):

end\_zero.append(si)

end\_one.append(si)

// Recursively call right\_or on non-empty end\_zero and end\_one vectors.

// If both vectors are empty, return empty string.

if end\_zero != empty and end\_one != empty:

iteration := iteration + 1

return str\_concat( right\_or(end\_zero, indexs, iteration), “0+”,

right\_or(end\_one, indexs, iteration), “1”)

else if end\_zero == empty and end\_one != empty:

iteration := iteration + 1

return str\_concat(right\_or(end\_one, indexs, iteration), “1”)

else if end\_zero != empty and end\_one == empty:

iteration := iteration + 1

return str\_concat(right\_or(end\_zero, indexs, iteration), “0”)

else:

return empty

Good example input to try code on:

[“0s0s0”, “s0s0s”]