Description of the implementation

Date: 10 de octubre de 2020

1 Experiments

1.1 Experiment 1

The following code checks that for all the cells e of the hypercube $[0,0]$ and $[0,0,0]$ in $2D$ and $3D$ respectively and for all possible configuration surrounding that cell, it is satisfied that $\chi(St_K v) = 0$ and $\chi(St_{\bar{K}} v) = 0 \Rightarrow \chi(St_K e) = 0$

```
ghci> [euler_star_all e | e <- e_list 2]
[[0],[0],[0],[0]]
ghci> [euler_star_all e | e <- e_list 3]
[[0],[0],[0],[0],[0],[0],[0],[0,0],[0],[0,0],[0,0],
[0,0,0,0,0,0],[0,0],[0,0,0,0,0,0],[0],[0,0],[0,0],[0,0,0,0,0,0]]
```

It was also checked in the fourth dimension where a counterexample was found.

```
ghci>[euler_star_all e | e <- e_list 4]
```

being the counter example the cell $[\frac{1}{2},\frac{1}{2},\frac{1}{2},0]$ in the configuration $[[0,0,0,0],[0,0,1,0],[0,1,0,0],[1,0,1,0],[1,1,1,0]]$.

1.2 Experiment 2

In this experiment, we checked that Euler Well Composedness implies Digitally Well Composedness until dimension 4.

1.3 Experiment 3

The configuration $[[0,0,0,0],[0,0,0,1],[0,0,1,1],[0,1,1,1],[1,1,1,0],[1,1,0,0],[1,0,0,0],[1,1,1,1]]$ is an example of a sDWC configuration that is not sχWC.

```
counter_example_init = [[0,0,0,0],[0,0,0,1],[0,0,1,1],[0,1,1,1],
  [1,1,1,0],[1,1,0,0],[1,0,0,0],[1,1,1,1.0]]
```

This example can be generalized to nD with the following code

```
counter_example 0 hs = hs
counter_example n hs = counter_example (n-1) [[x]++ys| x <-[0,1], ys <- hs]
```

and it was tedted until $n = 7$ with the following code

```
test = [(4+i,dwc (f i),dwc (dualHcubes (4+i) (f i)),euler_link (f i),
             euler_link (dualHcubes (4+i) (f i))) | i <- [1..2]]
     where
       f i = counter_example i counter_example_init
```

2 Code description

The module is given a name and the Haskell library for lists is imported `Data.List`.

```
module Nd_euler_dwc where
import Data.List
```

Next, we provide all the auxiliary functions:

```
noInt x = not (x == fromInteger (round x))

iteration 1 xs = xs
iteration n xs = iteration (n-1) [x ++ [a] | x <- xs, a <- [0,1]]

suma h = [take i h++[ t + (h!!i)]++(drop (i+1) h)| i <- [0..(length h)-1],t <- [-0.5,0.5],
          not (noInt (h!!i)) ]
suma_general = concat. map suma

removeItems xs ys = [x | x <- xs, not (elem x ys)]

remove xs xss = [ys| ys <- xss, xs/= ys]

hammingDistance :: Eq a => [a] -> [a] -> Int
hammingDistance = (sum .) . zipWith ((fromEnum .) . (/=))

combinations :: Int -> [a] -> [[a]]
combinations 0 _  = [ [] ]
combinations n xs = [ y:ys | y:xs' <- tails xs, ys <- combinations (n-1) xs']

intersection xss = nub (concat [intersect x y | x <- xss, y <- xss, x/=y])
```

The following function provides the list of all the hypercubes surrounding the vertex $\left[\frac{1}{2},\ldots,\frac{1}{2}\right]$

```
hypercubes_list n = iteration n [[a] | a <- [0,1]]
```

To compute the dual configuration of a given list of hypercubes

```
dualHcubes n xs = removeItems (hypercubes_list n) xs
```

Then, given a list of hypercubes, i.e., a configuration, we can compute the cells of a given dimension. In this case, `n` in `cell n hs` is `n` minus the desired dimension.

```
cell 0 hs = hs
cell i hs= nub (cell (i-1) (suma_general hs))
```

To compute the dimension of a cell we can use the function `dim e` where `e` is a cell.

```
dim e = sum [1 | i <- e, i==0||i==1]
```

Let us provide now some functions to determine the adjacency of cells in different situations.

```
not_adjacent xss = [xs | xs <- xss, (elem 0.5 xs == False)]
adjacents xss = [xs | xs <- xss, not (elem 0.5 xs == False)]
adjacent x xs = [y | y <- xs, hammingDistance x y == 1]
```

To compute the cubes in the link or the star of the vertex $\left[\frac{1}{2}, \ldots, \frac{1}{2}\right]$ in a given configuration we can use

```
cubes_link xss =  nub (concat [not_adjacent (suma xs)| xs <- xss])
cubes_star xss = nub (concat [adjacents (suma xs)| xs <- xss])
```

Then, to compute the Euler characteristic of the link

```
euler_link hs =  sum [(length (cell i cl))*(-1)^(i+1) | i <- [0..(n-1)]]
           where
                cl = cubes_link hs
                n = length (head hs)
```

To determine if there exists a path between two cells.

```
isthereapath :: (Eq a, Eq t, Num t) => t -> [a] -> [a] -> [[a]] -> Bool
isthereapath n x y xs | x == y = True
                      | n == 0 = False
                      | otherwise = or [isthereapath (n-1) t y xs | t <- (adjacent x xs)]
```

Is a configuration Digitally Well Composed?

```
dwc :: Eq a => [[a]] -> Bool
dwc xss = and [isthereapath (length xss) xs ys xss | ys <- xss, xs <- xss]
```

Determine if a cell e is a face of a cell c

```
face e c | dim c >= dim e = elem e (cell k [c])
         | otherwise = False
     where
        k = dim c - dim e
```

Computes the star of a given dimension of a cell e in a configuration hs.

```
star k e hs = [c | c <- cell k hs, face e c]
```

Computes the Euler characteristic of the star of a cell e in a configuration hs.

```
euler_star e hs =  sum [(length (star k e hs)) * (-1)^k | k <- [0..n] ]
           where
                n = length e
```

List of cells of [0..0]

```
e_list n = concat [cell k [replicate n 0] | k <- [1..n-1]]
v_list n = cell n [replicate n 0]
```

Euler characteristic of the previous list.

```
euler_e_list hs = [euler_star e hs | e <- e_list n]
             where
                n = length (head hs)
```

List of hypercubes that contains a cell e as a face.

```
adjacent_hips e = [h | h <- hypercubes_list_general n, face e h]
              where
                n = length e
```

All possible configurations of hypercubes that contain the cell e as a face.

```
all_conf e = [c | c <- concat [combinations k hs | k <- [1..length hs]], elem (replicate n 0) c, f e c]
         where
                hs = adjacent_hips e
                n = length e
                f e c = length c <= 2^(n-dim e)-1
```

Hypercubes surrounding a given vertex

```
hypercubes_list_general n = iteration n [[a] | a <- [0,1,-1]]
hs_list_gen_subset v = [h | h <- hypercubes_list_general n, face v h]
                   where
                        n = length v
dhs_gen v hs = (hs_list_gen_subset v) \\ hs
```

Euler characteristic of a cell e

```
euler_star_all e = [euler_star e hs | hs <- all_conf e, f hs]
               where
                f hs = sum [euler_star v hs + euler_star v (dhs_gen v hs) | v <- cell n hs] == 0
                n = length e

euler_star_all_list e = [hs | hs <- all_conf e, f hs]
               where
                f hs = sum [euler_star v hs + euler_star v (dhs_gen v hs) | v <- cell n hs] == 0
                n = length e
```