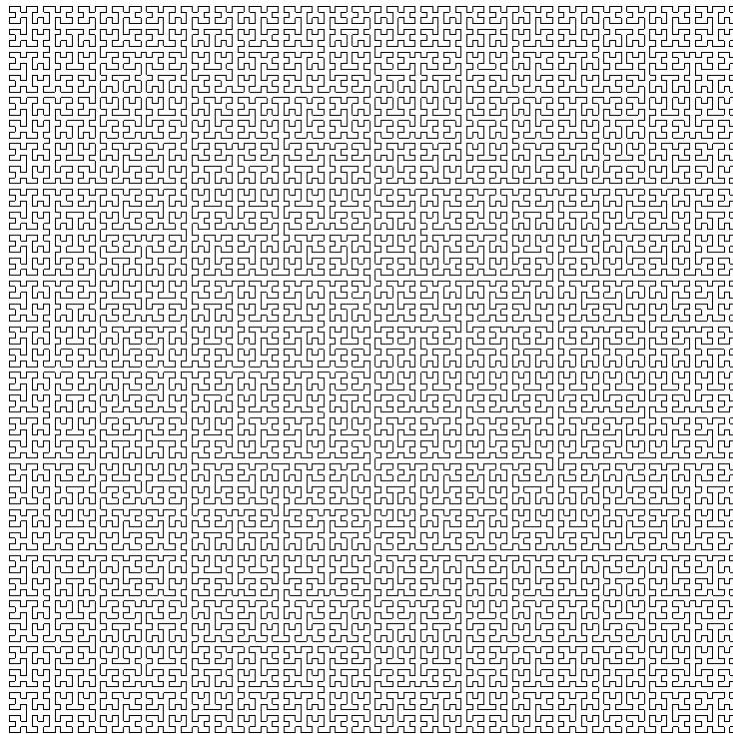




John the Artist



Francesc Barnola u1953660

Gerard Rovellat u1948291

Girona 2023

Índex

1	Introducció	3
2	Separa	4
3	Ajunta	5
4	Prop equivalents	6
5	Copia	7
6	Pentagon	8
7	Poligon	9
8	Espiral	11
9	Execute	13
10	Optimitza	15
11	Triangle	17
12	Fulla	18
13	Hilbert	20
14	Fletxa	22
15	Branca	24

1 Introducció

La practica de haskell consisteix en fer un programa que ens permeti pintar per pantalla diferents fractals proposats. Per aconseguir-ho haurem de començar dibuixant figures simples com son els espirals o els triangles i pentàgons per seguidament poder pintar fractals. També crearem un tipus Comanda que ve implementat parcialment.

Per la realització d'aquesta pràctica utilitzarem el compilador de Haskell en la versió 8.10.7 i llibreries de C que ens permetran veure els dibuixos realitzats com son OpenGL i GLUT. Addicionalment utilitzarem el paquet cabal, que a partir de fitxers .cabal podem fer un build del nostre projecte amb les dependències indicades dins el fitxer.

La hem penjat també a github: <https://github.com/GerardRovellat/PracticaHaskell>

2 Separa

Si la comanda és de la forma $p : \# : q$, cridem recursivament `separa` per a p i q , i concatenem les dues llistes de comandes obtingudes. Si la comanda és `Para`, retornem una llista buida `[]`, ja que volem eliminar el constructor `Para`. En qualsevol altre cas, retornem una llista amb una única comanda. Així, amb aquesta implementació, la funció `separa` transformarà la comanda original en una llista de comandes sense `#:` ni `Para`.

```
-- Separa una comanda en una lista de comandes individuals sense el seu infix :#:.
separa :: Comanda -> [Comanda]
-- Separar la comanda composta en les seves parts i concatenar les llistes resultants.
separa (p :#: q) = separa p ++ separa q
separa Para = [] -- Si la comanda és un 'Para', retornar una llista buida.
-- Per a qualsevol altra comanda, retornar-la en una llista individual.
separa comanda = [comanda]
```

Hem fet proves per comprovar el seu funcionament:

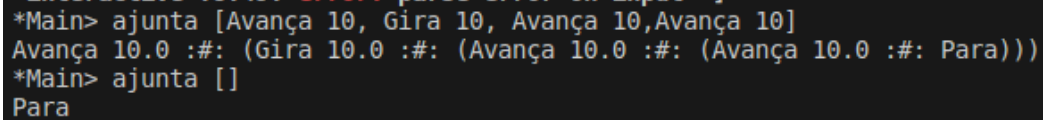
```
*Main> separa (Para :#: Avança 10 :#: Para :#: Gira 10 :#: Avança 10 :#: Para)
[Avança 10.0,Gira 10.0,Avança 10.0]
*Main> separa (Avança 10 :#: Gira 10 :#: Avança 10)
[Avança 10.0,Gira 10.0,Avança 10.0]
*Main> separa (Para)
[]
```

Figura 1: Prova funcionament problema 1

3 Ajunta

```
-- Junta una llista de comandes en una comanda amb l'infix :#: entre comandes.
ajunta :: [Comanda] -> Comanda
-- Si la llista és buida, retornar 'Para'.
ajunta [] = Para
-- Concatenar la primera comanda i l'infix :#:
-- amb la resta de la llista recursivament.
ajunta (x:xs) = x :#: ajunta xs
```

Hem fet algunes proves de funcionament:



```
*Main> ajunta [Avança 10, Gira 10, Avança 10,Avança 10]
Avança 10.0 :#: (Gira 10.0 :#: (Avança 10.0 :#: (Avança 10.0 :#: Para)))
*Main> ajunta []
Para
```

Figura 2: Prova funcionament problema 2

4 Prop equivalents

La funció `prop_equivalent` comprova la propietat de si dues comandes son equivalents, es a dir, si a la hora de aplicar la funció `separa` en les dues comandes, el resultat es igual.

```
-- Comprova si dues comandes són equivalents, és a dir
--, si es separen en la mateixa llista de comandes.
prop_equivalent :: Comanda -> Comanda -> Bool
prop_equivalent c1 c2 = (separa c1) == (separa c2)
```

La funció `prop_split_join` comprova la propietat de si en separar i ajuntar una comanda, s'obté la comanda original

La funció `prop_split` comprova que la funció `separa` retorna una llista complint els requisits de no tindre cap `Para` ni cap comanda composta (`infix ::` entre elles)

```
-- Comprova si en separar i després ajuntar una comanda s'obté la comanda original.
prop_split_join :: Comanda -> Bool
prop_split_join c = c == ajunta (separa c)

-- Comprova que la funcio separar ha retornat efectivament la llista de comandes
-- sense cap Para ni comanda composta (infix :: entre elles).
prop_split :: [Comanda] -> Bool
prop_split c = all comandaSimplif c
  where
    comandaSimplif (p :: q) = False -- Cas que tingui infix
    comandaSimplif Para = False -- Cas que tingui Para
    comandaSimplif _ = True -- Cas que estigui OK
```

Per comprovar el funcionament de la funció `prop equivalent` hem fet servir el mòdul `QuickCheck`

```
*Main> quickCheck (prop_equivalent Para Para)
+++ OK, passed 1 test.
```

Figura 3: Prova funcionament problema 3

5 Còpia

La funció copia com diu ella mateixa, copia una comanda donat un nombre d'iteracions a copiar.

```
-- Realitza una còpia d'una comanda donada un nombre d'iteracions.
copia :: Int -> Comanda -> Comanda
copia i c
  | i == 1    = c -- Cas base.
  | otherwise = c :#: copia (i-1) c -- Cas recursiu.
```

Podem observar el seu funcionament:

```
*Main> copia 5 (Avança 10 :#: Gira 10)
(Avança 10.0 :#: Gira 10.0) :#: ((Avança 10.0 :#: Gira 10.0) :#: ((Avança 10.0 :#: Gira 10.0) :#: ((Avança 10.0 :#: Gira 10.0) :#: (Avança 10.0 :#: Gira 10.0) :#: (Avança 10.0 :#: Gira 10.0))))
```

Figura 4: Prova funcionament problema 4

6 Pentagon

La funció pentàgon genera una comanda per dibuixar un pentàgon a partir d'una distància donada.

```
-- Genera una comanda per dibuixar un pentàgon a partir d'una distància donada.  
pentagon :: Distancia -> Comanda  
-- Fa la copia perquè te 5 costats amb distància d i gira 72 graus.  
pentagon d = copia 5 (Avança d :#: Gira 72)
```

Podem observar el seu funcionament:

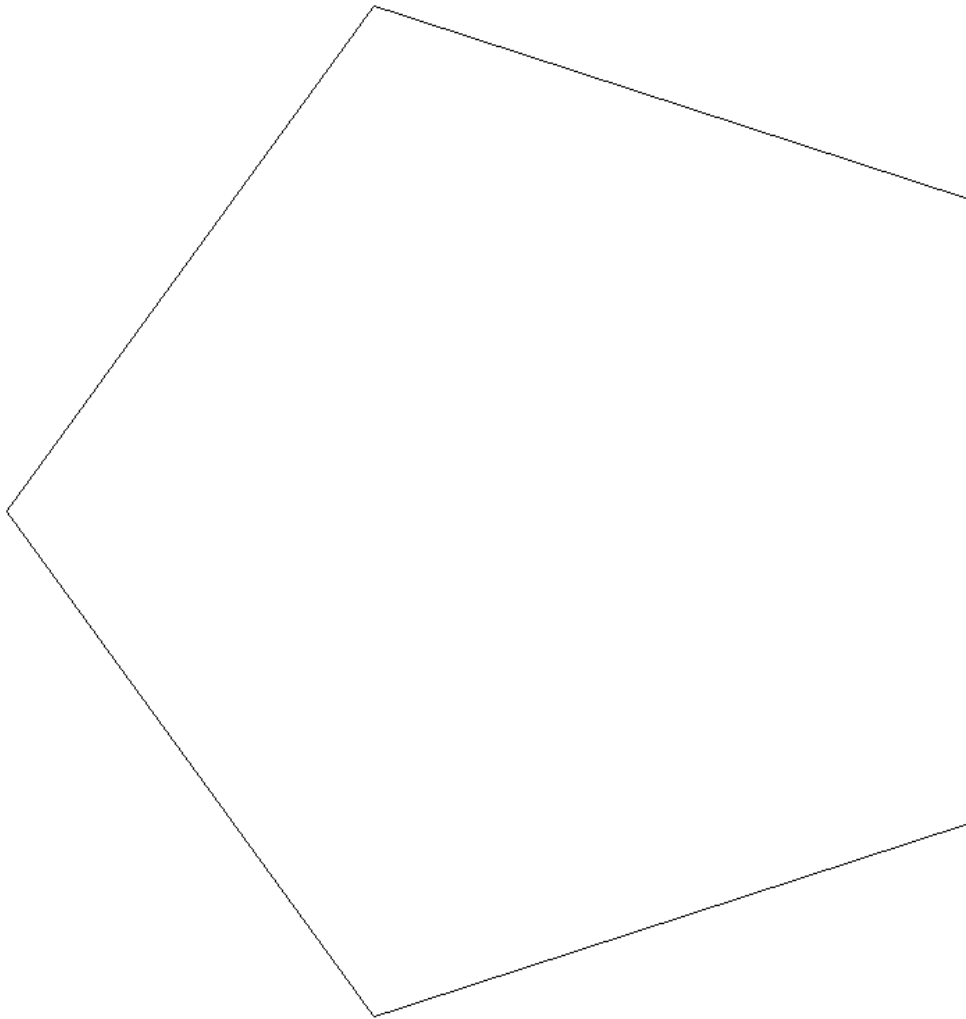


Figura 5: Funcionament pentagon

7 Polígon

La funció pentàgon genera una comanda per dibuixar un pentàgon a partir d'una distància donada.

```
-- Genera una comanda per dibuixar un polígon regular a partir de la distància,  
-- el nombre de costats i l'angle donats.  
poligon :: Distancia -> Int -> Angle -> Comanda  
poligon d n angle = copia n (Avança d :#: Gira angle)  
  
-- Comprova si la comanda generada per dibuixar un polígon regular amb distància  
-- d és equivalent a la comanda del pentàgon amb la mateixa distància.  
-- Verifica si les dues comandes són equivalents utilitzant la funció prop_equivalent,  
-- la qual compara les llistes de comandes generades.  
prop_poligon_pentagon :: Distancia -> Bool  
prop_poligon_pentagon d = prop_equivalent (poligon d 5 72)(pentagon d)
```

Podem observar el seu funcionament:

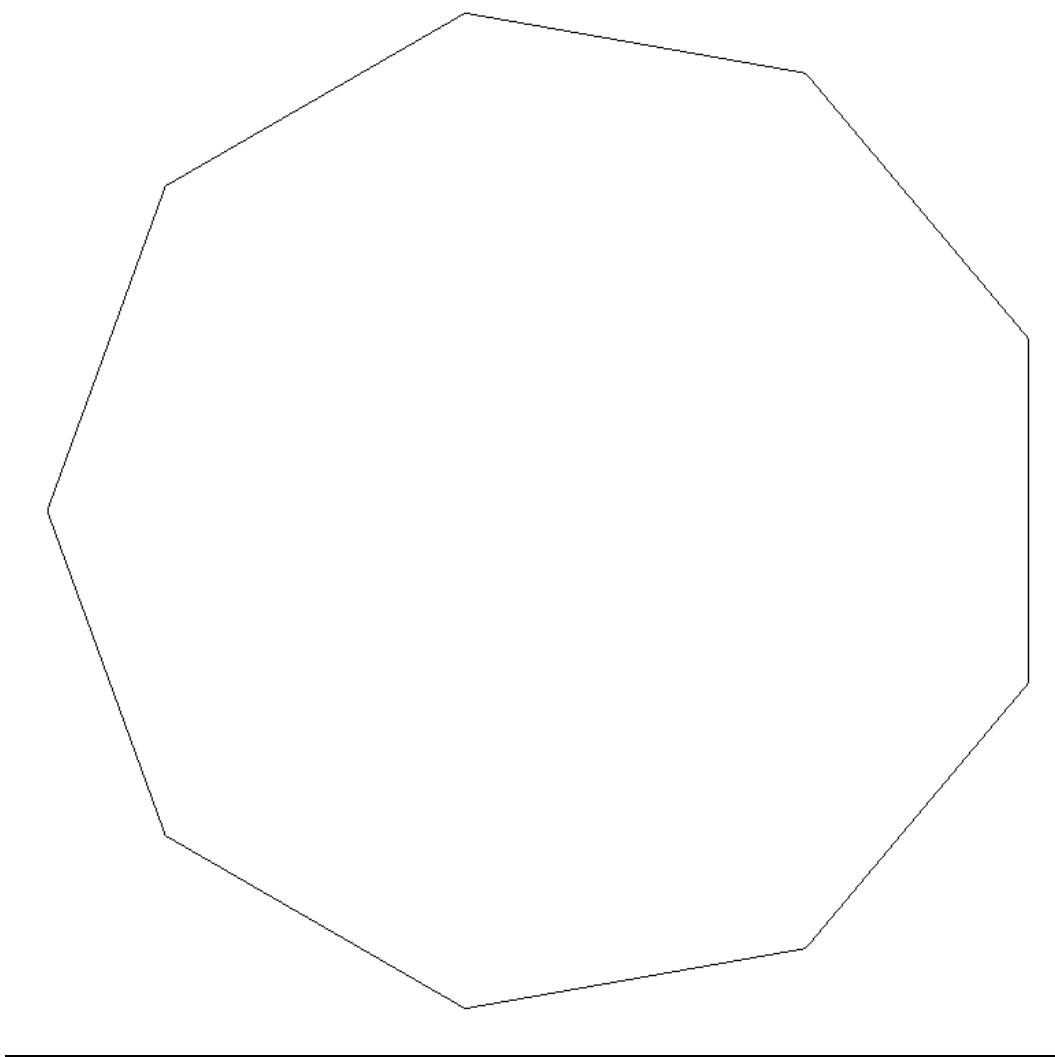


Figura 6: Funcionament poligon

8 Espiral

La funció `espiral` genera una llista de distàncies per a cada costat de l'espiral amb l'angle entrat i el número de segments.

```
-- Genera una llista de distàncies per a cada costat de l'espiral
espiral :: Distancia -> Int -> Distancia -> Angle -> Comanda
-- En cada iteració, pren la distància `d` de la llista generada per `take`
-- a partir de `costat`, incrementant en `pas` en cada pas
espiral costat n pas angle = foldr (:#:) Para [Avança d :#: Gira angle |
                                         d <- take n [costat, costat+pas ..]]
```

Podem observar el seu funcionament:

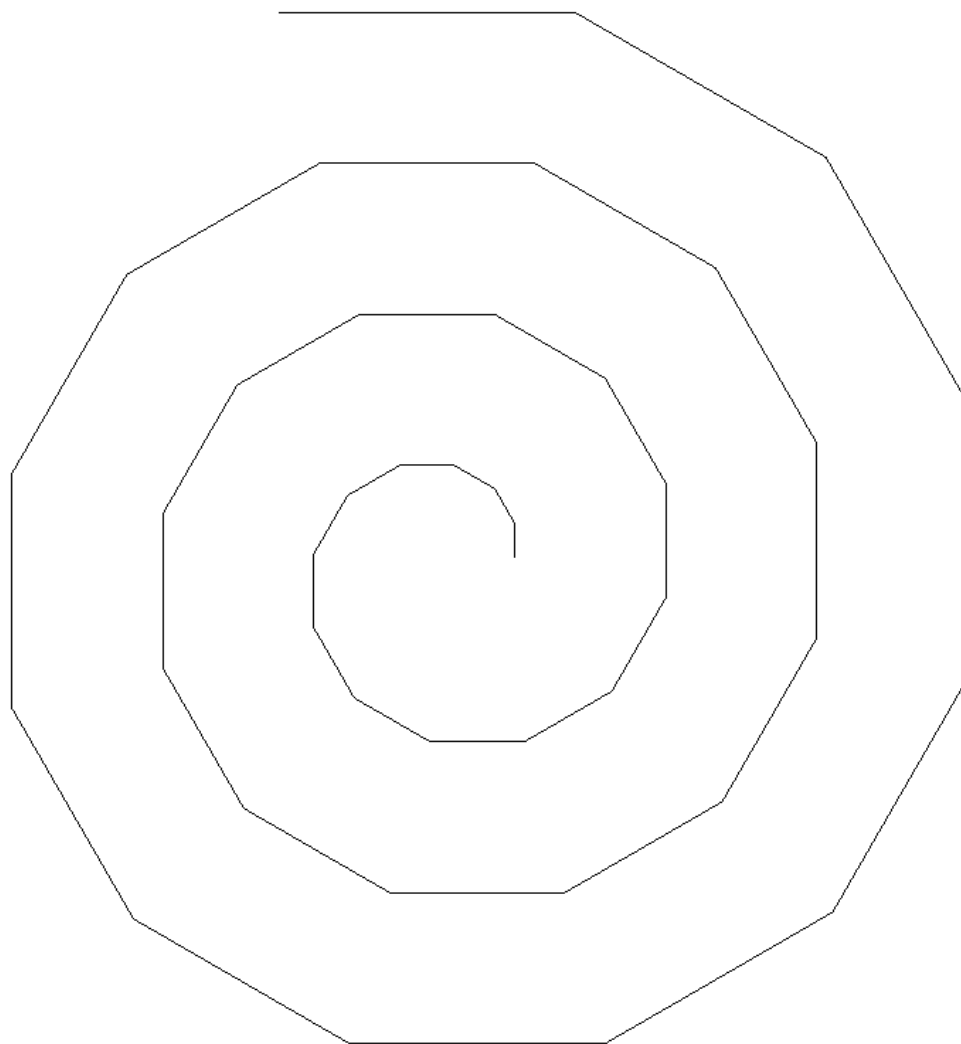


Figura 7: Funcionament espiral

9 Execute

La funció execute generarà la llista de línies a pintar amb el format demanat i l'estat del llapis actual amb valors per defecte de angle, color i punt inicial.

```
-- Declaració del tipus LlapisActual que es guardara l'estat en cada comanda del llapis
data LlapisActual = LlapisActual Llapis Angle Pnt

execute :: Comanda -> [Ln]
execute c = generarLiniesAPintar c LlapisActual negre 0 (Pnt 0.0 0.0)

-- Genera una llista de línies a pintar a partir d'una comanda i l'estat actual del llapis.
generarLiniesAPintar :: Comanda -> LlapisActual -> [Ln]
generarLiniesAPintar c act = case c of
    -- Modifica el color del del llapis actual
    (CanviaColor color) :# y -> generarLiniesAPintar y (modificarColorActual color act)
    -- Si trobem una branca, executem la comanda per separat
    (Branca br) :# y -> generarLiniesAPintar br act ++ generarLiniesAPintar y act
    -- Si el final es una branca, s'executa ella mateixa
    (Branca br) -> generarLiniesAPintar br act
    -- En cas de que hi hagi un Para, es pasa a la següent comanda disponible
    Para :# x -> generarLiniesAPintar x act
    (Avança n) :# x -> let punt2 = puntDesti (posicioLlapis act) (angleLlapis act) n in
        Ln (colorLlapis act) (posicioLlapis act) punt2 :
            generarLiniesAPintar x (modificarPosicioActual punt2 act)
        -- Calculem el punt desti i pintem la línia
    -- Si la comanda es Gira, es modifica el angle
    (Gira n) :# x -> generarLiniesAPintar x (modificarAngleActual n act)
    -- En cas de trobar una comanda composta, la transformem en una
    (x1 :# x2) :# y -> generarLiniesAPintar (x1 :# x2 :# y) act
    (Avança n) -> let punt2 = puntDesti (posicioLlapis act) (angleLlapis act) n in
        [Ln (colorLlapis act) (posicioLlapis act) punt2]
        -- Si trobem un Avança al final, pintem la línia
    _ -> [] -- Per finalitzar el pintar línies, retornem llista buida
where
    -- Modifica el color actual del llapis a l'estat actual.
    modificarColorActual :: Llapis -> LlapisActual -> LlapisActual
    modificarColorActual llapis (LlapisActual _ angle pos) = LlapisActual llapis angle pos

    -- Modifica la posició actual del llapis a l'estat actual.
    modificarPosicioActual :: Pnt -> LlapisActual -> LlapisActual
    modificarPosicioActual pos (LlapisActual llapis angle _) = LlapisActual llapis angle pos

    -- Modifica l'angle actual del llapis a l'estat actual.
    modificarAngleActual :: Angle -> LlapisActual -> LlapisActual
```

```

modificarAngleActual radians (LlapisActual llapis angle pos) =
    LlapisActual llapis (angle + radians) pos

-- Obté el color del llapis de l'estat actual.
colorLlapis :: LlapisActual -> Llapis
colorLlapis (LlapisActual llapis _ _) = llapis

-- Obté la posició del llapis de l'estat actual.
posicioLlapis :: LlapisActual -> Pnt
posicioLlapis (LlapisActual _ _ pos) = pos

-- Obté l'angle del llapis de l'estat actual.
angleLlapis :: LlapisActual -> Angle
angleLlapis (LlapisActual _ angle _) = angle

-- Pasa de graus a radians
graus2radians :: Float -> Float
graus2radians x = (x*2*pi)/360

-- Donat un punt de referència, un angle i una distància, retorna el punt corresponent.
puntDesti :: Pnt -> Angle -> Distancia -> Pnt
puntDesti (Pnt p1 p2) angle dist = Pnt (p1 + dist * cos (graus2radians angle))
    (p2 - dist * sin (graus2radians angle))

```

Podem observar el seu funcionament:

```

*Main> execute ((Avança 10' #: Para :#: Gira 10) :#: Avança 10)
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 10.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 10.0 0.0) (Pnt 19.848078 (-
1.7364819))]

```

Figura 8: Prova funcionament problema 8

10 Optimitza

La funció `optimitza`, optimitza una comanda aplicant l'optimització de `optimitzaComandes` repetidament fins a obtenir una comanda final optimitzada.

```
-- Optimitza una llista de comandes, reduint les repeticions i eliminant els
-- `Para` innecessaris.
optimitzaComandes :: [Comanda] -> Comanda
optimitzaComandes [] = Para -- Cas base: si la llista és buida, retorna `Para`
optimitzaComandes [c]
    -- Si l'única comanda de la llista és `Para`, retorna `Para`
    | equivalentPara c = Para
    -- Altrament, retorna la mateixa comanda
    | otherwise = c
optimitzaComandes (c1 : c2 : cs)
    -- Si la primera comanda és `Para`, la descarta i passa a l'element següent
    | equivalentPara c1 = optimitzaComandes (c2 : cs)
    -- Si la segona comanda és `Para`, descarta la segona comanda i passa a l'element següent
    | equivalentPara c2 = optimitzaComandes (c1 : cs)
    | otherwise = case (c1, c2) of
        -- Si les comandes són d'avançar, les combina en una sola comanda d'avanç
        -- amb la suma de les distàncies
        (Avança d1, Avança d2) -> optimitzaComandes (Avança (d1 + d2) : cs)
        -- Si les comandes són de girar, les combina en una sola comanda
        -- de gir amb la suma dels angles
        (Gira a1, Gira a2) -> optimitzaComandes (Gira (a1 + a2) : cs)
        -- En qualsevol altre cas, manté la primera comanda i continua
        -- amb la resta de la llista
        _ -> c1 : optimitzaComandes (c2 : cs)

-- Verifica si una comanda és equivalent a `Para`
equivalentPara :: Comanda -> Bool
equivalentPara Para = True -- `Para` és equivalent a `Para`
equivalentPara (Avança 0) = True -- L'acció d'avançar amb distància 0 és equivalent a `Para`
equivalentPara (Gira 0) = True -- L'acció de girar amb angle 0 és equivalent a `Para`
equivalentPara _ = False -- En qualsevol altre cas, la comanda no és equivalent a `Para`

-- Optimitza una comanda aplicant l'optimització de `optimitzaComandes` repetidament
-- fins a obtenir una comanda final optimitzada.
optimitza :: Comanda -> Comanda
-- S'utilitza dos cops el separa i l'optimitzaComandes per tal d'eliminar l'últim Para
optimitza c = optimitzaComandes (separa (optimitzaComandes (separa c)))
```

Podem observar el seu funcionament:

```
*Main> optimitza (Avança 10 :#: Para :#: Avança 20 :#: Gira 35 :#: Avança 0 :#: Gira 15 :#: Gira (-50))  
Avança 30.0
```

Figura 9: Prova funcionament problema 1

11 Triangle

La funció triangle dibuixa el fractal triangle fent les reescriptures necessàries.

```
-- Genera una comanda que expandeix la lletra 'F' en forma recursiva
expandirF :: Int -> Comanda
expandirF 0 = Avança 20 -- Cas base
expandirF n = expandirF (n-1) :#: Gira 90 :#: expandirF (n-1) :#:
    Gira (-90) :#: expandirF (n-1) :#: Gira (-90) :#:
    expandirF (n-1) :#: Gira 90 :#: expandirF (n-1)
    -- Reescriptura de F

-- Genera una comanda per dibuixar el fractal triangle utilitzant
-- la reescriptura de la lletra 'F'
triangle :: Int -> Comanda
triangle n = Gira 90 :#: expandirF n
```

Podem observar el seu funcionament:

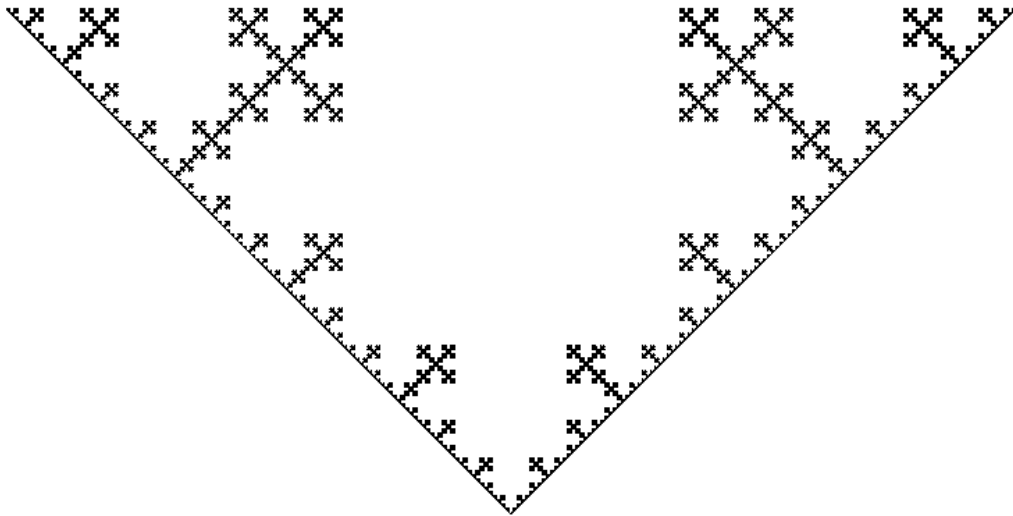


Figura 10: Funcionament triangle 6

12 Fulla

La funció fulla dibuixa el fractal fulla fent les reescriptures necessàries.

```
-- Genera una comanda que expandeix la lletra 'F' en forma recursiva
expandirF1 :: Int -> Comanda
expandirF1 0 = CanviaColor verd :#: Avança 5 -- Cas base
expandirF1 n = expandirG (n-1) :#: Branca (Gira (-45) :#:
    expandirF1 (n-1)) :#: Branca (Gira 45 :#:
    expandirF1 (n-1)) :#: Branca (expandirG (n-1) :#:
    expandirF1 (n-1))
    -- Reescriptura de F

-- Genera una comanda que expandeix la lletra 'G' en forma recursiva
expandirG :: Int -> Comanda
expandirG 0 = Avança 5 -- Cas base
expandirG n = expandirG (n-1) :#: expandirG (n-1)
    -- Reescriptura de G

-- Genera una comanda per dibuixar el fractal fulla
-- utilitzant la reescriptura de la lletra 'F'
fulla :: Int -> Comanda
fulla n = expandirF1 n
```

Podem observar el seu funcionament:

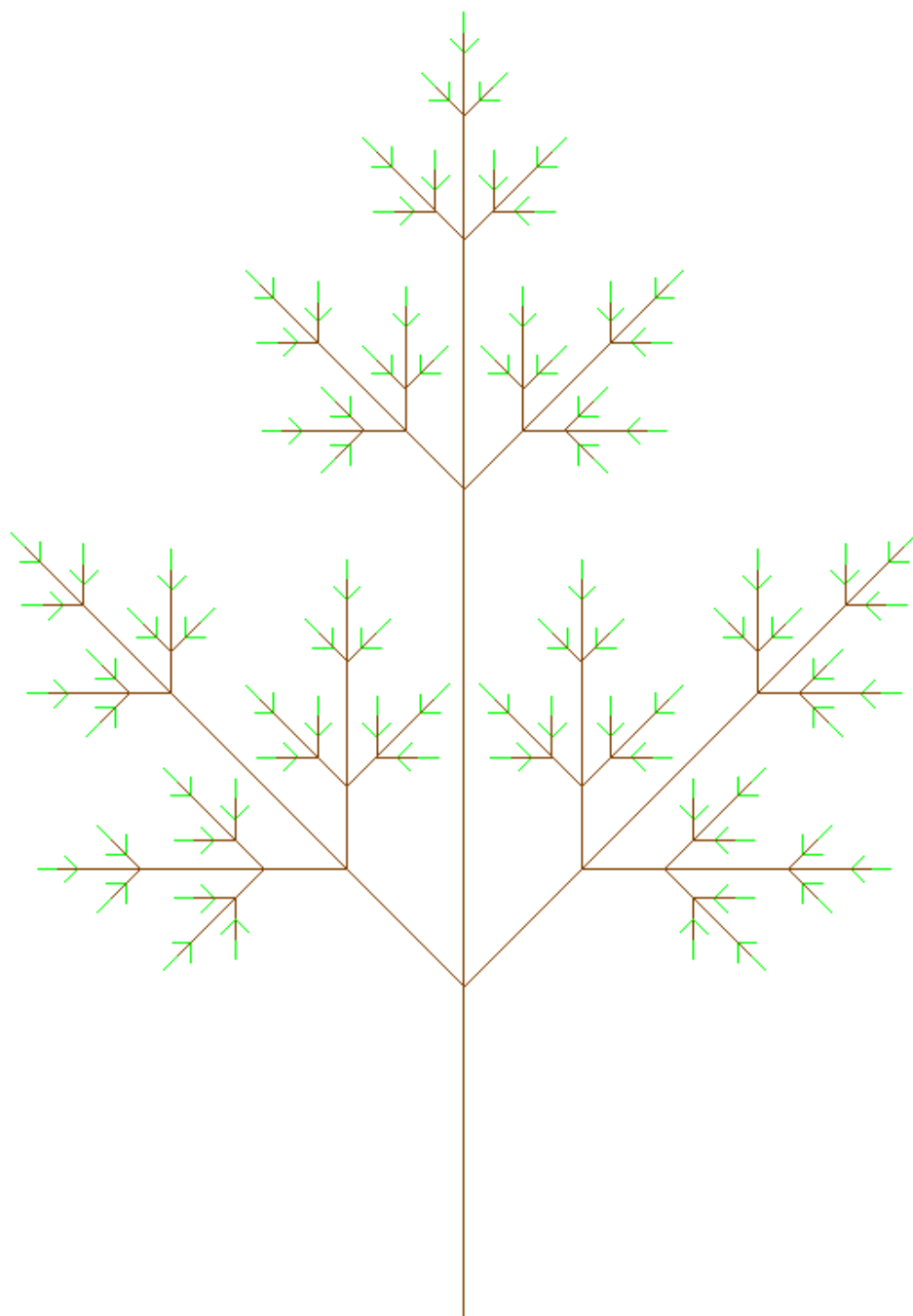


Figura 11: Funcionament fulla 5

13 Hilbert

La funció hilbert dibuixa el fractal hilbert fent les reescriptures necessàries.

```
-- Genera una comanda que expandeix la lletra 'L' en forma recursiva
expandirL :: Int -> Comanda
expandirL 0 = Para -- Cas base
expandirL n = Gira 90 :#: expandirR (n-1) :#: Avança 20 :#:
              Gira (-90) :#: expandirL (n-1) :#: Avança 20 :#:
              expandirL (n-1) :#: Gira (-90) :#: Avança 20 :#:
              expandirR (n-1) :#: Gira 90
              -- Reescriptura de L

-- Genera una comanda que expandeix la lletra 'R' en forma recursiva
expandirR :: Int -> Comanda
expandirR 0 = Para -- Cas base
expandirR n = Gira (-90) :#: expandirL (n-1) :#: Avança 20 :#:
              Gira 90 :#: expandirR (n-1) :#: Avança 20 :#:
              expandirR (n-1) :#: Gira 90 :#: Avança 20 :#:
              expandirL (n-1) :#: Gira (-90)
              -- Reescriptura de R

-- Genera una comanda per dibuixar el fractal hilbert
-- utilitzant la reescriptura de la lletra 'L'
hilbert :: Int -> Comanda
hilbert n = expandirL n
```

Podem observar el seu funcionament:

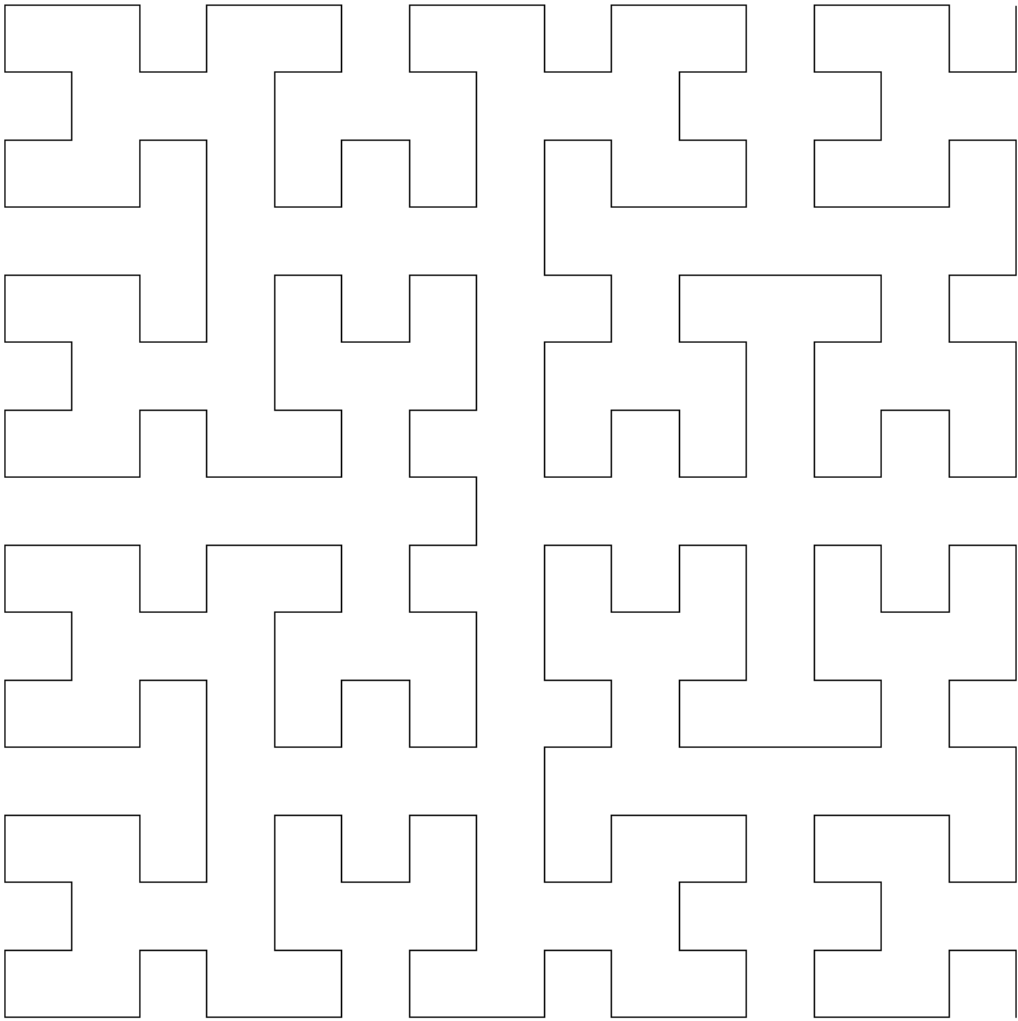


Figura 12: Funcionament hilbert 4

14 Fletxa

La funció fletxa dibuixa el fractal fletxa fent les reescriptures necessàries

```
-- Genera una comanda que expandeix la lletra 'F' en forma recursiva
expandirF2 :: Int -> Comanda
expandirF2 0 = Avança 10 -- Cas base
expandirF2 n = expandirG1 (n-1) :#: Gira 60 :#: expandirF2 (n-1) :#:
              Gira 60 :#: expandirG1 (n-1)
              -- Reescriptura de F

-- Genera una comanda que expandeix la lletra 'G' en forma recursiva
expandirG1 :: Int -> Comanda
expandirG1 0 = Avança 10 -- Cas base
expandirG1 n = expandirF2 (n-1) :#: Gira (-60) :#: expandirG1 (n-1) :#:
              Gira (-60) :#: expandirF2 (n-1)
              -- Reescriptura de G

-- Genera una comanda per dibuixar el fractal fletxa
-- utilitzant la reescriptura de la lletra 'F'
fletxa :: Int -> Comanda
fletxa n = Gira 90 :#: expandirF2 n
```

Podem observar el seu funcionament:

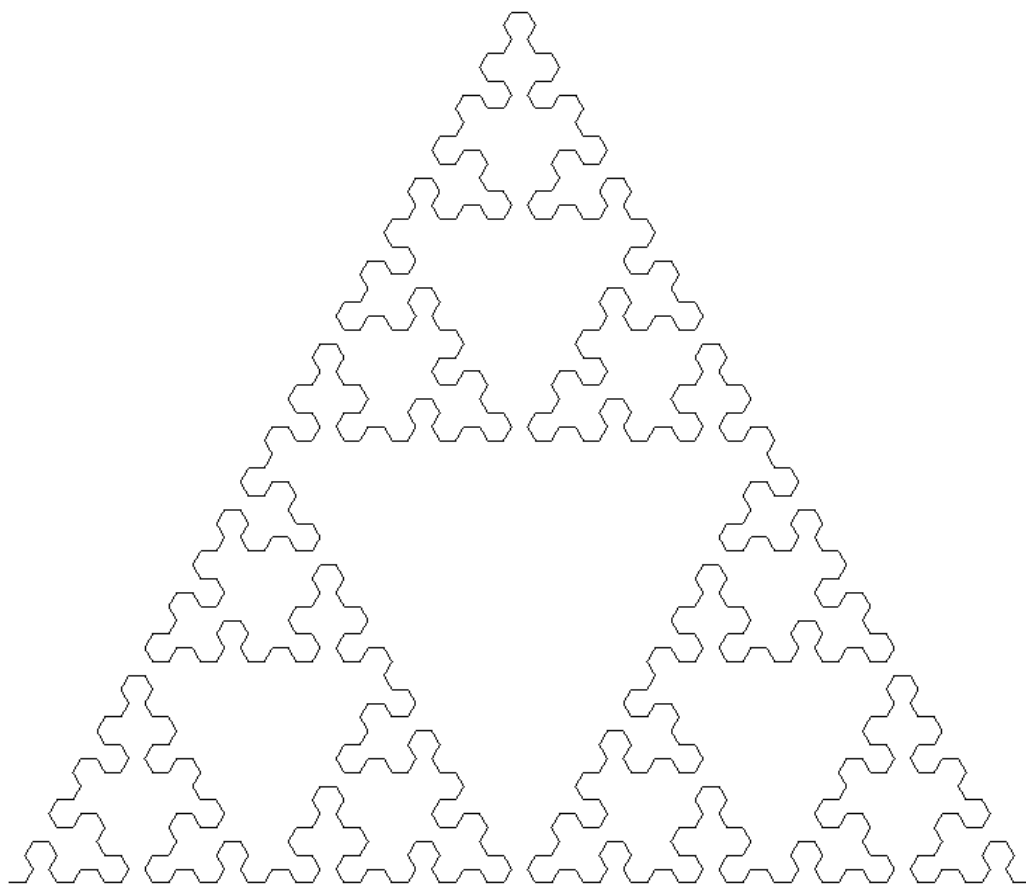


Figura 13: Funcionament fletxa 6

15 Branca

La funció branca dibuixa el fractal fletxa fent les reescriptures necessàries

```
-- Genera una comanda que expandeix la lletra 'G' en forma recursiva
expandirG2 :: Int -> Comanda
expandirG2 0 = Avança 10 -- Cas base
expandirG2 n = expandirF3 (n-1) :#: Gira (-22.5) :#: Branca
               (Branca (expandirG2 (n-1)) :#: Gira 22.5 :#:
                expandirG2 (n-1)) :#: Gira 22.5 :#:
               expandirF3 (n-1) :#: Branca (Gira 22.5 :#:
               expandirF3 (n-1) :#: expandirG2 (n-1)) :#:
               Gira (-22.5) :#: expandirG2 (n-1)
               -- Reescriptura de G

-- Genera una comanda que expandeix la lletra 'F' en forma recursiva
expandirF3 :: Int -> Comanda
expandirF3 0 = Avança 10 -- Cas base
expandirF3 n = expandirF3 (n-1) :#: expandirF3 (n-1)
               -- Reescriptura de F

-- Genera una comanda per dibuixar el fractal branca
-- utilitzant la reescriptura de la lletra 'G'
branca :: Int -> Comanda
branca n = expandirG2 n
```

Podem observar el seu funcionament:



Figura 14: Funcionament branca 5