Trabajo Práctico Final
DSL: Image Blending

Análisis de Lenguajes de Programación

Alumno: Navall, Nicolás Uriel N-1159/2

Indice

Descripción

El lenguaje se utiliza para representar secuencias de edición sobre imágenes, y en última instancia exportar una imagen resultado.

Entre sus características se encuentra la posibilidad de editar una sola imagen, mezclar imágenes, leer términos del lenguaje desde una línea de comandos o desde un archivo entre otras. Para más información lea las secciones “Manual de uso” y “Términos”.

Los modos de blending varían su implementación entre documentaciones, por lo que los resultados pueden ser levemente diferentes a algunos software de edición de imágenes.

Dependencias

Para poder utilizar el lenguaje deberá instalar los siguientes paquetes.

GHC es el compilador e intérprete de haskell, mientras que *cabal* es un instalador de paquetes de haskell.

```
sudo apt install ghc
```

```
sudo apt install cabal-install
```

Options es una biblioteca de haskell para poder utilizar flags en parsers y *hip* es una biblioteca de imágenes para haskell.

```
cabal install options
```

```
sudo apt-get install zlib1g
```

```
cabal install hip
```

Se requerirá instalar la biblioteca *zlib1g* de esta forma para utilizar hip ya que una de las bibliotecas que depende de hip utiliza zlib1g y cabal no puede instalar las dependencias que no corresponden a paquetes de haskell.

Manual de uso

Para ejecutar el programa debe correr el siguiente comando

```
runhaskell Main.hs COMANDO IMAGEN
```

Donde *COMANDO* hace referencia a los siguientes caracteres:

- **i (interpret)**: Si se desea escribir el término a evaluar en la terminal debe utilizar este comando. El término *IMAGEN* por lo tanto será una string con la expresión a evaluar.
- **f (file)**: Si lo que se busca es leer una expresión de un archivo se deberá utilizar dicho comando en conjunto con el nombre del archivo (puede escribirse una dirección relativa o absoluta) en lugar del término *IMAGEN*.

Además de los comandos presentados se puede utilizar diferentes flags con configuraciones opcionales:

- **--exec='s'**: Permite aplicar términos argumentos a un término leído. Tenga en cuenta que el término parseado debe ser una función y que 's' corresponde a una serie de nombre de un archivo o términos dependiendo si se utilizó el comando i o f respectivamente (estos separados por comas). El evaluador se encargará de agregar las aplicaciones necesarias.
- **--d='dir'**: Permite especificar la dirección en donde se guardará el archivo (relativa o absoluta) además del nombre y el formato de la imagen de salida. Se exportará el archivo como *output.png* de forma predeterminada en el directorio local.
- **--m='x'**: Permite elegir el modo de evaluación. Estos son:
    - **1**: Las imágenes que se usarán en funciones con dos argumentos necesitaran tener las mismas dimensiones. Este es el modo predefinido.
    - **2**: Las imágenes aplicadas en funciones binarias darán una imagen resultante con el menor tamaño de ambas.
    - **3**: Las imágenes aplicadas en funciones binarias darán una imagen resultante con el mayor tamaño de ambas".
- **-h**: Mensaje de ayuda con esta información.

Los modos de evaluación 2 y 3 recortan y aumentan los tamaños de las imágenes centrando la imagen mas chica sobre la más grande.

Alternativamente puede prescindir de haskell compilando el programa con *ghc Main.hs* y corriendo el ejecutable con los comandos anteriores utilizando *./Main* en lugar de *runhaskell Main.hs*.

Terminos

Los términos se basan en una combinación de operaciones "binarias" y "unarias" de imágenes (lease operaciones binarias a funciones que toman dos imágenes como argumentos y operaciones unarias como funciones que toman una imagen como argumento más un double que hará de "slider"). Además se agrega el potencial del lambda cálculo para definir funciones y variables.

- **Var**: Nombre de variable. Este debe ser alfanumerico.
- **Abs x y**: Definicion de funcion 'y' en donde su variable es 'x'.
- **App x y**: Aplicación de la expresión 'x' en 'y' ('y' debe ser una función, caso contrario se devolverá un error).
- **'<'s'>'**: Lectura de una imagen, en donde s es una dirección relativa o absoluta de un archivo.
- **Op x y**: 'Op' es una función binaria de blending que mezclara las imágenes resultantes de los términos 'x' e 'y'. Esto se interpreta como la imagen 'y' se aplica a la imagen 'x' con la operación de blending 'Op'. A continuación las funciones de blending disponibles:
    - **Normal**: No aplica ningún blending especial, solo mostrará la imagen 'y' por encima de la imagen 'x'.
    - **Add**: Suma los valores de los canales RGB de las imágenes.
    - **Difference**: Resta los valores de los canales RGB de las imágenes.
    - **Darken**: Da como resultado el componente más oscuro de cada canal entre dos imágenes.
    - **Lighten**: Da como resultado el componente más luminoso de cada canal entre dos imágenes.
    - **Multiply**: Multiplica las componentes de las dos imágenes canal por canal. El efecto es comparable al de poner dos filminas una encima de la otra y proyectarlas juntas. La luz, obligada a pasar por ambas filminas, es debilitada dos veces.
    - **Screen**: Función de mezclado opuesta a 'Multiply', multiplica los opuestos de las imágenes. Esto es: *Screen x y = (Complement Multiply (Complement x) (Complement y)).*
    - **Overlay**: Si la componente del canal de 'x' es menor a 0.5, los valores tonales se multiplican, si no se aplicará la función de mezclado de 'screen' (después de haber sido duplicados en ambos casos).
    - **Hard Light**: Este modo corresponde a realizar un mezclado con 'overlay' con las imágenes intercambiadas de lugar. Esto es *HardLight x y = Overlay y x.*

    - **Soft Light**: Similar a Overlay pero con efectos reducidos.
    - **Color Dodge**: El brillo de la imagen 'y' "protege" a la imagen 'x' de exposición.
    - **Color Burn**: El inverso de 'Color Dodge'. Es decir *ColorBurn x y= Complement (ColorDodge (Complement x) (Complement y))*
    - **Hue**: Da como resultado un píxel con el brillo y la saturación de 'x' pero el color (hue) de 'y'.
    - **Luminosity**: Da como resultado un píxel con el color y la saturación de 'x' pero el brillo de 'y'.
    - **Blend Color**: Da como resultado un píxel con el brillo de 'x' pero con el color (hue) y la saturación de 'y'.
    - **Blend Saturation**: Da como resultado un píxel con el color (hue) y el brillo de 'x' pero con la saturación de 'y'.
    - **Exclusion**: Un píxel brillante causa la inversión del otro pixel operando.

- **UOp x d**: 'Uop' es una función binaria que toma una imagen resultante de evaluar el término 'x' y un número flotante 'd'. Las funciones disponibles son las siguientes:
    - **Temperature**: Modifica la temperatura de una imagen (es decir cambian el tono de la imagen hacia los azules o hacia los naranjas). El rango sugerido para el argumento flotante es [1000, 40000].
    - **Saturation**: Modifica la saturación de una imagen. El rango sugerido para el argumento flotante es [-1, 1].
    - **Exposure**: Modifica la exposición de una imagen. El rango sugerido para el argumento flotante es [-1, 4].
    - **Contrast**: Modifica el contraste de una imagen. El rango sugerido para el argumento flotante es [-1, 1].
    - **Opacity**: Modifica el canal alpha de una imagen (es decir su transparencia). El rango sugerido para el argumento flotante es [-1, 1].
    - **Highlights**: Modifica la exposición de las luces. El rango sugerido para el argumento flotante es [-1, 1].
    - **Shadows**: Modifica la exposición de las sombras. El rango sugerido para el argumento flotante es [-1, 1].

- **Complement x**: Invierte los colores de la imagen resultante de evaluar la expresión 'x'.

Gramática

```
Term ::=  Var
  |  "Abs" String Term
  |  "App" Term Term
  |  '<' String '>'
  |  Op Term Term
  |  UOp Term Double
  |  "Complement" Term
  | '(' Term ')'

Var ::= String

Op ::= "Normal"
 | "Add"
 | "Difference"
 | "Darken"
 | "Lighten"
 | "Multiply"
 | "Screen"
 | "Overlay"
 | "HardLight"
 | "SoftLight"
 | "ColorDodge"
 | "ColorBurn"
 | "Hue"
 | "Luminosity"
 | "BlendColor"
 | "BlendSat"
 | "Exclusion"

UOp = "Temperature"
| "Saturation"
| "Exposure"
| "Contrast"
| "Opacity"
| "Shadows"
| "Highlights"
```

Sinonimos de las operaciones

Para facilitar el uso del lenguaje se ofrecen nombres abreviados de operaciones, estos se detallan a continuación:

Operación	Sinonimos
Normal	N
Add	A
Difference	Diff
Darken	D
Lighten	L
Multiply	M
Screen	S
Overlay	O
HardLight	Hard Light HL
SoftLight	Soft Light SL
ColorDodge	Color Dodge CD
ColorBurn	Color Burn CB
Hue	H
Luminosity	Lum
BlendColor	Blend Color Color BC
BlendSaturation	BlendSat Blend Saturation BS
Exclusion	E
Temperature	Temp T
Saturation	Sat
Exposure	Exp
Contrast	Cont
Opacity	
Complement	Comp

Propiedades de las operaciones

Se puede observar las definiciones de los modos de blending en el código y concluir (dada las operaciones matemáticas que se realiza entre los píxeles) que las siguientes operaciones son conmutativas:

- Add
- Darken
- Lighten
- Multiply
- Screen
- Exclusion

Bajo la misma idea se puede definir un elemento neutro en los siguientes modo de blending:

- Normal: Imagen transparente.
- Add: Imagen de color negro.
- Multiply: Imagen de color blanco.
- Screen: Imagen de color negro.
- Exclusion: Imagen de color negro.
- Overlay: Imagen de color gris es neutro a izquierda.
- HardLight: Imagen de color gris es neutro a derecha.
- ColorDodge: Imagen de color negro es neutro a derecha.
- ColorBurn: Imagen de color blanco es neutro a izquierda.

Interprétese una imagen de color blanco como una imagen en la cual todos sus píxeles tienen los componentes RGBA (0, 0, 0, -). Bajo la misma idea se interpreta a una imagen de color negro con pixeles RGBA (1 ,1, 1, -), una imagen de color gris con pixeles RGBA (0.5, 0.5, 0.5, -) y una imagen transparente con pixeles RGBA (-, -, -, 0).

Además se puede observar la existencia de elementos absorbentes en algunas operaciones:

- Normal: Cualquier imagen a la derecha.
- Luminosity: Imagen de color negro a derecha.
- Exclusion: Imagen de color gris.

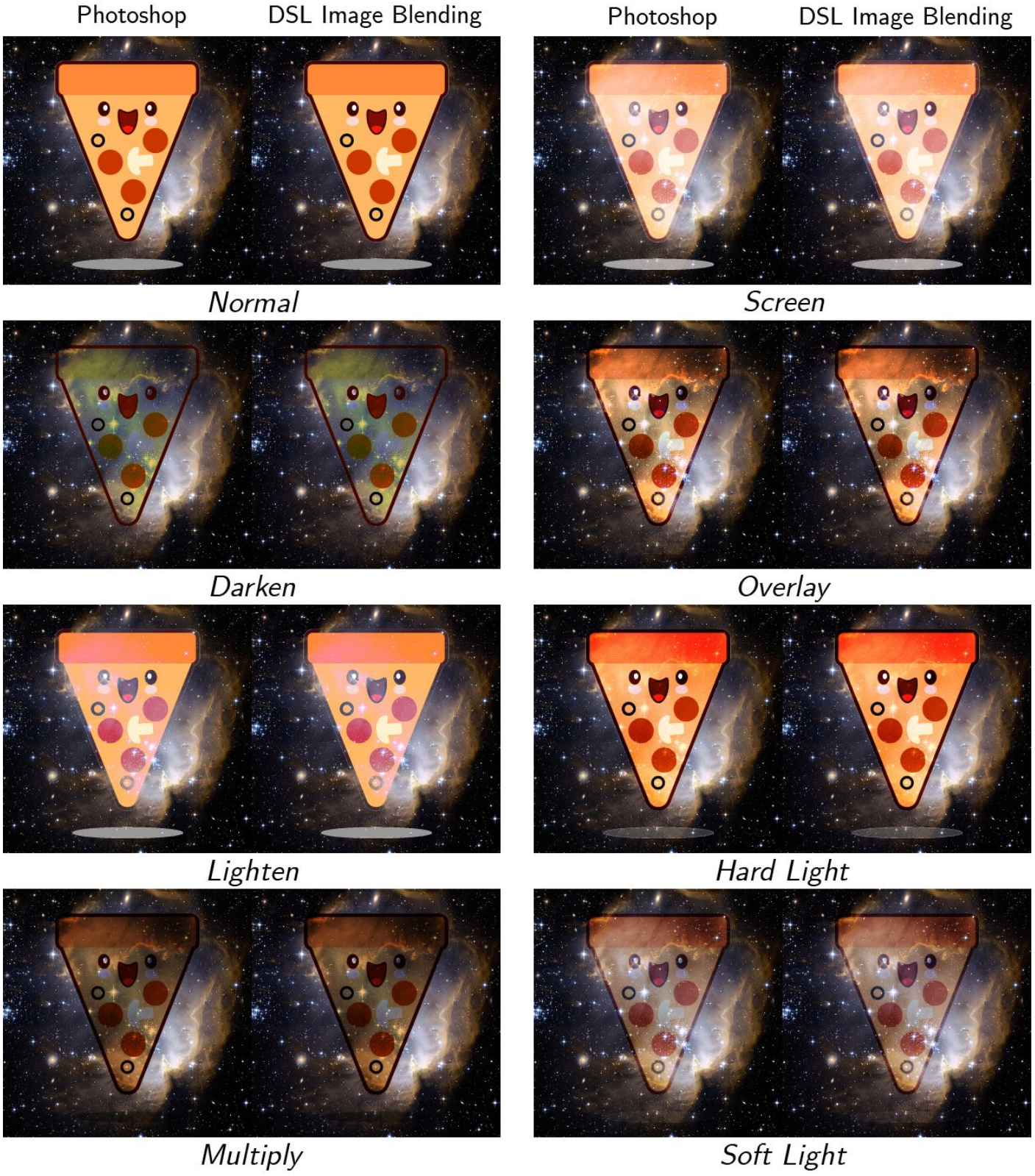
Con respecto a los operadores que toman una imagen y un double no es posible que estos posean la propiedad conmutativa por razones de tipos, pero si casi todos poseen al número 0 como elemento neutro a

derecha, con excepción el operador 'Temperature', el cual tiene a 6500 como elemento neutro a derecha.

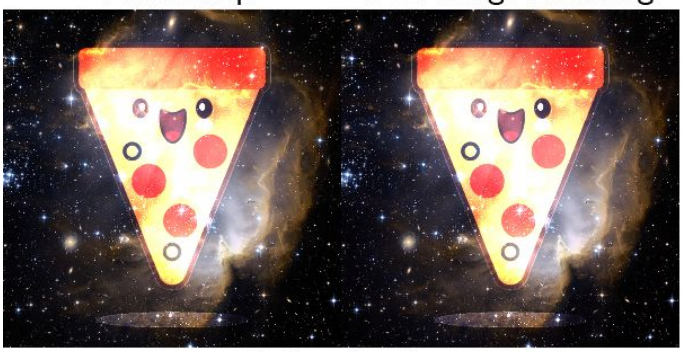
Todas las propiedades mencionadas son demostrables matemáticamente.

Comparación de resultados

Se presenta una tabla comparativa entre los resultados de mezclado de Photoshop y los obtenidos con el lenguaje:

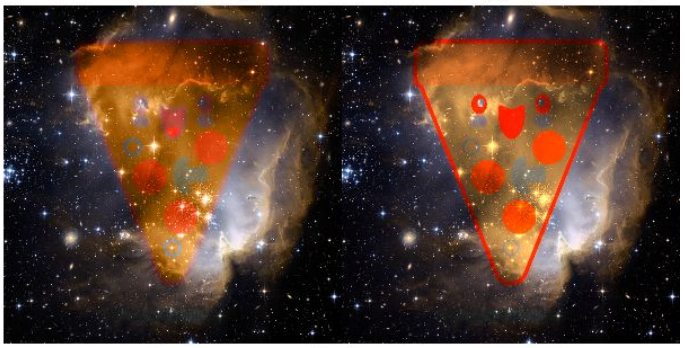


Photoshop DSL Image Blending

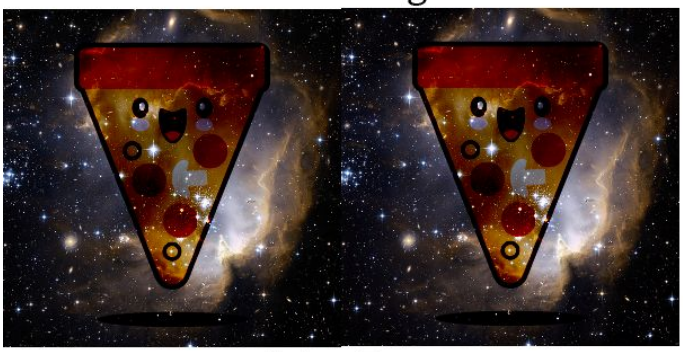


Color Dodge

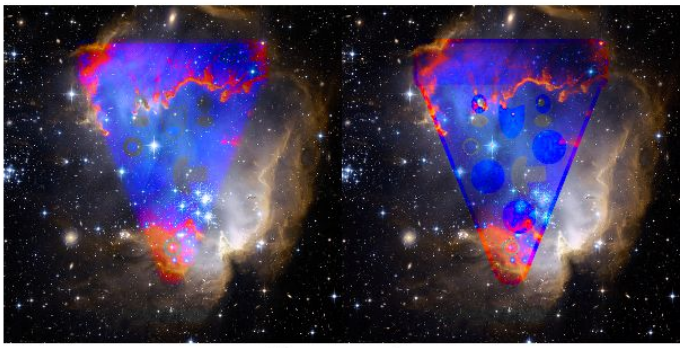
Photoshop DSL Image Blending



Color

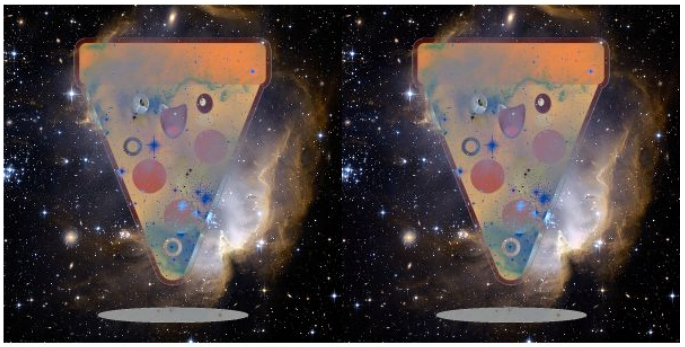


Color Burn



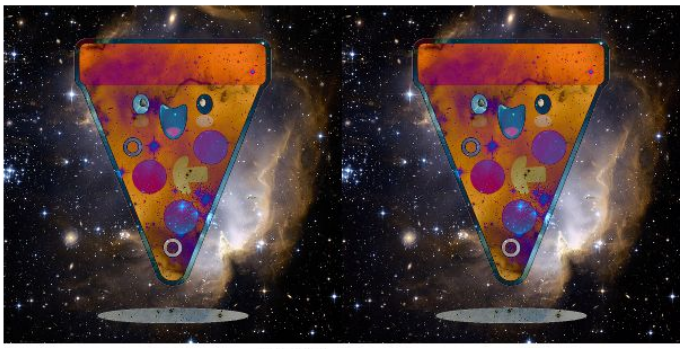
Saturation

Hue

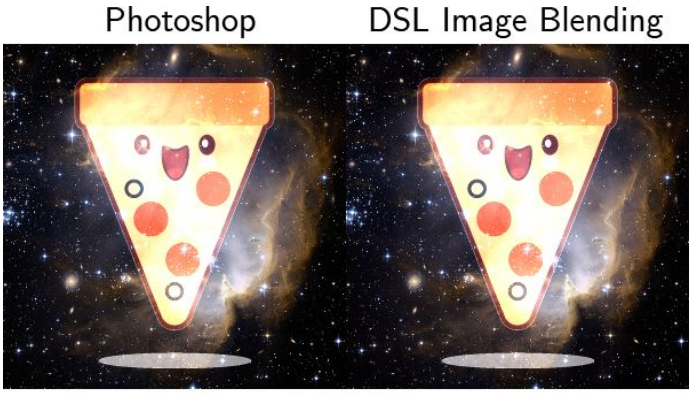


Exclusion

Luminosity



Difference



Addition

Distribución de módulos

A continuación una breve descripción de cada módulo:

- **Parser.hs**: Contiene las funciones asociadas al parseo de términos del lenguaje.
- **Common.hs**: Contiene los tipos con los que se maneja el lenguaje, como así también las funciones de mezclado y edición para píxeles e imágenes.
- **Eval.hs**: Contiene todas las funciones y monadas relacionadas a la evaluación de términos.
- **Main.hs**: Contiene los tipos y funciones necesarios para usar el lenguaje compilado con sus argumentos opcionales.

Decisiones de diseño

La implementación de lambda cálculo en el lenguaje me pareció necesaria ya que permite escribir funciones, y combinado con la posibilidad de leer archivos, facilita la interacción de un usuario con el lenguaje.

Opte por una implementación del lenguaje de tipo “Deep embedding” para facilitar el debugging y poder tener diferentes evaluadores.

El tipo LamTerm tiene un tipo Op y un tipo UOp entre los argumentos de sus constructores. Estos hacen referencia a una función de blending y edición respectivamente. Se optó por este enfoque en lugar de tener funciones asociadas a cada operación (Imagen->Imagen->Imagen y Imagen->Double->Imagen) como argumentos de los constructores para permitir su impresión en pantalla de ser necesario.

Se eligió “encapsular” en el lenguaje las direcciones de las imágenes con los caracteres ‘<’ ‘>’ ya que estos no son caracteres

válidos para nombres de archivo, por lo cual no se mezclaran con la dirección de una imagen.

La mayoría de las funciones de mezclado están conformadas por una función (que toma dos canales de dos imágenes y da un canal resultante) aplicada a otra función que permite la aplicación de la función descrita en los canales de un pixel (y en última instancia, en toda la imagen). Se prefirió que están definidas de esta manera ya que se puede observar muy fácilmente que hace cada función del lenguaje con los canales de un pixel. Las funciones que no están definidas de esta manera toman dos pixeles de dos imágenes y dan un píxel resultante; es necesaria esta definición ya que se necesita realizar una conversión a otro espacio de colores, necesitando así las 3 componentes de un pixel RGB. Lo mismo sucede con las funciones de edición.

Se escribió una función "Imagen -> Imagen -> Imagen" relacionada a cada operación para, además de facilitar la interpretación, poder realizar una clara distinción entre las funciones derivadas y primitivas.

A simple vista parece ser que 'Screen', 'ColorBurn' y 'Hard Light' pueden definirse a partir de otras funciones primitivas, siendo así funciones derivadas, y aunque ese es el caso de 'Screen' y 'ColorBurn', no sucede con 'Hard Light'. El modo de blending 'Hard Light' se define como aplicar el modo 'Overlay' con las imágenes argumento intercambiadas, pero la función que se encargará de aplicar los modos de blending a los pixeles debe tener en cuenta los pixeles alpha al aplicar el mezclado, por lo cual solamente intercambiar las imágenes argumento se interpreta como cambiar el orden de las "capas" (si 'x' estaba encima de 'y' ahora 'y' esta encima de 'x'), dando un resultado indeseado. Por lo que no podremos componer 'Hard Light' a partir de la función de 'Overlay' y pasará a ser una función primitiva.

Para la monada principal hice uso del concepto de "transformadores de monadas", que se basa en combinar los efectos de diferentes monadas. En mi caso necesitaba combinar una monada error con la monada IO. Dicha monada me permite, una vez realizado la evaluación del término, dar un resultado encapsulado en la monada IO () (ya sea teniendo una imagen resultado o un mensaje de error) necesario para que el resultado de la monada original se vea reflejado en la salida de la computadora. Se requirió el uso de transformadores de monadas ya que la biblioteca de imágenes utilizada devuelve una imagen leída encapsulada en una monada IO, por lo cual definir una nueva monada

que contenga los comportamientos de IO con una monada de error no era suficiente.

Decidí utilizar una biblioteca para parsear argumentos opcionales por que creo que es de gran utilidad darle control al usuario de donde se guarda la imagen resultado (como así también su nombre y su formato) entre otras posibilidades, apuntando de esta manera a facilitar el uso para usuarios que quieran más control sobre lo que hace el lenguaje sin volverlo abrumador para quien no está familiarizado con él.

Se intentó trabajar con imágenes con definición de canales en Float en lugar de Double, pero como la lectura de imágenes y conversiones entre espacios de colores daban como resultado imágenes y pixeles con precisión Double, la constante conversión requerida para trabajar con las imágenes en precision Float resultaba en una evaluación más lenta.

Bibliografia

- https://github.com/prod80/prod80-ReShade-Repository
- https://en.wikipedia.org/wiki/Alpha_compositing
- https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdf_reference_archive/blend_modes.pdf
- http://www.simplefilter.de/en/basics/mixmods.html
- https://en.wikibooks.org/wiki/Haskell/Monad_transformers
- https://hackage.haskell.org/package/hip
- https://hackage.haskell.org/package/options