Universidad de Santiago de Chile

Facultad de Ingeniería Informática

Paradigmas de Programación

Profesor Roberto González

Laboratorio 2
Paradigma Lógico

Nicolás Aguilera

Noviembre 2022

Índice

1. Introducción

El siguiente informe corresponde al desarrollo del Laboratorio 2 del curso *Paradigmas de Programación*, el cual consiste en la aplicación del *Paradigma Lógico* de programación a la resolución de un problema específico mediante el uso del lenguaje *Prolog* y su propio intérprete.

1.1. Descripción del problema

El problema consiste en la creación de predicados para el tratamiento imágenes RGB-D, similar a herramientas como GIMP y Adobe Photoshop, pero de forma simplificada. Para esto, se considera que las imagenes tienen un alto y un ancho (width x height) y que están compuestas por tres tipos de pixeles diferentes, pudiendo formar tres tipos de imagenes diferentes:

- **bitmap-d:** El valor que pueden tomar sus pixeles es de 0 o 1, indicando que el pixel se encuentra apagado o encendido (negro o blanco).
- **pixmap-d:** Los pixeles tienen tres canales de colores (R, G y B) los cuales toman valores de 0 a 255.
- **hexmap-d:** Los pixeles expresan sus canales RGB en formato hexadecimal.

La posición de cada pixel dentro de la imagen y la profundidad corresponden a valores enteros entre cero y el máximo, dependiendo de las dimensiones de la imagen y la profundidad de esta.

1.2. Descripción del paradigma

El paradigma lógico se enfoca en la creación de una base de conocimientos mediante la declaración de hechos y reglas (cláusulas) sobre objetos, y la relación entre estos, pudiendo ser entendidos mediante un lenguaje lógico-matemático. A diferencia del paradigma funcional en este caso si se cuenta con variables, las cuales pueden ser de diferentes tipos. A continuación se presentan algunos de los conceptos más importantes del paradigma:

- **Hechos y reglas:** Definen la relación entre objetos y conforman lo que se conoce como *base de conocimientos*, la que puede ser consultada posteriormente mediante el intérprete de Prolog. Mientras que los *hechos* son cláusulas siempre verdaderas, las *reglas* definen la relación entre objetos mediante la conjunción de metas, es decir el cumplimiento de otras cláusulas que se conocen como antecedentes.
- **Unificación y backtracking:** Proceso mediante el cual *Prolog* sustituye las variables por valores para que reglas y hechos consultados sean verdaderos al ser consultadas en la base de conocmientos, esto a través de un algoritmo de backtracking conocido como *SLD Resolution*.

- **Recursión:** Al igual que el paradigma funcional y a diferencia de paradigmas como el Imperativo, la inexistencia de **ciclos** para recorrer listas obliga el uso de la recursividad, en este caso natural.
- **Listas:** En *Prolog* las listas son la unidad base de estructuras más complejas las cuales se componen de una cabeza (head) y una cola (tail). Estas pueden ser conformadas por tipos de datos heterogéneos como podrían ser enteros, strings o átomos.

2. Desarrollo

2.1. Análisis del problema

El foco principal del problema es el tratamiento de imágenes mediante hechos y reglas utilizando la programación lógica. Por lo tanto, es necesario hacer mencionar y describir los elementos que conforman el problema y así orientar una solución hacia la construcción e interacción de estos elementos:

- **Imagen:** Unidad fundamental del problema. Está constituida por un alto y ancho, así como como pixeles, los cuales deben responder a las dimensiones de la imagen (alto x ancho).
- **Pixel:** Las imágenes están conformadas por sub-unidades llamadas *pixeles*, los cuales pueden ser de diferentes tipos: pixbit-d, pixrgb-d y pixhex-d, conformando los bitmap-d, pixmap-d y hexmap-d, respectivamente. Los pixeles a su vez, tienen una posición dentro de la imagen representada por coordendas x e y, un color (especificado anteriormente en *Descripción del problema*) y una profundidad.
- **Color:** Cada pixel tiene un color el cual es representado por un valor entero (o varios en el caso del pixrgb-d) o una cadena de caracteres (en hexadecimal) (en el caso de los pixhex-d).
- **Profundidad:** Además, los pixeles tienen una profundidad, que representa la "lejanía.[a] la que se encuentra ese pixel, desde el plano de la pantalla u hoja. Esta característica ayuda a la representación tridimensional de una imagen.

2.2. Diseño de la solución

2.2.1. TDAs implementados

Tal y como se mencionó anteriormente, la solución para el tratamiento de imágenes recae en una buena abstracción y representación los TDAs que son parte del problema, de tal manera que la implementación de la solución sea lo más sencilla posible y no caiga en redundancias. Esto significa que cada TDA debe estar compuesto por *Constructores*, *Funciones de Pertenencia*, *Selectores* y *Modificadores* cumpliendo cada uno sus roles específicos a la hora de implementar un TDA. En el caso particular de *Prolog*, para los TDA no fue necesaria la creación de selectores ni funciones de pertenencia dado que los

Constructores permiten la verificación de tipos de dato y la selección de un término en particular mediante el proceso de *unificación*.

Toda la implementación de los TDAs está sujeta a una representación basada en listas, por lo que fue importante considerar de qué manera se agrupaban los parámetros que conforman cada TDA. A continuación se describen los TDAs considerados para la solución del problema:

- **TDA image:** Su representación está dada por una lista que contiene los siguientes elementos: el **ancho** (width), el **alto** (height), el **color comprimido** de la imagen (el que será nulo si la imagen no está comprimida), una **lista de pixeles** que componen la imagen y por último una lista de **pixeles comprimidos** que contiene información reducida de los pixeles que no están en la imagen y será utilizada para su posterior recuperación (descompresión).
- **TDA Pixel:** Su representación está dada por una lista que contiene los siguientes elementos: una lista de dos elementos (**posición** x e y del pixel en la imagen), **color** del pixel (representación binaria, canales RGB y representación RGB hexadecimal) y **profundidad** del pixel.

Debido a que todos los tipos de pixeles contienen los mismos parámetros (posición, color y profundidad), se decidió considerar el **TDA Pixel** como TDA padre para la representación de los distintos tipos de pixeles. Esto con el fin de acceder de manera más sencilla a cada uno de los argumentos del pixel. Para un detalle más exhaustivo de los TDAs y sus predicados, revisar los Cuadros 1, 2 y 3 del *Anexo*.

2.2.2. Algoritmos y técnicas empleadas

Tal y como se especificó en el material entregado para el desarrollo del proyecto, todos los TDAs y algoritmos asociados a la implementación de funciones se basó en listas, por lo que los subproblemas que requerían de modificaciones de listas, utilizaron la *recursión natural* al ser el método que más se acomoda al lenguaje de programación dado que la mayoría de los subproblemas consistían en modificaciones y construcciones de listas a partir de otras listas.

2.3. Aspectos de implementación

2.3.1. Estructura del proyecto

Se utilizaron archivos diferentes para ambos TDAs implementados: **TDAImage.pl**[1] y **TDAPixel.pl**[2]. Además se tiene un tercer archivo, el cual es un script de prueba que muestra y ejemplifica el uso de todas las funciones requeridas para la implementación de la solución. Este archivo tiene el nombre de **prueba_19527704_AguileraGonzalez.pl**.

[1]Realmente llamado: **TDAImage_19527704_AguileraGonzalez.pl**
[2]Realmente llamado: **TDAPixel_19527704_AguileraGonzalez.pl**

2.3.2. Compilador y bibliotecas utilizadas

El lenguaje de programación utilizado es *SWI-Prolog* y su intérprete en su versión 8.4.3, del cual se utilizó toda la librería disponible en la página oficial del lenguaje, como parte de las exigencias del Laboratorio.

2.4. Instrucciones de uso

Para hacer uso de la implementación propuesta, se deben tener los tres archivos mencionados anteriormente y consultar la base de conocimiento en la terminal de *SWI-Prolog* haciendo uso del archivo de script de pruebas. En caso de no utilizar este script, se deben utilizar los *Constructores* del TDA Image y el TDA Pixel para construir una imagen. Luego, para tratar la imagen creada, se pueden consultar cualquiera de los predicados especificados en los *Requerimientos Funcionales* y que se encuentran en el Cuadro 1 y 2 del *Anexo*. Algunos ejemplos de estas funciones aplicadas a imagenes tipo *bitmap* y *pixmap* pueden encontrarse en las Figuras 1 y 2 de la Sección 2 del *Anexo*.

2.4.1. Resultados esperados

Se espera que el usuario pueda crear y modificar imagenes mediante las consultas a la base de conocimientos logrando rotar, voltear, comprimir e incluso mostrar los pixeles en pantalla, entre otras, visualizando los resultados en la terminal del intérprete de *SWI-Prolog* según la representación de los TDAs especificados con anterioridad.

2.4.2. Posibles errores

Dado que se asumió que el usuario utilizaría las funciones de forma adecuada y por lo tanto no introducirá valores fuera de los límites o tipos de datos inadecuados, puede que al hacer esto, las imágenes que cree el usuario y las modificaciones que le haga, provoquen errores que serán mostrados en la terminal del intérprete.

2.5. Resultados y autoevaluación

En general, los resultados obtenidos son los esperados, logrando la creación y manipulación de imágenes a través de la aplicación de funciones usando la programación funcional.

Los criterios y resultados de la *Autoevaluación* de requisitos funcionales pueden encontrarse en los Cuadros 5 y 6 del Anexo.

2.5.1. Funciones no completadas

Se cumplió con todos los requisitos funcionales de esta entrega, por lo que no hay funciones incompletas.

3. Conclusión

Luego del desarrollo del Laboratorio, se puede concluir que se cumplió con los requisitos generales y particulares del proyecto. En primer lugar, crear predicados para el tratamiento de imágenes utilizando la programación lógica a través del lenguaje de programación *SWI-Prolog* y su intérprete. También se cumplió con la creación de TDAs acorde a los requisitos funcionales del proyecto, logrando la abstracción y representación de cada tipo de dato, en particular, para el TDA Image y TDA Pixel implementados en la solución.

En cuanto a las dificultades para utilizar el paradigma lógico, al igual que para la entrega del Laboratorio 1, se encuentran la costumbre de utilizar lenguajes de programación basados en el paradigma imperativo, tales como *C* y *Python*, por lo que la transición a este paradigma prescindiendo de ciclos y hacer uso reiterativo de la recursión natural. Está también el tiempo invertido en aprender los conceptos que rodean el paradigma y el lenguaje de programación en particular y su notación poco común.

Por último, la solución propuesta para este laboratorio, fue en su mayoría la solución entregada para el Laboratorio 1 pero trasladada al paradigma lógico ya que el trabajo con listas era similar al realizado en Scheme.

4. Bibliografía y referencias

1. Escrit M., Toledo F. & Pacheco J. (2001). *El lenguaje de Programación Prolog*. Recuperado de: `http://mural.uv.es/mijuanlo/PracticasPROLOG.pdf`

2. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio*. Recuperado de: Enunciado General del Proyecto Semestral

3. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio. Laboratorio 2*. Recuperado de: Enunciado Laboratorio 2

5. Anexo

5.1. Sección 1

TDA Image - Requerimientos		
Nombre	**Tipo de predicado**	**Descripción**
image	Constructor	Crea una imagen de tipo bitmap-d, pixmap-d o hexmap-d
imageIsBitmap	Pertenencia	Determina si una imagen es tipo bitmap-d.
imageIsPixmap	Pertenencia	Determina si una imagen es tipo pixmap-d.
imageIsHexmap	Pertenencia	Determina si una imagen es tipo hexmap-d.
imageIsCompressed	Pertenencia	Determina si una imagen está comprimida.
imageFlipH	Modificador	Voltea una imagen de forma horizontal
imageFlipV	Modificador	Voltea una imagen de forma vertical
imageCrop	Modificador	Recorta una imagen dado un cuadrante determinado por dos puntos (x_1, y_1) y (x_2, y_2)
imageRGBtoHex	Modificador	Transforma una imagen tipo pixmap-d en una tipo hexmap-d
imageRotate90	Modificador	Rota una imagen 90 grados en sentido antihorario
imageCompress	Modificador	Comprime una imagen eliminando los pixeles con el color más frecuente
imageChangePixel	Modificador	Cambia un pixel de una imagen por otro.
imageInvertColorRGB	Modificador	Cambia el valor de los canales RGB al valor simétricamente opuesto
imageDecompress	Modificador	Descomprime una imagen, recuperando los pixeles previamente eliminados junto a su información
imageToString	Otras	Muestra una imagen con sus pixeles representados como string
imageDepthLayers	Otras	Crea una lista de imagenes que contienen pixeles de la misma profundidad
imageToHistogram	Otras	Entrega una lista de los colores que componen la imagen junto a su frecuencia

Cuadro 1: TDA Image y sus requerimientos funcionales.

TDA Image - Predicados adicionales		
Nombre	**Tipo de predicado**	**Descripción**
pixlistIsBit	Pertenencia	Determina si una lista de pixeles contiene pixbit.
pixlistIsRGB	Pertenencia	Determina si una lista de pixeles contiene pixrgb.
pixlistIsHex	Pertenencia	Determina si una lista de pixeles contiene pixhex.
pixelFlipH	Otras	Modifica la posición X de un pixel al ser volteado horizontalmente.
pixlistFlipH	Otras	Modifica la posición X de una lista de pixeles luego de ser volteados horizontalmente.
pixelFlipV	Otras	Modifica la posición Y de un pixel al ser volteado verticalmente.
pixlistFlipV	Otras	Modifica la posición Y de una lista de pixeles luego de ser volteados verticalmente.
insideCrop	Otras	Verifica si un pixel de la imagen se encuentra dentro del cuadrante de corte
pixlistCrop	Otras	Entrega una lista con los pixeles que se encuentran dentro del cuadrante.
newSize	Otras	Modifica las dimensiones de la imagen luego de ser cortada.
pixlistRGBtoHex	Otras	Cambia los pixrgb de una lista de pixeles a pixhex.
imageColors	Otras	Crea una lista con los colores de los pixeles de la imagen incluidos los repetidos.
removeDuplicateColors	Otras	Elimina los colores repetidos de una lista de colores.
colorFreqList	Otras	Crea una lista de frecuencias de los colores de la imagen.
colorFreq	Otras	Entrega la frecuencia de un color determinado a partir de una lista de colores.
histogram	Otras	Entrega una lista del tipo [Color,Freq] para todos los colores de una imagen.
mostFreqColor	Otras	Entrega el color más frecuente de la imagen
pixlistRotate90	Otras	Rota 90 grados todos los pixeles de una lista de pixeles.

Cuadro 2: Funciones adicionales del TDA Image.

TDA Image - Predicados adicionales		
Nombre	**Tipo de predicado**	**Descripción**
removeMostFreqColor	Otras	Remueve todos los pixeles de la imagen que tengan el color más frecuente.
compressedPixels	Otras	Retorna una lista con los pixeles comprimidos
pixlistToString	Otras	Retorna la imagen bitmap-d sin modificaciones si es de ese tipo
imageDepths	Otras	Entrega una lista con las profundidades que tiene la imagen incluidas las repetidas.
removeDuplicatedDepths	Otras	Elimina las profunidades repetidas de una lista de profundidades.
changeToWhitePixlist	Otras	Cambia el color a blanco y profunidad de un pixel.
depthImages	Otras	Entrega una lista de imagenes donde todos los pixeles de cada imagen tienen la misma profundidad.
imageDepthLayers	Otras	Entrega una lista de imagenes donde todos los pixeles de cada imagen tienen la misma profundidad.
backPixelsToPixlist	Otras	Entrega una lista de pixeles luego de agregar los pixeles comprimidos previamente.
decompressPixels	Otras	Entrega los pixeles comprimidos en un formato para ser integrados nuevamente a la imagen.

Cuadro 3: Funciones adicionales del TDA Image.

TDA Pixel		
Nombre	**Tipo de predicado**	**Descripción**
pixbit	Constructor	Crea un pixel tipo pixbit-d
pixrgb	Constructor	Crea un pixel tipo pixrgb-d
pixhex	Constructor	Crea un pixel tipo pixhex-d
bitColor	Pertenencia	Verifica si un color es de tipo bitColor (0 o 1).
rgbColor	Pertenencia	Verifica si un color es de tipo rgbColor.
hexColor	Pertenencia	Verifica si un color es de tipo hexadecimal.
changePixel	Otros	Cambia la información de un pixel de una imagen.
imageChangePixel	Modificador	Verifica si un color es de tipo rgbColor.

Cuadro 4: Funciones que conforman el TDA Pixel.

Evaluación	Descripción
0	No se cumple o no funciona
0.25	Falla la mayoría de las veces (funciona el 25 % de las veces)
0.5	Funcionamiento irregular (50 % de las veces)
0.75	Funcionamiento con problemas menores (75 % de las veces funciona)
1	Se cumple o funciona todo el tiempo

Cuadro 5: Evaluación y descripción utilizada para la Autoevaluación.

Función	Evaluación
TDAs	1
image	1
imageIsBitmap	1
imageIsPixmap	1
imageIsHexmap	1
imageIsCompressed	1
imageFlipH	1
imageFlipV	1
imageCrop	1
imageRGBtoHex	1
imageToHistogram	1
imageRotate90	1
imageCompress	1
imageChangePixel	1
imageInvertColorRGB	1
imageToString	1
imageDepthLayers	1
imageDecompress	1

Cuadro 6: Requerimientos funcionales y su evaluación.

5.2. Sección 2

```
?- pixbit( 0, 0, 1, 10, PA), pixbit( 0, 1, 0, 20, PB), pixbit( 1, 0, 0, 30, PC),
 pixbit( 1, 1, 1, 4, PD), image( 2, 2, [PA, PB, PC, PD], I), imageIsBitmap(I), i
mageToString(I, Str),write(Str).
1 0
0 1
PA = [[0, 0], 1, 10],
PB = [[0, 1], 0, 20],
PC = [[1, 0], 0, 30],
PD = [[1, 1], 1, 4],
I = [2, 2, [], [[[0, 0], 1, 10], [[0, 1], 0, 20], [[1, 0], 0, 30], [[1, 1], 1, 4
]], []],
Str = "1 0\n0 1\n"
```

Figura 1: Creación de una imagen pixmap e impresión en pantalla.

```
?- pixhex( 0, 0, "#FF0000", 20, PA), pixhex( 0, 1, "#FF0000", 20, PB), pixhex( 0
, 2, "#FF0000", 20, PC), pixhex( 1, 0, "#0000FF", 30, PD), pixhex( 1, 1, "#0000F
F", 4, PE), pixhex( 1, 2, "#0000FF", 4, PF), pixhex( 2, 0, "#0000FF", 4, PG), pi
xhex( 2, 1, "#0000FF", 4, PH), pixhex( 2, 2, "#0000FF", 4, PI), image( 3, 3, [PA
, PB, PC, PD, PE, PF, PG, PH, PI], I), imageCrop( I, 1, 1, 2, 2, I2), pixhex( 0,
 0, "#0000FF", 4, PE2), pixhex( 0, 1, "#0000FF", 4, PF2), pixhex( 1, 0, "#0000FF
", 4, PH2), pixhex( 1, 1, "#0000FF", 4, PI2), image( 2, 2, [PE2, PF2, PH2, PI2],
 I3).
PA = [[0, 0], "#FF0000", 20],
PB = [[0, 1], "#FF0000", 20],
PC = [[0, 2], "#FF0000", 20],
PD = [[1, 0], "#0000FF", 30],
PE = PI2, PI2 = [[1, 1], "#0000FF", 4],
PF = [[1, 2], "#0000FF", 4],
PG = [[2, 0], "#0000FF", 4],
PH = [[2, 1], "#0000FF", 4],
PI = [[2, 2], "#0000FF", 4],
I = [3, 3, [], [[[0, 0], "#FF0000", 20], [[0, 1], "#FF0000", 20], [[0, 2], "#FF0
000", 20], [[1, 0], "#0000FF", 30], [[1, 1], "#0000FF", 4], [[1, 2], "#0000FF",
4], [[2, 0], "#0000FF", 4], [[2, 1], "#0000FF", 4], [[2, 2], "#0000FF", 4]], []]
,
I2 = I3, I3 = [2, 2, [], [[[0, 0], "#0000FF", 4], [[0, 1], "#0000FF", 4], [[1, 0
], "#0000FF", 4], [[1, 1], "#0000FF", 4]], []],
PE2 = [[0, 0], "#0000FF", 4],
PF2 = [[0, 1], "#0000FF", 4],
PH2 = [[1, 0], "#0000FF", 4]
```

Figura 2: imageCrop aplicado a una imagen tipo hexmap.

```
?- pixrgb( 0, 0, 33, 33, 33, 10, PA), pixrgb( 0, 1, 44, 44, 44, 10, PB), pixrgb(
 1, 0, 55, 55, 55, 30, PC), pixrgb( 1, 1, 66, 66, 66, 30, PD), image( 2, 2, [PA,
 PB, PC, PD], I), imageDepthLayers(I, [PRIMERA, SEGUNDA]), pixrgb( 0, 0, 33, 33,
 33, 10, PA2), pixrgb( 0, 1, 44, 44, 44, 10, PB2), pixrgb( 1, 0, 255, 255, 255,
10, PC2), pixrgb( 1, 1, 255, 255, 255, 10, PD2), image( 2, 2, [PA2, PB2, PC2, PD
2], I2), pixrgb( 0, 0, 255, 255, 255, 30, PA3), pixrgb( 0, 1, 255, 255, 255, 30,
 PB3), pixrgb( 1, 0, 55, 55, 55, 30, PC3), pixrgb( 1, 1, 66, 66, 66, 30, PD3), i
mage( 2, 2, [PA3, PB3, PC3, PD3], I3).
PA = PA2, PA2 = [[0, 0], [33, 33, 33], 10],
PB = PB2, PB2 = [[0, 1], [44, 44, 44], 10],
PC = PC3, PC3 = [[1, 0], [55, 55, 55], 30],
PD = PD3, PD3 = [[1, 1], [66, 66, 66], 30],
I = [2, 2, [], [[[0, 0], [33, 33, 33], 10], [[0, 1], [44, 44, 44], 10], [[1, 0],
 [55, 55, 55], 30], [[1, 1], [66, 66, 66], 30]], []],
PRIMERA = I2, I2 = [2, 2, [], [[[0, 0], [33, 33, 33], 10], [[0, 1], [44, 44, 44]
, 10], [[1, 0], [255, 255, 255], 10], [[1, 1], [255, 255, 255], 10]], []],
SEGUNDA = I3, I3 = [2, 2, [], [[[0, 0], [255, 255, 255], 30], [[0, 1], [255, 255
, 255], 30], [[1, 0], [55, 55, 55], 30], [[1, 1], [66, 66, 66], 30]], []],
PC2 = [[1, 0], [255, 255, 255], 10],
PD2 = [[1, 1], [255, 255, 255], 10],
PA3 = [[0, 0], [255, 255, 255], 30],
PB3 = [[0, 1], [255, 255, 255], 30]
```

Figura 3: imageDepthLayers para una imagen tipo pixmap.