UNIVERSIDAD
DE SANTIAGO
DE CHILE

# INFORME DE LABORATORIO 2:
# SIMULACIÓN DE EDITOR DE IMÁGENES EN PROLOG

**Nombre:** Diego Armijo
**Profesor:** Victor Flores
**Fecha:** 03/11/2022

## TABLA DE CONTENIDOS

# 1. INTRODUCCIÓN

Un paradigma es todo aquel modelo, patrón o ejemplo que debe seguirse en determinada situación. En la programación, se refieren a este como una manera o estilo de programación de software. Existen diferentes formas de diseñar un lenguaje de programación y varios modos de trabajar para obtener los resultados que necesitan los programadores. Se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas computacionales. En este informe se introducirá un problema y su respectiva solución en el paradigma lógico, para ser más específicos, en el lenguaje de programación lógico Prolog.

## 1.1 DESCRIPCIÓN DEL PROBLEMA

Se pide desarrollar un programa tipo “Photoshop”, este tiene que permitir crear una imagen y hacerle ciertas modificaciones las cuales describiremos más adelante. Esta imagen se rige por ciertas propiedades que debemos tener en cuenta:

- **image:** En base a ciertos parámetros, se crea una imagen que contiene el ancho, alto y los píxeles de la imagen.
- **pixbit:** Tipo de píxel que acepta la imagen, este es de tipo bit.
- **pixrgb:** Tipo de píxel que acepta la imagen, este es de tipo rgb.
- **pixhex:** Tipo de píxel que acepta la imagen, este es de tipo hexadecimal.

## 1.2 DESCRIPCIÓN DEL PARADIGMA

El paradigma lógico es un paradigma declarativo y se basa en los conceptos de lógica matemática, donde se crean predicados que caracterizan o relacionan a los individuos involucrados y la deducción de las posibles respuestas a una determinada consulta. Contiene una base de conocimientos, la cual contiene los predicados de un programa. Los predicados son cosas que se quieren decir y estos están definidos por cláusulas (hechos y reglas). Dentro de este paradigma hay una serie de definiciones que se tienen que conocer, ya que son altamente relevantes a la hora de escribir código, y estas son:

- **Unificación:** Mecanismo que se emplea como paso de parámetros, donde las variables lógicas toman valor en Prolog. El valor que puede tomar una variable consiste en cualquier término.

- **Backtracking:** Cuando uno realiza una consulta, pueden existir más de un hecho a la base de conocimiento, por lo tanto si al momento de realizar una consulta existe más de una coincidencia o caso verdadero, Prolog realiza una “vuelta atrás” y devuelve el siguiente de los conocimientos que presenta la consulta.
- **Backfoward:** Resuelve los predicados cuando estos hayan quedado pendientes, es muy similar a la recursión de pila. Cuando hay estados pendientes, realiza la unificación.
- **Términos:** Son el único elemento del lenguaje, es decir, los datos son términos, el código son términos, incluso el mismo programa es un término. Un término se compone de un **functor/átomo** seguido de cero a N argumentos entre paréntesis y separados por comas.
- **Predicados:** Son las cosas que queremos decir. Los resultados o variables van en mayúscula.
- **Consultas:** Son aquellas preguntas que se hacen mediante consola y donde Prolog busca en su base de conocimiento para entregar una respuesta booleana.
- **Hechos:** También llamados cláusulas sin cuerpo. Si no existen condiciones para que una cláusula sea cierta se puede omitir el cuerpo. En tal caso solamente escribimos la cabeza terminada en un punto. Ejemplo: es un hecho que la edad de Diego es 23 años.

En Prólogo hay que evitar el problema de mundo cerrado que se puede generar al negar un predicado/hecho y a su vez que el resultado que se espere no se haya definido. Cuando existe un falso a una respuesta, esta no significa que sea un falso absoluto, solo significa que se dio el caso en que Prolog no fue capaz de encontrar un hecho/resultado que satisfaga la pregunta y, por lo tanto, retornar un falso.
Los hechos y/o reglas comienzan con mayúscula. Es muy importante saber esto ya que en Prolog las mayúsculas y minúsculas tienen funciones diferentes.

## 1.3 OBJETIVOS

El objetivo principal del proyecto es aprender sobre el paradigma y la programación lógica, para así obtener nuevas herramientas que nos faciliten la forma de resolver un problema y así también salir de la forma tradicional de programar. Otro objetivo es programar correctamente en Prolog y aprender a utilizar las herramientas que este nos brinda para completar el proyecto de laboratorio.

# 2. DESARROLLO

## 2.1 ANÁLISIS DEL PROBLEMA

Para generar una imagen, antes hay que crear los píxeles de esta. Cada píxel se rige por ciertas reglas y condiciones, es por eso que hay verificar que los parámetros que ingresa el usuario sean valores que estén dentro del rango aceptado por las condiciones del píxel. También, para poder generar una imagen es necesario saber las dimensiones, ya que en base a estas tendremos que verificar que las coordenadas de cada píxel que ingrese el usuario sean correctas y que la lista que contiene los píxeles sea del tamaño correcto. Una vez verificado todo podremos generar la imagen y manipularla según se nos solicite.

**pixbit:** Corresponde a un conjunto de asociaciones, donde un usuario genera un píxel de tipo bit. En Prolog se representará como una lista que contiene las coordenadas del píxel, el bit, la profundidad y el nombre del pixel generado (posX(int) x posY(int) x bit [0-1] x Depth(int) x PixelGenerado(list)).

**pixrgb:** Corresponde a un conjunto de asociaciones, donde un usuario genera un píxel de tipo rgb. En Prolog se representará como una lista que contiene las coordenadas del píxel, su color (R G B), la profundidad y el nombre del pixel generado (posX(int) x posY(int) x R(C) x G(C) x B(C) x D(int) x PixelGenerado(list)).

**pixhex:** Corresponde a un conjunto de asociaciones, donde un usuario genera un píxel de tipo hexadecimal. En Prolog se representará como una lista que contiene las coordenadas del píxel, su color representado de forma hexadecimal, la profundidad y el nombre del pixel generado (posX(int) x posY(int) x Hex(string) x D(int) x PixelGenerado(list)).

**image:** Corresponde a un conjunto de asociaciones, donde un usuario genera una imagen. En Prolog se representará como una lista que contiene las dimensiones de la imagen, una lista con los pixeles y el nombre de la imagen generada (Width(int) x Height(int) x [pixbit* | pixrgb* | pixhex*] x Image(list)). Es necesario tener los creadores de pixbit-d | pixrgb-d | pixhex-d para que la imagen pueda ser creada.

Cada **TDA** (Tipo de Dato Abstracto) está dentro de un archivo único, por lo tanto solo bastará con consultar al único y mismo archivo .pl.

## 2.2 DISEÑO DE SOLUCIÓN

Para el diseño de solución utilizamos los tipos de datos específicos (TDA), donde primero irá la representación, constructores, predicados de pertenencia, selectores, modificadores y por último otros predicados. Para la solución implementamos cuatro TDA:

**TDA pixbit**

- **Representación:** Para la creación de un pixbit se necesitan las coordenadas, el bit, la profundidad y el nombre del píxel generado.
- **Constructor:** Dado las coordenadas, el bit, la profundidad y el nombre del pixel generado, este retornara en un pixbit válido, el cual es una lista con los elementos de entrada y algunos adicionales.
- **Predicado de Pertenencia, Selectores, Modificadores y Otros predicados** (ver anexos).

**TDA pixrgb**

- **Representación:** Para la creación de un pixrgb se necesitan las coordenadas, el color (R G B), la profundidad y el nombre del píxel generado.
- **Constructor:** Dado las coordenadas, el color rgb, la profundidad y el nombre del píxel generado, este retornará un pixrgb válido, el cual es una lista con los elementos de entrada y algunos adicionales.
- **Predicados de Pertenencia, Selectores, Modificadores y Otros predicados** (ver anexos).

**TDA pixhex**

- **Representación:** Para la creación de un pixhex se necesitan las coordenadas, el color hexadecimal, la profundidad y el nombre del píxel generado.
- **Constructor:** Dado las coordenadas, el color hexadecimal, la profundidad y el nombre del píxel generado, este retornará un pixhex válido, el cual es una lista con los elementos de entrada y algunos adicionales.
- **Predicados de Pertenencia, Selectores, Modificadores y Otros predicados** (ver anexos).

**TDA image (principal)**

- **Representación:** Para la creación de una imagen se necesita el ancho, alto, la lista que contiene los píxeles y el nombre de la imagen generada.
- **Constructor:** Dada la altura, ancho, la lista con los pixeles y el nombre de la imagen creada, este retornará una imagen válida, la cual contendrá los elementos de entrada y algunos adicionales.
- **Predicado de Pertenencia, Selectores, Modificadores, Otros predicados y las operaciones obligatorias** (ver anexos).

## 2.3 ASPECTOS DE IMPLEMENTACIÓN

### 2.3.1 COMPILADOR

Para este proyecto se solicitó usar **Swi-Prolog versión 8.4 o superior.** Como alternativa se puede utilizar el Visual Studio Code con la extensión de Swi-Prolog o hacer uso de SWISH (Prolog online).

### 2.3.2 ESTRUCTURA DEL CÓDIGO

El código se separa en 4 archivos:

- **tda-pixbit:** Contiene el constructor, la función de pertenencia y otros predicados.
- **tda-pixrgb:** Contiene el constructor, la función de pertenencia y otros predicados.
- **tda-pixhex:** Contiene el constructor, la función de pertenencia y otros predicados.

- **tda-image(principal):** Es el main del código, donde se encuentran todas las operaciones obligatorias. Para importar los otros TDA se utilizó **include/1**.

## 2.4 INSTRUCCIONES DE USO

### 2.4.1 EJEMPLOS DE USO

Para el correcto funcionamiento se recomienda utilizar el comando “set_prolog_flag(answer_write_options,[max_depth(0)]).” Swi-Prolog, ya que con este se podrá visualizar con totalidad los datos.

Una vez ejecutado se puede cargar la base de conocimientos. Para esto se debe ir a la opción File->Consult y luego seleccionar el archivo “tda-image_20223138_ArmijoPalominos.pl”. Si no ocurren errores saldrá un mensaje en color verde, lo que significa que la base de conocimientos fue cargada con éxito. Ahora es solo cosa de abrir el script de prueba e ir copiando los ejemplos para ejecutarlos en Swi-Prolog.

### 2.4.2 RESULTADOS ESPERADOS

Se espera poder crear una imagen válida, a la cual se le puedan aplicar predicados para modificarla y así simular el uso de un editor de imágenes. Todo esto mediante el paradigma lógico y todos los recursos que nos aporta, los cuales los mencionamos en el apartado **1.2 DESCRIPCIÓN DEL PARADIGMA**.

### 2.4.3 POSIBLES ERRORES

Los errores pueden ocurrir si se llegase a utilizar mal el orden de los predicados de las operaciones o bien, si no se utilizan correctamente las variables a la hora de ejecutar el código. Por lo demás no deberían existir errores, ya que está el script de prueba y en el main cada predicado comentado.

## 2.5 RESULTADOS Y AUTOEVALUACIÓN

### 2.5.1 RESULTADOS OBTENIDOS

Los resultados obtenidos fueron los esperados, ya que se logró implementar los TDA con sus respectivos predicados. El programa funciona en su totalidad excluyendo algunos predicados que por mal manejo de tiempo no se pudieron

implementar en el código, por lo demás,  se logra dar una exitosa solución a un problema en el paradigma lógico.

### 2.5.2 AUTOEVALUACIÓN

Se realizó de la siguiente forma: 0: No realizado - 0.25: Funciona 25% de las veces - 0.5: Funciona un 50% de las veces - 0.75: Funciona 75% de las veces - 1: Funciona el 100% de las veces. (ver tabla en anexos).

## 3. CONCLUSIÓN

Tras realizar y finalizar el proyecto, se puede concluir que se cumplió con el objetivo principal de este, ya que se logró introducir y comprender el paradigma lógico.

Cabe destacar que al comienzo del laboratorio se sintió bastante complejo, ya que la forma de programar es muy diferente a cualquiera que se haya visto antes (imperativo/funcional). Entender la idea de que absolutamente todo es un predicado y como las variables van tomando valores mediante la unificación fue un poco raro, pero al ir programando y adentrandome en el lenguaje cada vez se hacia mas facil, sobretodo para abarcar problemas que con otros paradigmas se vuelven más complejos. Si bien no se logró la resolución completa del proyecto, se completó la gran mayoría.

## 4. BIBLIOGRAFÍA Y REFERENCIAS


- Miriam. (2020, June 9). *¿Qué son los paradigmas de programación?* Profile Software Services. https://profile.es/blog/que-son-los-paradigmas-de-programacion/
- Significados. (2013, August 8). *Significado de Paradigma*. Significados; Significados. https://www.significados.com/paradigma/
- *4. Paradigma Lógico*. (2014, November 16). Lenguajes de Programación; Lenguajes de Programación. https://kevinldp.wordpress.com/4-paradigma-logico/

- *Tutorial básico de programación en Prolog*. (n.d.). https://www.dsi.fceia.unr.edu.ar/downloads/IIA/recursos/Tutorial%20de%20%20Prolog.pdf

## 5. ANEXOS

### TABLA NRO 1: Predicados TDA image

| Tipo de predicado | Nombre | Descripción |
|---|---|---|
| Predicado de pertenencia | **verifyImage** | Predicado que verifica si una imagen es válida. |
| Selector | **getListTypePixels** | Predicado que obtiene la lista con el type de cada píxel de una imagen. |
| Selector | **getListPixels** | Predicado que obtiene la lista de píxeles de una imagen. |
| Otros predicados | **isPixbit** | Predicado que verifica que una lista contiene solo “pixbit”. |
| Operaciones obligatorias | **imageIsBitmap** | Predicado que permite determinar si una imagen corresponde a un bitmap. |
| Otros predicados | **isPixmap** | Predicado que verifica si una lista contiene solo “pixrgb”. |
| Operaciones obligatorias | **imageIsPixmap** | Predicado que permite determinar si una imagen corresponde a un pixmap. |
| Otros predicados | **isHexmap** | Predicado que verifica que una lista contiene solo “pixhex”. |
| Operaciones obligatorias | **imageIsHexmap** | Predicado que permite determinar si una imagen corresponde a un hexmap. |
| Otros predicados | **listLength** | Predicado que obtiene el tamaño de una lista. |
| Operaciones obligatorias | **imageIsCompressed** | Predicado que determina si una imagen está comprimida. |
| Otros predicados | **addElement** | Predicado que agrega un elemento a una lista. |
| Operaciones obligatorias | **imageFLipH** | Predicado que permite invertir una imagen horizontalmente. |
| Modificador | **pixelIsFlipH** | Predicado que invierte horizontalmente una lista de píxeles. |
| Operaciones obligatorias | **imageFlipV** | Predicado que invierte verticalmente una imagen. |
| Modificador | **pixelIsFlipV** | Predicado que invierte verticalmente una lista de píxeles. |
| Operaciones obligatorias | **imageCrop** | Recorta una imagen a partir de un cuadrante. |

| | | |
|---|---|---|
| Otros predicados | **crop** | Predicado que dada una lista de pixeles, esta va agregando los pixeles que se encuentran dentro de un rango. |
| Otros predicados | **filterCropX** | Predicado que verifica que la coord X se encuentre dentro de un rango. |
| Otros predicados | **filterCropY** | Predicado que verifica que la coord Y se encuentre dentro de un rango. |
| Otros predicados | **maxElement** | Predicado que devuelve el número mayor entre dos números. |
| Operaciones obligatorias | **imageRGBToHex** | Predicado que transforma una imagen desde una representación RGB a una HEX. |
| Modificador | **rgbTohex** | Predicado que dada una lista de píxeles rgb los transforma a hex. |
| Otros predicados | **preBit** | Predicado que ordena y hace encode a una lista de píxeles tipo bit. |
| Otros predicados | **extractRGB** | Predicado que extrae los colores R G B y los ordena. |
| Otros predicados | **preRGB** | Predicado que hace encode a R G B. |
| Otros predicados | **preHex** | Predicado que ordena y hace encode a una lista de píxeles tipo hex. |
| Operaciones obligatorias | **imageToHistogram** | Predicado que retorna un histograma de la imagen. |
| Otros predicados | **extractAllColors** | Predicado que extrae en una lista los colores de una lista de pixeles. |
| Operaciones obligatorias | **imageRotate90** | Predicado que permite rotar en 90 grados una imagen. |
| Modificador | **pixelIsRotate90Aux** | Predicado que rota en 90 grados una lista de píxeles. |
| Operaciones obligatorias | **imageChangePixel** | Predicado que permite reemplazar un píxel en una imagen. |
| Otros predicados | **preImageChangePixelBit** | Predicado que cambia un pixel tipo bit en una lista de píxeles tipo bit. |
| Otros predicados | **preImageChangePixelRGB** | Predicado que cambia un pixel tipo rgb en una lista de píxeles tipo rgb. |
| Otros predicados | **preImageChangePixelHex** | Predicado que cambia un pixel tipo hex en una lista de píxeles tipo hex. |

| | | |
|---|---|---|
| Operaciones obligatorias | **imageInvertColorRGB** | Predicado que permite obtener el color simétricamente opuesto en cada canal dentro de un píxel. |
| Otros predicados | **maplist** | Predicado que modifica una lista en base a otro predicado. |
| Otros predicados | **equals** | Predicado que verifica que dos algo sean iguales. |
| Otros predicados | **encode transform** | Predicado que recibe una lista y hace que los duplicados consecutivos se agrupen en términos [NroDuplicados, Elemento]. |
| Otros predicados | **pack transfer** | Predicado que recibe una lista y separa en sublistas los elementos que están repetidos. |

## TABLA NRO 2: TDA pixbit

| Tipo de predicado | Nombre | Descripción |
|---|---|---|
| Predicado de pertenencia | **verifyPixBit** | Verifica que el píxel sea de tipo pixbit. |

## TABLA NRO 3: TDA pixrgb

| Tipo de predicado | Nombre | Descripción |
|---|---|---|
| Predicado de pertenencia | **verifyPixrgb** | Verifica que el píxel sea de tipo pixrgb. |
| Selector | **getR** | Obtiene el color R de un pixrgb. |
| Selector | **getG** | Obtiene el color G de un pixrgb. |
| Selector | **getB** | Obtiene el color B de un pixrgb. |

## TABLA NRO 4: TDA pixhex

| Tipo de predicado | Nombre | Descripción |
|---|---|---|
| Predicado de pertenencia | **verifyPixhex** | Verifica que el píxel sea de tipo pixhex. |
| Otros predicados | **esHex** | Predicado que determina si un número es hex. |

## TABLA NRO 5: Autoevaluación

| Requerimientos Funcionales | 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 | 0.5 |
| imageChangePixel | 1 |
| imageInvertColorRGB | 1 |
| imageToString | 0 |
| imageDepthLayers | 0 |
| imageDecompress | 0 |