Informe Laboratorio 2

Paradigma Lógico – Lenguaje SWI-Prolog

Fabián Alejandro Lizama González

Departamento de Ingeniería Informática

Facultad de Ingeniería, Universidad de Santiago de Chile

Paradigmas de Programación

Profesor Roberto González Ibañez

3 de noviembre de 2022

Índice

Introducción

En el informe se presenta el concepto de Paradigma Lógico (concepto detallado en el cuerpo del documento) aplicado al desarrollo de un software de manipulación de imágenes bajo un enfoque simplificado, bajo el lenguaje de programación “SWI-Prolog”.

Se desarrollará este software presentando el problema, analizándolo y comentando aspectos del proceso de solución e implementación de éste junto a algunas instrucciones y ejemplos de uso para terminar con el apartado de resultados y conclusiones.

Descripción del problema

Se solicita diseñar un software de manipulación de imágenes del tipo RGBD, las cuales tienen la particularidad de adicionalmente a los canales de colores (red, green, blue) se tiene un canal de profundidad para representar imágenes en tres dimensiones, con el objetivo de poder manipular a gusto este tipo de archivos mediante distintas funciones. Para esto se requiere implementar la solución bajo el paradigma funcional e implementar una representación basada en “Tipos de Datos Abstractos” (TDAs).

Descripción del Paradigma

El Paradigma Lógico mencionado en la introducción pertenece a la familia de los paradigmas declarativos y consiste en trabajar principalmente con consultas a una base de conocimientos ya existente, la cual cuenta con hechos (verdades conocidas) y reglas (condiciones que se tienen que cumplir para que algo sea verdad), este paradigma está enfocado en el ¿Qué? Y no en el ¿Cómo?, a diferencia de otros más conocidos.

Algunos aspectos de este paradigma a tratar en la solución son los siguientes:

Consultas: Son las preguntas que se realizan a la base de conocimientos, siempre tienen respuesta de True o False.

Clausulas o Hechos: Como se expresó anteriormente, los hechos son la parte principal de la base de conocimiento expresando verdades a las cuales se pueden realizar consultas.

Análisis del Problema

La principal complejidad del problema radica en el realizar el software solo bajo el paradigma lógico, nuevamente evitando la programación imperativa y abordando un “cambio de paradigma” al haber desarrollado esta solución anteriormente bajo el paradigma funcional.

Se implementarán abstracciones apropiadas para el problema con estructuras basadas principalmente en listas, junto a sus operaciones asociadas de construcción, selección, modificación y otras.

Se busca que se logren manipular imágenes digitales bajo un enfoque simplificado, con el formato RGBD, es decir, canales de colores y además un cuarto canal de profundidad para representar imágenes en 3D, además de poder realizar las siguientes operaciones:

- Recortar imagen
- Invertir una imagen horizontalmente
- Invertir una imagen verticalmente
- Comprimir imagen en base a eliminación del color con mayor frecuencia.
- Convertir a hexadecimal
- Visualizar la imagen
- Rotar imagen en 90° a la derecha
- Rotar imagen en 90° a la izquierda
- Histograma
- Descomprimir imagen en base a restitución del color con mayor frecuencia
- Aplicar operaciones como las anteriores a un área seleccionada dentro de la imagen
- Convertir imagen a blanco y negro
- Convertir imagen a escala de grises
- Editar una imagen a partir de la aplicación de funciones especiales sobre los pixeles
- Separar capas de una imagen 3D en base a la profundidad. De esta forma desde una imagen 3D se puede devolver una lista de imágenes 2D
- Redimensionar imagen

Diseño de la Solución

Para el desarrollo de esta se utilizan cuatro TDAs:

1. TDA image:

   Tipo de dato abstracto que representa una imagen, está conformada por una lista de forma:

   (string (Color) X int (W) X int (H) X list [pixbit | pixrgb | pixhex] (Pixlist))

   a. El primer elemento corresponde al color más frecuente en la imagen cuando esta se comprime, en formato hexadecimal, si la imagen no está comprimida se inicializa en un string vacío “”.
   b. Los elementos W y H corresponden al ancho y al largo de la imagen.
   c. Finalmente, el elemento Pixlist corresponde a la lista de pixeles homogéneos que conforman la imagen en sí.

2. TDA pixbit:

   Tipo de dato abstracto que representa un píxel de formato pixbit-d, está conformado por una lista de forma:

   (int (X) X int (Y) X [0|1] (Bit) X int (D))

   a. Los elementos X e Y corresponden a las coordenadas del píxel en la imagen.
   b. Los elementos Bit y D representan el color del píxel que puede ser blanco o negro y la profundidad que tiene el píxel en la imagen.

3. TDA pixrgb:

   Tipo de dato abstracto que representa un píxel de formato pixrgb-d, está conformado por una lista de forma:

   (int (X) X int (Y) X int (R) X int (G) X int (B) X int (D))

   a. Los elementos X e Y corresponden a las coordenadas del píxel en la imagen.
   b. Los elementos R, G, B y D representan respectivamente el código RGB del píxel junto a su profundidad en la imagen.

4. TDA pixbit:

   Tipo de dato abstracto que representa un píxel de formato pixbit-d, está conformado por una lista de forma:

   (int (X) X int (Y) X string (Hex) X int (D))

   a. Los elementos X e Y corresponden a las coordenadas del píxel en la imagen.
   b. El elemento Hex representa el color del píxel en hexadecimal sin el “#” y en minúsculas, por ejemplo “ffffff” representa el color negro y el elemento D representa la profundidad que tiene el píxel en la imagen.

Aspectos de Implementación

Para el proyecto se utiliza el lenguaje “SWI-Prolog” en su versión 8.4.3-1 for Microsoft Windows y de editor de texto “Sublime Text” en su versión “Build 4126” junto a la extensión “Prolog”, el programa se organiza en un repositorio con los siguientes archivos:

image_21081166_LizamaGonzalez.pl

> Archivo de código en el lenguaje de programación Prolog con la implementación del TDA image y sus operaciones asociadas.

pixbit_21081166_LizamaGonzalez.pl

> Archivo de código en el lenguaje de programación Prolog con la implementación del TDA pixbit-d y sus operaciones asociadas.

pixrgb_21081166_LizamaGonzalez.pl

> Archivo de código en el lenguaje de programación Prolog con la implementación del TDA pixrgb y sus operaciones asociadas.

pixhex_21081166_LizamaGonzalez.pl

> Archivo de código en el lenguaje de programación Prolog con la implementación del TDA pixhex y sus operaciones asociadas.

pruebas_21081166_LizamaGonzalez.pl

> Archivo de código en el lenguaje de programación Prolog con ejemplos de los requerimientos funcionales.

repositorio_21081166_LizamaGonzález.txt

> Archivo de texto con el link del repositorio privado de GitHub con el proyecto.

autoevaluacion_21081166_LizamaGonzález.txt

> Archivo de texto con la autoevaluación de cada requerimiento funcional y no funcional.

informe_21081166_LizamaGonzález.pdf

> El presente informe.

Instrucciones de uso

Para crear y utilizar los TDAs respectivos hay que respetar el dominio de cada predicado especificado en el archivo de código y además en el presente informe.

Ejemplos de creación de TDAs:

TDA pixbit constructor:

> pixbit(0, 0, 1, 10, Pb).
>
> Pb se unificará con un pixbit de forma [0, 0, 1, 10], es decir un pixel bit-d de coordenadas (0, 0), color blanco y profundidad 10.

TDA pixrgb constructor:

> pixrgb(0, 0, 255, 255, 255, 10, Pr).
>
> Pr se unificará con un pixrgb de forma [0, 0, 255, 255, 255, 10], es decir un pixel rgb-d de coordenadas (0, 0), color blanco y profundidad 10.

TDA pixhex constructor:

> pixhex(0, 0, “ffffff”, 10, Ph).
>
> Ph se unificará con un pixbit de forma [0, 0, “ffffff”, 10], es decir un pixel hex-d de coordenadas (0, 0), color blanco y profundidad 10.

TDA image constructor:

> Para la construcción del TDA image hay que crear antes en la misma consulta los pixeles que utilizará:
>
> pixbit(0, 0, 1, 10, P1), pixbit(0, 1, 1, 10, P2), pixbit(1, 0, 1, 10, P3), pixbit(1, 1, 1, 10, P4), image(2, 2, [P1, P2, P3, P4], I).
>
> I se unificará con una imagen de tipo bitmap 2x2 de color blanco.

Resultados y autoevaluación

Se espera obtener un buen resultado sin fallos en el script de pruebas, podría llegar a fallar algún requisito funcional si no se respetan los dominios correspondientes, además se tomó la decisión de no implementar el predicado “imageDecompress” por falta de tiempo.

Conclusiones

Utilizar el paradigma Lógico fue un desafío importante luego de acostumbrarse al paradigma funcional realizando el laboratorio anterior, lo más complejo fue utilizar la recursión en los predicados ya que esta no se ve tan clara al momento de programar, a diferencia de otros lenguajes, se desarrolló la solución sin problemas mayores (omitiendo el imageDecompress ya que no implementarlo era lo esperado), por lo cual se obtuvieron los resultados esperados.