UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Paradigmas de programación

Laboratorio: *Paradigma lógico* (Stackoverflow)

Profesores: Roberto González

Camila Marquez

Alumna: Javiera Tapia

Santiago, Chile

03 de diciembre de 2020

Índice

Tabla de contenidos

Introducción

La programación es una disciplina enriquecida y los lenguajes de programación son usualmente complicados. Afortunadamente, las ideas más importantes de la programación son simples. Es así como nacen los paradigmas de programación, influenciando y diseñando el lenguaje por medio de un set de conceptos basados en la teoría matemática, presentando una serie de principios coherentes.

En este laboratorio se hablará específicamente sobre el *paradigma lógico* y su implementación.

Se presentará a continuación la creación de un clon de la aplicación **Stackoverflow** --sitio de preguntas y respuestas para programadores profesionales y entusiastas de la programación--, haciendo uso del lenguaje de programación Prolog (perteneciente al paradigma lógico) y sus lineamientos, que permitirá la construcción pertinente de Tipos de Dato Abstracto (TDA) y su implementación.

Descripción breve del problema

El stack tecnológico del foro Stackoverflow está, originalmente, basado en un lenguaje de programación orientado a objetos[1] (*server side*), sin embargo, es posible simular y reconstruir gran parte de sus componentes bajo otros lineamientos, por ejemplo, los que plantea el paradigma lógico. En esta ocasión, la problemática principal se centra en cubrir elementos claves como el manejo de usuarios registrados, preguntas, respuestas, etiquetas, etc y algunos de sus comportamientos (login, register, ask, vote, etc) que provee la plataforma real, haciendo uso del lenguaje Prolog.

Por otra parte, la implementación estará atada a requerimientos específicos como el uso de algunas técnicas y herramientas que reconstruirán estos componentes y comportamientos de la plataforma, tales como: recursión (natural y de cola), backtracking, unificación y estructura de datos basada en árboles.

1 Jeff Atwood, "What was Stackoverflow built with?", *The Overflow Blog*, https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/

Descripción breve del paradigma

Para pasar a describir las problemáticas y soluciones propuestas para cada uno de los conceptos requeridos por este laboratorio, es importante definir algunos elementos clave que le darán sentido y propósito a las distintas implementaciones y diseños escogidos:

1. ***TDA****:* Los *Tipo de Datos Abstractos* (TDA) se caracterizan por sus operaciones que pueden ser clasificadas en: *constructores, funciones de pertenencia, modificadores y otras implementaciones*. Un buen TDA es simple, coherente, adecuado, donde su representación es independiente[2].
2. ***Paradigma lógico****:* Paradigma que se basa en la lógica formal. Cualquier programa escrito bajo este paradigma se basará en una serie de sentencias en una forma lógica, expresando hechos y reglas sobre el dominio de un problema.
3. ***Backtracking:*** Backtracking es básicamente una forma de búsqueda. En el proceso de backtracking, se volverá atrás en búsqueda de una meta previa y luego, trataremos de encontrar otra forma de satisfacer esa meta.
4. ***Unificación:*** La manera en que Prolog coincide (*match*) dos términos se llama unificación. La idea es similar a la unificación en lógica: tenemos dos términos que queremos ver si representan la misma estructura.
5. ***Estructuras de datos basada en árboles:*** Un árbol es una estructura con una definición puramente recursiva ya que se puede definir como el elemento raíz, donde sus hijos, a su vez son árboles.

El paradigma lógico toma el *approach* declarativo para la resolución de problemas (no importa el *cómo*). Se hacen varias afirmaciones lógicas sobre una situación, estableciendo todos los hechos conocidos, lo que permitirá la realización de consultas. El computador es quien mantiene los datos y es quien realiza la deducción lógica.

[2] Amarasinghe Saman, Chlipala Adam, "Abstract Data Types", *Massachusetts Institute of Technology*, http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/

Análisis del problema (requisitos específicos)

El laboratorio requiere de la implementación de diversas funciones obligatorias que cumplen con la simulación del software Stackoverflow bajo el alero del paradigma lógico. El diseño de abstracciones y las características de éstas son fundamentales en primera instancia, puesto que las funciones requeridas deben combinarse de forma apropiada y actuar de forma consecuente al diseño planteado.

Requerimientos funcionales

1) **register*:*** Predicado que registra un nuevo usuario con sus respectivas credenciales (username y password), validación y verificación que evitarán duplicidad de usuarios registrados (usando username como identificador único). Este predicado se implementa como parte del TDA Stack, donde los usuarios registrados serán representados como la segunda lista dentro de la lista Stack. En el caso de ser una variable, esta será unificada con el Stack resultante luego de registrar al usuario, en caso de tratarse de un valor concreto, se verificará si el Stack2 cumple con ser el resultante luego de registrar al usuario.

2) **login*:*** Predicado que autentica a un usuario previamente registrado (dentro del stack), para lo anterior, se verifica la existencia del usuario en la lista Stack validando la existencia de éste en la primera lista del Stack. Si la autenticación es válida (username y password son correctos), el retorno es *true*. El usuario autenticado se agrega a la posición inicial de la lista Stack (string), puesto que será un elemento temporal y permitirá la fácil manipulación del mismo en el resultados de las operaciones.

3) **ask:** Predicado que permite a un usuario con sesión iniciada en la plataforma realizar una nueva pregunta. Cada pregunta registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación, la pregunta, un listado de etiquetas. El retorno es *true* si se puede satisfacer en "Stack2" el stack con la nueva pregunta incluida y sin la sesión activa del usuario que realizó la pregunta. Se valida fácilmente el usuario identificando el largo de la primera lista en el stack, estructura reservada para este elemento temporal. Esta función retornará el stack actualizado que incluirá esta pregunta y sus características, eliminando la sesión del usuario en su ejecución.

4) **answer:** Predicado que permite consultar el valor que toma un Stack al realizar la respuesta a una pregunta. Para esto se ingresa un stack inicial con sesión iniciada, la fecha de la respuesta, el id de la pregunta, el texto de la respuesta, un listado de etiquetas y el Stack resultante luego del registro. El retorno del predicado es *true* en caso que se pudo satisfacer la creación de esta respuesta en el Stack resultante. El nuevo stack actualizado incluirá la respuesta sin el usuario logueado.

5) **accept:** Predicado que busca permitir a un usuario logueado aceptar una respuesta a una de sus preguntas, se debe validar que la pregunta pertenezca a él. El retorno será un stack actualizado que actualizará los puntajes que correspondan a esta acción y que no incluirá al usuario logueado. Se debe hacer uso de predicado de corte **!**.

4) **stackToString:** Predicado que recibirá un stack y entregará su representación como un string que será comprensible a un usuario dada su estructura. Se integran saltos de línea "\n" para mejorar su visualización.

5) **vote:** Predicado que permite consultar el valor que toma un Stack al ocurrir un voto del usuario con sesión activa a una pregunta o respuesta. Se identificará a qué entidad corresponde esta votación (pregunta y respuesta) y cálculo asociado a esta acción.

Diseño de la solución

Respecto al diseño de la solución planteada para este laboratorio, se tomó como idea principal el desarrollo correcto y apropiado de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software. Idear y conceptualizar los requerimientos *a priori*, siguiendo un orden claro nos facilita la implementación de código, sin lugar a dudas. Antes de programar, es necesario analizar y aplicar una arquitectura correcta para evitar refactorizaciones innecesarias, documentar nuestro código (Base de conocimiento, dominio, hechos, reglas y metas), aplicando buenas prácticas que no sólo nos favorece a quienes desarrollamos, sino también a nuestros colegas.

De acuerdo a la idea anterior, se esquematiza la abstracción del Stack como una **lista de listas** (*ver Anexo 5: implementación simbólica de TDAs*) o contenedor de data. Sin embargo, eso no fue suficiente, también se requería reservar listas vacías que aseguran la posición de cada estructura dentro de este contenedor, esta no puede ser arbitraria ya que el acceso a cada una de ellas debe estar definido, evitando efectos secundarios y errores de compilación.

Respecto a la descomposición de los problemas, cada requerimiento se manejó como un subcomponente y se trabajó de forma aislada, sin embargo, se reconoció que la mayoría de las acciones debían ser una consecuencia del login correcto de un usuario. Por otra parte, se integraron técnicas consecuentes con el paradigma, como el uso práctico de la programación declarativa, mecanismos de inferencia (unificación, backtracking, árboles SLD, etc), uso pertinente de hechos, reglas y consultas a la base de conocimiento, entre otros.

Aspectos de implementación

Estructura del proyecto

La estructura del proyecto se basa en la modularización de cada TDA elegido, para ello, cada archivo maneja su propia lógica, siguiendo los principios de encapsulación. Adicionalmente, se provee de una sección **utils**, donde se definen predicados auxiliares que se utilizarán en uno o más TDAs. Es importante señalar que se provee de los predicados fundamentales en el archivo *lab_prolog_180051106_TapiaBobadilla.pl* que será de uso exclusivo del usuario, donde se exponen los ejemplos y casos de uso para cada requerimiento funcional.

Bibliotecas empleadas

Swi-Prolog y Prolog proveen de muchas utilidades que están disponibles en su documentación, sin embargo, se limitará (según requerimiento) sólo a aquellas utilidades para manejo de strings tales como: **string_concat**, **atomics_to_string** y **atom_string** para facilitar la manipulación y formateo de listas solicitados por el predicado **stackToString**.

Algunas de los predicados disponibles se recrearon *from scratch* como funciones auxiliares dentro de la sección **utils** del archivo principal.

Detalles del compilador e intérprete

SWI-Prolog Version 7, sin embargo, para la mayor parte del desarrollo de este laboratorio, se hizo uso del sitio web SWISH SWI-Prolog, puesto que ofrece un gran tooling para el uso de Prolog (terminal integrado para queries), es aquí dónde se corrieron los ejemplos y ejercicios desde el browser.

Instrucciones de uso

Instrucciones generales

Existen diversos entornos de trabajo para manejar archivos de Prolog (.pl), sin embargo, se prefiere el uso de la plataforma SWI-Prolog for SHaring, puesto que el proyecto se desarrolló localmente en sistema operativo Mac OSX, donde para el efectivo uso del entorno se requería la instalación mediante Homebrew. Para evitar problemas cross platform, se sugiere seguir las siguientes indicaciones o bien, seguir las indicaciones en "Downloads" (https://www.swi-prolog.org/Download.html) que se ajuste a la máquina local que testeará este código.

- Importar archivo en plataforma SWI-Prolog for SHaring. Correr las queries necesarias en la terminal provista.
- Clickear en botón **Run**, esquina inferior derecha de la terminal.

Ejemplos

Se definen 3 ejemplos para cada uno de los requerimientos funcionales y, en algunos casos, la implementación de ejemplos de casos específicos (usuario no logueado intentando preguntar, intento de registro de un usuario previamente registrado, etc.) Los resultados de cada ejemplo funcionarán como entrada de los siguientes, esto para demostrar el correcto funcionamiento de creación múltiple y, además, para ejecutar acciones como las de recompensa con un stack que ya posea preguntas previamente creadas. (ver *Anexo 4: Resultado de pruebas*).

Resultados esperados

Se espera que cada uno de los ejemplos cumpla las expectativas mencionadas en los requerimientos funcionales, exponiendo una estructura coherente y clara.

Posibles errores

No se reconocen posibles errores, puesto que los ejemplos se ajustan a los requerimientos. Se recomienda mantener la estructura de los ejemplos para evitar efectos secundarios en la ejecución de los predicados.

Resultados y autoevaluación

Se presenta a continuación un resumen de los resultados obtenidos a través de este laboratorio, los cuales serán plasmados y presentados en la siguiente tabla:

Requerimientos	Grado de alcance (%)	Pruebas realizadas	% de pruebas exitosas	% de pruebas fracasadas
stackRegister	100%	(3) Casos de registros exitosos. (1) Caso particular (validación de existencia)	100%	0%
stackLogin	100%	(3) Casos de login exitosos y (1) caso particular (validación de usuario registrado)	100%	0%
ask	100%	(3) casos de creación de preguntas exitosas y (1) caso particular (creación de preguntas múltiples).	100%	0%
answer	100%	(2) casos de creación de respuestas exitosas y (1) caso particular (creación de preguntas múltiples) y (1) caso que valida existencia de pregunta.	100%	0%
accept	100%	(2) casos de aceptación de respuestas caso particular y (1) caso que valida la aceptación de una respuesta perteneciente al usuario.	100%	0%
stackTostring	95%	(2) casos expuestos con **write** exitosamente. No se valida usuario logueado.	95%	5%
vote	98%	(4) casos de votación en contra y favor tanto para preguntas como para respuestas.	98%	2%

En la tabla anterior, no se presentan las funciones que no se completaron dado que en el laboratorio se trabajaron todos los requerimientos obligatorios y **uno** de aquellos opcionales (vote) en su totalidad.

[3] Durante el desarrollo del laboratorio, muchas pruebas fueron un fracaso, sin embargo, se trabajó en reducir al máximo los bugs y efectos secundarios hasta lograr los resultados esperados.

Conclusiones del trabajo

Este trabajo ha permitido conocer en detalle el alcance del paradigma funcional, el que podemos ver se basa más en el *"qué hacer"* y no en *"cómo se debe hacer"*,lo que permite que cualquier persona tenga la posibilidad de comprender, fácilmente, un código de estas características. Cabe mencionar que también hay algunas ventajas del paradigma, en las que se pueden mencionar: fácil creación de una base de datos sin un gran esfuerzo programático, manejo sencillo de listas lo que hace sencillo hacer uso de cualquier algoritmo que las involucre, el pattern-matching es sencillo dado que la búsqueda se basa en recursión. Sin embargo, y, como todo, hay una desventaja, por ejemplo, que LISP en general tiene mejor features I/O (Input/Output) que Prolog, además, el orden en el que las reglas se ingresen generan un gran impacto en la eficiencia del lenguaje.

Asimismo, cabe resaltar que en el paradigma hay una interesante característica de unificación y backtracking, siendo ambas bastante elementales. Respecto a las dificultades, se puede mencionar que la carencia de shortcuts y tooling del IDE SWI-Prolog y la mala compatibilidad con ciertos sistemas operativos (Ej: OSX) realmente ralentizaron el desarrollo del trabajo. También, conviene decir que hubo dificultades al momento de trabajar con el format de lista a string *from Scratch* fue necesario el uso de cierto tooling provisto por la librería.

A pesar de todo, hay cosas positivas que reconocer, como el haber trabajado con un lenguaje fuertemente orientado a la matemática, lo que permitió conocer más profundamente el paradigma lógico y un lenguaje no tan popular como Prolog, que sí permitió comprender de mejor manera cómo algunos motores de base de datos funcionan (Ej: SQL). En ese sentido, mencionar la importancia del uso de Git en las tareas diarias que tiene cualquier desarrollador, pues permite organizar mejor nuestras ideas y trabajos.

Referencias

Atwood, Jeff. (21 de septiembre de 2008). *What was Stackoverflow built with?*. The Overflow Blog.
https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/

Amarasinghe, Saman., Chlipala, Adam., Devadas, Srini., Ernst, Michael., Goldman, Max., Guttag, John., Jackson, Daniel., Miller, Rob., Solar-Lezama, Armando. (s.f.). *Abstract Data Types*. MIT.
http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/

Algunos aspectos de este trabajo fueron vistos de manera general en (https://stackoverflow.blog)

Documentación Prolog extraída desde (https://www.swi-prolog.org/pldoc/index.html)

Ehud Sterling Leon; Shapiro . (Enero de 1994). *The Art of Prolog.*

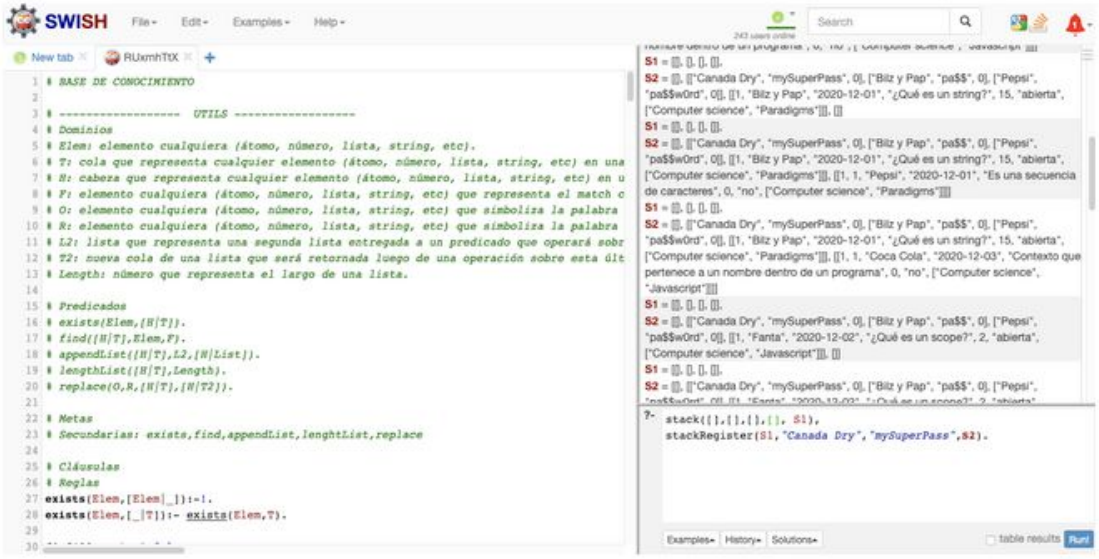
Van Roy, Peter. (Abril de 2012). *Programming Paradigms for Dummies: What Every Programmer Should Know*. Researchgate.
https://www.researchgate.net/publication/241111987_Programming_Paradigms_for_Dummies_What_Every_Programmer_Should_Know

Anexos

Anexo 1.Tabla de construcción de TDAs

Abstracción	Contexto	Características (Qué)
Stack	Estructura base que funciona como contenedor de usuarios, preguntas, recompensas, respuestas y búsqueda.	Lista de listas que se encarga de contener una serie de estructuras (del tipo lista), tales como: usuarios, preguntas, recompensas, respuestas y búsqueda dentro de la plataforma.
Usuario (User)	Persona que publica, edita y elimina tanto preguntas como respuestas, ofrece recompensas, vota, reporta, entre otros.	- Lista que posee credenciales de acceso y reputación - Ofrece reputación, pregunta y responde, da ranking, etc. dentro de la plataforma.
Pregunta (Question)	Enunciado interrogativo realizado por un usuario sobre diversas tecnologías relacionadas a la ingeniería de software en la plataforma.	- Lista que puede ser editada, modificada y eliminada. - Es creada por un usuario y recibe ranking por medio del mismo. - Posee textos con formato (negrita, cursiva) imágenes, enlaces y código - Posee un ID, votaciones (positivas y negativas), visualizaciones, etiquetas, título, contenido, fecha de publicación, fecha de última modificación , etc. - Es referenciada por una respuesta a través de su ID.
Respuesta (Answer)	Enunciado que responde a una pregunta realizada por un usuario sobre diversas tecnologías relacionadas a la ingeniería de software en la plataforma	- Lista que recibe ranking y referencia una pregunta. - Es creada por un usuario, quien puede recibir una bonificación (reputación) si esta es aceptada. - Posee un autor (usuario), fecha de publicación, votos (favor y contra), estado de aceptación (sí/no), reportes de ofensa y categorías (mejor respuesta, peor respuesta, etc).
Recompensa (Reward)	Bonificación que genera un usuario y que compensa la utilidad de una pregunta dentro de la plataforma.	- Lista que posee una descripción, id de pregunta y usuario que la otorgó. - Bonifica una pregunta y posee diversas reglas en su aplicación.
Búsqueda (Search)	Motor de búsqueda dentro de la plataforma que resulta de utilidad para encontrar información rápidamente por medio de una coincidencia parcial de texto.	- Búsqueda por etiqueta (entre corchetes) - Búsqueda palabra exacta (entre comillas) - Búsqueda por cantidad de respuestas (answers: numero_x) - Búsqueda por puntaje (score: numero_x)

Anexo 2. Entorno swish.swi-prolog



Anexo 3: Diagrama simbólico de los TDAs