**Universidad de Santiago de Chile**
**Facultad de Ingeniería**
**Departamento de Ingeniería Informática**

## Paradigmas de programación

# Laboratorio 2

# Paradigma Lógico

Christofer Rodríguez

Profesor: Víctor Flores Sánchez

Santiago – Chile
2-2020

# Tabla de contenidos

# Introducción

Stack Overflow es un famoso foro de preguntas y respuestas relacionadas, principalmente con la programación e informática; este foro cuenta con diferentes funcionalidades, tales como: preguntas, responder preguntas, votar, entre otras. En base a esto, es necesario hacer una simulación de un sistema de fotos tomando como referencia la funcionalidad de Stack Overflow en el lenguaje de programación Prolog, esto bajo el paradigma lógico. En este documento primero se contextualizará con respecto al paradigma utilizado, se analizará el problema en cuestión, luego se explicará la solución creada para resolver este problema, para finalmente con los resultados obtenidos evaluar el grado de alcance de cada requerimiento.

## Descripción del problema

Se necesita crea una representación de un foro de preguntas y respuestas basado en Stack Overflow, esto debe estar diseñado en el lenguaje de programación Prolog, utilizando el paradigma lógico. Esta representación debe contener características esenciales del foro como usuarios registrados, preguntas realizadas y respuestas a estas preguntas, por esto se necesita realizar una representación abstracta de este foro y confeccionar hechos y reglas que simulan la interacción entre capa parte de esta abstracción, de esta manera se espera replicar el funcionamiento del foro original, en específico es necesario implementar el registro de un usuario en el foro, iniciar sesión, realizar preguntas, responder preguntas, aceptar una respuesta como la mejor para una pregunta, votar a favor o en contra de una pregunta o respuesta y una manera de visualizar de una manera más grafica esta simulación de foro.

## Descripción del paradigma

El paradigma lógico consiste en realizar consultas a una base de conocimientos, estas consultas solamente responden si la consulta realizada es verdadera o falsa.

Al igual que en el paradigma funcional, bajo el paradigma lógico también se debe respetar la transparencia referencial, esto quiere decir que, cada vez que realizamos una consulta con los mismos parámetros, la respuesta debe ser siempre la misma, esto ya que las consultas realizadas no tienen efecto colateral, en otras palabras, no modifican el valor de elementos fuera de los predicados involucrados.

Ya que las consultas realizadas a una base de conocimientos solamente pueden ser respuestas con un verdadero o falso, se utiliza un concepto esencial en este paradigma llamado unificación. La unificación puede ser utilizada de diferentes maneras, pero su principal uso para este proyecto es para simular un “retorno” por parte de una regla, de esta manera podremos consultar cuál es el valor que tomaría una variable luego de realizar cierta acción, se modifica una estructura y luego se unifica con una variable para “retornar” cierto elemento, si lo comparamos con otros paradigmas, esta manera de utilizar la unificación sería el asignar un valor a una variable.

Otro de las funcionalidades importantes de este paradigma es el backtracking, con esto nos referimos a que, si se realiza una consulta a la base de conocimientos y no se logra una igualdad entre ambos términos, Prolog comienza a probar con diferentes términos para lograr una unificación, en el caso de que luego de probar con todos los términos no se llega a unificar, la respuesta a esta consulta será falsa.

## Análisis del problema

Al igual que bajo el paradigma funcional, en el paradigma lógico no podemos almacenar los valores de nuestra representación y modificarlos cuando se requiera, sino que tendremos que utilizar la información contendida en una representación entregada para crear una nueva versión de este con la información actualizada.

Si analizamos los requerimientos para este problema, podemos notar que será necesario implementar una representación del foro en base a listas, tanto para el foro en sí, como para casi la totalidad de sus componentes, estos siendo los usuarios, preguntas y respuestas. Para la interacción de cada uno de estos componentes es necesario crear reglas que nos entreguen el valor que debe tomar nuestra estructura para que sea verdadera la consulta. Gracias a las características del lenguaje de programación Prolog, solo será necesario crear un par de reglas para acceder a cada elemento parte de los componentes del foro; además debemos crear reglas capaces de añadir elementos, modificar valores, filtrar listas, verificar si es que podemos realizar ciertas consultas, entre otras.

Para esto será de gran ayuda la unificación, ya que esta será utilizada para retornar elementos, comparar valores y para la recursión dentro de las reglas.

## Diseño de la solución

Para llevar a cabo la simulación del foro, se utilizaron 8 TDA diferentes los cuales serán detallados a continuación:

1. TDA usuario: Un usuario estará representada por una lista de elementos que representen cada aspecto de un usuario de la plataforma, este contendrá: su nombre de usuario (string), su contraseña (string), una lista con el identificador de cada pregunta que él haya realizado (lista de enteros) y su reputación (entero).

2. TDA lista de usuarios: Este TDA será una lista que almacenar a cada usuario registrado en la plataforma, cada elemento de esta lista es un TDA usuario diferente.

3. TDA pregunta: Representación de una pregunta realizada en el foro, este TDA está representado por una lista que contiene: la fecha en la que se creó (TDA fecha), el contenido de esta pregunta (string), sus etiquetas (lista de strings), su identificador (entero), su estado (string), el autor de la pregunta (string) y sus votos (entero).

4. TDA lista de preguntas: Será una lista conformada por TDA preguntas, cada uno de sus elementos es una pregunta diferente con sus respectivos datos.

5. TDA respuesta: Respuesta a una pregunta del foro, consistirá en una lista con su fecha de creación (TDA fecha), el identificador de la pregunta a la que responde (entero), su contenido (string), sus etiquetas (lista de strings), su autor (string), su identificador (entero), sus votos (entero) y su estado (string).

6. TDA lista de respuestas: Lista que guardará un TDA respuestas diferente en cada una de sus componentes.

7. TDA fecha: Lista que contiene el día, mes y año de creación de una pregunta o respuesta.

8. TDA stack: El más importante de todos los TDA, representa al foro mismo, lista en la que estarán almacenados los usuarios registrados (TDA lista de usuarios), las preguntas (TDA lista de preguntas), las respuestas (TDA lista de respuestas), el usuario con sesión activa (TDA usuario), el identificador de la última pregunta realizada (entero) y el identificador de la última respuesta creada (entero).

Debido a que Prolog es un lenguaje de programación débilmente tipado, se decidió utilizar listas para representar la mayoría de los componentes de la representación del foro, de esta manera es posible trabajar con diferentes tipos de datos contenidos en un solo lugar de manera ordenada y fácil acceso.

De manera general, para abordar este proyecto se decidió implementar una solución en la que cada vez que se desea modificar un elemento dentro de la estructura del foro, es necesario crear una versión actualizada de este stack. Lo anterior se realizó utilizando la información contenida por el stack original y se entregaban los componentes que se debían modificar a los diferentes predicados que se encargaban de cada subproblema, una vez modificados los componentes, estos eran unificados con una variable sin valor entregada por el usuario.

Los principales subproblemas presentes en la implementación de este proyecto se implementaron de la siguiente manera:

Predicados de TDA's: Para facilitar el trabajo con los TDA se crearon predicados esenciales para interactuar con ellos, constructores, selectores, modificadores, pertenencia; estas fueron el pilar fundamental para poder implementar los algoritmos descritos a continuación.

Modificar: En esta implementación existen dos tipos de modificadores. El primero de ellos es un modificador "local", este tipo de predicado fue utilizado para modificar valores dentro de TDA's unitarios como un usuario, una pregunta o una respuesta. Estos predicados consistían entregar un TDA, el nuevo valor para uno de sus componentes y una variable sin valor para que el resultado se unificará con ella, una vez comprobado que el nuevo valor cumple con el tipo de dato, se unifica el resto de los componentes del TDA y el nuevo valor en el lugar correspondiente.
El segundo de estos modificadores es un modificador para aquellos TDA que contienen otros TDA, por ejemplo: una lista de preguntas, una lista de respuestas o una lista de usuarios. Para realizar esto se recorría la lista mientras se intentaba unificar la cabeza de la lista con el elemento buscado, una vez encontraba el elemento buscado se unificaba el resto de los elementos de la lista y la nueva versión del elemento a modificar con la variable sin valor entregada.

Agregar: Este tipo de modificador era solamente utilizado por aquellos componentes representados por una lista, por lo anterior estos predicados solamente consistían en recorrer la lista mediante recursión y una vez se llegaba al final de esta, se unificaban todos los elementos ya recorridos junto con el elemento a agregar al final de ellos con una variable sin valor entregada como "argumento".

Verificar: Este tipo de predicados eran utilizados para comprobar si es que se podía realizar la consulta de otro predicado, principalmente se utilizó la unificación de valores, si estos eran capaces de unificarse significaba que se podía realizar o que no se podía realizar, esto dependiendo de lo que se buscaba comprobar.

Buscar: Al igual que la verificación, el buscar un elemento dentro de una lista se realizaba recorriendo la lista e intentando unificar la cabeza de esta con el valor buscado, si se lograba unificar, por lo general este elemento se unificaba con una variable sin valor para ser "retornada" por el predicado.

Filtrar: Para filtrar listas, también se utilizó la unificación mientras se recorría una lista de elementos, aquellos elementos que lograban unificarse eran unificados dentro de una lista, que a su vez se unificaba con una variable sin valor para ser "retornada" por el predicado.

## Aspectos de implementación

Para la implementación de este proyecto se utilizó la SWI-Prolog como intérprete del lenguaje de programación Prolog, solamente fueron utilizados predicados pertenecientes a la librería estándar de Prolog. La estructura que se siguió para la base de conocimientos fue, primero explicar brevemente los componentes de cada TDA, explicar el dominio de cada predicado que no tengo un nombre auto explicativo, luego todos los predicados presentes en la base de conocimientos, las metas principales, los hechos, las reglas y finalmente 3 ejemplos de consultas para cada meta principal.

## Instrucciones de uso

Si se desea utilizar el simulador del foro Stack Overflow, es necesario que se habrá el intérprete SWI-Prolog y se consulte a la base de conocimientos almacenada en el archivo llamado "lab_git_20239786_RodriguezAllup.pl", se recomienda utilizar el intérprete nombrado anteriormente, ya que este fue el utilizado para probar cada una de estas consultas en su desarrollo.

A continuación, se mostrarán ejemplos de las principales consultas del programa que interactúan con el foro, estas consultas (texto en negrita) deben ser copiadas dentro de la consola una vez se consulte a la base de datos:

Registrar un nuevo usuario en el foro: **stack(StackInicial, 2), stackRegister(StackInicial, "Christofer", "123456", StackResultante).** , esta consulta mostrará el valor que toma un stack luego de registrar un usuario en el foro.

Iniciar sesión de un usuario registrado: **stack(StackInicial, 1), stackLogin(StackInicial, "jaime", "456", StackResultante).** , esta consulta mostrará el valor que toma un stack luego de iniciar la sesión de un usuario previamente registrado en el foro, es necesario realizar previamente esta consulta para el correcto funcionamiento de las consultas: ask, answer, accept y vote.

Realizar una pregunta: **stack(StackInicial, 1), stackLogin(StackInicial, "chris", "chris123", StackResultante), ask(StackResultante, "06-12-2020", "Como pregunto?", ["ayuda"], NuevoStackResultante).** , esta consulta mostrará el valor que toma un stack luego de que un usuario con sesión iniciada realice una pregunta en el foro.

Responder a una pregunta: **stack(StackInicial, 1), stackLogin(StackInicial, "chris", "chris123", StackResultante), answer(StackResultante, "06-12-2020", 2, "Asi se responde ?", ["ayuda"], NuevoStackResultante).** , esta consulta mostrará el valor que toma un stack luego de que un usuario con sesión iniciada responda una pregunta del foro.

Aceptar una respuesta a mi pregunta como la mejor: **stack(StackInicial, 1), stackLogin(StackInicial, "jaime", "456", StackResultante), accept(StackResultante, 2, 2, NuevoStackResultante).** , esta consulta mostrará el valor que toma un stack luego de aceptar una respuesta a la pregunta del usuario loggueado como la mejor y además se modificará positivamente la reputación de los usuarios involucrados

Mostrar el foro: **stack(StackInicial, 1), stackTostring(StackInicial, StackStr).** , esta consulta mostrará una representación gráfica del foro. En caso de que se consulte con un usuario loggueado, solo se mostrara la información de dicho usuario y sus preguntas.

Votar una pregunta o respuesta: **stack(StackInicial, 1), stackLogin(StackInicial, "chris", "chris123", StackResultante), getQuestion(StackResultante, 1, Pregunta), vote(StackResultante, Pregunta, true, NuevoStackResultante).** , esta consulta muestra el valor que toma el stack luego de que un usuario loggueado vota positiva o negativamente una pregunta o respuesta.

## Resultados y autoevaluación

Se obtuvieron resultados satisfactorios al poner a prueba cada consulta, para comprobar su correcto funcionamiento se realizaron consultas con distintos valores, en los casos de ingresar valores validos estas consultas funcionaron correctamente y los valores inválidos fueron detectados correctamente. Las únicas pruebas que fallaron fueron cuando se introducía un stack externo con valores incorrectos dentro de él, por ejemplo: preguntas con el mismo ID, en estos casos se producía un error en el funcionamiento dentro de algún proceso.

| | |
|---|---|
| TDA's | Se logró implementar cada TDA necesario de una manera satisfactoria y que permitió un fácil manejo de ellos. |
| Hechos | Se incluyeron todos los hechos requeridos dentro de la base de conocimientos |
| stackRegister | Se logro implementar correctamente y es capaz de detectar si es que existe un usuario previamente registrado con el nombre entregado. |
| stackLogin | Se logró implementar correctamente, comprueba que los datos ingresados sean los del usuario registrado. |
| Ask | Se crea correctamente la pregunta, se registra correctamente con su identificador y se actualiza el contador de preguntas en el stack. |
| Answer | Se crea correctamente la respuesta, se registra correctamente con su identificador, junto con el identificador de la pregunta y se actualiza el contador de respuestas en el stack. |
| Accept | Se implemento correctamente, se actualiza el estado de la pregunta y respuesta, además se actualiza la reputación de cada usuario involucrado. Debido a que no se especificó, un usuario puede aceptar su propia respuesta a su propia pregunta. |
| stackTostring | Muestra de manera correcta la información de todo el stack o del usuario en cuestión cuando se consulta con sesión iniciada. |
| Vote | Se modifica correctamente los votos de la pregunta o respuesta y se modifica la reputación de los usuarios involucrados en cada caso. Debido a que no se especificó, un usuario puede votar negativamente sus propias respuestas, recibiendo un doble descuento de reputación. |

# Conclusión

Al igual que en el proyecto anterior, es difícil adaptarse a un nuevo paradigma y lenguaje, pero debido a las similitudes, en cuanto a la transparencia referencial que tiene este paradigma con el paradigma funcional, no fue tan difícil el cambio de paradigma. Debido a la poca costumbre con el lenguaje de programación Prolog, existieron ciertas limitaciones o una ineficiente utilización del lenguaje, aunque también sirvió para ver distintas soluciones a un problema. Al igual que bajo el paradigma funcional fue de vital importancia saber dividir los problemas en distintos subproblemas, esto para una adecuada utilización del paradigma y para realizar un trabajo más ordenado y eficiente, esto junto con un diseño claro de cada TDA y la manera de interactuar con cada uno de ellos. Por lo anteriormente descrito se considera que el proyecto fue realizado de una manera correcta y su desarrollo será de vital importancia para la formación como programador.

## Referencias

Javatpoint. (2020). Backtracking in Prolog. (Recuperado el 06/12/2020) https://www.javatpoint.com/backtracking-in-prolog

Merino, M. (2020). El lenguaje Prolog: un ejemplo del paradigma de programación lógica. (Recuperado el 06/12/2020) https://www.genbeta.com/desarrollo/lenguaje-prolog-ejemplo-paradigma-programacion-logica

Universidad de Chile (2020). Unificación. (Recuperado el 06/12/2020) https://users.dcc.uchile.cl/~abassi/Cursos/41a/unificacion.html

# Anexo

## Ejemplos detallados de consultas

Si se desea modificar los datos de cada consulta, se debe seguir las siguientes consideraciones:

Primero, dentro de la base de conocimientos existen dos stack y pueden ser consultados de la siguiente manera:

*stack(NombreDelStack, 1).* , el primer elemento corresponde al nombre que se le dará al stack, es necesario que el nombre comience con mayúscula y no tenga espacio entre su nombre. El segundo elemento corresponde al identificador del stack, el numero uno corresponde a un stack con usuarios, preguntas y respuestas previamente registradas en él.

*stack(NombreDelStack, 2).* , al igual que el anterior, primero tenemos el nombre del stack, seguido por su identificador. Este stack corresponde de un stack vacío, en caso de querer comenzar desde cero, se recomienda utilizar este stack.

*stackRegister(StackInicial, "Username", "Pass", StackResultante).* , el primer elemento es el nombre con el que se "guardó" la consulta de alguno de los 2 stack iniciales, luego viene el nombre de usuario con su contraseña, ambos deben ir entre comillas y separados por una coma, finalmente viene el nombre con el que se guardará el stack resultante, este debe comenzar con mayúscula.

*stackLogin(StackInicial, "Username", "Pass", StackResultante).* , el primer de sus elementos es un stack, luego viene el nombre de usuario y la contraseña de un usuario previamente registrado que quiera iniciar sesión, ambos deben ir entre comillas y separados por una coma, en caso de que el usuario exista y los datos ingresados sean correctos, el stack resultante contendrá el stack con la sesión iniciada del usuario, en caso de que alguno de los datos sea incorrecto, se mostrará que la consulta fue falsa.

*Para las consultas siguientes es necesario consultar el predicado *stackLogin* previamente para su correcto funcionamiento.

*ask(StackInicial, "día-mes-año", "Contenido", ["Etiqueta1", "Etiqueta2"], StackResultante).* , el primer elemento es un stack inicial, el segundo es la fecha de creación de la pregunta, esta debe ir entre comillas y se recomienda seguir el formato para mantener la consistencia con la estructura, luego viene el contenido de la pregunta entre comillas, después entre corchetes se debe agregar las etiquetas para la pregunta, estas deben ir entre comillas y separadas por una coma entre ellas, en caso de no querer agregar etiquetas, dejar solamente los corchetes, finalmente se escribe el nombre del stack resultante.

*anwer(stackInicial, "día-mes-año", IdPregunta, "Contenido", ["Etiqueta1", "Etiqueta2"], StackResultante).* , primero tenemos el stack inicial, luego viene la fecha de creación de la respuesta entre comillas y respetando la estructura, luego el identificador numérico de la pregunta a la que va dirigida la respuesta, seguido del contenido entre comillas, las etiquetas que deben seguir la misma estructura explicada en *ask* y finalmente el nombre del stack resultante.

*accept(StackInicial, IdPregunta, IdRespuesta, StackResultante).* , primero tenemos el nombre del stack inicial, luego tenemos el identificador numérico de una pregunta realizada por el usuario que inicio sesión, seguido del identificador numérico de la respuesta que se desea marcar como: mejor respuesta, finalmente tenemos el nombre del stack resultante.

*stackTostring(StackInicial, StackStr).* , primero tenemos el stack que deseamos visualizar y luego el nombre que se le dará al string resultante de la consulta. Si el stack inicial tiene un usuario con una sesión iniciada, solamente se mostrará la información de dicho usuario y sus preguntas; en cambio, si el stack inicial no tiene un usuario con sesión iniciada, se mostrará toda la información de contenida en dicho stack.

En caso de querer votar positiva o negativamente una pregunta se debe realizar lo siguiente:

Primero se debe consultar un stack inicial, luego consultar el siguiente predicado:

*getQuestion(StackInicial, IdPregunta, Pregunta).* , primero ingresamos el stack inicial, luego el identificador numérico de la pregunta que deseamos votar y al final el nombre que le daremos a la pregunta, esta debe comenzar con mayúscula y no contener espacios en su nombre.

Ahora para vota se debe ingresar la consulta anterior seguida de una coma y consultar el predicado vote:

*getQuestion(StackInicial, IdPregunta, Pregunta), vote(StackInicial, Pregunta, TipoVoto, StackResultante).* , el primer elemento es el mismo stack inicial donde se buscó la pregunta, luego el nombre con el que se almaceno la pregunta buscada, después se debe escribir *true* en caso de que sea un voto positivo o *false* en caso de que sea un voto negativo, finalmente tenemos el nombre del stack resultante.

En caso de querer votar positiva o negativamente una respuesta se debe realizar lo siguiente:

Primero se debe consultar un stack inicial, luego consultar el siguiente predicado:

*getAnswer(StackInicial, IdPregunta, IdRespuesta, Respuesta).* , primero ingresamos el stack inicial, luego el identificador numérico de la pregunta a la que responde la respuesta a votar, seguido del identificador numérico de la respuesta a votar y finalmente el nombre que se le dará a la respuesta a votar.

Ahora para vota se debe ingresar la consulta anterior seguida de una coma y consultar el predicado vote:

*getAnswer(StackInicial, IdPregunta, IdRespuesta, Respuesta), vote(StackInicial, Respuesta, TipoVoto, StackResultante).*, primero tenemos el nombre de este stack donde se buscó la respuesta, el nombre que se le dio a la respuesta a votar luego viene el tipo de voto, para esto se debe ingresar *true* en caso de que sea un voto positivo o *false* si es un voto negativo, al final se debe ingresar el nombre del stack resultante.

*Se debe tener en consideración que, si es que se ingresa un valor en específico en lugar de un nombre para el stack resultante, la consulta arrojará verdadero o falso, en caso de que el stack resultante sea igual al valor ingresado.