Universidad de Santiago de Chile

Facultad de ingeniería

Departamento de ingeniería en informática

Informe laboratorio N°2
Paradigma Lógico

Nombre: Jordán Arias Hurtado

Profesor: Víctor Flores Sánchez

Sección: B-2

INDICE

Introducción:

En este informe esta seccionado por una descripción del problema a trabajar, una descripción del paradigma empleado y porque el uso de este, posteriormente un análisis del problema ya planteado, seguido de un diseño de solución para tratar el dilema, aspectos en la implementación de la solución, instrucciones necesarias para el uso del programa, resultados obtenidos, y finalmente conclusiones pertinentes al tema.

El fin de este proyecto es el de poder crear una red de metro funcional, problema el cual cada vez se complejiza más, cada año con cada nueva línea que construyen, o estaciones que añaden a estas, por lo que se complica cada vez más, poder mantener en orden y correcto funcionamiento dicha red. Para abarcar este dilema se empleó un nuevo paradigma, el paradigma lógico, implementado en el lenguaje de programación prolog.

Descripción del problema:

Este proyecto se origina dese la necesidad de poder diseñar, organizar, y modificar una red de metro de manera eficiente. El programa en su fase final, debe poder conseguir modelar por completo una red de metro operativa, teniendo siempre en consideración, el uso del paradigma lógico, y evitando el uso de programación de manera imperativa, y en lugar de eso, utilizar el paradigma planteado y lograr la abstracción necesaria para que al realizar las consultas, el programa funcione como es debido.

Descripción del paradigma:

El paradigma lógico consiste en un paradigma de tipo declarativo donde basa su abstracción en la lógica matemática, esto causa que tengamos que pensar en modelar las situaciones a través de la lógica. Su principal fuerte es que se centra en el “qué” de los problemas, utilizando una base de conocimientos como un “todo” que existe y se cumple, donde lo que está fuera de este “todo” es falso, en la base de conocimiento se declaran los hechos y se crean relaciones entre ellos, ósea las “verdades”, mientras que por otro lado, las consultas son, preguntas que se responden en torno al “todo” que se definió anteriormente por los hechos.

Prolog posee dos elementos muy importantes para lograr una correcta búsqueda de elementos, siendo estos la inferencia y unificación. La primera sería consultar directamente sobre algo ya conocido que es verdadero o falso. Y el segundo es, que, en base de las relaciones existentes de los hechos, se construyan los elementos propiamente necesarios para obtener tal elemento, mientras este sea verdadero. Por último, cabe mencionar que este cuenta con Backtracking automático, que en palabras simples, sería un proceso recursivo que revisa en cada uno de los hechos, y clausulas hasta lograr una unificación.

Análisis del problema:

Los requisitos específicos que se deben de cubrir son el de crear estaciones, que en pares componen secciones, y un conjunto de estas secciones junto a cierta información adicional crean una línea de metro. Crear vagones o carros, que la unión de estos crea un tren. Un conductor habilitado para la conducción de los trenes, Y finalmente un la creación de un subway, que reúne las líneas, trenes y conductores creados. Teniendo siempre en cuenta que estamos operando bajo el paradigma lógico.

También hay clausulas específicas a cubrir, estos son:

- station: Crea una estación con un id, nombre, tipo de estación que debe ser terminal, combinación, mantenimiento, o recorrido, y un tiempo de parada.
- section: Crea una sección, compuesta de 2 estaciones diferentes, la distancia entre ellas, y un costo monetario asociado.
- line: Crea una línea con un id, un nombre, el tipo de rieles, y una lista de secciones
- lineAddSection: Añade una sección al final de una línea.
- isLine: verifica que una línea sea consistente en su estructura, ósea si empieza y termina con estaciones de tipo terminal, a no ser que sea circular.
- lineLength: Calcula la cantidad total de estaciones de una línea, la distancia total entre las secciones de esta, y el costo mismo asociado.
- lineSectionLength: Calcula el recorrido que hay entre 2 estaciones, además de la distancia de este camino, y su costo asociado
- isPcar: Crea un vagón de tren, con un id, su capacidad individual, el modelo, y si es de termino o central.
- train: Crea un tren con un id, una manufacturadora, tipo de riel, velocidad máxima, y una lista de vagones
- isTrain: Verifica si un tren posee la estructura adecuada, y si comienza y termina con vagones terminales, y su cuerpo es de vagones centrales
- trainAddCar: Añade un vagón a una posición especifica de un tren
- trainRemoveCar: Remueve un vagón de una posición especifica de un tren
- trainCapacity: calcula la capacidad máxima de pasajeros del tren
- driver: Crea un conductor con un id, un nombre, y los tipos de tren que está habilitado a conducir en base a la manufacturadora
- subway: Crea una red de metro con un nombre y un id
- subwayAddTrain: añade trenes a la línea de metro
- subwayAddLine: añade líneas a la línea de metro
- subwayAddDriver: añade conductores a la linea de metro

Diseño de la solución:

Para comenzar a diseñar la solución se empezó planteando los TDA necesarios para lograrlo, el primero fue el station, la base y unidad más básica de una línea de metro. Este fue necesario para poder crear el TDA sections, capaz de relacionar 2 estaciones y añadir distancia y costo asociado. A continuación, se creó el TDA del line como tal, que siendo la unión de varias secciones, el cual permite ver de manera clara y transparente que la creación de la línea sea coherente, que en caso de ser una linea imposible o con defecto retornara un falso. Luego se creó el TDA modificador de la línea, lineAddSection con el que

se verifico la correcta creación de una línea a partir desde una línea vacía, posteriormente se crearon las otras funciones lineLength y lineSectionLength, con el que gracias a clausulas auxiliares que se diseñaron, como verificar la conexión completa de la linea, o el recorrido de esta, se procedio a concluir la linea con el de pertenencia isLine, que gracias a las clausulas auxiliares mencionas, se pudo crear un verificador de consistencia de una linea.

El siguiente TDA concebido fue el pcar, ya que con este se logro crear exitosamente, del mismo modo que con linea, un vagon de tren, la unidad mas simple de este, y que a diferencia de linea, estos se añaden directamente al siguiente diseñado, el TDA train que permite crear un tren tomando como base una lista de vagones. Siguiendo el mismo modelo que con la línea, se siguió con los modificadores trainAddCar y trainRemoveCar, que se logro verificar su funcionalidad a través de su uso. Luego se siguió con otras funciones y particularmente el desarrollo trainCapacity. Y para concluir este TDA, se creo el de pertenencia isTrain, que permite verificar la correcta construcción del tren antes de darle algún uso a futuro

Posteriormente se diseño al conductor de trenes en su propio TDA driver.

Finalmente se creo el TDA subway, en el cual a través de los modificadores subwayAddTrain, subwayAddLine, subwayAddDriver, implementamos directa e indirectamente, todos los TDA previamente creados en una convergencia para crear una red de metro operativa que se debe operar por medio de la consola de Prolog.

Consideraciones de la implementación:

Para la implementación en este laboratorio se estructuró en 7 TDA, siendo estos station, section, line, pcar, train, driver y subway. Estos TDA’s fueron necesarios para la implementación total del ambiente esperado, ya que estos son los requerimientos principales obligatorios.

Cada uno de estos TDA poseen sus respectivas representaciones, constructores, selectores, modificadores, otras operaciones y funciones de pertenencia necesarios para el correcto funcionamiento y uso de cada uno

Se utilizó SWI-Prolog de 64 bits, versión 9.2.4 en lenguaje de Prolog, las bibliotecas empleadas fueron solamente las nativas de Prolog sin ninguna externa o uso del comando import.

Instrucciones de uso:

para poder utilizar y ejecutar el programa y crear una red de metro, se requiere que se utilice por medio de la consola de SWI-Prolog, los predicados para conformar las estaciones, secciones, linea, vagones, trenes, conductores, un ejemplo sería el Script de Pruebas [Figura 1], de esta forma ya se tiene creado por completo 3 redes de metro cada una con una linea distinta. Cabe destacar que hay que escribir todo de manera continua con comas como separación entre predicados y de esta forma puede con el respectivo predicado, manipular y operar la red de metro como se desee.

Resultados: Probando con todos y cada uno de los predicados realizados (hasta subwayAddDriver), cumplen su función en un 100%, sin embargo no se pudo lograr finalizar

por completo todo lo esperado. Pero cada uno de los predicados hechos fueron testeados con cada posible complicacion que podrían tener, .

Los predicados no implementados fueron subwayToString, subwayAssignTrainToLine, subwayAssignDriverToTrain, whereIsTrain, y subwayTrainPath que por falta de tiempo y practica con el paradigma lógico para abordarlos, no se lograron implementar.

Conclusiones:

El uso del paradigma logico, resulto mas practico para la implementación de una solución al problema respecto al paradigma funcional con el que se trabajo previamente, ya que en vez de emplear multiples funciones y anidarlas para obtener una respuesta, se realiza todo mediante consultas, y aprovechar el “que necesitan para que sea verdad”.

En retrospectiva, el trabajo realizado logró implementar lo solicitado en gran medida, pudiendo ahondar en el entendimiento de este curioso paradigma.

Referencias:

-SWI-Prolog Solutions. (s.f.). swi-prolog.org. Obtenido de https://www.swi-prolog.org/

-Serra, S. (2009). uvirtual.usach.cl. Obtenido de https://uvirtual.usach.cl/moodle/pluginfile.php/369075/mod_resource/content/1/Unidad%20VI%20-%20Paradigma%20LOGICO.pdf

Anexos: