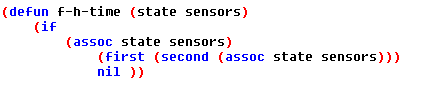
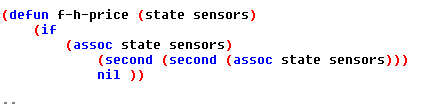
***Inteligencia Artificial: Práctica 2***

***Ejercicio 1:***

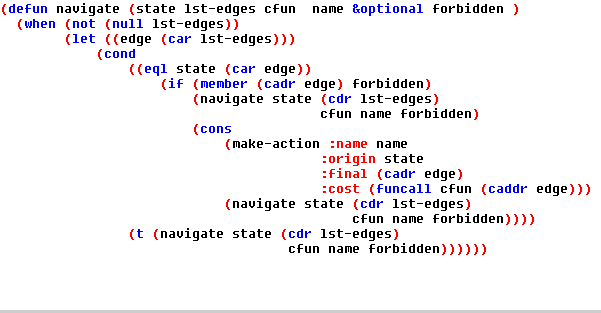
Para el ejercicio, hemos desarrollado dos funciones que calculan los valores de las dos posibles heurísticas:

1. **F-h-time**: Si en la lista de heurísticas está el estado pedido, devolvemos el primer elemento de la segunda sublista (la que contiene los valores), que es el que corresponde al tiempo.
2. **F-h-price**: Si en la lista de heurísticas está el estado pedido, devolvemos el segundo elemento de la segunda sublista (la que contiene los valores), que es el que corresponde al coste.



***Ejercicio 2:***

En este segundo ejercicio, se nos pide hacer una función que devuelva las posibles acciones desde un nodo, dado el estado (nombre del nodo). Para ello, hemos desarrollado una función general, y seguidamente una serie de funciones secundarias que, depende de lo pedido, llaman a esta primera con distintos argumentos. Las funciones son las siguientes:

1. **Navigate**: Función general, que recibe como argumentos un estado, la lista de conexiones del grafo, la función para acceder al valor buscado (en el caso del ejemplo, tiempo o precio), el nombre a dar a las acciones generadas y opcionalmente, una lista de ciudades a las que no se puede llegar en tren. Esta función recorre los elementos de la lista de conexiones, comprobando si el origen de cada conexión es el estado pasado como argumento. En tal caso, si el destino no está en la lista de prohibidos, crea una acción con los campos correspondientes, y sigue recorriendo las conexiones. En cualquier otro caso, ignora esa conexión y sigue recorriendo.
2. **Navigate-canal-time:** Obtiene las posibles acciones por canales y según el tiempo desde un estado dado. Usa Navigate.



1. **Navigate-canal-price**: Obtiene las posibles acciones por canales y según el precio desde un estado dado. Usa Navigate.



1. **Navigate-train-time:** Obtiene las posibles acciones en tren y según el tiempo desde un estado dado. Usa Navigate.

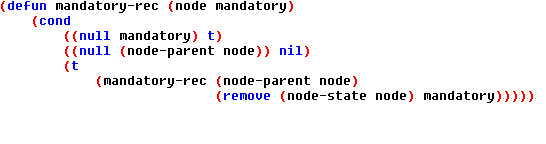


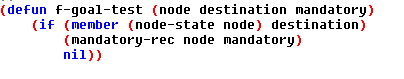
1. **Navigate-train-price:** Obtiene las posibles acciones en tren y según el precio desde un estado dado. Usa Navigate.



***Ejercicio 3:***

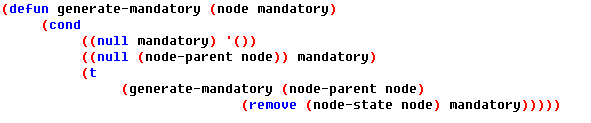
Se nos pide desarrollar una función que compruebe si se ha alcanzado el objetivo. Para ello, hay que comprobar que el estado del nodo dado es un estado final, y que ha pasado por todas las ciudades obligatorias. Para lo segundo, hemos creado una función auxiliar. En total tenemos estas dos funciones:

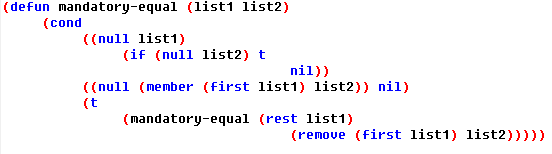
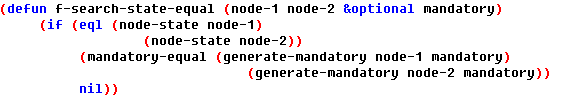
1. **Mandatory-rec:** Va recorriendo los antecesores del nodo dado, comprobando en cada iteración si la ciudad visitada está en la lista de obligatorias y, en tal caso, eliminándolas de esta. La función devolverá si se han visitado todas o si se ha llegado al final y no ha pasado por todas.
2. **F-goal-test:** Si el estado del nodo es un estado final, devuelve lo que devuelva Mandatory-rec, en otro caso devuelve nil.



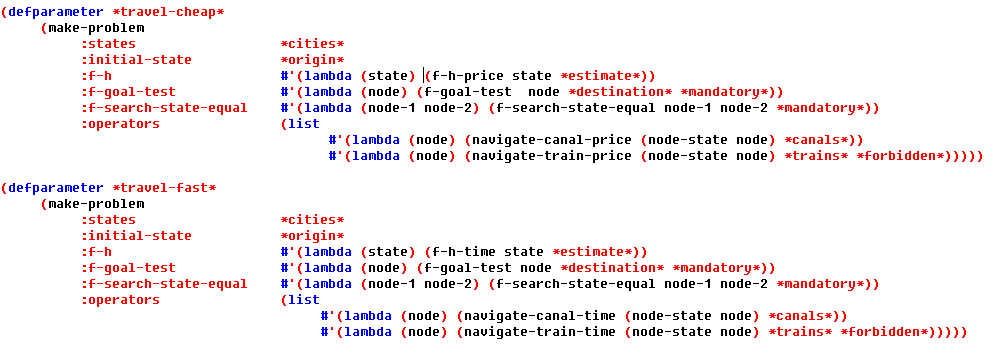
***Ejercicio 4:***

En este ejercicio, se nos pide codificar una función que compare dos nodos , comprobando si son iguales a nivel estados de búsqueda. Para ello, hemos desarrollado dos funciones auxiliares, utilizadas por una tercera general que da lugar al resultado buscado. Las funciones son:

1. **Generate-mandatory:** Recorre todos los antecesores del nodo dado, comprobando si están en la lista de obligatorios, y en tal caso, borrándolos de la misma. Al terminar, la función devolverá las ciudades obligatorias que le quedan al nodo por visitar.

1. **Mandatory-equal:** Recibe dos listas y las compara, recorriendo la primera elemento a elemento y comprobando si cada uno está en la otra.
2. **F-search-state-equal:** Función general que, usando las dos anteriores, compara si dos nodos son iguales. Para ello, comprueba primero que sus estados sean iguales, y seguidamente genera la lista de estados obligatorios aun por recorrer de cada nodo, y las compara. En caso de ser iguales, ambos nodos son idénticos a nivel de estado de búsqueda.

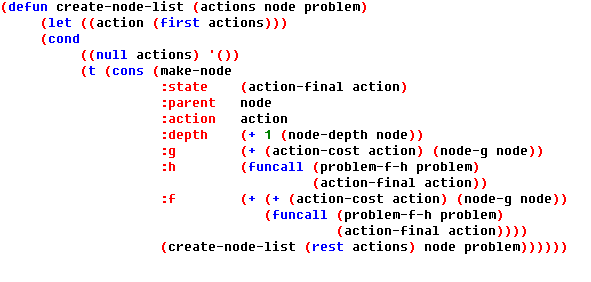
***Ejercicio 5:***

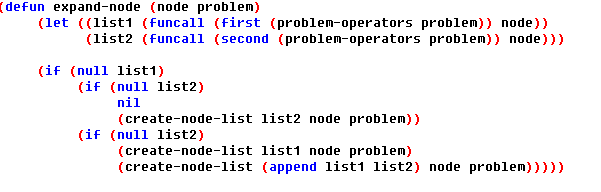
Para este ejercicio hemos creado dos estructuras de tipo problema, una para el que optimiza el precio y otra para el que optimiza el tiempo. Por lo tanto, los campos los hemos rellenado con las funciones correspondientes, las que obtienen las acciones y las heurísticas correspondientes a cada problema, y las funciones de comparación y de estado final ,que coinciden junto al origen y al conjunto de estados en ambos problemas.

***Ejercicio 6:***

En este ejercicio, se nos pide desarrollar la función de expansión de un nodo. Para ello, hemos codificado una función auxiliar que genera a partir de una lista de acciones, los nodos que surgen. En total, tenemos dos funciones:

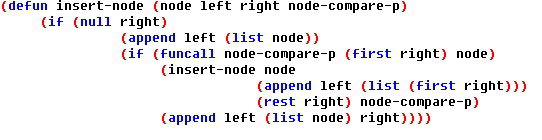
1. **Create-node-list:** Recibiendo un nodo y una serie de acciones como parámetro, genera los nodos que tienen a este primero como padre y que surgen a partir de las acciones dadas.

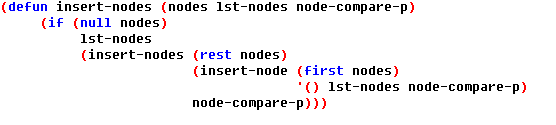


1. **Expand-node:** Primeramente, se generan las listas de acciones a raíz de los operadores del problema. Estas listas se le pasan a create-node-list, y esta genera la lista de nodos. Lo que realmente hace esta función es comprobar si alguna de las listas generadas por los operadores es nil, y en tal caso le pasa a la función auxiliar solo la que corresponda, o un cons de ambas.

***Ejercicio 7:***

En este caso, hemos decidido dividir el trabajo en tres funciones, a cada cual más general, y siendo el objetivo final introducir una serie de nodos en orden en una lista ordenada de nodos. Las funciones son:

1. **Insert-node:** Recorre la lista, comparando el nodo a insertar usando la función dada, y determinando que nodos van antes (izquierda) y cuales después (derecha) de este mismo, situándolo en su lugar.
2. **Insert-nodes:** Va insertando los nodos de una lista no vacía en la lista ordenada, llamando recursivamente a insert-node.



1. **Insert-nodes-strategy:** Llama a insert-nodes, extrayendo la función de comparación de la estrategia dada.

***Ejercicio 8:***

En este ejercicio, simplemente definimos la función de comparación de la f de dos nodos, y la incluimos en la definición de la estrategia A estrella, la cual declaramos como parámetro.

