Trabajo Práctico 2: Razonamiento

Grupo 4:

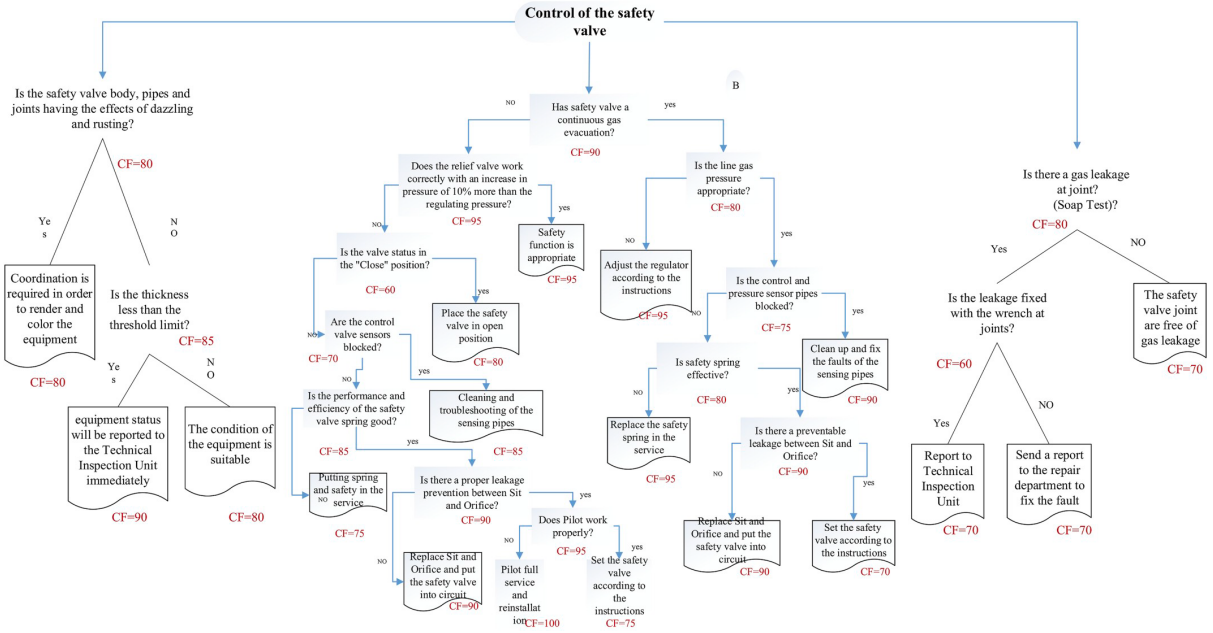
. Araujo Croizier, Juan Ignacio 10649

. Linares, Ramiro 10777

Ejercicio 1

Se desarrolló una Knowledge Base en lenguaje PROLOG para modelar el comportamiento de un sistema que se ocupa de la evaluación y el mantenimiento de una Válvula de Seguridad en una Estación de reducción de presión de gas natural.

Se eligió este dominio porque es en base al cual se explicó en clase este tema de la materia, y porque cuenta con un diagrama de árbol que muestra con gran claridad muchas de las decisiones que pueden tomarse frente a posibles situaciones. Estas últimas pueden corresponder tanto a un funcionamiento normal como a uno con errores presentes en él.



El código está conformado por 2 grandes partes. La parte A engloba todos los axiomas, los predicados invariantes que conforman la Base del Conocimiento. La parte B es mucho más corta, y está compuesta por los Ground Facts que representan los diversos estados que pueden existir en el sistema en cada caso particular.

Dentro de la parte A, hay a su vez 3 divisiones, las cuales se corresponden con cada una de las 3 grandes “ramas” del árbol de decisiones. La Rama Izquierda es la que parte de la consulta “¿El cuerpo de la Válvula de Seguridad, las Tuberías y las Junturas han sufrido las consecuencias del desgaste y el óxido?”, y tiene el mismo tamaño que la Rama Derecha. Esta última se inicia con la acción de “Verificar, mediante la Prueba del Jabón, si hay una fuga de gas en la Juntura”, y sus correspondientes bifurcaciones, de acuerdo al resultado de esta consulta.

A continuación, se muestra la Rama Izquierda, con sus decisiones posibles, en el código:

```
% RAMA IZQUIERDA

verificar(espesor_menor_que_umbral) :-
    estado(espesor_menor_que_umbral, correcto), writeln('El estado del equipo debe ser reportado inmediatamente a la Unidad de Inspecciones Tecnicas').


verificar(espesor_menor_que_umbral) :-
    estado(espesor_menor_que_umbral, incorrecto), writeln('La condicion del equipo es adecuada').



verificar(espesor_menor_que_umbral) :-
    estado(espesor_menor_que_umbral, desconocido),
    (
        (
            estado(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste, correcto),
            writeln('Se requiere coordinar para renderizar y pintar el equipo')
        )
        ;
        (
            estado(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste, incorrecto),
            writeln('Verificar si el espesor de las tuberias es menor que el umbral limite')
        )
        ;
        verificar(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste)
    ).



verificar(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste) :-
    estado(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste, desconocido),
    writeln('Verificar si el cuerpo de la Valvula de Seguridad, las Tuberias y las Junturas han sufrido las consecuencias del desgaste y el oxido').
```

En esta captura puede observarse un “resumen” de cómo trabaja el código: de acuerdo a como trabaja el motor de PROLOG, se comienza por los predicados correspondientes a las “hojas” del árbol de decisión. En este caso, la hoja más profunda es la que solicita verificar que “el espesor de las tuberías es menor al del umbral límite”, y desemboca en 2 posibles caminos, dependiendo de si el resultado es correcto o incorrecto.

Al ir descendiendo en el código, nos acercamos al nodo raíz, siguiendo para cada predicado axiomático una estructura similar a la mostrada; de acuerdo a cómo está armada esta KB, siempre se tienen 3 estados posibles para cada coyunto que representa uno de los nodos: **desconocido**, **correcto** e **incorrecto**. Así, cada predicado “verificar (nodo)” en particular se busca demostrar sólo cuando el estado que tiene dicho nodo es **desconocido**. Luego, se tienen 3 caminos posibles, los cuales dependen del nodo padre del evaluado. Esto es porque el nodo antecesor puede a su vez tener un estado **correcto**, **incorrecto** o, nuevamente, **desconocido**. En este último caso, lo que hace el algoritmo es proseguir a la demostración del predicado “verificar (nodo)” del nodo antecesor. De esta manera, el motor de PROLOG puede hacer un backtracking en el árbol de decisiones cada vez que el estado de un nodo es desconocido, pudiendo llegar incluso a los nodos raíz.

La Rama Central, por ser mucho más grande y estar a su vez subdividida en su comienzo por la posibilidad de que “la Válvula de Seguridad tenga evacuación continua” o no, está levemente particionada en el código, en “Rama Central Izq” y “Rama Central Der”. En el último tramo, y correspondiendo al nodo raíz mencionado anteriormente, ambas partes de la Rama Central convergen en el código, como se muestra a continuación:

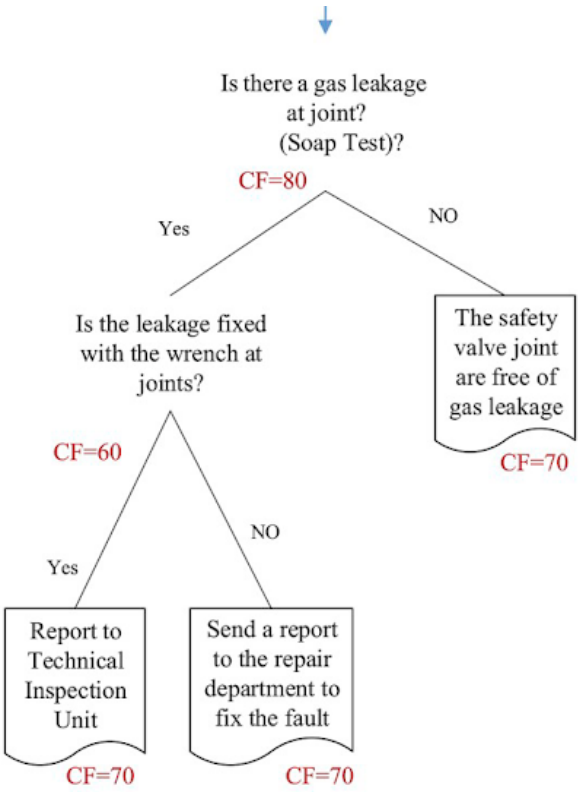
```
208  verificar(valvdeAlivio_ok_con_incremento_presion) :-
209      estado(valvdeAlivio_ok_con_incremento_presion, desconocido),
210      (
211          (
212              estado(valvdeSeg_tiene_evac_cont, incorrecto),
213              writeln('Verificar que la Valvula de Alivio esta bien con un incremento de la P de 10%')
214          )
215          ;
216          (
217              estado(valvdeSeg_tiene_evac_cont, correcto),
218              writeln('Verificar si la Linea de gas tiene la Presion correcta')
219          )
220          ;
221          verificar(valvdeSeg_tiene_evac_cont)
222      ).
```

```
verificar(presion_linea_apropiada) :-
    estado(presion_linea_apropiada, desconocido),
    (
        (
            estado(valvdeSeg_tiene_evac_cont, incorrecto),
            writeln('Verificar que la Valvula de Alivio esta bien con un incremento de la P de 10%')
        )
        ;
        (
            estado(valvdeSeg_tiene_evac_cont, correcto),
            writeln('Verificar si la Linea de gas tiene la Presion correcta')
        )
        ;
        verificar(valvdeSeg_tiene_evac_cont)
    ).


%----------------------


% RAMA CENTRAL

verificar(valvdeSeg_tiene_evac_cont) :-
    estado(valvdeSeg_tiene_evac_cont, desconocido),
    writeln('Verificar que la Valvula de Seg tiene evacuacion continua').
```

Puede observarse en las imágenes cómo ambos predicados llevan a la corroboración del predicado "verificar(valvdeSeg_tiene_evac_cont)".

En la parte final del código, la B, se enuncian los Ground Facts. Éstos buscan representar los valores que se obtendrían, en una situación real, de mediciones electrónicas de distintos puntos del proceso. En este ejercicio, los estados pueden configurarse libremente, al adjudicarles alguno de los valores que se manejan en la KB (desconocido, correcto o incorrecto). De este modo, puede lograrse y probarse cualquier combinación de situaciones, y el sistema experto responderá en consecuencia.

A continuación, se muestra un ejemplo de cómo funciona el código. Dado que en la clase se realizó el ejemplo con el Nodo Hoja "Piloto funciona correctamente", aquí configuraremos los Ground Facts de tal manera que lo que se busca es obtener un resultado para cuando hay una fuga de gas y ésta no es solucionable con una llave inglesa.



Para ello, los GF deberían ser, en el código, los siguientes (el resto se ha fijado con el valor "desconocido", ya que no influyen):

```
%B. Ground Facts que definen el estado incial:

estado(piloto_ok, desconocido). %RCI    %Nodo Hoja

estado(prev_fuga_entre_Sit_y_Orif, desconocido). %RCI

estado(rendim_resorte_valvdeSeg, desconocido). %RCI
estado(fuga_evitable_entre_Sit_y_Orif, desconocido).  %RCD  %Nodo Hoja

estado(sensores_valvdeControl_bloq, desconocido).    %RCI
estado(resorte_seg_es_efect, desconocido).  %RCD

estado(valv_cerrada, desconocido).   %RCI
estado(tuberias_sensor_bloq, desconocido).  %RCD

estado(espesor_menor_que_umbral, desconocido).  %RI  %Nodo Hoja
estado(valvdeAlivio_ok_con_incremento_presion, desconocido). %RCI
estado(presion_linea_apropiada, desconocido).   %RCD
estado(se_arregla_fuga_con_llave, incorrecto). %RD %Nodo Hoja

estado(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste, desconocido).   %RI
estado(valvdeSeg_tiene_evac_cont, desconocido).  %RC
estado(fuga_gas_en_junt_pruebaJabon, correcto).   %RD

estado(todo,correcto).
%Estos GF han determinado que el estado de TODOS los sistemas es "desconocido"
%Representan una situacion particular del sistema.
```

La consulta correspondiente, y su resultado, serán:

```
?- verificar(se_arregla_fuga_con_llave).
Enviar reporte al Departamento de Reparaciones para que arreglen la falla
true
```

Como se sugirió en clase, y dado que la interfaz no era muy cómoda para el usuario, se añadió finalmente, al inicio del programa, el siguiente "metapredicado":

```
%A. Axiomas

verificar(todo) :-
    estado(todo,correcto),
    (
    writeln('.Respecto al funcionamiento correcto del piloto:'),
    verificar(piloto_ok),
    writeln('\n.Respecto a si hay una fuga evitable entre Sit y Orificio:'),
    verificar(fuga_evitable_entre_Sit_y_Orif),
    writeln('\n.Respecto a si el espesor de las tuberias es esta por debajo del umbral limite:'),
    verificar(espesor_menor_que_umbral),
    writeln('\n.Respecto a si hay una fuga de gas que puede arreglarse con la llave inglesa:'),
    verificar(se_arregla_fuga_con_llave)
    ).
```

Así, ya no es necesario que el operador recuerde los nombres específicos de los procesos, ya que al realizar una consulta general, puede conocer las decisiones que debería tomar para cada uno de los 4 nodos hoja principales. A modo de ejemplo, se configurar los GF de manera semi aleatoria, como sigue:

```
%B. Ground Facts que definen el estado incial:

estado(piloto_ok, desconocido). %RCI    %Nodo Hoja

estado(prev_fuga_entre_Sit_y_Orif, incorrecto). %RCI

estado(rendim_resorte_valvdeSeg, desconocido). %RCI
estado(fuga_evitable_entre_Sit_y_Orif, desconocido).  %RCD  %Nodo Hoja

estado(sensores_valvdeControl_bloq, correcto).    %RCI
estado(resorte_seg_es_efect, desconocido).  %RCD

estado(valv_cerrada, desconocido).   %RCI
estado(tuberias_sensor_bloq, incorrecto).  %RCD

estado(espesor_menor_que_umbral, correcto).  %RI  %Nodo Hoja
estado(valvdeAlivio_ok_con_incremento_presion, desconocido). %RCI
estado(presion_linea_apropiada, incorrecto).   %RCD
estado(se_arregla_fuga_con_llave, correcto). %RD %Nodo Hoja

estado(valvdeSeg_tuberias_y_junt_sufriendo_consec_desgaste, desconocido).   %RI
estado(valvdeSeg_tiene_evac_cont, incorrecto).  %RC
estado(fuga_gas_en_junt_pruebaJabon, correcto).   %RD
```

La consulta general y su respuesta serán, luego:

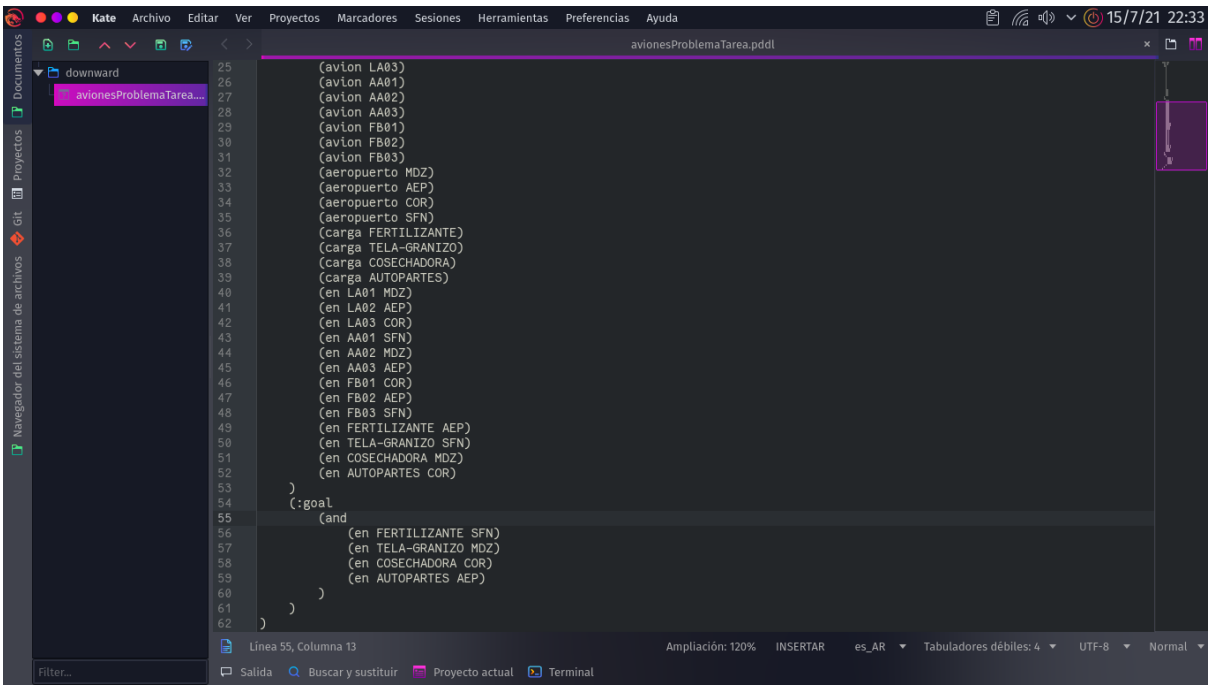
```
?- verificar(todo).
.Respecto al funcionamiento correcto del piloto:
Reemplazar Sit y Orificio, y colocar la Valvula de Seguridad en el circuito (RAMA IZQ)

.Respecto a si hay una fuga evitable entre Sit y Orificio:
Verificar si el Resorte de Seguridad es aun efectivo

.Respecto a si el espesor de las tuberias es esta por debajo del umbral limite:
El estado del equipo debe ser reportado inmediatamente a la Unidad de Inspecciones Tecnicas, dado que el espesor

.Respecto a si hay una fuga de gas que puede arreglarse con la llave inglesa:
La fuga no es reparable. Enviar reporte a la Unidad de Inspecciones Tecnicas
true
```

De esta manera, se puede tener una visión integral, sencilla y veloz de todo el sistema, sus posibles fallas y las acciones que deberían tomarse para compensarlas.

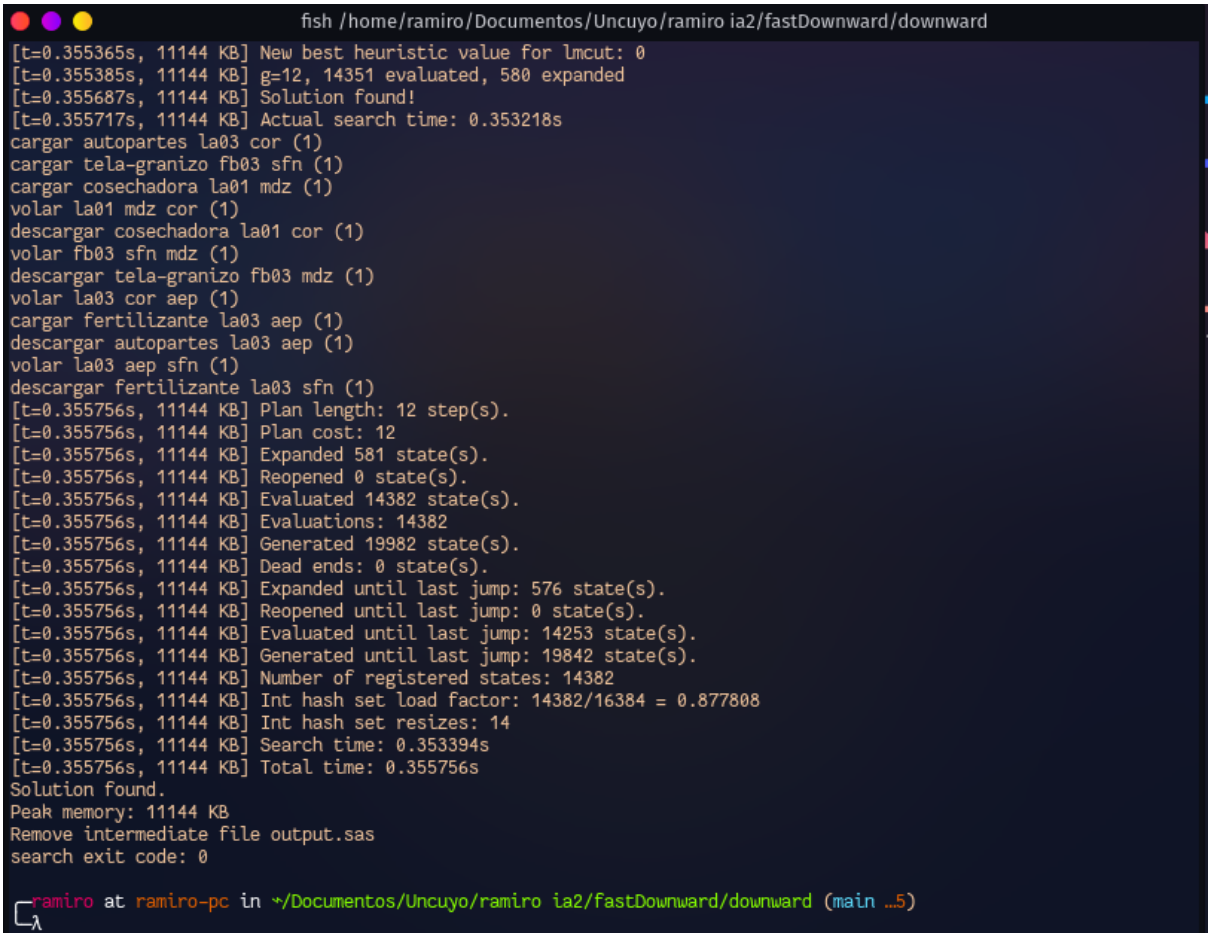
Ejercicio 2

1. **Utilizando el planificador [Fast Downward]**
   a. **Modelar en PDDL el dominio de transporte aéreo de cargas y definir algunas instancias del problema**

Se modela el ejercicio de dominio de transporte aéreo dado en clases, y la meta se define en la **tarea**.



Se definen aviones, aeropuertos y cargas. Se inicializa cada una de estas entidades con valores, y ya teniendo definido el estado objetivo, se ejecuta el planificador Fast-Downard



Se puede observar que encontró la solución al problema en 0.353218s con un costo de plan de 12, esto se debe a que cada paso tiene un costo de 1. Para que los costos sean más “reales” se podría hacer que la carga y descarga valga 1 y todos los vuelos tengan un costo grande, también que al mismo tiempo dependa de la distancia entre aeropuertos su valor de costo.

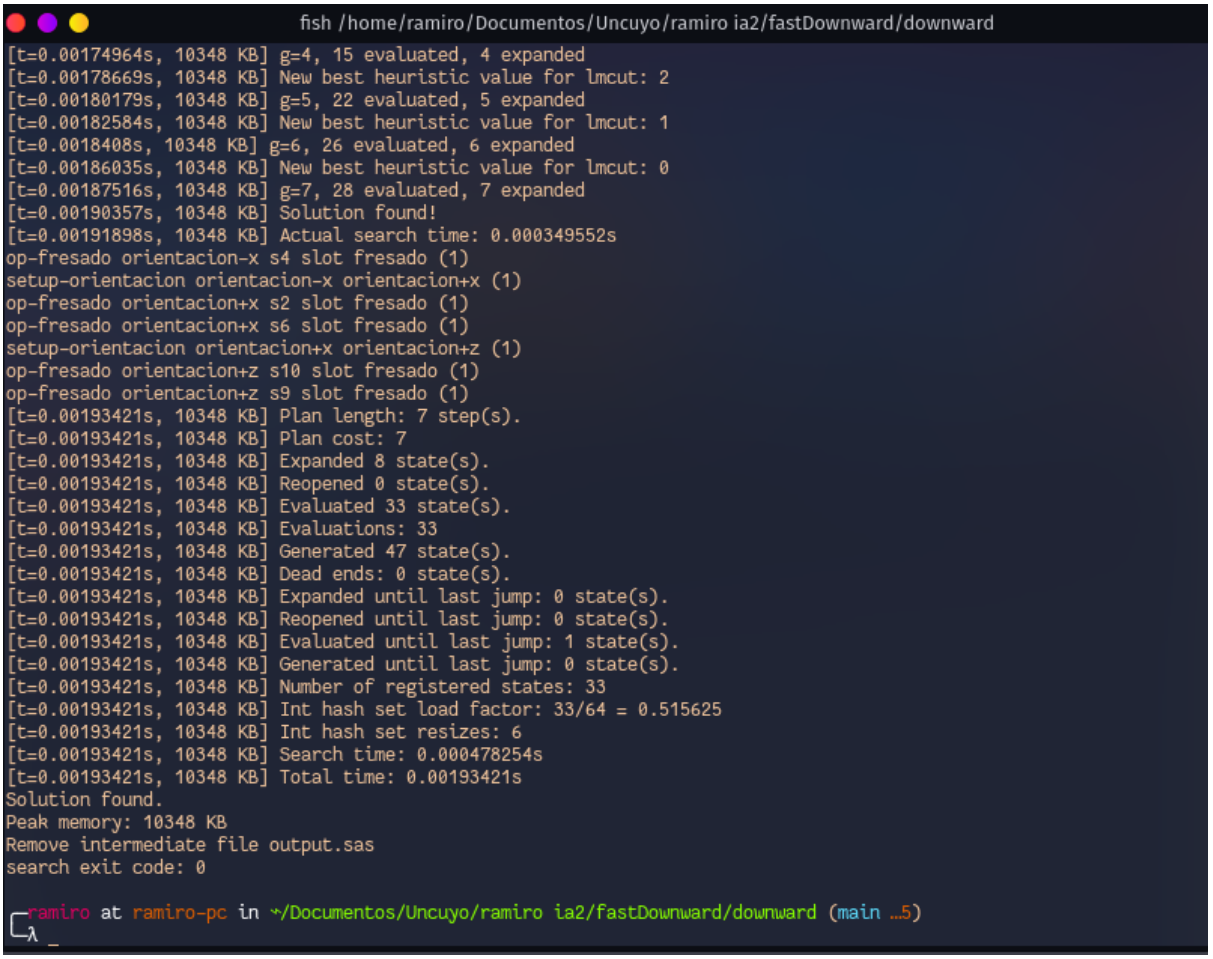
En resumen, la solución del ejercicio por parte del planificador fue cargar las 3 cargas, volar al aeropuerto de 1 carga, descargar en su estado objetivo correspondiente, y así sucesivamente para descargar las 3 cargas cumpliendo la meta que le fue impuesta en la tarea.

b. Modelar en PDDL y definir al menos una instancia del problema para alguna de las siguientes opciones

- **Un dominio a su elección**
- **Planificación de Procesos Asistida por Computadora (CAPP, Computer-Aided Process Planning).**

En el modelado de CAPP, vemos que tenemos diferentes tipos de entidades o predicados, los cuales son orientación (de la pieza), features (definidos en la imagen del ejercicio), tipo (que sería si es agujero pasante, ciego, o ranura), feature-tipo relaciona un feature con un agujero en particular, orientación-pieza (define el estado de la pieza), orientación-feature (orientación de la herramienta), fabricable-tipo (tipo de feature con la que se fabrica la pieza), fabricada (cuando la pieza ya esta fabricada con los feature que deseábamos en el goal).

Una vez definidos el dominio y la tarea, ejecutamos el planificador:



Encuentra la solución en 0.0003s, con un costo de 7 pasos, nuevamente no definimos costos distintos por lo que cada paso tiene un costo de 1.

Ejercicio 3

Implementar un sistema de inferencia difusa para controlar un péndulo invertido

- Asuma que el carro no tiene espacio restringido para moverse
- Definir variables lingüísticas de entrada y salida, particiones borrosas, operaciones borrosas para la conjunción, disyunción e implicación, reglas de inferencia (cubrir todas las posibles combinaciones de valores borrosos de entrada en la base de reglas)
- Utilice el modelo del sistema carro-péndulo

Tabla FAM (Fuzzy Abstract Machine), donde NG es negativo grande, NP negativo pequeño, Z cero, PP positivo pequeño, PG positivo grande

Velocid/ Theta	NG	NP	Z	PP	PG
NG	**PG**	**PG**	**PG**	**PP**	**Z**
NP	**PG**	**PG**	**PP**	**Z**	**NP**
Z	**PG**	**PP**	**Z**	**NP**	**NG**

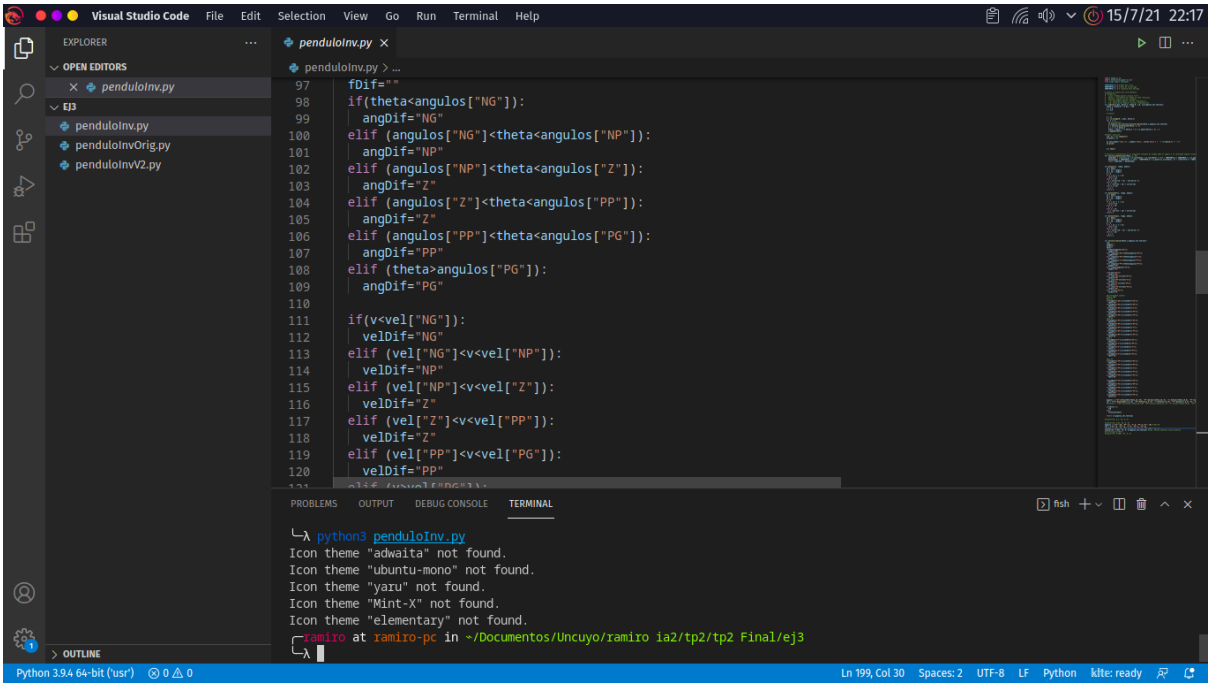
PP	**PP**	**Z**	**NP**	**NG**	**NG**
PG	**Z**	**NP**	**NG**	**NG**	**NG**

Las reglas equivalentes serían:

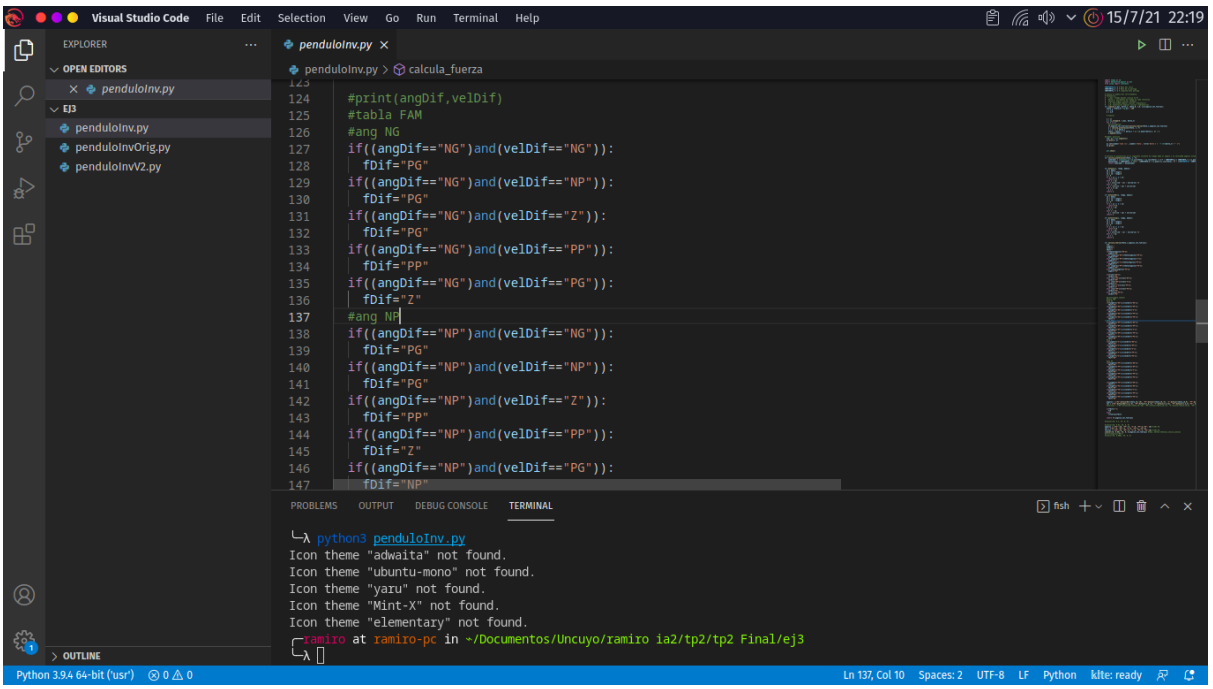
Velocidad es "NP", theta es "PP", entonces F es "Z"

Haciendo la misma analogía con el resto de los valores de la tabla podemos obtener las reglas equivalentes.

Lo primero que se realizó fue pasar los valores numéricos de theta y velocidad a sus equivalentes



Una vez obtenido theta y velocidad se aplicó la tabla FAM misma:



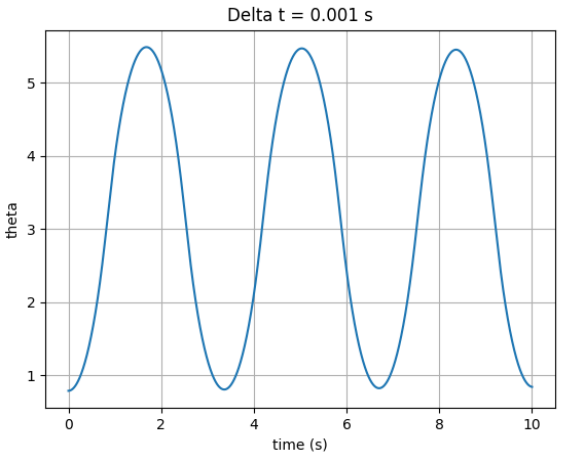
Por una cuestión de extensión del informe no se mostrará la totalidad de dicha tabla.

Una vez planteadas las reglas de la tabla FAM, aplicamos un control mediante lógica difusa. Al tener las dos entradas que son theta y velocidad, calculamos una respuesta difusa.

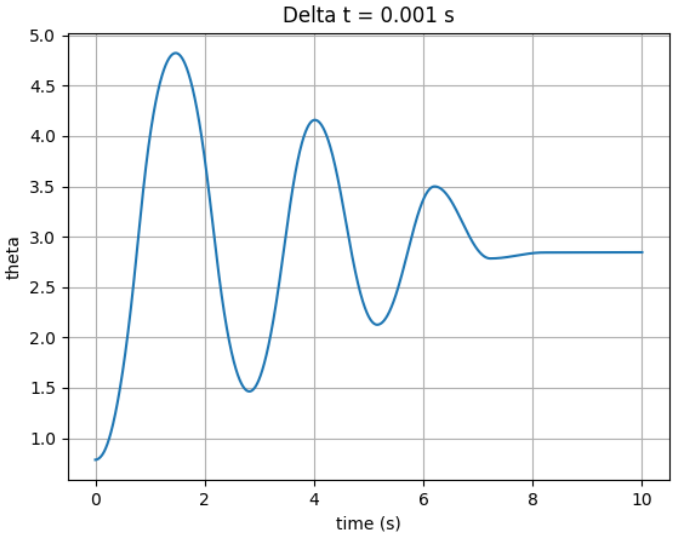
Una vez obtenida podemos conocer la magnitud y dirección de la fuerza que debemos aplicar, realizando así una simulación de un sistema de control que tiende a la posición que definimos en el ejercicio y en clases.

Comparando el ejercicio simulado dado por el profesor y la respuesta dada (ambos graficos con deltat=0.001s)

Original sin control



Pendulo invertido controlado aplicando tabla FAM



Se observa que el sistema tiene una respuesta amortiguada.