Trabajo Práctico Nº 2
Razonamiento

EJERCICIOS

Ejercicio 1. **Modelado de una base de conocimientos (KB, Knowledge Base) en PROLOG.**
Además de la base de conocimientos, plantee preguntas que el sistema puede contestar, incluyendo preguntas cerradas (verdadero/falso) y preguntas abiertas (ej: qué debe hacerse?). Además de las reglas axiomáticas, incluya algunos ground facts (hechos) necesarios para codificar una instancia del problema.

Resolución del ejercicio:
Para el desarrollo de dicho ejercicio se elige como alternativa el modelado de evaluación y mantenimiento de una válvula de seguridad de una estación de reducción de presión de gas.
Para hacer el modelado del problema en lenguaje PROLOG se utilizó el siguiente árbol de búsqueda:

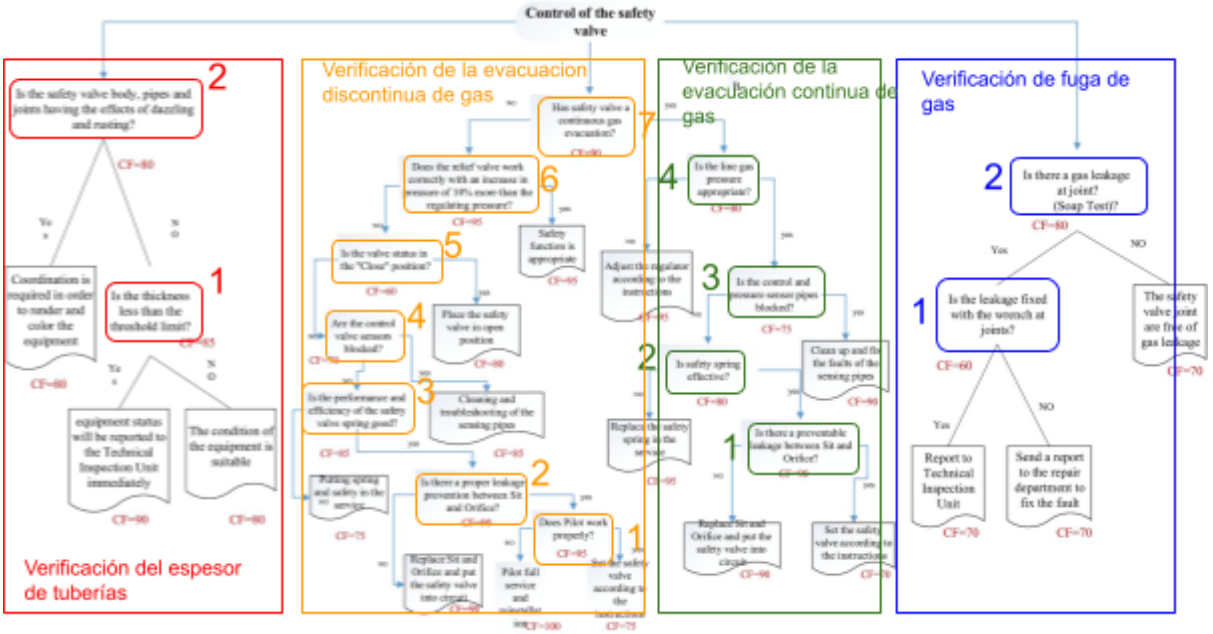

Figura 1.

Se han dividido en 4 colores distintos las ramas que dicho árbol presenta. En el siguiente cuadro se representan los ground facts que se implementaron para modelar cada rama.

Ground Facts:

Rama		Ground Facts	Variable	Valores Posibles
1 - Verificación del espesor de tuberías	1	**estado(thickness, X) :- X is (-1)**	X	(-1) - Sin valor asignado (0 - 3) - Espesor no aceptado (>3) - Espesor aceptado
	2	**estado(safety_system_having_dazzling_rusting_efects, desconocido)**	No	1- desconocido 2- no 3- yes
2 - Verificación de la evacuación discontinua de gas	1	**estado(pilot_ok, desconocido)**	No	1- desconocido 2- no 3- yes
	2	**estado(sit_and_orifice_ok, desconocido)**	No	1- desconocido 2- no 3- yes
	3	**estado(efficiency_of_safety_valve_spring, X) :- X is (-1)**	Si	1- (-1) Desconocido 2- (X<70) Eficiencia no aceptada 3- (X>=70) Eficiencia aceptada
	4	**estado(control_valve_sensors, Tension) :- Tension is (-1)**	Si	1- (-1) Desconocido 2- (X<5) Tensión baja 3- (5<=X<=12) Tensión aceptada
	5	**estado(valve_status, opened) :- valve_opening_status(O), O>=60**	Si. "Valve_status" se define según sea el valor de O.	1- desconocido: O>=(-1) 2- closed: 0<O<60% 3- opened: O>=60%
		estado(valve_status, closed) :- valve_opening_status(O), O<60, O>0		
		estado(valve_status, desconocido) :- valve_opening_status(O), O==(-1)		

	6	**estado(pressure_of_the_relief_valve, X) :- X is (-1)** **regulating_pressure(RP) :- RP is 300**	Si	1- (-1) Desconocido 2- (X<(RP+(RP*(10/100)) Presion de valvula baja 3- (X>=(RP+(RP*(10/100)) Presion de valvula ok
	7	**estado(safety_valve_has_continuous_evacuation, desconocido)**	No	1- desconocido 2- no 3- yes
3 - Verificación de la evacuación continua de gas	1	**estado(line_gas_pressure, X) :- X is (-1)** **regulating_pressure(RP) :- RP is 300**	Si	1- (-1) desconocido 2- (0<X<RP) Presión baja 3- (X=RP) Presion ok
	2	**estado(pressure_sensor_pipes_blocked, desconocido)**	No	1- desconocido 2- no 3- yes
	3	**estado(safety_spring_ok, desconocido)**	No	1- desconocido 2- no 3- yes
	4	**estado(preventable_leakage_between_sit_and_orifice, desconocido)**	No	1- desconocido 2- no 3- yes
4 - Verificación de fuga de gas	1	**estado(gas_leakage_at_joint, desconocido)**	No	1- desconocido 2- no 3- yes
	2	**estado(leakage_fixed_with_wrench, desconocido)**	No	1- desconocido 2- no 3- yes

Los ground facts se encuentran numerados de abajo hacia arriba, de acuerdo a los que se indica en Figura 1. Se enumeran de ese modo para establecer una congruencia entre el modelo y la lógica de búsqueda de PROLOG, que realiza búsqueda por encadenamiento hacia atrás y en profundidad. Por lo que, la definición de cada elemento de la base de conocimientos se resuelve de abajo hacia arriba. De manera que, en la estructura del programa quedan los nodos raíz de cada rama al final de las definiciones, y las hojas al comienzo.

Estructura del programa:

Se observa que el programa se estructura en términos de funciones "verificar(ElementoDelSistema)" como la siguiente:

```
verificar(line_gas_pressure) :-
    (
        (estado(line_gas_pressure, P), P==(-1),
estado(safety_valve_has_continuous_evacuation, yes), writeln('Verificar
line gas pressure'));
        (estado(line_gas_pressure, P), regulating_pressure(RP), P<RP,
P>0, writeln('Please adjust the regulator acording to the
instructions'));
        verificar(safety_valve_has_continuous_evacuation)
    ).
```

las cuales consisten en cláusulas de Horn que indicarán "True" siempre que alguna de las secuencias de literales se cumpla. Para éste caso en particular "verificar(line_gas_pressure)=True" siempre que:

a. Se tenga una línea de gas con presión P, P=desconocida y la válvula de seguridad presente evaluación continua, se mostrará al usuario el mensaje "Verificar line gas pressure"
b. Se tenga una línea de gas con presión P, una presión de regulación RP de referencia y 0<P<RP, se mostrará al usuario el mensaje "Please adjust the regulator according to the instructions"
c. Si no se cumple ninguna de las condiciones anteriores significa que no hay una asignación consistente de valores a las variables "line_gas_pressure" y "safety_valve_has_continuous_evacuation", por lo que se infiere que esta última no está definida y se la debe verificar.

Se observa que cada uno de los ítems anteriores representa una conjunción de literales, es decir que los componentes de una sentencia se separan entre sí por y lógico (","); y a su vez las sentencias se separan entre sí por disyunciones (";").

Del mismo modo se estructuran todos los componentes que el sistema de mantenimiento debe verificar para definir las condiciones en que se encuentra, y poder indicarle al operario el procedimiento a realizar.

Para verificar el sistema completo se definió una función "verificar(sistema)":

```
verificar(sistema) :-
    estado(sistema, desconocido), writeln('Analizando sistema...'),
    (
        verificar(thickness), verificar(leakage_fixed_with_wrench),
        verificar(pilot_ok);
        verificar(pressure_of_the_relief_valve)
    ).
```

que resulta en una conjunción en la cual los literales son las funciones de verificación de cada una de las ramas. Es decir que dicha función ejecuta los procedimiento de verificación de cada una de las hojas del árbol, comenzando por la rama de la izquierda

"verificar(thickness)" y finalizando con la de la derecha "verificar(leakage_fixed_with_wrench)". Se observa que el nodo central del árbol se resuelve como una disyunción entre las ramas "verificar(piloto_ok)" y "verificar(pressure_of_the_relief_valve)", dado que no pueden ocurrir ambas en simultáneo.

Preguntas cerradas:

Son preguntas que se realizan al sistema para que indique "True" en caso de que la respuesta sea afirmativa o "False", de ser negativa.

Pueden realizarse preguntas sobre el estado de alguno de los elementos del sistema,por ejemplo:

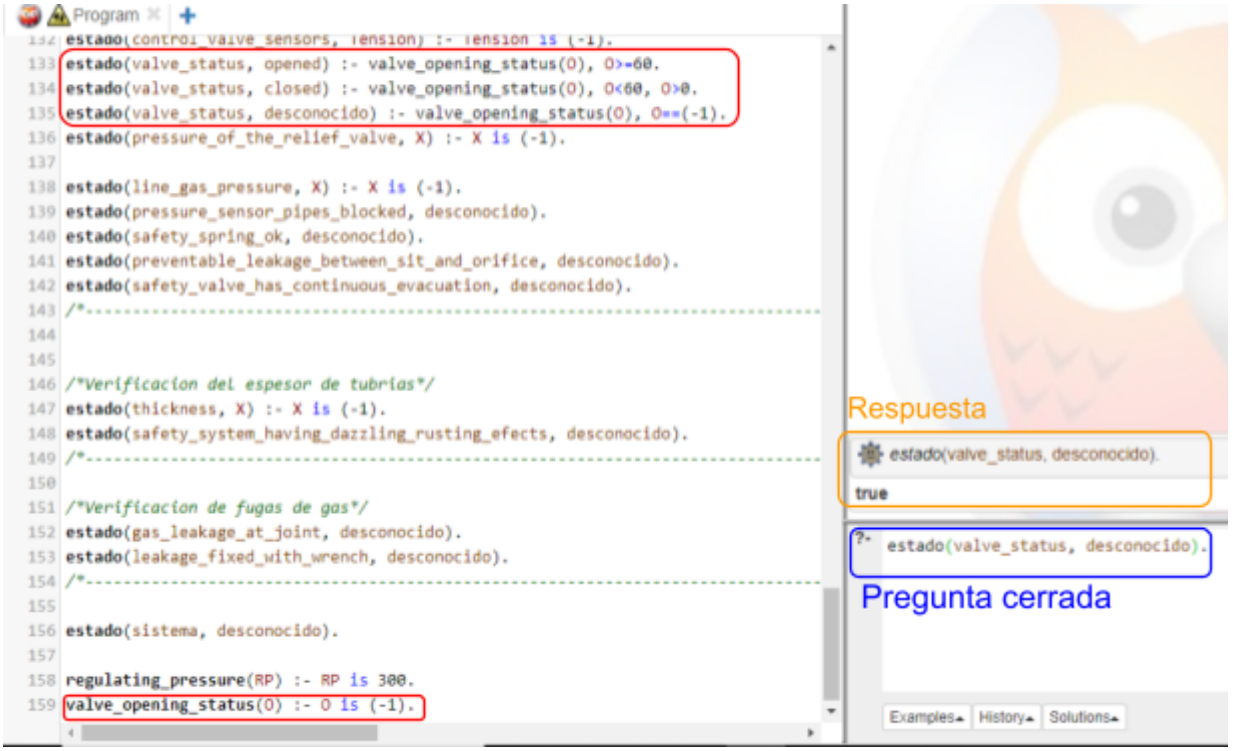

Figura 2.

En caso de que se pregunte si el estado de la válvula es cerrado, cuando en realidad es desconocido, el sistema devolverá "false" porque la variable O=(-1), es decir, es desconocida.

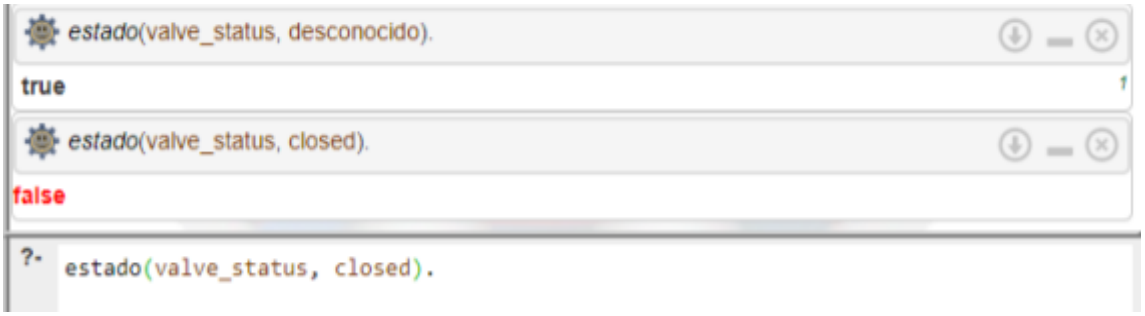

Figura 3.

Preguntas abiertas:
En este caso, puede consultarse al sistema que verifique algún elemento en particular, para que nos indique cual es la acción a realizar para que el funcionamiento sea óptimo.

```
verificar(safety_system_having_dazzling_rusting_efects) :-
                (
                    (estado(safety_system_having_dazzling_rusting_efects, desconocido), writeln('Verificar safety system dazzling and rusting efects'));
                    (estado(safety_system_having_dazzling_rusting_efects, yes), writeln('Coordination is required in order to render and color the equipment'))
                    /*verificar(piloto_ok)*/
                ).

verificar(thickness) :-
                (
                    (estado(thickness, Esp), Esp==(-1), estado(safety_system_having_dazzling_rusting_efects, no),writeln('Verificar system thickness'));
                    (estado(thickness, Esp), Esp<3, Esp>0, writeln('Please report to technical inspection unit, little tickness'));
                    (estado(thickness, Esp), Esp>=3, writeln('No dazzling and rusting efects , equipment in good conticions'));
                    verificar(safety_system_having_dazzling_rusting_efects)
                ).
```

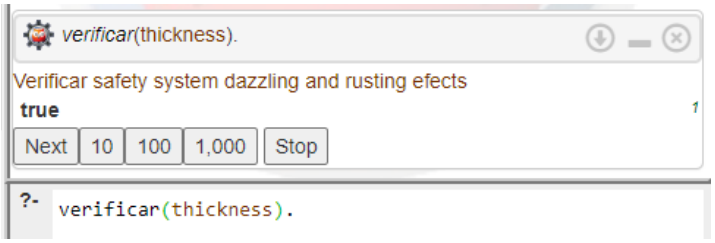

Figura 4.

El sistema nos está indicando que previo a verificar el espesor de las tuberías, deben verificarse los efectos del desgaste sobre éstas. Esto se debe a que la variable “Esp=(-1)”, es desconocida.
También puede consultarse por el sistema completo realizando la consulta “verificar(sistema)”, de manera que el sistema indique todas las acciones que deben realizarse sobre los elementos que lo componen.

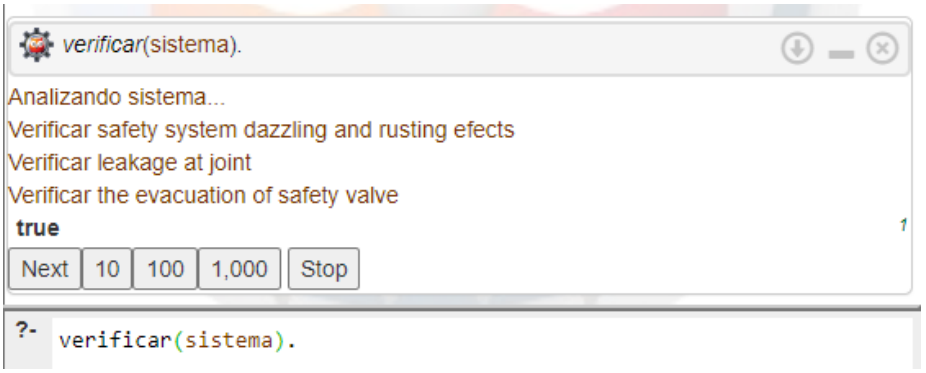

Figura 5.

Ejercicio 2. Uso de PDDL con el planificador **Fast Downward**

A) Modelar en PDDL el dominio de transporte aéreo de cargas y definir algunas instancias del problema.

Para resolver este ejercicio utilizamos los siguientes archivos en nuestro repositorio de github:

- Dominio-Aviones
- Problema-Aviones

Y ejecutamos el problema con el siguiente solver online Fast-Downward:

- Estado Inicial:

  (en FERTILIZANTE AEP)
  (en TELA-GRANIZO SFN)
  (en COSECHADORA MDZ)
  (en AUTOPARTES COR)

- Estado Objetivo:

  (en FERTILIZANTE SFN)
  (en TELA-GRANIZO MDZ)
  (en COSECHADORA COR)
  (en AUTOPARTES AEP)

- Solución:
  - (cargar tela-granizo aa01 sfn)
  - (volar aa01 sfn mdz)
  - (cargar cosechadora aa01 mdz)
  - (descargar tela-granizo aa01 mdz)
  - (volar aa01 mdz cor)
  - (cargar autopartes aa01 cor)
  - (descargar cosechadora aa01 cor)
  - (volar aa01 cor aep)
  - (descargar autopartes aa01 aep)
  - (cargar fertilizante aa01 aep)
  - (volar aa01 aep sfn)
  - (descargar fertilizante aa01 sfn)
  - ; cost = 12 (unit cost)

Para ver cómo funciona hicimos también unos cambios en el objetivo del problema:

- Estado Inicial (***igual que antes***):

  (en FERTILIZANTE AEP)
  (en TELA-GRANIZO SFN)
  (en COSECHADORA MDZ)
  (en AUTOPARTES COR)

- Estado Objetivo (***cambiamos destinos de las cargas***):

  (en FERTILIZANTE MDZ)
  (en TELA-GRANIZO COR)
  (en COSECHADORA AEP)

(en AUTOPARTES SFN)

- Solución:
  - (cargar tela-granizo aa01 sfn)
  - (volar aa01 sfn cor)
  - (cargar autopartes aa01 cor)
  - (descargar tela-granizo aa01 cor)
  - (volar aa01 cor sfn)
  - (descargar autopartes aa01 sfn)
  - (cargar cosechadora aa02 mdz)
  - (volar aa02 mdz aep)
  - (descargar cosechadora aa02 aep)
  - (cargar fertilizante aa02 aep)
  - (volar aa02 aep mdz)
  - (descargar fertilizante aa02 mdz)
  - ; cost = 12 (unit cost)

cargar autopartes la03 cor (1)
volar la03 cor aep (1)
cargar fertilizante la03 aep (1)
descargar autopartes la03 aep (1)
volar la03 aep sfn (1)
cargar tela-granizo la03 sfn (1)
descargar fertilizante la03 sfn (1)
volar la03 sfn mdz (1)
descargar tela-granizo la03 mdz (1)
cargar cosechadora la03 mdz (1)
volar la03 mdz cor (1)
descargar cosechadora la03 cor (1)

- **VUELOS-VACUNAS-COVID**

Luego hicimos unas pruebas modificando el ejercicio de los vuelos, para aplicarlo al problema de los vuelos para buscar vacunas de COVID. Fue a modo de probar el planificador, los archivos usados fueron los siguientes:

- Dominio-Vuelos-Vacunas
- Problema-Vuelos-Vacunas

El problema consiste en **buscar paquetes** de **vacunas** en sus países de **origen** y **llevarlos** a sus países de **destino**. Se probó con algunos pocos porque sino el planificador daba error, en especial el fast-downward online (que nunca anduvo).

Los datos usados en el problema fueron los siguientes:

- Estado Inicial (origen de las vacunas):

(en COVISHIELD INDIA)
(en SINOPHARM CHINA)
(en ASTRAZENECA UK)
(en SPUTNIK-V RUSIA)
(en PFIZER EEUU)
(en MODERNA EEUU)

- Estado Objetivo (destino de las vacunas):

(en ASTRAZENECA ARG)
(en SINOPHARM ARG)
(en SPUTNIK-V ARG)
(en COVISHIELD ARG)
(en PFIZER ESP)
(en MODERNA ESP)

Para la solución vamos a probar con varios métodos de búsqueda para ver distintos resultados:

- Solución ***lazy_greedy([ff()], preferred=[ff()])*** (por defecto) - Tiempo ***0.0504558s***:
  - (volar aa03 esp eeuu)
  - (cargar pfizer aa03 eeuu)
  - (cargar moderna aa03 eeuu)
  - (volar aa03 eeuu esp)
  - (descargar pfizer aa03 esp)
  - (descargar moderna aa03 esp)
  - (volar aa01 arg china)
  - (cargar sinopharm aa01 china)
  - (volar aa01 china arg)
  - (descargar sinopharm aa01 arg)
  - (volar aa01 arg india)
  - (cargar covishield aa01 india)
  - (volar aa01 india arg)
  - (descargar covishield aa01 arg)
  - (volar aa01 arg rusia)
  - (cargar sputnik-v aa01 rusia)
  - (volar aa01 rusia arg)
  - (descargar sputnik-v aa01 arg)
  - (volar aa01 arg uk)
  - (cargar astrazeneca aa01 uk)
  - (volar aa01 uk arg)
  - (descargar astrazeneca aa01 arg)
  - **; cost = 22 (unit cost)**

- Solución ***astar(lmcut())*** - Tiempo ***9.97665s***:

- (volar la03 esp china)
- (cargar sinopharm la03 china)
- (volar la03 china india)
- (cargar covishield la03 india)
- (volar la03 india rusia)
- (cargar sputnik-v la03 rusia)
- (volar la03 rusia uk)
- (cargar astrazeneca la03 uk)
- (volar la03 uk arg)
- (descargar sputnik-v la03 arg)
- (descargar sinopharm la03 arg)
- (descargar covishield la03 arg)
- (descargar astrazeneca la03 arg)
- (volar la03 arg eeuu)
- (cargar pfizer la03 eeuu)
- (cargar moderna la03 eeuu)
- (volar la03 eeuu esp)
- (descargar pfizer la03 esp)
- (descargar moderna la03 esp)
- **; cost = 19 (unit cost)**

- Solución ***astar(ipdb())*** - Tiempo ***+120s (No terminó antes de los 2 min)***
- Solución ***astar(blind())*** - Tiempo ***+120s (No terminó antes de los 2 min)***
- Solución ***lazy_greedy()*** - Con otros ***lazy_greedy*** directamente daba error (no se ejecutaba).

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).**

Para resolver este ejercicio utilizamos los siguientes archivos en nuestro repositorio de github:
- Dominio-CAPP
- Problema-CAPP

Y ejecutamos el problema con el siguiente solver online Fast-Downward:

- Estado Inicial:

  (orientacion-pieza orientacion-X)

  (orientacion-feature s2 orientacionX)
  (orientacion-feature s4 orientacion-X)
  (orientacion-feature s6 orientacionX)

(orientacion-feature s8 orientacion-X)
(orientacion-feature s9 orientacionZ)
(orientacion-feature s10 orientacionZ)

(orientacion-feature h1 orientacionZ)
(orientacion-feature h3 orientacionZ)
(orientacion-feature h5 orientacionZ)
(orientacion-feature h7 orientacionX)
(orientacion-feature h9 orientacionX)
(orientacion-feature h11 orientacionX)
(orientacion-feature h12 orientacionX)

(fabricable slot fresado)
(fabricable through-hole taladrado)

- Estado Objetivo:

(fabricada s2)
(fabricada s4)
(fabricada s6)
(fabricada s8)
(fabricada s9)
(fabricada s10)

(fabricada h1)
(fabricada h3)
(fabricada h5)
(fabricada h7)
(fabricada h9)
(fabricada h11)
(fabricada h12)

- Solución (con Fast-Downward Online):
    - (op-fresado orientacion-x s4 slot fresado)
    - (op-fresado orientacion-x s8 slot fresado)
    - (setup-orientacion orientacion-x orientacionx)
    - (op-fresado orientacionx s2 slot fresado)
    - (op-fresado orientacionx s6 slot fresado)
    - (op-taladrado orientacionx h11 through-hole taladrado)
    - (op-taladrado orientacionx h12 through-hole taladrado)
    - (op-taladrado orientacionx h7 through-hole taladrado)
    - (op-taladrado orientacionx h9 through-hole taladrado)
    - (setup-orientacion orientacionx orientacionz)
    - (op-fresado orientacionz s10 slot fresado)
    - (op-fresado orientacionz s9 slot fresado)

- (op-taladrado orientacionz h1 through-hole taladrado)
- (op-taladrado orientacionz h3 through-hole taladrado)
- (op-taladrado orientacionz h5 through-hole taladrado)
- ; cost = 15 (unit cost)

- Solución (con Visual Studio):

Figura 6.

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 siguiente modelo del sistema carro-péndulo

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - ml\dot{\theta}^2 Sin\theta}{M+m} \right)}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{M+m} \right)}$$

$\theta' = \theta' + \theta''\Delta t$

$\theta = \theta + \theta'\Delta t + (\theta''\Delta t2)/\ 2$

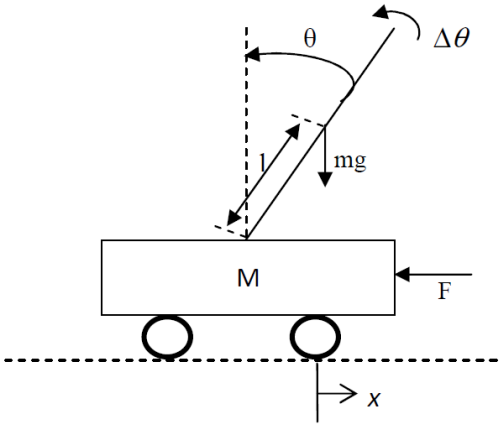

Figura 7.

Variables lingüísticas:

- theta (posición angular del péndulo):

  Conjuntos borrosos: [‘NG’:Negativo grande, ‘NP’: Negativo pequeño, ‘Z’: cero,’PP’: positivo pequeño, ‘PG’: positivo grande]

  Universo del discurso: [-0.75,0.75]

  Particiones borrosas: ['NG':(-0.75,-0.25), 'NP':(-0.5,0), 'Z':(-0.25,0.25), 'PP':(0,0.5), 'PG':(0.25,0.75)]

- dtheta (velocidad angular del péndulo):

  Particiones borrosas: [‘NG’:Negativo grande, ‘NP’: Negativo pequeño, ‘Z’: cero,’PP’: positivo pequeño, ‘PG’: positivo grande]

  Universo del discurso:[-3,3]

  Particiones borrosas: ['NG':(-3,-1), 'NP':(-2,0), 'Z':(-1,1), 'PP':(0,2), 'PG':(1,3)]

- F (Fuerza aplicada sobre el carro):

  Particiones borrosas: [‘NG’:Negativo grande, ‘NP’: Negativo pequeño, ‘Z’: cero,’PP’: positivo pequeño, ‘PG’: positivo grande]

  Particiones borrosas: ['NG':(-150,-50), 'NP':(-100,0), 'Z':(-50,50), 'PP':(0,100), 'PG':(50,150)]

Universo del discurso: [-150,150]

Operaciones borrosas:

- Conjunción: min()
- Disyunción: max()

Reglas de inferencias:

[('NG','NG','PG'),('NG','NP','PG'),('NG','Z','PG'),('NG','PP','PP'),('NG','PG','PP'),
('NP','NG','PG'),('NP','NP','PG'),('NP','Z','PP'),('NP','PP','PP'),('NP','PG','NP'),
('Z','NG','PG'),('Z','NP','PP'),('Z','Z','Z'),('Z','PP','NP'),('Z','PG','NG'),
('PP','NG','PG'),('PP','NP','PP'),('PP','Z','NP'),('PP','PP','NP'),('PP','PG','NG'),
('PG','NG','PP'),('PG','NP','NP'),('PG','Z','NP'),('PG','PP','NG'),('PG','PG','NG')]

Donde los primeros dos elementos de cada tupla son los antecedente (theta, dtheta) y el último es el consecuente (Fuerza) y se debe leer como:

“Si theta es NG y dtheta es NG entonces F es PG”

Luego se combinaron las reglas que tienen el mismo consecuente para así achicar la base de conocimiento.

Para llevar a cabo la lógica difusa se creó la clase controlador la cual tiene los siguiente métodos:

- borrosificador (xtheta, xdtheta): este recibe con argumentos valores de posición y velocidad y mediante el método membership define el valor de pertenencia a cada conjunto borroso de estas variables.
- membership (a1,a2,X,i): calcula el valor de pertenencia de una variable a un conjunto borroso
- motordeinferencias (xtheta, xdtheta): realiza las inferencias haciendo uso de la base de conocimientos.
- desborrificador(): Desborrifica el valor de la variable de salida en este caso la Fuerza.

Resultados:

En la figura 1 se puede observar la oscilación del sistema sin ningún tipo de control.

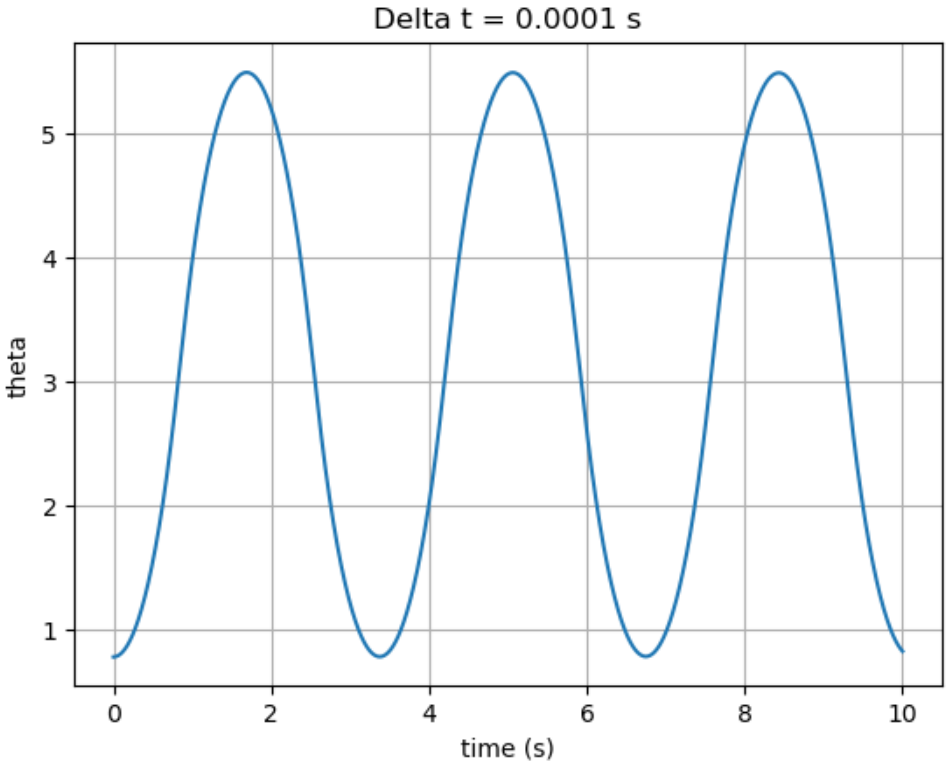

Figura 8.

A continuación en la figura 2 se puede observar la respuesta del sistema con el control, el figura 3 se muestra la velocidad, en la 4 la aceleración y en la 5 la acción de control.

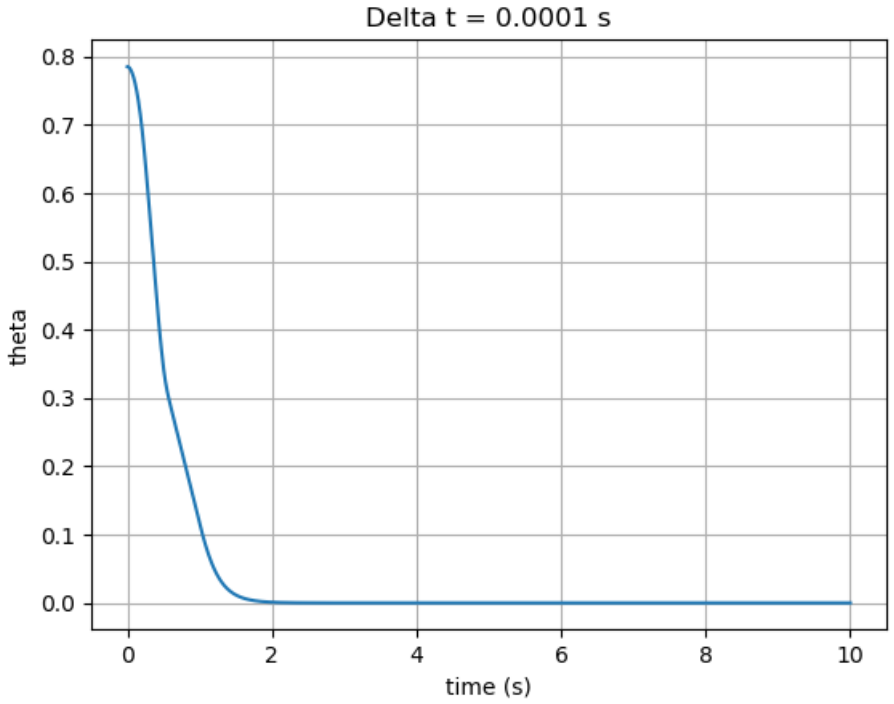

Figura 9.

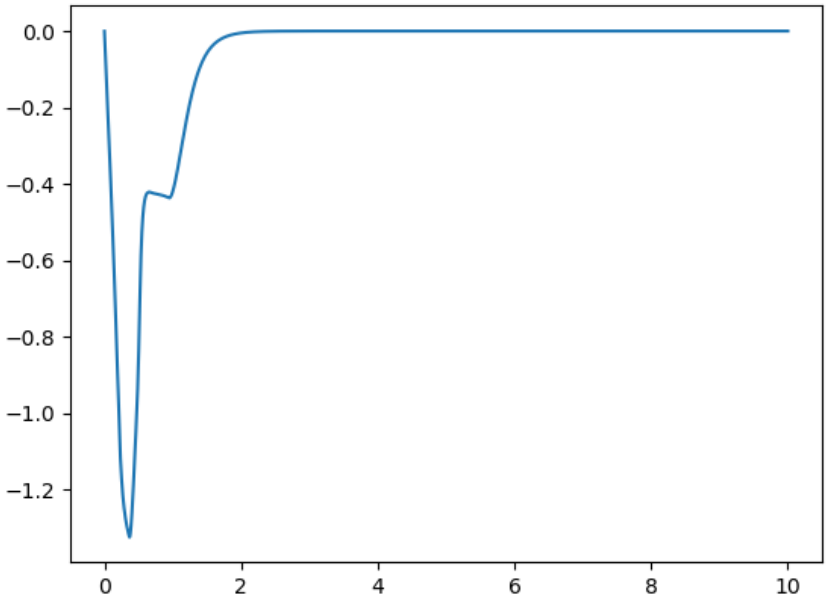

Figura 10.

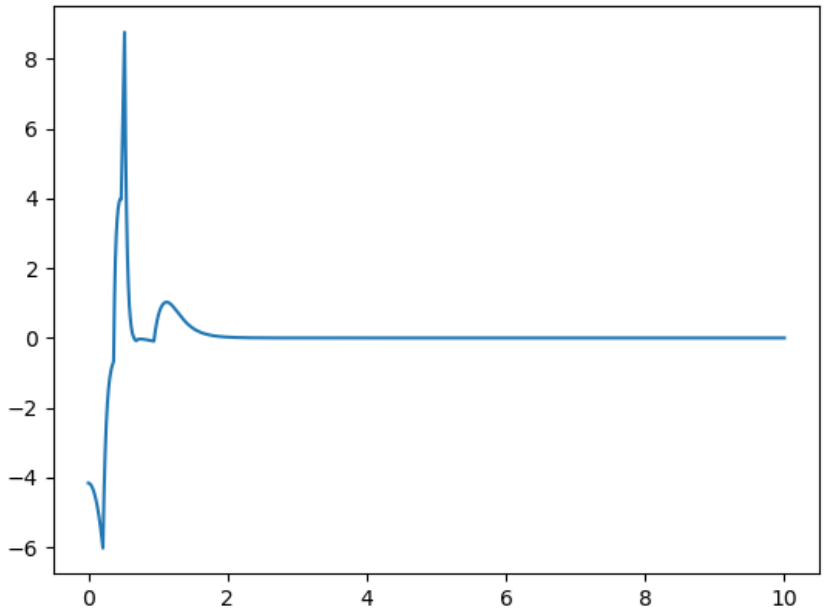

Figura 11.

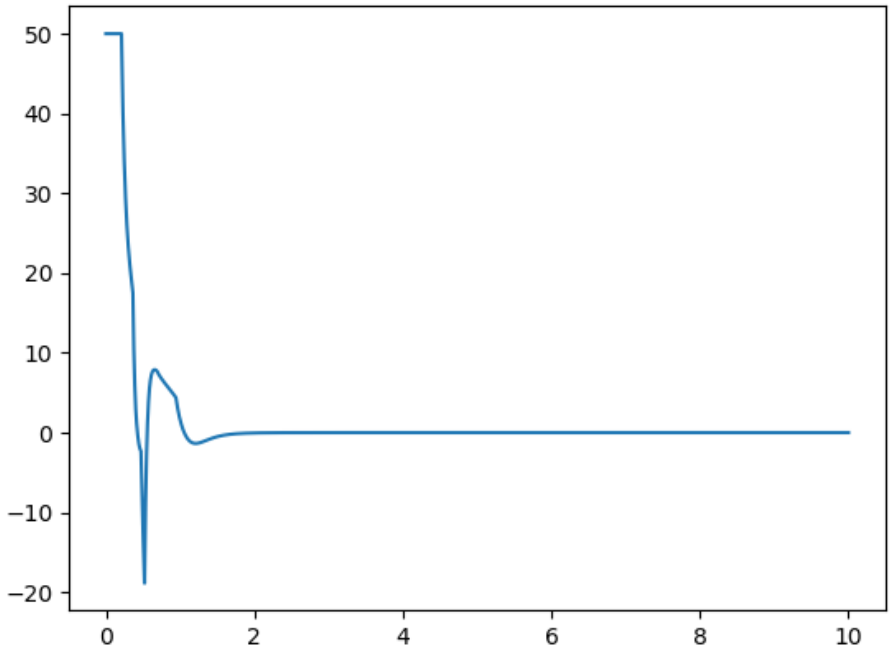

Figura 12.