Técnicas, entornos y aplicaciones de inteligencia artificial

Práctica 1 – Axel Guzman Godia

## Parte 1 : Resolver uno de los dos problemas propuestos mediante un mecanismo de razonamiento backward.

La primera parte de la práctica consiste en resolver uno de los dos problemas propuestos. En este caso he seleccionado el problema de los animales. Dicho problema consiste en construir un sistema capaz de descubrir en que animal estas pensando mediante una serie de preguntas.

#### 1.1 Estructura del programa

Para realizar esta parte se han modelado las reglas que se proponen en el enunciao. De este modo tenemos un total de 9 reglas que son suficientes para descubrir el animal daod el dominio de la aplicación ( 4 animales ). Las reglas modeladas son las siguientes:

(defrule mamifero1

(cubre pelo)

=>

(assert (tipo mamifero)))

(defrule mamifero2

(leche si)

=>

(assert (tipo mamifero)))

(defrule ave

(cubre plumas)

(reproduce oviparo)

=>

(assert (tipo ave)))

(defrule ave1

(pico si)

(reproduce oviparo)

=>

(assert (tipo ave)))

(defrule carnivoro

(tipo mamifero)

(garras si)

(comer carne)

=>

(assert (comer carnivoro)))

(defrule guepardo

(comer carnivoro)

(color pardo)

(piel manchas)

=>

(printout t "El animal es el guepardo")

(halt))

(defrule tigre

(comer carnivoro)

(color pardo)

(rayas negras)

=>

(printout t "El animal es el tigre")

(halt))

(defrule pinguino

(tipo ave)

(vuela mal)

(nada bien)

=>

(printout t "El animal es el pingüino")

(halt))

(defrule gaviota

(tipo ave)

(vuela muy\_bien)

=>

(printout t "El animal es la gaviota")

(halt))

Dichas reglas permiten al sistema deducir información sobre el animal. Sin embargo el sistema necesita conseguir información para poder hacer estas deducciones. A este efecto se han modelado las siguientes reglas que permiten al sistema preguntar al usuario la información necesaria:

(defrule preguntar-cubre

(need-cubre ?)

=>

(printout t "Como se cubre el animal? ")

(assert (cubre (read))))

(defrule preguntar-piel

(need-piel ?)

=>

(printout t "Como es la piel del animal? ")

(assert (piel (read))))

(defrule preguntar-leche

(need-leche ?)

=>

(printout t "Da leche el animal? ( si o no)")

(assert (leche (read))))

(defrule preguntar-reproduce

(need-reproduce ?)

=>

(printout t "Como se reproduce el animal? (mamifero,oviparo) ")

(assert (reproduce (read))))

(defrule preguntar-pico

(need-pico ?)

=>

(printout t "Tiene pico el animal? (si o no) ")

(assert (pico (read))))

(defrule preguntar-garras

(need-garras ?)

=>

(printout t "Tiene garras el animal? (si o no) ")

(assert (garras (read))))

(defrule preguntar-comer

(need-comer ?)

=>

(printout t "Que dieta sigue el animal? (carnivoro, omnivoro....) ")

(assert (comer (read))))

(defrule preguntar-color

(need-color ?)

=>

(printout t "De que color es el animal? ")

(assert (color (read))))

(defrule preguntar-rayas

(need-rayas ?)

=>

(printout t "Tiene rayas el animal? en caso afirmativo de que color son?")

(assert (rayas (read))))

(defrule preguntar-vuela

(need-vuela ?)

=>

(printout t "Vuela el animal? Como vuela? (mal, bien, muy bien...)")

(assert (vuela (read))))

(defrule preguntar-nada

(need-nada ?)

=>

(printout t "Sabe nadar el animal? Como nada? (bien,mal....)")

(assert (nada (read))))

Se puede observar que la estructura de las reglas es la misma . El sistema pregunta por una información i a continuación la “asserta”, con lo cual se activan las reglas que permiten deducir información. Una vez se deduce que animal es , el sistema se detiene.

#### 1.2 Conclusiones

La elaboración de este programa resulta bastante sencilla ya que se basa en una serie de preguntas i deducciones que se indican en lenguaje natural en el enunciado. La única complicación es tener en cuenta que se esta utilizando el proceso de backward-chainning, por lo que hay que añadir esta información en el inicio del programa mediante (do-backward-chaining ) . La realización de esta parte no ha supuesto un gran esfuerzo y he podido acabarla en las horas de laboratorio sin más compliación.

## Parte 2 : Modelar y resolver un problema de complejidad moderada como un SBC.

#### 2.1Introducción al problema escogido

Para desarrollar esta parte de la práctica he decidido modelar un sistema experto en turismo. Dicho sistema debe ser capaz de proporcionar un itinerario de actividades al usuario según las preferencias de este. Dicho problema se ha modelado de una manera simplificada dividiendo los días en dos mitades. Para cada mitad el sistema recomendará actividades, teniendo en cuenta además de las preferencias del usuario, las franjas horarias adecuadas para cada actividad (El sistema no recomendará ir a una discoteca por la mañana)

Este sistema experto en turismo se ha diseñado para Peñiscola, localidad de la que soy natural.

#### 2.2Estructura del sistema

Para representar el problema se hace uso de dos tipos de “templates “ diferentes. El primer template, “plan” sirve para modelar actividades. Contiene información sobre el nombre, la descripción, el tipo de turismo al que se asocia, la duración aproximada, el precio, el momento apropiado del día y si requiere de vehículo privado. Su implementación es la siguiente:

(deftemplate plan

(slot nom(type string))

(slot descripcio(type string))

(multislot tipo (type string))

(slot duracio (type integer))

(slot preu (type integer))

(multislot moment(type string))

(multislot vehicle(type string))

)

El conocimiento inicial del sistema se basa en una serie de hechos “plan” que modelan actividades que el sistema puede recomendar. Estas actividades están todas definidas en “deffacts”

El segundo template , más sencillo, sirve para representar los días. Su estructura es la siguiente:

(deftemplate dia

(slot numero(type integer))

(slot mati (type string))

(slot tarde(type string)))

#### 2.3 Reglas y funcionamento

Una vez están definidas las estructuras que usará el sistema es necesario definir como obtendrá el sistema la información del usuario y que hará con ella. Para obtener la información del usuario se han propuesto cinco preguntas que permiten conocer si el usuario ha visitado con anterioridad el pueblo, que tipo de turismo desea hacer, el presupuesto con el que cuenta, cuanto tiempo desea estar y si dispone de vehículo propio. Dichas reglas contienen declaraciones de prioridad para poder asegurar que se hacen en un orden concreto.

La implementación de dichas reglas es la siguiente:

(defrule pregunta0

(declare(salience 5))

=>

(printout t "Has estat alguna vegada a Peñiscola?")

(assert (visitat (read))))

(defrule pregunta1

(declare(salience 4))

=>

(printout t "Quin tipus de turisme vols fer? (Familiar, oci, cultural, natural,parella)")

(assert (tipo (read)))

)

(defrule pregunta2

(declare(salience 3))

=>

(printout t "Quants diners vols destinar a les teues vacançes? (en euros)")

(assert(presupost (read))))

(defrule pregunta3

(declare(salience 2))

=>

(printout t "Quant de temps estarás de vacançes? (Dies, SENSE INCLOURE el dia d'arribada i el dia de sortida)")

(bind ?i (read))

(while (> ?i 0)

(assert (dia(numero ?i) (mati "") (tarde "") ))

(-- ?i)

)

)

(defrule pregunta4

(declare(salience 1))

=>

(printout t "Disposes de vehicle privat?")

(assert (vehicle (readline) )))

Una vez obtenida la información básica, el sistema tiene la información necesaria para poder empezar a generar el itinerario. Esto se hace mediante el uso de la siguiente regla:

(defrule itinerari

(vehicle ?vehicle)

(tipo ?tipo)

?dia <- (dia (numero ?n)(mati "")(tarde ""))

?activitatMati <- (plan (nom ?n1) (descripcio ?descr1)(tipo $? ?tipo $?) (duracio ?t1) (preu ?p1)(moment $? "mati" $?)(vehicle $? ?vehicle $? ))

?activitatTarde <- (plan (nom ?n2) (descripcio ?descr2) (tipo $? ?tipo $?) (duracio ?t2)(preu ?p2)(moment $? "tarde" $?)(vehicle $? ?vehicle $? ))

(presupost ?pres&:(> ?pres (+ ?p1 ?p2))) ;Comprovar que no se sobrepassa el presupost

(test(<> ?n1 ?n2)) ;Cal asegurar-se de no repetir una activitat el mateix dia

;Comprovar que no se sobrepassa el presupost

=>

(bind ?cost (+ ?p1 ?p2))

(bind ?\*itinerari\* (str-cat ?\*itinerari\*

"------------------------------- Dia " ?n " matí: " ?n1 ". Temps aproximat: " ?t1 " minuts ---------------------------------------

" ?descr1 ))

(bind ?\*itinerari\* (str-cat ?\*itinerari\*

"------------------------------- Dia " ?n " tarde: " ?n2 ". Temps aproximat: " ?t2 " minuts ---------------------------------------

" ?descr2 ))

(assert (presupost (- ?pres (+ ?p1 ?p2))))

(bind ?\*cost\* (+ ?\*cost\* ?cost));Acumular el cost total

(retract ?activitatMati)

(retract ?activitatTarde)

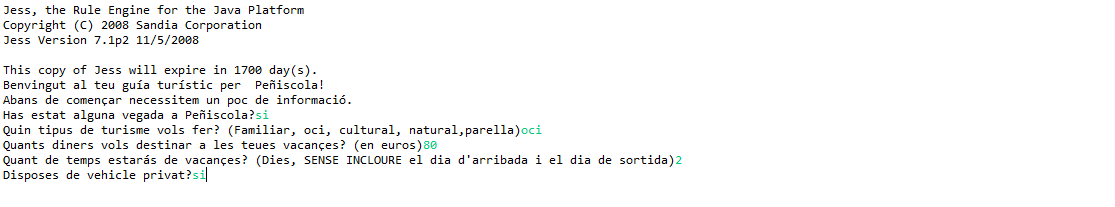
(retract ?dia)

)

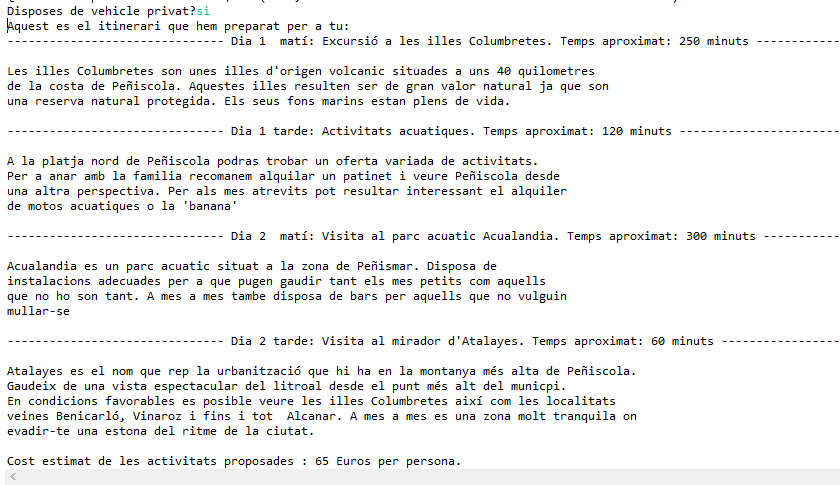
Esta regla se encarga de ir asignando actividades a los días de tal manera que cumplan los requisitos establecidos por el usuario. Así se comprueba que no se sobrepase le presupuesto o que el tipo de turismo coincida con la regla. Todas las actividades se van acumulando en un String que es lo que se imprime al final para devolver el resultado.

#### 2.4 Evaluación y pruebas

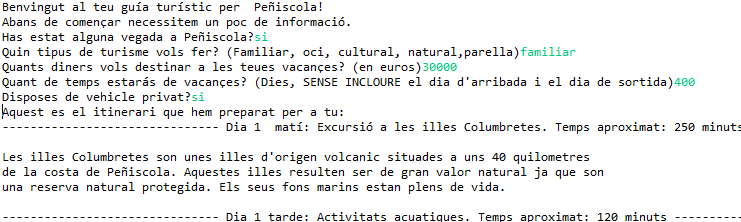
Vamos a observar el comportamiento del sistema con las siguientes entradas:



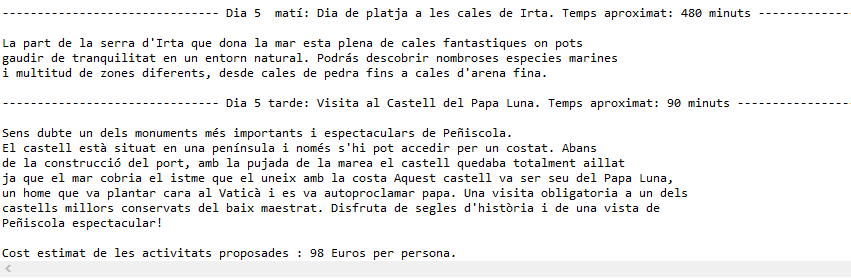
A lo que el sistema devuelve el siguiente resultado:



Se puede observar que no se ha rebasado el presupuesto que hemos indicado. Además hemos obtenido una serie de actividades coherentes con nuestras preferencias.

Por otro lado si indicamos al sistema un número inusualmente grande de días, el resultado es diferente: 

Hemos solicitado un itinerario para 400 días, pero el sistema no es capaz de preparar un itinerario tan largo y devuelve solo un itinerario para cinco dias. Esto se debe a que el sistema esta diseñado para no repetir planes y además la base de actividades con la que cuenta es reducida ya que no se trata de un programa profesional.



#### 2.5 Conclusiones y críticas

Tras haber realizado el proyecto inmediatamente se dejan ver algunas posibles mejoras sobre el sistema. Por ejemplo el sistema no tiene en cuenta que ciertas actividades pueden ir combinadas con otras, como por ejemplo actividades cansadas por la mañana y actividades de relax por la tarde. Otro aspecto susceptible de mejora es la inclusión de diversas actividades en una mitad del día.

En cuanto a las sensaciones que deja la realización de la práctica hay diversos aspectos destacables. Primero que nada cabe destacar que el entorno de depuración no funciona y puede llegar a ser muy frustrante programar sin una herramienta de depuración. Esto ha hecho que el tiempo de este proyecto se haya dilatado bastante llegando a casi 15 horas de trabajo extra fuera del laboratorio. Por otro lado uno de los aspectos que más positivamente valoro es la libertad que se deja para realizar la segunda parte, ya que una idea propia siempre suele motivar más.