Sistema experto CLIPS

Recomendador de ocio

Inteligencia Artificial

Josué Toledo Castro
María Nayra Rodríguez Pérez

INDICE:

1. Introducción

▪ Objetivo de la práctica

Se pretende desarrollar una actividad práctica basada en la utilización de CLIPS para diseñar un Sistema Experto basado en Reglas. El tipo y los objetivos del SE serán elegidos por los alumnos, constituyendo como algunos posibles ejemplos de sistemas expertos los siguientes: diagnóstico, recomendador, clasificación, control, alarma, etc.

▪ Requisitos mínimos

Los requisitos mínimos del Sistema Experto son los siguientes:

- Debe contener hechos ordenados y no ordenados.
- El Sistema debe realizar preguntas y explicaciones al usuario.
- Debe incluir alguna clase de objetos.
- Se debe hacer uso de atributos múltiples.
- Se han de utilizar variables en los casos que sea necesario.

2. CLIPS

Clips es un entorno completo para la construcción de sistemas expertos basados en reglas y/o objetos. Ofrece paradigmas heurísticos y procedimentales para representar el conocimiento.

En comparación con los lenguajes procedurales, la ejecución en estos puede tener lugar sin datos, es decir, las sentencias son suficientes para desencadenar la ejecución de un programa, mientras que en CLIPS se requieren de unos datos o hechos para provocar tal efecto.

▪ Reglas.

Se utilizan para la representación de heurísticos que especifican un conjunto de acciones a realizar para una situación dada. En este sentido, un SE dispondrá de un conjunto de reglas cuyo propósito principal será la resolución de problemas.

A modo de entender mejor el concepto y la funcionalidad de las mismas, éstas actúan de una manera similar a las sentencias IF-THEN de los lenguajes procedimentales como C, aunque suelen entenderse y concebirse de una manera más generalizada como sentencias SIEMPRE QUE-ENTONCES.

- **Hechos no ordenados.**

Los hechos no ordenados proporcionan al usuario la habilidad de abstraerse de la estructura del hecho, asignando a un nombre a cada campo del mismo. De esta forma, podemos acceder a los diferentes campos por su nombre.

Un hecho no ordenado es una secuencia de cero o más campos con nombre separado por espacios y delimitado por paréntesis.

- **Hechos ordenados.**

El campo inicial suele expresar una relación entre los campos siguientes. Los campos de un hecho ordenado pueden ser de cualquier tipo primitivo de datos, excepto el primero, que debe ser un símbolo.

No es necesario declararlos. No existe restricción alguna en el orden de los campos, pero para ser usados en una regla sus campos deben aparecer en el mismo orden que indique la regla. Es decir, los hechos ordenados “codifican” la información según la posición. Para acceder a esa información, el usuario debe saber la información que almacena el hecho y qué campo la contiene.

3. Descripción del Sistema Experto

Este sistema experto tiene por objetivo recomendar lugares de ocio en San Cristóbal de La Laguna a los usuarios interesados. Concretamente nos referimos a cafeterías, bares y restaurantes, tomando en consideración la posibilidad de solicitar comida a domicilio o reservar mesa en un restaurante.

En primer lugar, al usuario se le realizarán una serie de preguntas a partir de las cuales el sistema experto concretará qué establecimientos en La Laguna son más acordes a sus gustos y necesidades. En primer lugar, el interesado especificará si prefiere comer o beber. En el primer caso, el sistema le ofrecerá la posibilidad de distinguir entre optar a un servicio de entrega a domicilio o comer en el propio restaurante, dándole en este caso la opción de elegir un establecimiento con o sin terraza.

Además, se tendrá en cuenta el tipo de comida deseado por el cliente:

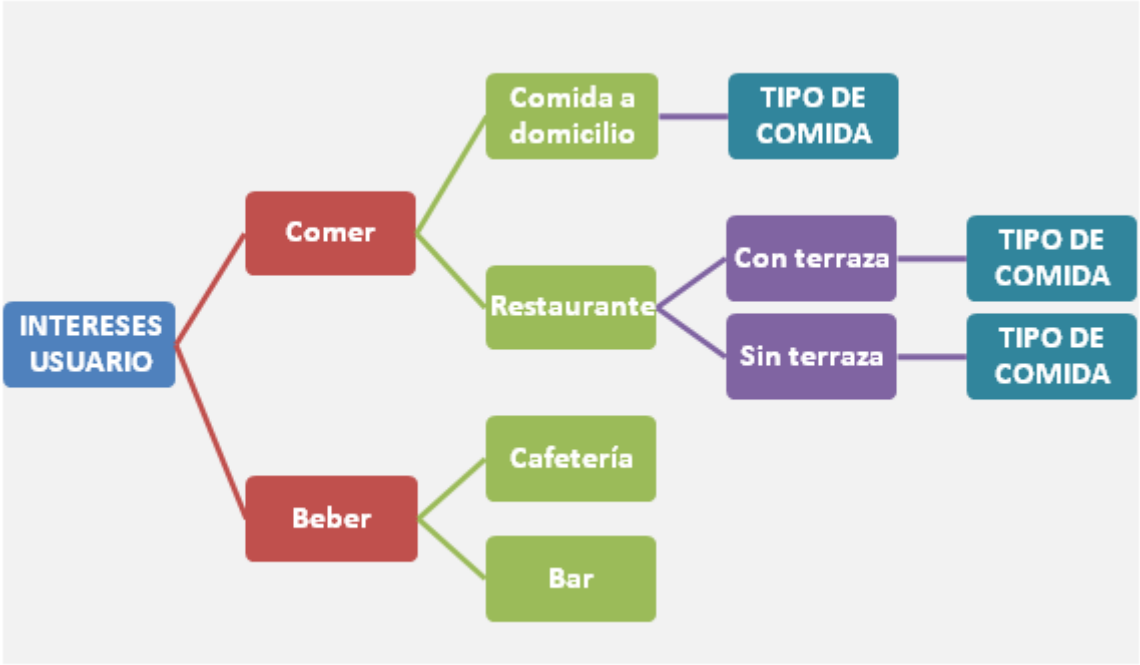
- Comida canaria.
- Comida turca.
- Comida italiana.
- Comida mexicana.

Por otro lado, en el caso de que usuario escoja beber, se le dará la oportunidad de elegir entre dos tipos de establecimientos: bar o cafetería.

Finalmente, se expondrá una lista de resultados atendiendo a las especificaciones introducidas por dicho usuario, incluyendo otros datos de interés como pueden ser los siguientes:

- Dirección del establecimiento.
- Puntuación de otros clientes.

A continuación puede observarse un esquema representativo de los aspectos tratados y la dirección de las distintas cuestiones planteadas al usuario por parte del SE siguiendo el planteamiento descrito:



4. Descripción del Código

▪ Definición de clase establecimiento.

Cada objeto de tipo “establecimiento” representará todo aquel lugar de ocio que será expuesto por parte del SE al usuario como consecuencia de un proceso de recopilación de información a través de preguntas encadenadas que irá respondiendo este último.

En primer lugar se procede a la definición de la clase:

```
(defclass establecimiento(is-a USER)(role concrete)
```

Seguidamente, se definen los distintos campos (“slot”) que se conciben o comparan con los atributos de una clase en un lenguaje procedimental como C. De este modo, puede observarse la definición de diferentes slot para determinar el nombre del establecimiento, la comida ofrecida por el mismo, la posibilidad de que en el establecimiento se pueda fumar o no y, de este modo, la posibilidad de que disponga de terraza, la posibilidad de que el establecimiento tenga servicio a de entrega a domicilio, un slot para indicar de qué tipo es el establecimiento (cafetería, restaurante o bar), y, finalmente, un campo para determinar la puntuación que algunos usuarios han otorgado al establecimiento y un slot de tipo string para definir la dirección del lugar:

1. Definición del campo nombre_establecimiento de tipo STRING.
```
(slot nombre_establecimiento
    (type STRING)
)
```

2. Definición del campo comida_establecimiento de tipo STRING
```
(slot comida_establecimiento
    (type STRING)
)
```

3. Definición del campo posibilidad_fumar, de tipo de dato SYMBOL, el cual permitirá los símbolos “si” y “no”.
```
(slot posibilidad_fumar
    (type SYMBOL)
    (allowed-symbols si no)
)
```

4. Definición del campo posibilidad_terraza de tipo de dato SYMBOL.
```
(slot posibilidad_terraza
    (type SYMBOL)
```

```
        (allowed-symbols si no)
    )
```

5. Definición del campo comida_adomicilio, de tipo de dato SYMBOL, el cual determinará si el establecimiento dispone de servicio de entrega de comida a domicilio.

```
    (slot comida_adomicilio
        (type SYMBOL)
        (allowed-symbols si no)
    )
```

6. Definición del campo tipo_establecimiento de tipo SYMBOL, el cual, a diferencia de los anteriores casos en los que se permitían como valores posibles “si” o “no”, en este caso los valores permitidos son “Bar”, “Cafeteria” o “Restaurante”, indicando así el tipo de establecimiento y los servicios así como productos ofrecidos por el mismo.

```
    (slot tipo_establecimiento
        (type SYMBOL)
        (allowed-symbols Bar Cafeteria Restaurante)
    )
```

7. Definición del slot puntuacion_establecimiento de tipo de dato FLOAT, que contendrá una puntuación decimal de cada establecimiento otorgada por los usuarios en distintos recursos webs.

```
    (slot puntuacion_establecimiento
        (type FLOAT)
    )
```

8. Definición del campo dirección_establecimiento de tipo de dato STRING, con información detallada acerca del número y nombre de la calle dónde se encuentra el establecimiento.

```
    (slot direccion_establecimiento
        (type STRING)
    )
) ;Paréntesis de cierre de la deffclass establecimiento
```

- **Definición de instancias en la base de datos**

Similarmente a los hechos iniciales, se han definido una serie de instancias o casos que se crearán cada vez que ejecuta el comando “reset”.

La sintaxis es la siguiente:

(definstances <definstances-name> [active] [<comment>] <instance-template>)*

A continuación, puede observarse como se ha definido el constructor de instancias BD_establecimientos y algunos ejemplos de establecimientos del municipio de San Cristóbal de la Laguna. Resulta preciso nombrar la estrecha relación entre los campos o slot definida o explicada previamente en el apartado anterior y los campos de cada una de las instancias, concediéndose del mismo modo que en los lenguajes procedurales como objetos o instancia de clase, en este caso de la clase “establecimiento”:

```
(definstances BD_establecimientos  ;Creacion de BD_establecimientos
        (Granero of establecimiento  ;Definición de la instancia para el bar El Granero
        (nombre_establecimiento "El Granero")
        (comida_establecimiento "-------")
        (tipo_establecimiento Bar)
        (posibilidad_fumar no)
        (posibilidad_terraza no)
        (puntuacion_establecimiento 8.0)
        (direccion_establecimiento "Calle Maria del Cristo Ossuna")

        (Tintin of establecimiento   ;Definición de la instancia para el Bar Tintin
        (nombre_establecimiento "El Rincon del Titin")
        (comida_establecimiento "-------")
        (tipo_establecimiento Bar)
        (posibilidad_fumar si)
        (posibilidad_terraza si)
        (puntuacion_establecimiento 7.6)
        (direccion_establecimiento "Plaza de la Concepcion,7")
        )
        (LaBourmet of establecimiento ;Definición de la instancia para La Bourmet
        (nombre_establecimiento "La Bourmet")
        (comida_establecimiento Americana)
        (tipo_establecimiento Restaurante)
        (posibilidad_fumar si)
        (posibilidad_terraza si)
        (puntuacion_establecimiento 8.7)
        (direccion_establecimiento "Calle San Agustin,42")
        )
        ;;;;;;; Resto de instancias definidas ;;;;;;
```

- ## Definición de reglas

a. Definición de la regla estado_inicial.

Esta primera regla se ejecutará inicialmente una vez que produzca la carga del sistema experto y ejecutado los comandos “reset” y “run”. La funcionalidad principal de esta regla no es otra que dar un mensaje de bienvenida al usuario y capturar el nombre del mismo una vez que lo introduce a través del teclado. Como puede observarse en la última línea, se establece el hecho *usuario es interesado* , hecho que constituye el antecedente de otra regla que se explicará seguidamente.

```
(defrule estado_inicial
(initial-fact)
=>
(printout t "-----------------SE Recomendacion de Lugares de Ocio-----------------"
crlf)
(printout t "Hola! ¿Como te llamas?: " crlf)
(bind ?nombre(read))
(printout t "Bienvenido " ?nombre crlf)
(assert(usuario es interesado))
)
```

b. Definición de una regla para determinar el interés del usuario.

Si se produce el hecho usuario es interesado, implica la salida en pantalla de la pregunta ¿Quiere comer o solamente tomar algo?. Una vez capturada la respuesta introducida por teclado del usuario, mediante la sentencia condicional if-else, el SE comprueba que el texto en ?respuesta se corresponda con Comer o Beber, lanzándose un mensaje en el caso de que la palabra introducida por el usuario no sea adecuada y provocándose la salida forzosa del sistema en tal caso: *(halt).*

```
(defrule interes_del_usuario
(usuario  es interesado)
=>
(printout t "¿Quiere comer o solamente tomar algo?(Comer|Beber)" crlf)
(bind ?respuesta(read))
(if(or (eq ?respuesta Comer)(eq ?respuesta comer))
then
        (assert(interes_usuario Comer))
else
        (if (or(eq ?respuesta Beber)(eq ?respuesta beber))
        then
```

```
            (assert(interes_usuario Beber))
        else
            (printout t "No ha introducido ninguna opción posible" crlf)
            (halt)
            )
        )
)
```

c. Definición de otras reglas para recopilar información acerca de los intereses del usuario.

Siguiendo el mismo esquema y la estructura que en la explicación anterior, podemos observar otras reglas con una configuración similar.
Se determina el interés del usuario o no en un servicio de comida a domicilio:

```
(defrule interes_comer
(interes_usuario Comer)
=>
(printout t "Usted ha seleccionado la opcion comer" crlf)
(printout t "¿Desea solicitar comida a domicilio?(S/N)" crlf)
(bind ?respuesta_domicilio(read))
(if (or (eq ?respuesta_domicilio S) (eq ?respuesta_domicilio s) (eq
?respuesta_domicilio si) (eq ?respuesta_domicilio SI))
 then
        (assert(comida_domicilio si))
else
        (assert(comida_domicilio no))
))
```

Se determina el interés del usuario en cafeterías o bares:

```
(defrule interes_beber
(interes_usuario Beber)
=>
(printout t "Usted ha seleccionado la opcion tomar algo" crlf)
(printout t "¿Que desea tomar?(Cafeteria|Bar)" crlf)
(bind ?respuesta(read))
(if(or (eq ?respuesta Cafeteria)(eq ?respuesta cafeteria))
 then
        (assert(tipo_establecimiento Cafeteria))
        (printout t "Usted ha seleccionado " ?respuesta crlf)
else
        (if(or (eq ?respuesta Bar) (eq ?respuesta bar))
        then
            (assert(tipo_establecimiento Bar))
```

```
                (printout t "Usted ha seleccionado " ?respuesta crlf)
        else
                (printout t "No ha introducido ninguna opcion posible..." crlf)
                (halt)
        )
))
```

Como otro ejemplo representativo de las reglas definidas podemos observar la determinación del tipo de comida deseada por el usuario siempre que éste haya manifestado previamente su interés por un servicio de entrega de comida a domicilio:

```
(defrule domicilio_si
(comida_domicilio si)
=>
(printout t "¿Que tipo de comida prefieres?(Italiana|China|Americana|Turca)"
crlf)
(bind ?respuesta(read))
(if (or (eq ?respuesta Italiana)(eq ?respuesta italiana))
 then
        (assert(comidad_cliente Italiana))
        (printout t "Usted ha escogido " ?respuesta crlf)
else
        (if (or (eq ?respuesta China)(eq ?respuesta china))
        then
                (assert(comidad_cliente China))
                (printout t "Usted ha escogido " ?respuesta crlf)
        else
                (if (or (eq ?respuesta Americana) (eq ?respuesta americana))
                then
                        (assert(comidad_cliente Americana))
                        (printout t "Usted ha escogido " ?respuesta crlf)
                else
                        (if(or (eq ?respuesta Turca)(eq ?respuesta turca))
                        then
                                (assert(comidad_cliente Turca))
                        else
                                (printout t "No esta disponible" crlf)
                                (halt)
                        )
                )
        )
)
```

Finalmente, como otra regla representativa de las configuradas en el SE, podemos observar el caso en el que el usuario no manifiesta interés en un servicio de entrega de comida a domicilio, cuestionándole el sistema seguidamente si desea terraza y el tipo de comida que le apetece:

```
(defrule domicilio_no
(comida_domicilio no)
=>
(printout t "¿Desea usted restaurante con terraza?(S/N)" crlf)
(bind ?respuesta(read))
(if (or (eq?respuesta S) (eq?respuesta Si) (eq?respuesta si) (eq?respuesta s))
then
        (assert(comidar_terraza si))
else
        (assert(comidar_terraza no))
)
(printout t "¿Qué comida prefieres?(Italiana|China|Americana|Canaria|Turca)")
(bind ?respuesta(read))
(if (or(eq ?respuesta Italiana)(eq ?respuesta italiana))
then
        (assert(comidar_cliente Italiana))
        (printout t "Usted ha escogido " ?respuesta crlf)
 else
        (if (or(eq ?respuesta China)(eq ?respuesta china))
        then
                (assert(comidar_cliente China))
                (printout t "Usted ha escogido " ?respuesta crlf)
        else
                (if (or(eq ?respuesta Americana)(eq ?respuesta americana))
                then
                        (assert(comidar_cliente Americana))
                        (printout t "Usted ha escogido " ?respuesta crlf)
                else
                        (if(or(eq ?respuesta Turca)(eq ?respuesta turca))
                        then
                                (assert(comidar_cliente Turca))
                                (printout t "Usted ha escogido " ?respuesta crlf)
                        else
                                (if(or(eq?respuesta Canaria)))
                                then
                                        (assert(comidar_cliente Canaria))
                                        (printout t "Ha escogido " ?respuesta)
                                else
                (printout t "La comida que desea no esta disponible" crlf)
                (halt)))))))
```

d. Definición de reglas para la proyección y exposición de los resultados.

Finalmente, baśandonos fundamentalmente en los intereses del usuario, se han configurado las reglas en base a su deseo de servicio a domicilio o no, a la posibilidad de terraza y de fumar, tipo de comida o interés en cafeterías o bares para ingestas más ligeras o bebidas simplemente.

Como ejemplo de regla para evidenciar como se ha efectuado el proceso de salida en pantalla de los resultados finales, puede observarse que se ejecuta en el caso de que el usuario manifieste un interés determinado, un tipo de comida particular y la posibilidad de un servicio de entrega de comida a domicilio:

```
(defrule resultados_domicilio
(interes_usuario ?interes_usuario)
(comida_domicilio si)
(comidad_cliente ?comidad_cliente)
=>
(printout t "--------------------------------" crlf)
(printout t "Le recomendamos:" crlf)
(printout t "--------------------------------" crlf)
```

A continuación, para todas las instancias declaradas o definidas inicialmente, se comprueba mediante el elemento *eq* la similitud del contenido de ?interes_usuario y ?comidad_cliente con aquellos establecimientos definidos en BD_establecimientos cuyo campo *comida_adomicilio* tome el valor *si:*

```
(do-for-all-instances ((?ins establecimiento))(and(eq
?ins:comida_establecimiento ?comidad_cliente)(eq ?ins:comida_adomicilio si))
```

Finalmente, proyectamos los resultados en la pantalla , es decir, todos los datos de aquellos establecimientos cuyas características se correspondan con los características de los usuarios:

```
  (printout t "Nombre del establecimiento: " ?ins:nombre_establecimiento crlf)
  (printout t "Comida del establecimiento: " ?ins:comida_establecimiento crlf)
  (printout t "Direccion: " ?ins:direccion_establecimiento crlf)
  (printout t "Puntuacion: " ?ins:puntuacion_establecimiento crlf)
  (printout t "Comida a domicilio: " ?ins:comida_adomicilio crlf)
  (printout t "Terraza: " ?ins:posibilidad_terraza crlf)
  (printout t "Fumar: " ?ins:posibilidad_fumar crlf)
  (printout t "--------------------------------" crlf)
)
```

Finalmente, la ejecución del programa acaba:

```
(halt)
) ;Paréntesis de cierre de la defrule
```

5. Duración y reparto del trabajo en el desarrollo del sistema experto.

El desarrollo del recomendador de ocio se ha realizado fundamentalmente la semana comprendida entre el 8 y el 15 de Diciembre, llevando a cabo los primeros días de la misma una dedicación principalmente orientada a la documentación y a la realización de pequeños ejemplos a modo de aprender los conceptos básicos requeridos.

Seguidamente, en el término de los días 11 al 13 de Diciembre se procedió a la creación de la estructura del sistema, priorizando en la definición de la clase “establecimiento”, las instancias de la base de datos (inclusión de los distintos establecimientos o lugares de ocio disponibles en el sistema experto y recomendados al usuario en función de sus intereses) y ,finalmente, las reglas principales. El día 14 de Diciembre se dedicó exclusivamente para la parte de proyección de los resultados al usuario y para la realización de pruebas a modo de comprobar fallos o errores y corregirlos.

Todo el trabajo, desde el tiempo dedicado a la documentación y a la realización de pequeños ejemplos para la familiarización con CLIPS y para el desarrollo del SE, se ha distribuido en distintas sesiones caracterizadas por un trabajo colectivo por parte de los dos miembros del grupo. En cuanto a las horas dedicadas a cada una de las fases de desarrollo puede observarse la siguiente lista:

1. Consulta de recursos disponibles, documentación y realización de ejemplos: 4 horas.
2. Definición de la clase, declaración de instancias en la base de datos y creación y configuración de las reglas principales: 5 horas.
3. Proyección de resultados en función de los intereses del usuario: 3 horas.
4. Realización de pruebas, detección de errores y corrección de los mismos: 2 horas.

6. Bibliografía.

En ese apartado del informe se expondrán los principales recursos y enlaces accedidos por parte del grupo para la asimilación de los conceptos necesarios para el desarrollo del Sistema Experto:

1. https://es.wikipedia.org/wiki/Sistema_experto
2. http://clipsrules.sourceforge.net/
3. https://es.wikipedia.org/wiki/CLIPS
4. http://www.uco.es/users/sventura/misc/TutorialCLIPS/Practica1.htm
5. https://www.cs.us.es/cursos/ia2-2004/temas/tema-01.pdf
6. http://www.cs.upc.edu/~bejar/ia/transpas/lab/clips-n.pdf
7. Apuntes en el campus virtual.