PRÁCTICA DE LABORATORIO 6 – CLIPS Semestre 2022-1

El presente documento pretende servir de guía para la práctica de laboratorio del curso Inteligencia Artificial, grupo 2, a realizarse el 04 de Julio del 2022. Se presentan algunos tópicos del lenguaje CLIPS con sus respectivos ejemplos, y se proponen algunos ejercicios.

1. OBJETIVOS DE LA PRÁCTICA

- Que el alumno pueda conocer la programación en CLIPS a través de los ejms presentados y especialmente mediante la resolución de los ejercicios propuestos.
- Se espera que pueda servir de ayuda para entender mejor el tema de sistemas expertos, el cual está incluido en el sílabo del presente curso.

2. PREPARATIVOS DE LA PRÁCTICA

Para instalar CLIPS, caso no estuviese instalado, se debe descargar su programa de instalación desde la página web oficial de CLIPS: http://clipsrules.sourceforge.net/ .
Luego efectuar los sgts pasos:

a) Descargar el Windows Installer Package "clips_windows_64_bit_executables_640" en carpeta C:\CLIPS.
b) Ejecutar el archivo y seguir las instrucciones que aparecen en pantalla (las rutinarias de una instalación en Windows).
c) Crear acceso directo en escritorio para el programa.

Los archivos CLIPS utilizan la extensión *.clp* y se pueden crear con cualquier editor de archivos de texto aunque se recomienda usar el editor del propio CLIPS.

Con la finalidad de observar el resultado de las acciones a efectuar, abra las ventanas **Agenda** y **Facts**. Para ello, elija **Facts (MAIN)** y **Agenda (MAIN)** de la opción **Window** que aparece en el menú principal.

3. INTRODUCCION A CLIPS

CLIPS es una herramienta computacional para la creación de sistemas expertos. CLIPS es el acrónimo de *C Language Integration System* (Sistema de Producción integrada en lenguaje C), y se diseñó utilizando el lenguaje de programación C en el Centro Espacial Johnson de la NASA a mediados de los 80. CLIPS es más que una herramienta software, es un lenguaje de programación que proporciona soporte para programación basada en reglas, orientada a objetos y por procedimientos.

La finalidad del sistema experto es realizar la tarea del experto humano reproduciendo el mismo modelo que éste utiliza para la resolución de problemas pertenecientes a su ámbito de saber. En todo sistema experto se pueden distinguir las siguiente partes:

- Base de Hechos: lista de datos ingresados (información base sobre el campo en el que se desarrollarán las actividades del experto) e inferidos.
- Base de Conocimiento: reglas (contempla las formas de utilizar la información citada en el punto anterior), funciones.
- Mecanismo o motor de Inferencias, controla la ejecución del SE (modela el proceso de razonamiento mediante el cual el experto humano realiza su tarea).

CLIPS permite crear sistemas expertos basados en reglas. Las reglas son una forma de representar los conocimientos del experto, constan normalmente de una parte condicional que, si se cumple, tiene como consecuencia la ejecución de una acción asociada. Cada vez que se aplica una de las reglas, los datos guardados en el sistema pueden variar, dando lugar a que se activen otras reglas que en la situación anterior no lo estaban. La resolución del problema planteado pasa por la aplicación sucesiva de reglas hasta que se llega a la condición de finalización. A este proceso se

le denomina conducido por los datos o **encadenamiento hacia adelante**, ya que las acciones sólo se producen cuando hay datos que las justifican.

La nomenclatura seguida en CLIPS para designar las principales partes del sistema experto son:

- Hechos (facts) son los que forman la base de información o datos del sistema.
- Reglas (rules) forman la base de conocimiento.

Formas de trabajo con CLIPS

- Con el prompt
- Con archivos de texto (archivos por lotes)

Notación

- Las palabras claves y las funciones propias de CLIPS van en minúsculas
- Distingue mayúsculas y minúsculas. Ej.: jUAN, Juan, JUAN

Caracteres delimitadores

;	Comentario
"	Inicio o final de un string
(	Inicio de una expresión
)	Final de una expresión
?	Inicio de una variable comodín
$	Inicio de una variable multicampo

Tipos de datos I

Numéricos

- Reales (FLOAT): Ej.: 1.5, -0.7, 3.5e-10
- Enteros (INTEGER): Ej.: 1, -1, +3, 65

Simbólicos

- Símbolos (SYMBOL): Cualquier secuencia de caracteres que no siga el formato de un número, excluyendo ciertos caracteres. Ej: casa, arbol, perro
- Cadenas (STRING): Cualquier secuencia de caracteres entre comillas. Ej: “Juan Molina”, “Casa del arbol”

Tipos de datos II

Tipos de datos propios de CLIPS, permiten almacenar direcciones de los hechos, direcciones externas, instancias de nombres e instancias de direcciones

- Fact-address Direcciones de hechos
  - Lista de hechos: hechos referenciados por su posición o por un nombre.
  - Se imprime como <fact XXX> (XXX: índice del hecho en memoria)
- External-address Direcciones externas
  - Dirección de una estructura de datos externa devuelta por una función escrita en C o Ada, que ha sido integrada con CLIPS.
  - Se imprime como <pointer XXX>

Funciones

Función: algoritmo identificado con un nombre que puede o no devolver valores a otras partes del programa

Tipos de funciones:

- Funciones internas: Definidas en CLIPS
- Funciones externas: Escritas en un lenguaje distinto a CLIPS

Se definen con el constructor: deffunction
Llamado de funciones:
Las funciones son llamadas en notación prefija entre paréntesis

Funciones matemáticas:
+, -, *, /, div, max, min, abs, float, integer

Ejemplos de llamadas a las funciones + y *:
(+ 34 5 1.3)
(* 2 3)
(+ 3 (* 5 2) 10)

Constructores
Constructor: Permite al programador añadir elementos a la base de hechos y a la base de conocimiento (funciones, reglas, hechos, clases).

Los más importantes son:
- deffunction: Para definir funciones
- defglobal: Para definir variables globales
- deftemplate: Para definir plantillas
- deffacts: Para definir hechos
- defrule: Para definir reglas
- defmodule: Para definir módulos

Hay también constructores para definir objetos.

Comandos
Comandos: Realizan acciones con efectos colaterales sin devolver valor alguno y son llamados entre paréntesis.

Algunos comandos:
- (exit)
- (assert ...)
- (watch ...)
- (run)
- (reset)
- (clear)

Entorno de desarrollo de CLIPS
La herramienta CLIPS incluye un entorno de desarrollo para programar sistemas de forma completa. En la ventana de comandos aparece el prompt:
`CLIPS>`
En este prompt se pueden introducir comandos directamente, en lo que se denomina “alto nivel”.

Para salir de CLIPS se usa el comando exit.
`CLIPS> (exit)`
También se puede usar la entrada del menú desplegable `File -> Exit`.

En CLIPS, todos los elementos deben ir encerrados entre paréntesis, teniendo una sintaxis parecida a la de LISP.

4. PROCEDIMIENTOS INTERACTUANDO CON EL SISTEMA CLIPS

1.- Qué responderá **CLIPS** a lo siguiente?
Tener en cuenta que la sintaxis para las operaciones aritméticas es similar a la usada en Lisp.

```
CLIPS> (* 10 (div 33 2 3 5) (+  4 (* 2 3 4) 6) (**  8 2))
```

Sugerencia: efectuar los cálculos parciales.

```
CLIPS>  (* (/ (max 3.0 4 2.0)(abs -2))(integer 4.0))

CLIPS>  (*( log 2.718281828459045)(sqrt  9))
```

2.- Ingrese los siguientes hechos. Qué responde **CLIPS?**
Un hecho (*fact*) es una pieza de información que se almacena en la llamada lista de hechos (*fact-list*). Los hechos constituyen los datos de partida del problema cuando éste se empieza a ejecutar y además recoge el estado del sistema en cualquier momento de su ejecución.

Sintaxis: `(<simbolo> <valor>)`

```
CLIPS>   (altitud es 2400 metros)

CLIPS>   (alumnos Melendez Carpio Salazar Reyes)
```

Ahora ingréselos con:

```
CLIPS>   (assert (altitud es 2400 metros))
CLIPS>   (assert (alumnos Melendez Carpio Salazar Reyes))
```

y describa lo que pasa:

¿Es posible ingresarlos con un solo ASSERT?

También agregue estos hechos:

```
CLIPS>   (assert (nombre "Alberto"))
CLIPS>   (assert (edad 21) )
CLIPS>   (assert  (a) (b) (c) (d))
```

A cada hecho en la lista se le asocia un identificador (*fact identifier*) de la forma *f-XXX*, donde el entero *XXX* es un índice de hecho (*fact index*), que indica la posición del hecho en la lista de hechos.

¿Cuál es la diferencia entre los cuatro primeros hechos cargados y los cuatro últimos?

3.- Como verifica si un hecho esta saliendo o entrando a la memoria (lista de hechos)?
Use los siguientes hechos:

```
(es-modelo  Millet-Figueroa)
(es-modelo  Gutty-Carrera)
```

4.- Remueva los hechos (a) (b) (c) (d) de la lista de hechos, use la primitiva RETRACT cuya sintaxis es: `(retract <indice-de-hecho>)`. Observe lo que pasa en la ventana Facts (MAIN).

Los hechos pueden ser ordenados y no ordenados.

- Hechos Ordenados:
  - Están formados por un símbolo seguido de cero o más campos separados por espacios, y todo ello delimitado por paréntesis.
  - Podemos interpretar el primer campo (un símbolo) como una relación y el resto de los campos como los términos relacionados.
  - En una relación el orden de los términos es importante.
  - Por ejemplo:
    - (abuelo "Juan" "Pedro") vs. (abuelo "Pedro" "Juan")
    - (color carro rojo) vs. (carro color rojo)
- Hechos No Ordenados o Estructurados:
  - Asignan un nombre a cada campo del hecho.
  - Cada campo contiene un nombre (el nombre del campo) y el valor que toma ese campo, todo ello entre paréntesis.
  - Ahora el orden de los campos no es importante.
  - Por ejemplo:
    - (persona (nombre "Juan")(apellido "Perez"))
    - (persona (apellido "Perez")(nombre "Juan"))

PLANTILLAS

Antes de crear hechos no ordenados, debe informarse a CLIPS del modelo o plantilla de hechos: el nombre de la relación y el nombre de los términos relacionados (*slots*).

Una plantilla se define mediante el constructor `deftemplate` el cual consiste en una lista de campos con nombres llamados slots. `deftemplate` permite el acceso por el nombre antes que mediante la especificación del orden de los campos.

Sintaxis o formato:

```
(deftemplate <nombre-plantilla>
  [<comentario-opcional>]
  <definición-de-slot1>
  <definición-de-slot2>
  …
  <definición-de-slotN>)
```

Una estructura `deftemplate` se define indicando el nombre de la plantilla, en el ejm `persona`, luego un comentario opcional entre comillas, en este caso "informacion vital", luego el nombre, el tipo y el valor por defecto de cada slot que contenga la estructura. El primer slot `nombre` es de tipo STRING; el segundo slot llamado `apellidos` es de tipo simbólico. En el slot `edad` el tipo de campo es NUMBER, el valor por defecto del campo es 25 y los valores pueden variar entre 1 y 100 años. Para el último slot idénticamente, se indica el nombre del campo `profesion`, el tipo de campo es SYMBOL, y los valores simbólicos que puede asumir son `futbolista`, `periodista`, y `artista`.

EJM.

```
(deftemplate persona
"informacion vital"
  (slot nombre
      (type STRING))
  (slot apellido
      (type SYMBOL))
  (slot edad
      (type NUMBER)
      (default 30)
      (range 1 100))
  (slot profesion
      (type SYMBOL)
      (allowed-symbols
      futbolista periodista
      artista))
)
```

Si no se definen valores explícitos, CLIPS inserta los **valores por defecto** del DEFTEMPLATE cuando se da un `(reset)`. Se pueden especificar los valores de campo ya sea explícitamente haciendo una lista de ellos o dando un rango de valores.
Las ranuras de un hecho que se especifican con la palabra clave `slot` en sus DEFTEMPLATE sólo pueden contener un valor (ranuras de un solo campo). En cambio cuando se especifican con la palabra clave `multislot` pueden contener 0 o más valores (ranuras con varios campos).

```
(deftemplate alumno "Una deftemplate de ejm"
      (multislot nombre)
      (slot edad)
      (slot genero))

CLIPS> (assert (alumno (nombre Mario Irribarren Mantilla)(edad 22)
(genero masculino))
```

Restricciones sobre los valores para los slots
`allowed-symbols` <palabras de caracteres sin comillas>
`allowed-strings` <cadenas de caracteres entre comillas>
`allowed-numbers` <números enteros, reales y exponenciales>
`allowed-integers` <números enteros>
`allowed-floats` <números reales>
`allowed-values` <cualquier tipo de valor>

5.- Defina la plantilla siguiente.

```
(deftemplate persona "Una deftemplate de ejm"
  (slot nombre)
  (slot edad)
  (slot ocupacion))
```

Ingrese ahora los siguientes hechos no ordenados y observe la respuesta del sistema:

```
  CLIPS> (persona (nombre "paolo guerrero") (edad 32) (ocupacion futbolista))
  CLIPS> (persona (nombre "juliana oxenford") (edad 34) (ocupacion
periodista))
```

Ahora ingréselos con el comando DEFFACTS asignando un nombre a la lista de hechos a cargar:

```
(deffacts lista-de-personas
      (persona (nombre "gianluca lapadula") (edad 27) (ocupacion futbolista))
      (persona (nombre "magaly medina") (edad 45) (ocupacion  periodista)))
```

Haga a continuación un RESET con:

```
(reset)
```

y describa lo que pasa:

6.- Genere ahora un grupo de datos referido a un determinado dominio (político, futbolístico, periodístico, etc.). Dicha data constituirá el conocimiento inicial (hechos que son ciertos antes de ejecutar un programa. Defina una o más plantillas ad hoc para cargar los hechos iniciales.

7.-Borre todo (hechos y reglas) mediante el comando CLEAR

```
CLIPS> (clear)
```

y luego ingrese las siguientes reglas:

```
(defrule nombreJuan
   (nombre Juan)
=>
   (printout t "Tu nombre de pila es Juan" crlf)
)

(defrule nombreyapellidos
   (nombre Juan)
   (apellido-1 Perez)
   (apellido-2 Lopez)
=>
   (printout t "Te llamas Juan Perez Lopez" crlf)
)
```

Puede hacerlo directamente en el Dialog Window o puede usar el editor incorporado en el CLIPS. En este caso haga lo sgte:

i. Abre una ventana de edición haciendo clic en el ícono de la hoja en blanco ubicado en el extremo superior izquierdo del entorno.
ii. Edite las reglas en la ventana de edición.
iii. Seleccione las reglas, haga clic derecho y finalmente haga clic en la opción **Load Selection** (si no hay errores se cargarán al interpretador).

Compruebe que han sido cargadas correctamente mediante:

```
CLIPS> (rules)
```

Ingrese ahora los siguientes hechos no ordenados:

```
(assert (nombre <ponga su nombre de pila>)
(assert (nombre Juan))
(assert (apellido-1 <ponga su primer apellido>))
(assert (apellido-1 Perez) (apellido-2 Lopez))
(assert (Introduza ahora otros hechos))
```

¿Se ha activado alguna regla?. ¿En qué orden se han activado?.

Comience el ciclo de ejecución. Para ello, escriba `(run)`. ¿Qué reglas se han ejecutado?. ¿En qué orden lo han hecho?. ¿Por qué cree que se han ejecutado en ese orden?

Reinicie el sistema con `(reset)`. ¿Qué hubiera pasado si reiniciamos con `(clear)` en lugar de con `(reset)`?.

8. Introduzca los hechos anteriores utilizando el comando `(deffacts)`. ¿Se activa alguna regla ahora?. ¿Cuántos hechos hay en la ventana de hechos?. ¿Por qué?

REGLAS EN CLIPS

Una regla de producción SI-ENTONCES consta de un antecedente -también denominado parte "SI" o lado izquierdo de la regla (LHS, left hand side) y de un consecuente -también denominado parte "Entonces" o lado derecho de la regla (RHS, right hand side).

El antecedente está formado por un conjunto de elementos condicionales (EC) que deben satisfacerse para que la regla sea aplicable. (Existe un *and* implícito entre todas las condiciones en la parte izquierda de la regla).

Sintaxis o formato:

```
(defrule <nombre_regla>
[<comentario-opcional>]
     (elemento-condición 1)
     (elemento-condición 2)
     ...
     (elemento-condición N)
=>
     (acción 1)
     (acción 2)
     ...
     (acción M))
```

La cabecera está formada por el nombre de la regla que debe ser único y, opcionalmente, por un comentario entre comillas (en el ejm: "Prueba de comentario"). Luego comienza el LHS de la regla que está formado por los elementos declarados antes del símbolo "=>", el cual es llamado flecha. Esta representa el comienzo de del RHS de la regla que está formada por los elementos que son declarados justamente después del símbolo: "=>". Cada uno de los patrones y **acciones** de la regla y ésta misma, deben estar encerrados entre paréntesis. En el ejemplo caso la acción es imprimir el string `"Te llamas Juan Perez Lopez"`.

EJM.

```
(defrule nombreJuan
  "Prueba de comentario"
   (nombre Juan)
=>
   (printout t "Tu nombre
de pila es Juan" crlf)
)

(defrule nombreyapellidos
"Prueba de comentario"
   (nombre Juan)
   (apellido-1 Perez)
   (apellido-2 Lopez)
=>
   (printout t "Te llamas
Juan Perez Lopez" crlf)
)
```

Recuerde para comenzar el ciclo de ejecución se debe escribir `(run)` y que previamente se deben haber cargado en el interpretador la Base de Reglas y los hechos (vía los `deftemplate` y `deffacts` considerados). Además se debe ejecutar `(reset)` para, entre otras cosas, añadir al sistema el conocimiento contenido en los `deffacts`.

Cargar en el interpretador los constructores

i. Seleccionar en la ventana de edición los constructores a cargar (los `deftemplate`, `deffacts` y `defrules` deseados).
ii. Hacer clic derecho sobre la región seleccionada, luego ejecutar `Load Selection (Ctrl+K`. Los comandos que no sean para adicionar constructs no son tenidos en cuenta por CLIPS.

RAMP