Universidad Complutense de Madrid.
Facultad de Ingeniería Informática
Inteligencia Artificial 1.

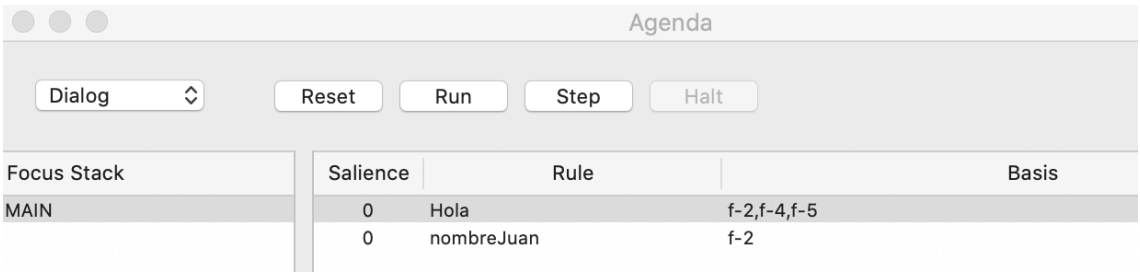
Práctica 4.
Sistemas basados en reglas.

Grupo 5:
Frederick Ernesto Borges Noronha
Víctor Manuel Cavero Gracia

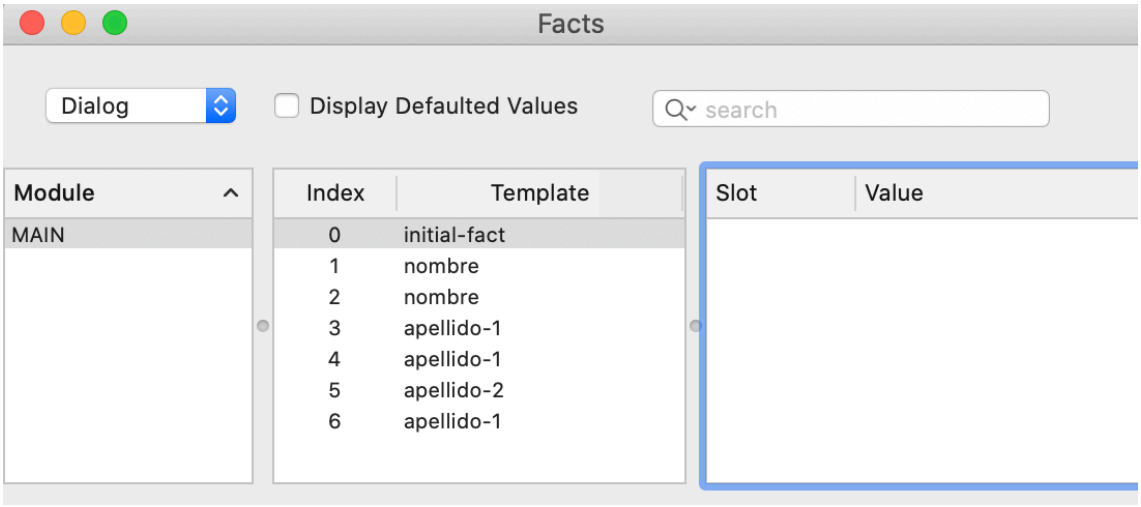
APARTADO 1

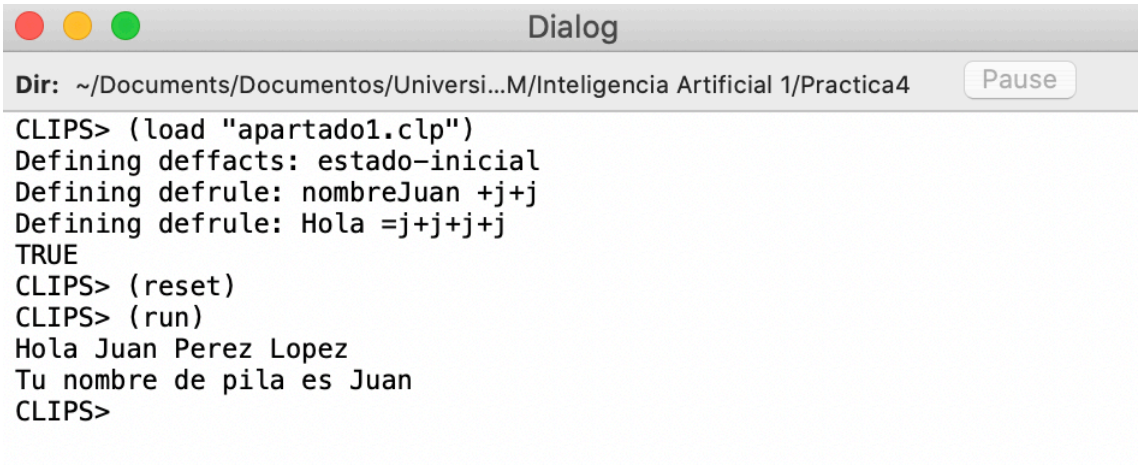
- **¿Se ha activado alguna regla? ¿En qué orden se han activado?**

Si, se han activados ambas reglas, y en cuanto el orden de activación se activó primero la regla nombreJuan y luego la regla Hola.



- **Para comenzar la ejecución hay que escribir (run) o usar la opción run del menú ¿Qué reglas se han ejecutado? ¿En qué orden lo han hecho? ¿Por qué crees que se han ejecutado en ese orden?**





Se han ejecutado ambas reglas en orden LIFO, ya que así está configurado el entorno

- **Reiniciar el sistema con (reset). ¿Qué hubiera pasado si reiniciamos con (clear) en lugar de con (reset)?**

Al reiniciar el sistema con reset se reestablece el sistema volviendo a cargar los mismo he los hechos y reglas, sin embargo, si escribimos clear se borra todo y tendremos un entorno completamente limpio.

APARTADO 2

Añade las reglas que añaden todos los hechos (primos p1 p2) que indica que p1(h o m) es primo hermano de p2 (h o m).

```
(defrule primosHermanos1
    (dd ?padre1 ?madre1 ?primo1 ?)
    (dd ?padre2 ?madre2 ?primo2 ?)
    (hermano ?padre1 ?padre2)
    =>
    (assert (primo ?primo1 ?primo2))
)

(defrule primosHermanos2
    (dd ?padre1 ?madre   1 ?primo1 ?)
    (dd ?padre2 ?madre2 ?primo2 ?)
    (hermano ?padre1 ?madre2)
    =>
    (assert (primo ?primo1 ?primo2))
)

(defrule primosHermanos3
    (dd ?padre1 ?madre1 ?primo1 ?)
    (dd ?padre2 ?madre2 ?primo2 ?)
    (hermano ?madre1 ?madre2)
    =>
    (assert (primo ?primo1 ?primo2))
)

(defrule primosHermanos4
    (dd ?padre1 ?madre1 ?primo1 ?)
    (dd ?padre2 ?madre2 ?primo2 ?)
    (hermano ?madre1 ?padre2)
    =>
    (assert (primo ?primo1 ?primo2))
)
```

APARTADO 3

En el archivo dado ej6.clp se proporciona un sistema en Clips que da soporte a un servicio de búsqueda de pareja. El sistema dispondrá de datos iniciales de distintas personas que estarán establecidos como hechos. Por ejemplo, el nombre, el sexo, la edad y el tipo de pareja que busca (hombre o mujer). Además, se dispone de datos sobre su número de amigos en Facebook (si no tiene cuenta entonces valor 0) y sobre sus gustos (música, lectura, cine, teatro). El enunciado completo lo tienes en

la hoja de ejercicios. Ejecuta el sistema, indica cuál es el fallo y corrígelo, vuelve a hacer las pruebas necesarias y comenta los resultados.

Partiendo del archivo **ej6.clp** nos damos cuenta que existen varios fallos en las reglas *extrovertido*, *introvertido* y *noClasificable, e*n ellas ocurría que al modificarse la propia persona, cuando cumplía las condiciones de amigos en Facebook necesarias, provocaba que se volviese a llamar a la misma regla para asignarle ese igual valor, quedando así un bucle infinito. Para solucionar esto, simplemente comprobamos que el tipo de carácter de la persona no sea ya el nuevo que le vamos a asignar. Código:

```
(defrule extrovertido
?p <- (persona (numAmigosFacebook ?num) (caracter ?c))
(test (>= ?num 50))
(test (neq ?c extrovertido)) ;Añadido
=>
(modify ?p (caracter extrovertido))
)

(defrule introvertido
?p <- (persona (numAmigosFacebook ?num) (caracter ?c)) ;Añadido
(test (<= ?num 50))
(test (> ?num 0))
(test (neq ?c introvertido)) ;Añadido
=>
(modify ?p (caracter introvertido))
)

(defrule noClasificable
?p <- (persona (numAmigosFacebook ?num) (caracter ?c)) ;Añadido
(test (eq ?num 0))
(test (neq ?c noDefinido)) ;Añadido
=>
(modify ?p (caracter noDefinido))
)
```

Además existía otro fallo, fue necesario añadir en la regla *afines* que comprobase que no existieran ya dos personas afines y para ello había cerciorarse también del simétrico.

```
(defrule afines
(persona(nombre ?nom1)(gustos ?gustos1))
(persona(nombre ?nom2)(gustos ?gustos2))
(test (neq ?nom1 ?nom2))
(test (> (calculaÀfinidad ?gustos1 ?gustos2) 0.5))
(not (afines ?nom2 ?nom1))
(not (afines ?nom1 ?nom2)) ;Añadido
=>
(assert (afines ?nom1 ?nom2))
)
```

Una vez corregidos los fallos del sistema podemos observar como la salida, dados unos hechos específicos, es la esperada. Hechos:

```
(deffacts Personas
        (persona (nombre Pepe) (sexo H) (edad 26) (tipoPareja M)(numÀmigosFacebook 81)(gustos teatro))
        (persona (nombre Pepa) (sexo M) (edad 25) (tipoPareja H)(numÀmigosFacebook 51)(gustos teatro))
        (persona (nombre Luis) (sexo H) (edad 35) (tipoPareja M)(numÀmigosFacebook 0)(gustos teatro))
)
```

Tras ejecutar las reglas sobre los hechos obtenemos los siguientes hechos:

```
CLIPS> (load ej6.clp)
Defining deftemplate: persona
Defining deffacts: Personas
Defining defrule: extrovertido +j+j
Defining defrule: introvertido +j+j
Defining defrule: noClasificable +j+j
Defining deffunction: calculaÀfinidad
Defining defrule: afines +j+j+j+j+j
Defining defrule: compatibles +j+j+j+j
Defining defrule: cital +j+j+j+j
TRUE
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0     (initial-fact)
f-4     (persona (nombre Luis) (sexo H) (edad 35) (tipoPareja M) (numÀmigosFacebook 0) (gustos teatro) (caracter noDefinido))
f-5     (afines Luis Pepa)
f-6     (afines Luis Pepe)
f-7     (persona (nombre Pepa) (sexo M) (edad 25) (tipoPareja H) (numÀmigosFacebook 51) (gustos teatro) (caracter extrovertido))
f-8     (afines Pepa Pepe)
f-9     (persona (nombre Pepe) (sexo H) (edad 26) (tipoPareja M) (numÀmigosFacebook 81) (gustos teatro) (caracter extrovertido))
f-10    (compatibles Pepa Pepe)
f-11    (cita Pepa Pepe)
For a total of 9 facts.
CLIPS> _
```

Estos hechos son los esperados y podemos ver como las únicas dos personas *compatibles* y que tienen *cita* son Pepa y Pepe, ya que Luis y Pepe tienen *caracter* distintos; en el caso de Luis y Pepa tampoco tienen *cita* o son *compatibles* ya que no encajan por su *tipoPareja*. Además se cumple que todos entre sí son afines.

APARTADO 4

Dado el archivo cocina.clp se trata de analizar y comprender el funcionamiento del Sistema Experto denominado STOVE, desarrollado

por Thad Fiebich, de la Universidad Johannes Kepler (Linz, Austria). Se trata de un especialista en reparación de cocinas, tanto de gas como eléctricas, cuyo funcionamiento se basa en un sistema de preguntas alternativas (generalmente SI/NO), mediante el cual el programa tratará de determinar el problema de la cocina y su posible solución. Realiza un análisis del sistema y de su funcionamiento y describe el árbol de decisión asociado.

El funcionamiento del sistema es sencillo, comienza con una regla *iniciar* que se ejecuta con el *initial-fact f-0* que es el que siempre se crea por defecto y en base a este se insertan dos nuevos hechos, el de *empezar* y que *necesita profesional no*.

```
;*********************************
;regla 1:
;       Inicializa el programa
;*********************************

(defrule iniciar
 (declare (salience 9980))
 ?x <- (initial-fact)
=>
 (retract ?x)
 (assert (necesita profesional no))
 (assert (empezar))
)
```

Los hechos *parar* y que *necesita profesional si* detienen la ejecución del sistema.

```
;*********************************************************
;regla 2:
;       Usada para finalizar el programa y llamar a
;       la regla inicial despues de resetear el programa
;
;*********************************************************

(defrule fin1
 (declare (salience 9200))
 ?w <- (parar)
 =>
 (retract ?w)
 (reset)
 (halt)
)
```

```
;*****************************************************************
;regla 10:
;       dice al usuario que deje a un profesional el problema
;       para evitar peligros
;
;*****************************************************************

(defrule pida-ayuda
 (declare (salience 9999))
 (necesita profesional si)
=>
 (printout t crlf crlf
        "Este problema puede ser serio y deberia dejarse a un " crlf
        "profesional. Seguir intentando reparar la cocina podria ser" crlf
        "peligroso." crlf crlf)
 (assert (parar))
)
```

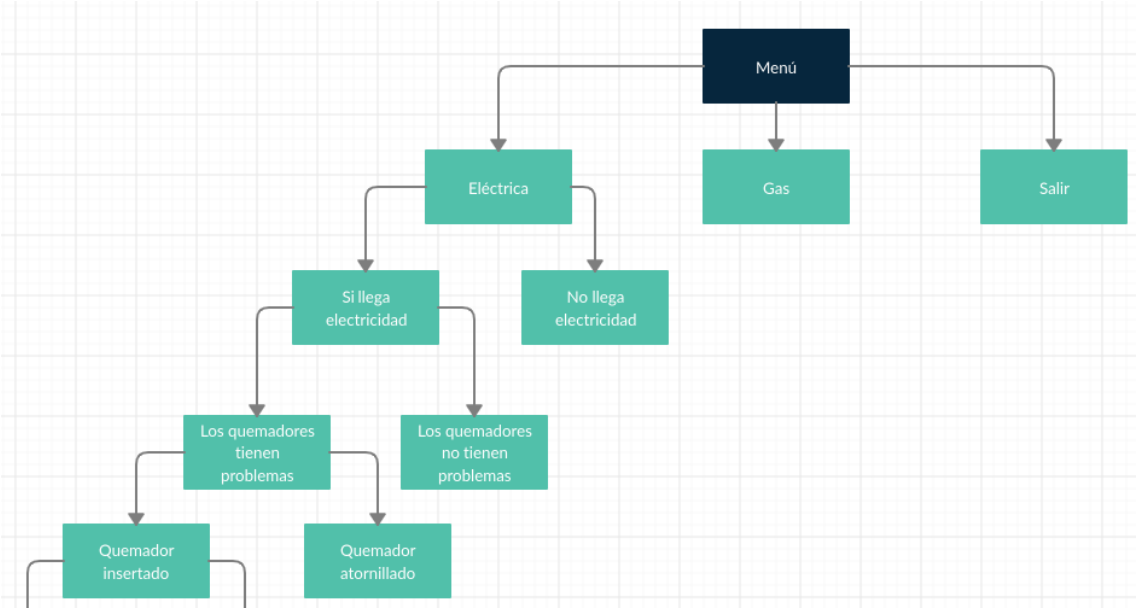
El hecho *empezar* desemboca la regla *inicio* que es el menú principal.

```
;*****************************************************************
;regla 3:
;       Da el mensaje de apertura y pregunta que tipo de cocina tiene
;
;*****************************************************************

(defrule inicio
 (declare (salience -9000))
 ?x <- (empezar)
=>
 (printout t crlf crlf crlf
        "El programa esta diseñado para ayudarlo a reparar una cocina electrica" crlf
        "o de gas.  El programa le indicara paso a paso los procedimientos" crlf
        " que ha de realizar para reparar la cocina." crlf crlf
        "Tiene usted una" crlf
      "  1) Cocina electrica" crlf
        "  2) Cocina de gas" crlf
      "  3) o salir del programa" crlf
        "Elija 1 - 3 -> ")
 (retract ?x)
 (assert (tipo cocina =(read)))
)
```

De esta manera, **mediante hecho insertados por las reglas y las propias reglas que se ejecutan por los cambios en los hechos**, se va desarrollando el árbol de decisiones.

Este sistema de hechos y reglas crea algo como esto, en él sólo se han expandido una de sus ramas debido a la gran longitud del mismo.



Continuación de la rama quemador insertado