Representación del conocimiento
Tarea 2

Arturo Márquez Flores,
David Martínez Galicia,
José Alberto López López

Universidad Veracruzana
Centro de Investigación en Inteligencia Artificial
Sebastian Camacho 5, Xalapa, Ver., México 91000

Resumen El presente documento pretende exponer de manera simple el concepto, funcionamiento y aplicaciones del software CLIPS, un shell empleado para modelar la experticia y el conocimiento humano. Con el fin de implementar las técnicas vistas dentro del curso "Representación del conocimiento" en la Maestría en Inteligencia Artificial (MIA) de la Universidad Veracruzana, las secciones posteriores introducen la sintaxis y el funcionamiento de CLIPS, un caso práctico derivado de una base de conocimiento médica y finalmente la implementación de las técnicas.

Keywords: Sistema experto · Representación del conocimiento · CLIPS

Un sistema experto es un programa que se comporta como un experto humano de algún dominio específico, usando conocimiento referente al dominio mismo. Para simplificar su funcionamiento, es conveniente dividir a un sistema experto en tres módulos: la base de conocimiento que incluye los hechos y reglas que modelan el dominio; la máquina de inferencia que contiene los procedimientos para emplear la información de la base y la interfaz de usuario que funge como un puente de comunicación entre el usuario y el sistema. La unión de estos dos últimos módulos se conoce como shell, para desarrollar las técnicas de interes se utilizarán las herramientas de CLIPS.

1. CLIPS

Desarrollado en el Centro Espacial Johnson de la NASA desde 1985 a 1996, CLIPS es un lenguaje de programación basado en reglas, útil para crear sistemas expertos y otros programas donde una solución heurística es más fácil de implementar y mantener que una solución algorítmica. De esta forma facilita la implementación de software que modela el conocimiento y la experiencia humana.

El nombre de CLIPS es un acrónimo derivado de C Language Integrated Production System que resalta una de sus características más importantes, la integración con otros lenguajes de programación como C y Java. Esto permite

a CLIPS definir prodecimientos que pueden ser ocupados desde un lenguaje procedural y funciones externas que llaman métodos de lenguajes procedurales.

Entre sus aplicaciones más relevantes se encuentran: un sistema de entrenamiento inteligente para controladores de vuelo de transbordadores espaciales (IAAI-89), HUB SIAASHING: un sistema basado en el conocimiento para la reducción temporal del horario de las aerolíneas (IAAI-92), un sistema experto de procesamiento de seguridad del personal de la NASA (IAAI-96), un sistema experto para el reconocimiento de acciones faciales y su intensidad (IAAI-00) y un sistema híbrido basado en el conocimiento para la optimización multiobjetivo de las operaciones de sistemas de distribución de energía (IAAI-05).

1.1. CLIPS como shell

Aunque CLIPS es considerado una herramienta para el desarrollo de sistemas expertos por sus características como un editor integrado y herramientas de debugging, generalmente es más conocido por sus características de shell el cual realiza inferencias y razona. Especificamente el shell de CLIPS provee los elementos básicos de un sistema experto:

- Una memoria global para almacenar datos (**fact-list**, **instance-list**).
- Una base de reglas (**knowledge-base**, **rule-base**).
- Un mecanismo para el control de la ejecución de reglas (**inference engine**).

1.2. Representación del conocimiento

CLIPS ofrece tres formas de representación, las cuales pueden ser ocupadas dentro del mismo o en conjunción con lenguajes procedurales.

- Hechos y reglas, usados principalmente en el conocimiento heurístico basado en la experiencia.
- Funciones, destinadas principalmente para el conocimiento procedural.
- Programación orientada a objetos, empleada para conocimiento procedural que acepte las funciones de clases, manejadores de mensajes, abstracción, encapsulación, herencia y polimorfismo.

Un programa escrito en CLIPS consiste de hechos, reglas y objetos. El mecanismo de inferencia decide qué regla debe ser ejecutada y cuando. También, es importante resaltar que un sistema experto basado en reglas de CLIPS es un programa que es manejado por los datos, es decir, los hechos, reglas y objetos estimulan la ejecución via el mecanismo de inferencia.

1.3. Ejecutando CLIPS

Todos los comandos en CLIPS comienzan y terminan en paréntesis:(comando). Esta sintaxis significa que para poder ejecutar el comando *comando*, deberá de introducirse (comando) tal y como se muestra. Sin embargo, cada comando presenta una sintaxis propia y una serie de opciones. Para poder utilizar una notación general para todos ellos se utilizará el siguiente convenio.

- Los corchetes, '[]' indicarán que lo que se encuentra en su interior es opcional.
- Lo que se encuentre entre los símbolos <y >, indica que debe de sustituirse necesariamente, incluidos los símbolos <y >, por algún valor del tipo especificado.
- El símbolo * se asocia a un tipo, e indica que la descripción puede reemplazarse por cero o más ocurrencias del tipo especificado. En general se presenta de la forma <tipo>*.
- El símbolo + se asocia a un tipo, e indica que la descripción puede reemplazarse por uno o más ocurrencias del tipo especificado. En general se presenta de la forma <tipo>+. Es equivalente a <tipo><tipo>*.
- La barra vertical | indica que debe hacerse una elección entre los elementos que se encuentran separador por la barra. En general se presenta de la forma opcion-1 | opcion-2 | ... | opción-n.
- El símbolo ::= se utiliza para definir los términos que aparecen en una expresión. En general presenta la forma <Termino-a-definir>::= <Definicion-del-termino>.

1.4. Tipos de datos primitivos

CLIPS provee ochos tipos de datos primitivos para la representación de la información que pueden ser observados en la Figura 1.

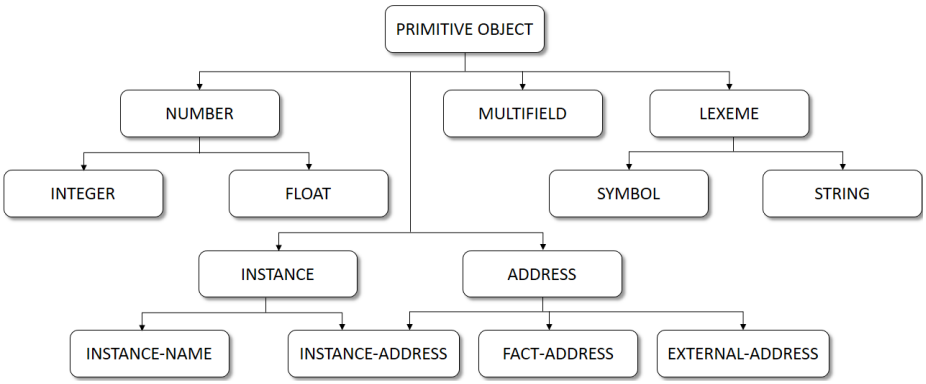

Figura 1. Diagrama jerarquico del tipo de objetos.

- Un **number** (número) consiste sólo de digitos del 0 al 9, un punto decimal (.), un signo (+ o -) y, opcionalmente, un símbolo (e) para la notación exponencial con correspondiente signo. Cualquier número que consiste en un signo (opcional) seguido de sólo digitos se almacena como un **integer**, representado como un *long integer de C*, en cambio cualquier otro número es un **float** , representado como *double-presicion float de C*.

- Un **symbol** (símbolo) es cualquier secuecia de caracteres que comienza con cualquier símbolo ASCII imprimible seguido por 0 o más símbolos ASCII imprimibles. Cuando un delimitador es encontrado, el símbolo finaliza. Los delimitadores se componen de cualquier símbolo ASCII no imprimible, espacios, tabuladores, retornos de carro, linea nueva, doble comilla, parantesis, ampersand, barra vertical, menor que, entre otros.
- Un **string** (cadena) es un conjunto de caracteres que comienza con una doble comilla (") y se sigue de cero o más cracteres imprimibles. Una cadena termina con doble comilla. Las doble comillas pueden estar dentro de una cadena si antes le precede un backslash (\), en cambio si se quiere un backslash dentro de la cadena es necesario repetir dos veces el símbolo.
- Un **external-address** es una dirección de una estructura de datos externa que regresa una función de cierto lenguaje procedural que ha sido integrada a CLISP. La representación impresa de una dirección externa dentro de CLIPS es *<Pointer-XXXXX>*.
- Un **fact-address** es una dirección de un hecho, que dentro de CLIPS es impresa como *<Fact-XXX>*. Las sección 1.7 da más detalles sobre los hechos.
- Un **instace** (instancia) es un objeto o ejemplo especifico de una clase. Los objetos en CLIPS spm definidos como floats, integers, símbolos, strings, valores multicampos, direcciones externas, direcciones de hechos o instancia de una clase definida por el usuario. Un **instance-name** es un nombre de la instancia que se forma encerrando un símbolo dentro de corchetes y finalmente un **instance-address** es una dirección de una instacia con el formato *<Instance-XXX>*.

1.5. Funciones

Una función en CLIPS es una pieza de código ejecutable identificada por un nombre específico que regresa un valor o realiza una acción. Existen distintos tipos de funciones, como las definidas por el sistema que han sido codificadas internamente por el ambiente de CLIPS y las definidas por el sistema que han sido codificadas externamente del ambiente de CLIPS.

El constructor **deffunction** permite al usuario definir nuevas funciones directamente en el ambiente CLIPS usando su sintaxis. Los constructores **defgeneric** y **defmethod** permiten al usuario definir funciones genéricas. Estas funciones hace posible la ejecución de distitas piezas de código dependiendo de los argumentos brindados.

Un punto importante a resaltar, es la llamada de funciones en CLIPS, debido a que se emplea una notación prefija, es decir, los argumentos de una función siempre aparecen después del nombre y la llamada debe comenzar con un parantesis izquierdo y terminar con un parentesis derecho.

1.6. Constructores

Varios constructores son definidos en CLIPS: defmodule, defrule, defacts, deftemplate, defglobal, deffclass, definstances, defmessage-handler, defgeneric y

defmethod. Todos estos constructores son encerrados en parentesis. Definir un constructores expliciamentes altera el ambiente de CLIPS agregando un objeto a la base de conocimiento de CLIPS.

1.7. Hechos

Los hechos son una de las formas básicas de alto nivel para representar información en un sistema CLIPS. Cada hecho representa una pieza de información que ha sido agregada en la lista de hechos, llamada **fact-list**. Los hechos son la unidad de datos fundamental usada por la reglas.

Comando	Descripción
assert	Agrega un hecho a fact-list.
retract	Remueve un hecho de fact-list. Es necesario definir el hecho a remover.
modify	Modifica un hecho de fact-list. Es necesario definir el hecho a modificar.
duplicate	Duplica un hecho de fact-list. Es necesario definir el hecho a duplicar.
reset	Reinicia los índices y agrega los hechos definidos con deffacts.
clear	Reinicia los índices de los hechos y elimina el conocimiento declarado.

Cuadro 1. Comandos de CLIPS para modificar fact-list.

El Cuadro 1 muestra los comando que a través de la interacción específica del usurario o de la ejecución de un programa CLIPS modifican la lista de hechos. Un identificador de hecho es una notación abreviada para mostrar un hecho. Consiste en el carácter "f", seguido de un guión y del índice de hechos del hecho.

Un hecho se almacena en uno de dos formatos: ordenado o no ordenado.

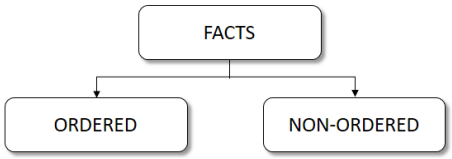

Figura 2. Tipo de hechos.

Los **datos ordenados** consisten en un símbolo seguido de una secuencia de cero o más campos separados por espacios y delimitados por un paréntesis de apertura a la izquierda y un paréntesis de cierre a la derecha. El primer campo de un hecho ordenado especifica una relación"que se aplica a los campos restantes en el hecho ordenado.

Los **hechos no ordenados** (o deftemplate) proporcionan al usuario la capacidad de abstraer la estructura de un hecho mediante la asignación de nombres a cada campo del hecho. El constructor deftemplate se usa para crear una plantilla

que luego se puede usar para acceder a los campos por nombre. La construcción deftemplate permite que el nombre de una plantilla se defina junto con cero o más definiciones de campos o espacios con nombre. A diferencia de los hechos ordenados, las ranuras de un hecho de deftemplate pueden estar restringidas por tipo, valor y rango numérico. Además, se pueden especificar valores predeterminados para una ranura.

```
(deftemplate <deftemplate-name> [<comment>]
   <slot-definition>*)

<slot-definition> ::= <single-slot-definition> |
                      <multislot-definition>

<single-slot-definition>
               ::= (slot <slot-name>
                         <template-attribute>*)

<multislot-definition>
               ::= (multislot <slot-name>
                              <template-attribute>*)

<template-attribute> ::= <default-attribute> |
                         <constraint-attribute>

<default-attribute>
               ::= (default ?DERIVE | ?NONE | <expression>*) |
                   (default-dynamic <expression>*)
```

Figura 3. Sintaxis de los templares.

Básicamente, un hecho no ordenado consta de un nombre de la relación (campo simbólico) seguido de cero o más casillas (también campos simbólicos). Un ejemplo de hecho es el siguiente:

```
(perro   (nombre "Maya")
         (raza "Labrador")
         (color-ojos marrones)
         (peso 27) )
```

Figura 4. Ejemplo de deftemplate.

Cada hecho, al igual que cada casilla, está delimitado por los paréntesis ( ). El símbolo perro es el nombre de la relación y el hecho consta de cuatro campos: nombre, raza, color-ojos y peso. El valor de la casilla nombre es "Maya", el de la casilla raza es "Labrador", el de la casilla color-ojos es marrones y el de la casilla peso es 27. El orden de las casillas es irrelevante.

1.8. Reglas

Uno de los métodos principales para representar el conocimiento en CLIPS es una regla. Las reglas se utilizan para representar heurísticas, o reglas de oro", que especifican un conjunto de acciones a realizar para una situación dada. Una **regla** está compuesta de un **antecedente** y un **consecuente**. El antecedente de una regla también se conoce como la porción if o el lado izquierdo (LHS) de la regla. El consecuente de una regla también se conoce como la porción then o el lado derecho (RHS) de la regla.

```
(defrule <rule-name> [<comment>]
   [<declaration>]              ; Rule Properties
   <conditional-element>*       ; Left-Hand Side (LHS)
   =>
   <action>*)                   ; Right-Hand Side (RHS)
```

Figura 5. Sintaxis de las reglas.

El antecedente de una regla es un conjunto de **condiciones** que deben cumplirse para que la regla sea aplicable. En CLIPS, las condiciones de una regla se satisfacen en función de la existencia o no existencia de hechos especificados en la lista de hechos o instancias especificadas de clases definidas por el usuario en la lista de instancias. Un tipo de condición que se puede especificar es un **patrón**. Los patrones consisten en un conjunto de restricciones que se utilizan para determinar qué hechos u objetos satisfacen los condiciones especificadas por el patrón. El proceso de hacer coincidir los hechos y los objetos con los patrones se denomina **pattern-matching**.

CLIPS proporciona un mecanismo, llamado motor de inferencia, que hace coincidir automáticamente los patrones con el estado actual de la lista de hechos o la lista de instancias y determina qué reglas son aplicables.

El consecuente de una regla es el conjunto de acciones que se ejecutarán cuando la regla sea aplicable. Las acciones de las reglas aplicables se ejecutan cuando se le indica al motor de inferencia CLIPS que comience a ejecutar las reglas aplicables. Si se aplica más de una regla, el motor de inferencia usa una estrategia de **resolución de conflictos** para seleccionar qué regla debe ejecutar sus acciones. Las acciones de la regla seleccionada se ejecutan (lo que puede afectar la lista de reglas aplicables) y luego el motor de inferencia selecciona otra regla y ejecuta sus acciones. Este proceso continúa hasta que no queden reglas aplicables.

Aunque en muchos sentidos, las reglas se pueden utilizar como declaraciones IF-THEN, estas condiciones en un lenguaje procedural sólo se evalúan cuando el flujo de control del programa está directamente en la instrucción. En contraste, las reglas actúan como declaraciones WHENEVER-THEN. El motor de inferencia siempre realiza un seguimiento de las reglas que tienen sus condiciones

satisfacidas y, por lo tanto, las reglas se pueden ejecutar de inmediato cuando son aplicables.

1.9. Objetos

COOL (Lenguaje orientado a objetos de CLIPS por sus siglas en inglés) es un lenguaje híbrido con características de muchos sistemas OOP diferentes y nuevas ideas. Una mezcla de ideas de Smalltalk, CLOS y otros sistemas forman la base de los mensajes.

```
(defclass <name> [<comment>]
  (is-a <superclass-name>+)
  [<role>]
  [<pattern-match-role>]
  <slot>*
  <handler-documentation>*)

<role>  ::= (role concrete | abstract)

<pattern-match-role>
        ::= (pattern-match reactive | non-reactive)

<slot>  ::= (slot <name> <facet>*) |
            (single-slot <name> <facet>*) |
            (multislot <name> <facet>*)

<facet> ::=  <default-facet> | <storage-facet> |
             <access-facet> | <propagation-facet> |
             <source-facet> | <pattern-match-facet> |
             <visibility-facet> | <create-accessor-facet>
             <override-message-facet> | <constraint-attributes>
```

Figura 6. Sintaxis de las clases.

Un **defclass** es una constructor para especificar las propiedades (ranuras) y el comportamiento (manejadores de mensajes) de una clase de objetos. Un defclass consta de cinco elementos: 1) un nombre, 2) una lista de superclases de las que la nueva clase hereda ranuras y manejadores de mensajes, 3) un parámetro que indica si se permite o no la creación de instancias directas de la nueva clase, 4) un parámetro que indica si las instancias de esta clase pueden coincidir o no con los patrones de objetos en el LHS de las reglas y 5) una lista de ranuras específicas a la nueva clase. Todas las clases definidas por el usuario deben heredar de al menos una clase, y para este fin, COOL proporciona clases de sistema predefinidas para su uso como base en la derivación de nuevas clases. Cualquier espacio especificado explícitamente en la orden de anulación anula los que se obtienen de la herencia.

1.10. Ejecución de reglas

Originalmente, CLIPS era un lenguaje de reglas basado en el Algoritmo Rete, que hace reconocimiento de patrones de manera eficiente para implementar un sistema de producción. En la quinta versión introdujo dos nuevos paradigmas

de programación: la programación imperativa y la programación orientada a objetos.

Generalmente cuando se desean ejecutar la reglas en CLIPS, se introduce el comando (run [<máximo de reglas a ejecutar>]). Si <máximo de reglas a ejecutar>no se especifica se ejecutan todas las reglas hasta que no haya más reglas que aplicar. Si se especifica el parámetro, se ejecutarán como máximo tantas reglas como el valor indicado.

1. Mientras el límite de ejecución de reglas no se alcance.
2. La agenda se actualiza atendiendo a la lista de hechos de la memoria de trabajo.
3. Se selecciona la mejor regla atendiendo a la estrategia de resolución de conflictos y las prioridades de las reglas.
4. La instancia seleccionada de una regla se dispara, y es eliminada de la agenda.
5. Regresar al paso 1.

2. Caso práctico

El principal objetivo este documento es exponer las diversas técnicas de representación del conocimiento que pueden ser empleadas en los sistemas expertos, para lograr este cometido, se propone un pequeño caso práctico derivado de una base de conocimiento médica. Dentro de la base se busca medicar a un paciente dado el cuadro clínico que este presenta, tomando encuenta tanto aspectos personales como su estado de salud y sintomas, como aspectos relacionado con organismos bacterianos como su morfología y dónde se localiza. La Figura 7 ejemplifica cada unas propiedades y objetos necesarios para entender la jerarquía de esta base de conocimientos.

Las Figuras 8 y 9 representan el gráfico AND/OR de las reglas que involucran el sistema experto médico.

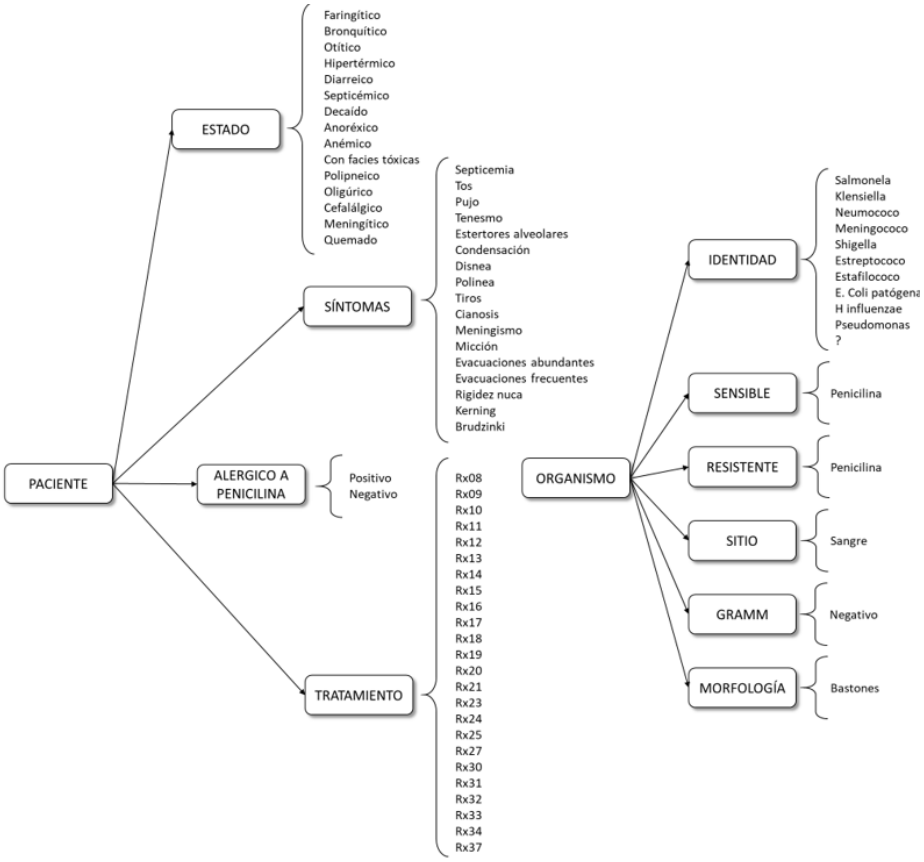

Figura 7. Diagrama de relaciones.

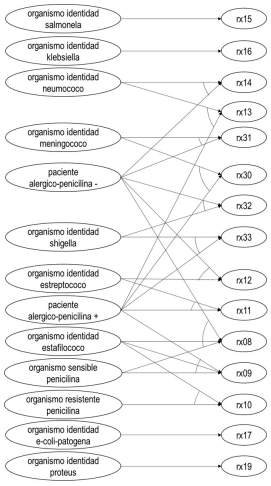

Figura 8. Gráfico AND/OR

Figura 9. Gráfico AND/OR

3. Técnicas de representación

Esta sección expone 4 técnicas de representación del conocimiento que pueden ser aplicadas a sistemas experto, cada una se implementa a partir de modificaciones a la base de conocimiento propuesta en la sección 2. El siguiente listado muestra las técnicas a tratar.

- Representación sencilla.
- Grados de certeza.
- Redes semánticas.
- Frames-marcos.

3.1. Representación sencilla

La primer técnica de representación de conocimiento consiste en hacer uso de los hechos y las reglas que proporciona CLIPS. En el caso específico del conocimiento médico la parte principal es el conjunto de reglas que se utiliza para generar conocimiento a partir de los datos (hechos). En específico podemos observar un ejemplo en la Figura 10.

```
(defrule regla012
        (organismo identidad estreptococo)
        (paciente alergico-penicilina -)
=>
        (assert (paciente rx12 penicilina-g 5 eritromicina 4 lincomicina 4))
        (printout t "paciente rx12 penicilina-g 5 eritromicina 4 lincomicina 4" crlf))
```

Figura 10. Ejemplo de Regla

Este conjunto de reglas establece la relación entre los datos que reflejan información sobre el organismo y el paciente y el tratamiento que debe recibir el paciente. Por ejemplo, en la Figura 10, dado que se tienen los hechos (organismo identidad estreptococo) y (paciente alergico-penicilina -), el paciente debe seguir el tratamiento 12, que receta penicilina-g 5 eritromicina 4 lincomicina 4. La parte derecha de esta regla especifica que se debe agregar (con assert) el hecho que define el tratamiento a la base de conocimientos o lista de hechos. Por otro lado también se utiliza la función printout para imprimir este resultado.

De manera similar se definen todas las reglas de la base, formando una correspondencia entre los elementos del gráfico AND/OR y las reglas definidas.

A partir de esto, en la interacción con el sistema si el usuario agrega los hechos (organismo identidad estreptococo) y (paciente alergico-penicilina -) a la lista de hechos con assert y tras la ejecución de *(run)*, CLIPS activará la regla 12.

3.2. Grados de certeza

Cuando se desea conocer de forma precisa qué tan posible es que un evento ocurra, las matemáticas, y en particular la probabilidad son la mejor herramienta que alguien podría elegir para tal fin. Sin embargo existen casos en los que, por determinadas circunstancias, un análisis probabilistico es inviable o incluso indeseable. Las razones para lo anterior pueden ser muy diversas, como lo es el que los datos obtenidos no sean suficientes, que la fuente de información no sea confiable, que la experiencia del sujeto que hace el estudio no concuerde con los modelos probabilísticos establecidos, etc.

En muchas de estas situaciones no se tiene más remedio que recurrir a la subjetividad; es decir, realizar suposiciones acerca de la información faltante, las cuales pueden estar basadas en la experiencia o en lo que el sujeto crea que es la mejor opción para su análisis. En estas situaciones la probabilidad no es una propiedad física tangible, obtenida a partir de hechos del mundo real y por tanto objetiva de los sucesos que son de interés, sino una percepción o grado de creencia en la verosimilitud de la persona que asigna la probabilidad sobre la plausibilidad de ocurrencia del suceso.

Dichos grados de creencia o certeza se pueden expresar como descriptores de una infinidad de conceptos, por ejemplo:

- Muy alto
- Frío
- Caliente
- Muy oscuro
- Poco probable

Los grados de certeza también se pueden expresar como números que expresan alguna probabilidad, sin embargo como se ha explicado, dicha probabilidad tiene un caracter subjetivo.

Por ejemplo, se podría establecer el siguiente modelo para conocer la certeza c de que los eventos E_1 y E_2 ocurran individualmente y en combinación. Si $c(E_1)$ y $c(E_2)$ es la certeza de que ocurran individualmente los eventos E_1 y E_2 respectivamente, se podría definir (subjetivamente):

$$c(E_1 \wedge E_2) = min(c(E_1), c(E_2))$$

$$c(E_1 \vee E_2) = max(c(E_1), c(E_2))$$

Como es de esperar, el punto debíl de estas formulaciones está en que no tienen más fundamentos que los que su creador pueda establecer, sean lógicos o

no, por lo que es deseable que los grados de certeza o creencía sean establecidos de la forma de una forma imparcial.

Utilizando la base de datos se simuló la atribución de diferentes grados de certeza a los padecimientos presentes en esta base, los cuales conllevan algún tratamiento o bien son síntomas de alguna otra enfermedad. Por ejemplo si se tiene que una enfermedad es diagnosticada mediante la presencia de tres síntomas, entonces se podría pensar que si se presentan los tres síntomas el médico diagnosticará la enfermedad con más certeza o confianza que si solo se presentan dos síntomas. Esto queda representado en las respuestas que da el programa a las consultas que contienen la información de los padecimientos que sufre el paciente.

Para realizar dicha labor se utilizó el constructor $deftemplate$. Debido a que la base de datos utilizada solo contiene dos estructuras de datos; *organismo* y *paciente*, solo se definieron dos plantillas para la elaboración del programa, las cuales se muestran a continuación (debido a su cantidad, no se muestran todos los atributos de la plantilla *paciente*):

```
1
2  ;Plantilla para almacenar la informacion de los organismos.
3
4  (deftemplate organismo
5
6          (slot identidad
7          (default x))
8
9          (slot sensible
10         (default x))
11
12         (slot resistente
13         (default x))
14
15         (slot sitio
16         (default x))
17
18         (slot gramm
19         (default x))
20
21         (slot morfologia
22         (default x)))
23
24 ;Plantilla para almacenar la informacion de los pacientes.
25
26 (deftemplate paciente
27
28         (slot alergico-penicilina
29         (default x))
30
31         (slot hipertermia
32         (default x))
33
34         (slot miccion
35         (default x))
36
37         (slot faringitico
38         (default x))
39
40         (slot bronquitico
41         (default x))
42
43         (slot otitico
44         (default x))
45                 .
46                 .
47                 .
```

Utilizando estas plantillas se pueden establecer *hechos*, con los cuales se le informa al sistema que se desea conocer cuales son los efectos que pueden desencadenar uno o varios de estos y con qué certeza. Algunas de las reglas que permiten conocer esta información se muestran a continuación:

```
1
2  ;Si se pregunta por una o mas instancias de la regla, esta asignara
3  un valor de certeza (porcentaje) y las sumara para conocer cual es
4  la certeza de que dichas instancias provoquen un determinado efecto.
5
6  (defrule regla008
7     (organismo (identidad ?estafilococo))
8     (organismo (sensible ?penicilina))
9     (paciente (alergico-penicilina ?negativo))
10
11    =>
12
13    (progn
14    (bind ?certeza 0)
15    (if (eq ?estafilococo estafilococo) then (bind ?certeza (+ ?certeza 23)))
16    (if (eq ?penicilina penicilina) then (bind ?certeza (+ ?certeza 12)))
17    (if (eq ?negativo negativo) then (bind ?certeza (+ ?certeza 36)))
18    (if (> ?certeza 0) then (printout t "El␣tratammiento␣a␣seguir␣es:
19 ␣␣␣␣penicilina-g␣5.␣Con␣una␣certeza␣de:␣" ?certeza "%" crlf))
20    ))
21
22 ;Algunas enfermedades o tratamientos son causados por un solo hecho, por lo
23 que se podria pensar que dado ese hecho, la certeza de que se de el tratamiento
24 o enfermedad es alta.
25
26 (defrule regla015
27    (organismo (identidad ?salmonela))
28 =>
29    (progn
30    (bind ?certeza 0)
31    (if (eq ?salmonela salmonela) then (bind ?certeza (+ ?certeza 90)))
32    (if (> ?certeza 0) then (printout t "El␣tratammiento␣a␣seguir␣es:
33 ␣␣␣cloromicetina␣5␣ampicilina␣5.␣Con␣una␣certeza␣de:␣" ?certeza "%" crlf))
34    ))
```

Por último se presentan algunas consultas que activan las reglas anteriores:

```
1
2  ;Algunos hechos tienen mas de un efecto.
3
4  CLIPS> (assert (organismo (identidad estafilococo) (sensible penicilina))
5  (paciente))
6  <Fact-1>
7  CLIPS> (run)
8  El tratammiento a seguir es: penicilina-g 5. Con una certeza de: 35%
9  El tratammiento a seguir es: eritromicina. Con una certeza de: 67%
10 El tratammiento a seguir es: cefalotina cefaloridina 5 cloxacilina
11 dicloxacilina meticilina naficilina oxacilina 5 eritromicina. Con una
12 certeza de: 10%
13
14
15 CLIPS> (assert (organismo (identidad salmonela)) (paciente))
16 <Fact-2>
17 CLIPS> (run)
18 El tratammiento a seguir es: cloromicetina 5 ampicilina 5. Con una
19 certeza de: 90%
```

3.3. Redes semánticas

Una red semántica es una forma de representación de conocimiento en la cual los conceptos y sus interrelaciones se representan mediante un grafo dirigido o no dirigido. Los conceptos son representados mediante nodos, mientras que los arcos representan las relaciones semánticas entre los conceptos. Son ampliamente utilizadas en el área del procesamiento del lenguaje natural en aplicaciones como analizadores gramaticales semánticos y en general en el mundo de la computación para recuperar información de gigantescas bases de datos en forma rápida y eficiente. Se ha verificado que las redes semánticas juegan un rol importante en las aplicaciones enfocadas al entendimiento y la representación de síntesis de textos.

Las redes semánticas fueron creadas por Richard H. Richens, de la Cambridge Language Research Unit en 1956; y posteriormente desarrolladas por Robert F. Simmons, Sheldon Klein, Karen McConologue, M. Ross Quillian.

Figura 11. M. Ross Quillian, Robert F. Simmons y Sheldon Klein.

Existen varios tipos de relaciones entre conceptos utilizados para elaborar redes semánticas. Una relación muy común que une a dos conceptos es la relación *esun*: A *esun* B, la cual significa que A es un concepto menos general que B. Existen otras relaciones comúnes, como *tiene*, *es*, *causa*, etc. Más aparte las que pudiera definir el modelador de la red semántica.

Una de las propiedades más importantes de las redes semánticas es la herencia, lo cual permite que un nodo herede las propiedades de conceptos más altos. Cuando esto sucede se dice comúnmente que el nodo que hereda dichas propiedades es *subclase de* el nodo que las otorga.

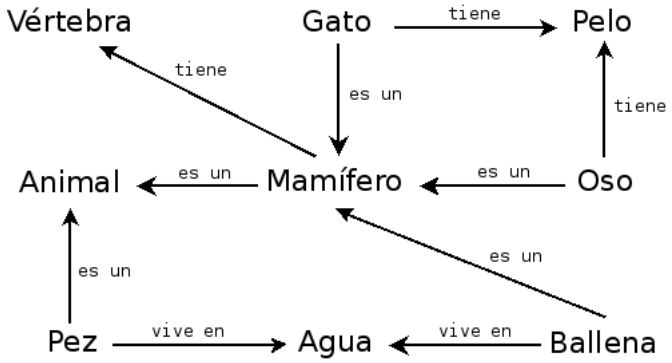

Figura 12. Ejemplo de una red semántica.

Una propiedad interesante de estas redes para la inteligencia artificial es la posibilidad de representar descripciones lógicas por medio de ellas, al hecer uso de grafos existenciales o grafos conceptuales. Ya que estos tienen un poder expresivo igual o superior que la lógica de primer orden o de predicados. Utilizando estas representaciones, una red semántica puede ser usada como un confiable deductor lógico automatizado.

La implementación de una red semántica en $CLIPS$ se puede realizar mediante el constructor $deftemplate$, mediante el cual se crea un tipo de estructura que consta de un identificador y un número arbitrario de atributos y sus respetivos valores para ese identificador. Dichos atributos se declaran anteponiendo utilizando el término *slot*. Esta herramienta se utilizó para realizar una red semántica con la base de datos .

Los *templates* que se definieron para la realización de la red semántica son los mismos que se usaron en la sección *gradosdecerteza*. Sin embargo las reglas difieren un poco. Estas fueron elaboradas con el objetivo de conocer cules son las clases que contienen a determinada instancia; si es que se les puede llamar así, ya que lo que se considera como una clase padre en esta red son los tratamientos que corresponden a determinados padecimientos, así como las enfermedades que son causadas por otro conjunto de padecimientos, lo cual no necesariamente tiene la característica de ser una instancia más general de sus clases hijas. Algunos ejemplos de las reglas definidas para la red son las siguientes:

```
1
2  (defrule regla008
3
4    ;Esta regla contiene las siguientes instancias de una clase superior.
5
6     (or (organismo (identidad estafilococo))
7     (organismo (sensible penicilina))
8     (paciente (alergico-penicilina -)))
9
10    =>
11
12    ;Si se consulta alguna de las instancias de la superclase se imprimira
13    la identidad de esta (en este caso el tratamiento que se lleva a cabo).
14
15    (printout t "La␣superclase␣es␣el␣tratamiento:␣penicilina-g␣5" crlf))
16
17
18
19
20 (defrule regla022
21
22    (or (paciente (hipertermia prolongada))
23    (paciente (miccion alterada)))
24
25    =>
26
27    ;Algunas clases son una condicion de un paciente que a su vez contienen
28    a las condiciones necesarias para que estas se disparen.
29
30    (progn
31    (printout t "La␣superclase␣es␣(paciente␣(infec-urinaria␣+))" crlf)
32    (assert (paciente (infec-urinaria +)))))
```

Ejemplos de consultas que activan las reglas anteriores se muestran a continuación:

```
1
2  ;Algunas instancias pertenecen a mas de una clase.
3
4  CLIPS> (assert (organismo (identidad estafilococo)))
5  <Fact-0>
6  CLIPS> (run)
7  La superclase es el tratamiento: penicilina-g 5
8  La superclase es el tratamiento: eritromicina
9  La superclase es el tratamiento: cefalotina cefaloridina 5
10 cloxacilina dicloxacilina meticilina naficilina oxacilina 5 eritromicina
11
12 ;Otras tienen como clase mas instancias que a su vez tambien consultan la
13 clase a la que pertenecen.
14
15 CLIPS> (assert (paciente (miccion alterada)))
16 <Fact-1>
17 CLIPS> (run)
18 La superclase es (paciente (infec-urinaria +))
19 La superclase es el tratamiento: tetraciclina 3 kanamicina 3 lincomicina 3
```

3.4. Frames-Marcos

Dentro de esta sección se aborda la ténica de representación propuesta por Marvin Minsky en su artículo *A Framework for Representing Knowledge*. Este tipo de representación puede verse como un antecedente de la programación orientada a objetos, en la cual se propone una estructura de datos llamada **marcos** que se encuentra conformada por **ranuras**. Una característica sobresaliente de los marcos es la capacidad de almacenar valores, referencias a otros marcos y procedimientos que regresan valores.

Derivado del caso práctico, se propone el uso de clases CLIPS para representar entidades abstractas que conforman la base de conocimiento. Aunque el término clase esté más ligado al paradigma de objetos, en el ambiente CLIPS comparten ciertas similitudes con los marcos. La propuesta define inicialmente la clase *organismos* que se refiere a bacterias junto con sus principales atributos y la clase *pacientes* que registra los posibles sintomas, estados y otras características de una persona con una patología.

```
(defclass organismos (is-a USER)
        (slot identidad (allowed-symbols desconocido salmonela klebsiella
        meningococo shigella estreptococo estafilococo e-coli-patogena
        proteus h-influenzae pseudomonas neumococo))
        (slot sensible (allowed-symbols nil penicilina))
        (slot resistente (allowed-symbols nil penicilina))
        (slot sitio (allowed-symbols nil sangre))
        (slot gramm (allowed-symbols nil + -))
        (slot morfologia (allowed-symbols nil bastones)))
```

Cuadro 2. Clase organismos definida en CLIPS.

Los Cuadros 2 y 3 ejemplifican las implementaciones de las clases y las restricciones impuestas para la creación de instancias. Como es observado, las ranuras sólo pueden recibir valores tipo símbolo, y a su vez, cada ranura tiene un listado de símbolos permitidos. Cuando se genera una instancia de una clase y no se especifica los valores de sus ranuras, esta hereda el primer símbolo de la lista de símbolos permitidos que en la mayoría de los casos es *nil*.

```
(defclass pacientes (is-a USER)
        (slot tos (allowed-symbols nil + -))
        (slot pujo (allowed-symbols nil + -))
        (slot tenesmo (allowed-symbols nil + -))
        .
        .
        .
        (slot diarreico (allowed-symbols nil + -))
        (slot diarr-bact (allowed-symbols nil + -))
        (slot rigidez-nuca (allowed-symbols nil + -)))
```

Cuadro 3. Clase pacientes definida en CLIPS.

Para aproximar la representación al planteamiento conceptual, se propone el uso de reglas para que en este caso se imprima en pantalla el posible tratamiento que se le puede asignar a un paciente.

El Cuadro 4 muestra tres reglas, la primera (iniciar) comienza el proceso de inferencia preguntando cuáles son las instancias a considerar para la derivación de nuevos hechos. En dado caso que sólo se considere una instancia se puede usar el valor *nil* para la instancia a omitir. Por otro lado, la introducción de las intancias debe ser del tipo *name-instance* con el formato *[NOMBRE]*. Una vez introducidas las intancias, se agregan los hechos (iniciado), (caso-organismo [instancia1]) y (caso-paciente [instancia2]) que conducirán la inferencia.

Las reglas *regla008* y *regla009* inducen nuevos hechos modificando ranuras en las instancias, todo este proceso se dirige gracias a los hechos agregados anteriormente debido a que las reglas sólo realizan el pattern matching como el nombre de las instancias en los hechos (caso-organismo ?) y (caso-paciente ?).

```
(defrule iniciar
        (not (iniciado))
=>
        (printout t "Introduzca␣la␣instancia␣del␣organismo␣a␣consultar" crlf)
        (bind ?org (read))
        (printout t "Introduzca␣la␣instancia␣del␣paciente␣a␣consultar" crlf)
        (bind ?pac (read))
        (assert (iniciado))
        (assert (caso-organismo ?org))
        (assert (caso-paciente ?pac)))

(defrule regla008
        (caso-organismo ?org)
        (caso-paciente ?pac)
        ?o1 <- (object (is-a organismos) (name ?org) (identidad estafilococo)
        (sensible penicilina))
        ?o2 <- (object (is-a pacientes) (name ?pac) (alergico-penicilina -))
=>
        (assert (paciente rx08 penicilina-g 5))
        (printout t "paciente␣rx08␣penicilina-g␣5" crlf))

(defrule regla009
        (caso-organismo ?org)
        (caso-paciente ?pac)
        ?o1 <- (object (is-a organismos) (name ?org) (identidad estafilococo)
        (sensible penicilina))
        ?o2 <- (object (is-a pacientes) (name ?pac) (alergico-penicilina +))
 =>
        (assert (paciente rx09 eritromicina))
        (printout t "paciente␣alergico-penicilina␣+" crlf))
```

Cuadro 4. Reglas definidas en CLIPS.

Como último punto a abordar se definen prefivamente 5 instancias de la clase organismos y 5 instancias de la clase paciente con *definstances* (Cuadro 5).

```
(definstances baseMedica
        (org1 of organismos (identidad salmonela))
        (org2 of organismos (identidad estafilococo) (resistente penicilina))
        (org3 of organismos (identidad estreptococo))
        (org4 of organismos (identidad desconocido))
        (org5 of organismos (identidad pseudomonas) (sitio sangre)
        (gramm +) (morfologia bastones))
        (pac1 of pacientes (alergico-penicilina +))
        (pac2 of pacientes (hipertermia prolongada) (miccion alterada))
        (pac3 of pacientes (diarreico +) (hipertermico +))
        (pac4 of pacientes (faringitico +) (septicemico +) (quemado +))
        (pac5 of pacientes (rigidez-nuca +) (kernig +) (brudzinski +)
        (hipertermico +) (cefalalgico +)))
```

Cuadro 5. Instancias definidas en CLIPS.

En el Cuadro 6 se expone una corrida del sistema experto donde se seleccionan los objetos [org4] y [pac5] obteniendo un tratamiento para el caso clínico.

```
CLIPS> (reset)
CLIPS> (run)
Introduzca la instancia del organismo a consultar
[org4]
Introduzca la instancia del paciente a consultar
[pac5]
paciente meningismo +
paciente meningitico +
paciente rx37 penicilina-g 3 lincomicina 2 tetraciclina 3
paciente grave endovenosa
CLIPS>
```

Cuadro 6. Ejemplo de ejecución.

Comentario

Aunque existe una representación para la inferencia bayesiana, su desarrollo es considerablemente más complejo y por lo tanto se omite en estas notas.

Referencias

1. Culbert, C., Riley, G., Donnell, B. (2015). CLIPS–Reference Manual Volume I Basic Programming Guide (Version 6.30).