Proyecto Inter ciclo de la materia de Sistemas Expertos (diciembre de 2020)

Edison huinaizaca
ehuinaizaca@est.ups.edu.ec

Resumen – Proyecto desarrollado para la materia de Expertos que consiste el desarrollo de un sistema experto basado en Reglas utilzando CLIPS y PROLOG,donde se implementara las siguientes comprobaciones rutinarias a realizar a un coche que entra en un taller, para averiguar averías fácilmente detectables y sugerir una posible reparación.

Índice de Términos – lenguaje de programación, algorítmico de lenguajes naturales,Reglas,Condisiones

I. INTRODUCCION

El problema a tartar de este proyecto es el desarrollar a partir del uso del software de clips y Prolog ,un sistema experto basado en reglas que tratara de resolver los problemas que presenta un coche al hacer una revisión rutinaria .Además de ver las características y capacidades de cada software tanto a nivel estructural como a nivel lógico de estas dos aplicaciones las cuales se basan en estructuras de conocimiento de hechos y reglas ,esto significa que las reglas son los elementos que permitirán que el sistema evolucione, mediante la modificando hechos.

II. QUE ES PROLOG Y CLIPS

1) PROLOG

Se trata de un lenguaje de programación ideado a principios de los años 70 en la Universidad de Aix-Marseille I (Marsella, Francia). Creado por Alain Colmerauer y Philippe Roussel, este proyecto nació de un proyecto que no tenía como objetivo la traducción de un lenguaje de programación, sino el tratamiento algorítmico de lenguajes naturales. Alain Colmerauer y Robert Pasero trabajaban en la parte del procesado del lenguaje natural y Jean Trudel y Philippe Roussel en la parte de deducción e inferencia del sistema. Interesado por el método de resolución SL, Philippe Roussel persuadió a su autor, Robert Kowalski para que colaborara con el proyecto, dando lugar a una versión preliminar del lenguaje Prolog a finales de 19712 y apareciendo la versión definitiva en 1972.3 Esta primera versión de Prolog fue programada en ALGOL W.
Inicialmente se trataba de un lenguaje totalmente interpretado hasta que, en 1983, David H.D. Warren desarrolló un compilador capaz de traducir Prolog en un conjunto de instrucciones de una máquina abstracta denominada Warren Abstract Machine, o abreviadamente, WAM. Desde entonces Prolog es un lenguaje semi-interpretado.

Documento recibido el 13 de diciembre de 2020.

a) Para que sirve SWI-Prolog

La mayoría de las implementaciones del lenguaje Prolog están diseñadas para servir a un conjunto limitado de casos de uso. SWI-Prolog no es una excepción a esta regla. SWI-Prolog se posiciona principalmente como un entorno que actúa como "pegamento" entre componentes. Al mismo tiempo, SWI-Prolog tiene como objetivo proporcionar un entorno productivo de creación rápida de prototipos

b) Operadores aritméticos.

Con estos podemos llevar a cabo operaciones aritméticas entre números de tipo entero o real, sin embargo se tuvieron en cuenta sólo los básicos, pero existen para las funciones trigonométricas, valor absoluto, piso, techo, entre otros muchas más.

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/ y //	División real y entera
^ y **	Potencia
+	Positivo
-	Negativo

TABLA. 1.Tabla de Operadores aritméticos

c) Operadores relacionales con evaluación

Este tipo de operadores recibe valores numéricos y/o expresiones antes de realizar unificación o comparaciones evalúa el valor de la expresión

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
=:=	Igualdad	10 + 2 =:= 5 + 7
=\=	Desigualdad	10 + 2 =\= 5 + 8
>	Mayor que	11 * 3 > 3 ^ 2
<	Menor que	2 ** 10 < 5 * 2
>=	Mayor o igual que	99.0 >= 0
=<	Igual o menor que	-15 =< 15

TABLA. 2.tabla de relacionales con evaluación

d) Operadores de listas:

Las operaciones en listas nos permiten consultar alguna propiedad de una lista, así como realizar modificaciones.

Operador	Significado	Ejemplos
=	Unificación	[X, Y, Z] = [a, 1, 2.0]
member(term, list)	term ∈ list	member(4.0, [c, 3, 4.0]). member(X, [c, 3, 4.0]).
append(list1, list2, result)	Une list1 con list2	append([h, o], [l, a], X). append([h, o], X, [h, o, l, a])
length(list, result)	Calcula la longitud de la lista	length([3, 0.0, x], X).
sort(list, result)	Ordena la lista	sort([4, a, 3], X).
is_list(term)	Comprueba si term es lista	is_list([a, list]).

TABLA. 3.Tabla de listas

e) Backtracking

Cuando Prolog intenta demostrar un predicado aplica el algoritmo de backtracking. Este algoritmo recorre todas las soluciones posibles, pero de forma más inteligente que la fuerza bruta. Backtracking intenta conseguir una solución hasta que un predicado falla, en ese momento, Prolog va hacia atrás y continúa por otra vía que pueda seguir.

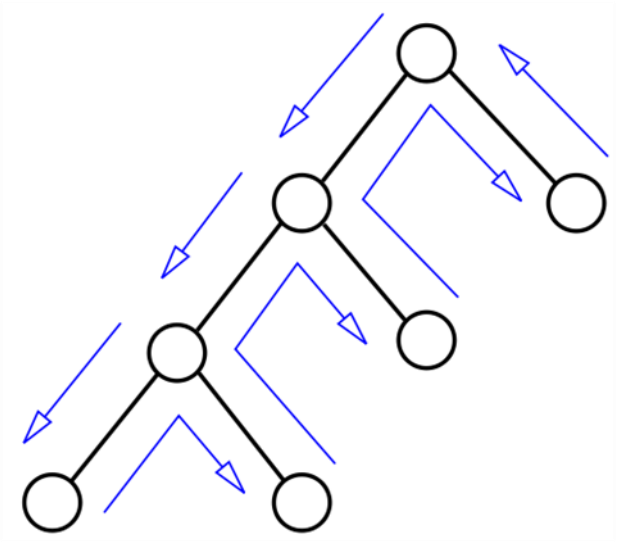

Fig. 1. Backtracking

Cada predicado es un nodo. Si un predicado falla se vuelve atrás. Esto es muy interésate ya que Prolog técnicamente puede ejecutar código hacia atrás.

2) CLIPS

CLIPS es un acrónimo de C Language Integrated Production System (Sistema de Producción Integrado en Lenguaje C). Es un lenguaje programado en C, un lenguaje de programación universal tiene comunicación con otros lenguajes como C y ADA y mantiene similitudes al lenguaje C y LISP, es un sistema experto desarrollado por la NASA durante la década de los ochenta, proporcionaba un entorno completo para la construcción de sistemas expertos basados en reglas u objetos. Fue creado como una opción distinta a LISP .Porque LIPS en muchas ocasiones era incompatible con otras herramientas o aplicaciones. En un principio la NASA propuso que el desarrollo de una herramienta así fuese llevado a cabo por una empresa externa. Sin embargo, los costes del proyecto eran demasiado altos, así que se le encargó a la unidad de IA de la NASA.
A partir de mediados del año 1986 CLIPS fue liberado para grupos y usuarios que no perteneciesen a la NASA haciendo que se diese a conocer y fuese ganando más popularidad poco a poco. Con los años este sistema experto se ha continuado usando gracias al trabajo de sus desarrolladores, que han seguido actualizándolo y añadiendo algunas mejoras a pesar de que éste sea dominio público.

a) Para que sirve

En cuanto a su funcionamiento, al igual que otros sistemas expertos, CLIPS se basa en una serie de conocimientos y reglas. Los conocimientos son la información del entorno que es percibida y las reglas son las pautas que tiene el sistema para que sea posible su evolución. Puede que al principio CLIPS se tratase de una herramienta de entrenamiento para la construcción de sistemas expertos sin embargo también servía

para el desarrollo y ejecución de ellos. Tras completarse el desarrollo de su primera versión y ser utilizado durante un año como periodo de prueba se demostró que era una alternativa mas barata a otras herramientas que hacían el mismo trabajo

b) Características

Algunas de las características de CLIPS son:

- *Representación del Conocimiento*: permite manejar una amplia variedad de conocimiento, soportando tres paradigmas de programación que son: el declarativo, el imperativo, y el orientado a objetos. Esto permite que el conocimiento sea representado como reglas heurísticas.
- *Portabilidad:* CLIPS fue escrito en C con el fin de hacerlo más portable y rápido,esto permite ser ejecutado en cualquier sistema con un compilador ANSI de C, o un compilador de C++ en cualquier sistema operativo . además de que el código fuente de CLIPS puede ser modificado en caso que el usuario lo considere necesario, con el fin de agregar o quitar funcionalidades.
- *Integrabilidad:* CLIPS puede ser embebido en código imperativo, invocado como una sub-rutina, e integrado con lenguajes como C, Java, FORTRAN y otros. Además de que incorpora un completo lenguaje orientado a objetos para la elaboración de sistemas expertos.
- *Desarrollo Interactivo*: La versión estándar de CLIPS provee un ambiente de desarrollo interactivo y basado en texto; este incluye herramientas para la depuración, ayuda en línea, y un editor integrado.
- *Verificación/Validación*: CLIPS contiene funcionalidades que permiten verificar las reglas incluídas en el sistema experto que está siendo desarrollado, incluyendo diseño modular y particionamiento de la base de conocimientos del sistema.

c) Estructura

Hay tres formas de representar el conocimiento en CLIPS:
• Reglas, que están destinadas principalmente al conocimiento heurístico basado en experiencia.
• Defunciones y funciones genéricas, que están destinadas principalmente al conocimiento procedimental.
• Programación orientada a objetos, también destinada principalmente al conocimiento de procedimientos.

El shell CLIPS proporciona los elementos básicos de un sistema experto:
1. lista de hechos y lista de instancias: memoria global para datos
2. base de conocimientos: contiene todas las reglas, base de reglas
3. motor de inferencia: controla la ejecución general de las reglas

Comandos básicos usados en clips

- (exit)salir de CLIPS
- (clear)borra de CLIPS todos los hechos, reglas y definiciones.
- (reset)pone el sistema en su estado inicial.
- (initial_fact)así como todos los que el usuario defina por defecto.
- (run)ejecuta el programa cargado en CLIPS
- (load “nombrefichero.clp”)Carga un programa CLIPS a partir del fichero nombrefichero.clp

B. Fact

CLIPS proporciona varios comandos para ayudarlo a depurar programas. Un comando le permite ver continuamente los hechos afirmados y retractados. Esto es más conveniente que tener que escribir un comando (hechos) una y otra vez y tratar de averiguar qué ha cambiado en la lista de hechos. Para comenzar a ver hechos, ingrese el comando (ver hechos) como se muestra en el siguiente ejemplo.
CLIPS> (borrar)
CLIPS> (ver hechos)
CLIPS> (afirmar (animal-es pato)) ==> f-1 (animal-is duck)
<Fact-1>
CLIPS>
El símbolo de la flecha doble derecha, ==>, significa que un hecho está entrando en la memoria mientras que la flecha doble izquierda indica que un hecho está saliendo de la memoria, como se muestra siguiendo.
CLIPS> (restablecer) <== f-1 (el animal es un pato)
CLIPS> (afirmar (el animal es un pato)) ==> f-1 (el animal es un pato)
<Fact-1> CLIPS> (retraer 1) <== f-1 (animal-is pato)
CLIPS> (hechos)
CLIPS>
El comando (ver hechos) proporciona un registro que muestra la dinámica; o cambiar el estado de la lista de hechos. Por el contrario, el comando (hechos) muestra el estado estático de la lista de hechos, ya que muestra el contenido actual de la lista de hechos. Para desactivar la visualización de hechos, ingrese (dejar de ver hechos).

C. Funciones

Una función en CLIPS es un fragmento de código ejecutable identificado por un nombre específico que devuelve un valor útil o realiza un efecto secundario útil (como mostrar información). En toda la documentación de CLIPS, la palabra función se usa generalmente para referirse solo a funciones que devuelven un valor. Las funciones definidas por el usuario y las funciones definidas por el sistema son fragmentos de código que se han escrito en un lenguaje externo y se han vinculado con el entorno CLIPS. Las funciones definidas por el sistema son aquellas funciones que han sido definidas internamente por el entorno CLIPS.A continuación se muestran algunos ejemplos de llamadas a funciones que utilizan las funciones de suma (+) y multiplicación (*).
(+ 3 4 5)
(* 5 6.0 2)
(+ 3 (* 8 9) 4)
(* 8 (+ 3 (* 2 3 4) 9) (* 3 4))
Mientras que una función se refiere a un fragmento de código ejecutable identificado por un nombre específico, una

expresión se refiere a una función que tiene sus argumentos especificados (que pueden ser o no llamadas a funciones también).).

D. Objetos

Un objeto en CLIPS se define como un símbolo, una cadena, un número entero o de punto flotante, un valor de campo múltiple, una dirección externa o una instancia de una clase definida por el usuario. Algunos ejemplos de objetos y sus clases son:

Objeto (representación impresa)	Clase
Rolls-Royce	SÍMBOLO
"Rolls-Royce"	STRING
8.0	INTEGER
(8.0 Rolls-Royce 8 [Rolls-Royce])	MULTIFIELD
<Pointer-00CF61AB>	EXTERNAL-ADDRESS
[Rolls-Royce]	CARRO (una clase definida por el usuario)

TABLA.4.Tabla de Objetos

Los objetos en CLIPS se dividen en dos categorías importantes: tipos primitivos e instancias de clases definidas por el usuario. Estos dos tipos de objetos se diferencian en la forma en que se hace referencia a ellos, se crean y se eliminan, así como en la forma en que se especifican sus propiedades.

III. CIPS VS PROLOG

PROLOG	CLIPS
Herramienta para programar artefactos electrónicos mediante el paradigma lógico	Herramienta de desarrollo y manejo de sistemas expertos.
Prolog está basado en la lógica de primer orden, es aquella que resuelve problemas formulados con una serie de objetos y relaciones entre ellos	CLIPS fue escrito en C con el fin de hacer lo más portable y rápido
Se compone de hechos y un conjunto de reglas, es decir, relaciones entre objetos de la base de datos.	Se compone básicamente de reglas definidas, y está basado en texto que incluyen herramientas.
Los comentarios se definen entre los símbolos/*y*/.	Los comentarios se colocan detrás de un punto y coma.
Una llamada concreta a un predicado o a una determinada función, con unos argumentos concretos.	Una función comienza con un paréntesis izquierdo, seguido por el nombre de la función y a continuación le siguen los argumentos de la función separados por uno o más espacios. La llamada a la función finaliza con un paréntesis de cierre.
Tiene una sintaxis y semántica simples. Sólo busca relaciones entre los objetos creados, las variables y las listas ,que son sus estructuras básicas	Su sintaxis está basada en la sintaxis de ART (otra herramienta para el desarrollo de sistemas expertos).

TABLA.5.Tabla de comparacion

IV. DESCRIPCIÓN DEL PROBLEMA

A. Planeación del problema

Se desea realizar un programa en CLIPS y Prolog que implemente las siguientes comprobaciones rutinarias a realizar a un coche que entra en un taller, para averiguar averías fácilmente detectables y sugerir una posible reparación:
- Lo primero es comprobar si el coche arranca y funciona perfectamente . o si arranca pero su funcionamiento no es satisfactorio. También podría ocurrir que no arrancara en absoluto.
- Otro punto a examinar es la rotación del motor: si el coche arranca, sabemos que el motor gira. Si el coche no arranca, hay que comprobarlo.
- También hay que examinar la carga de la batería. Si el motor arranca, la batería está cargada. Si el motor no gira, hay que comprobarlo.
- El encendido de las bujías: si el coche arranca y funciona, sabemos que las bujías estaban bien, sin necesidad de comprobarlo. Si el coche no arranca, pero gira el motor, el encendido es defectuoso. Si el motor no gira, el encendido está completamente estropeado. Si el coche funciona, pero no del todo bien, hay que comprobar el estado del encendido.

Una vez realizadas las comprobaciones de los cuatro puntos anteriores, se podrá pasar a las comprobaciones finales, que nos llevarán a las reparaciones a realizar:
- En el caso de que el coche funciona, pero no satisfactoriamente, cuatro posibles comprobaciones tendrían sentido:
  - La suciedad del filtro de gasolina
  - Las bujías
  - Bloqueo en el motor
  - Bajada de potencia del motor.
- En el caso de que el coche no arrancara, pero el motor girara, una comprobación rutinaria sería ver si tiene gasolina
- Si no arrancara y las bujías estuvieran defectuosas, o bien simplemente la potencia es baja, entonces hay que mirar los contactos de las bujías, que podrían estar de tres formas posibles: normales, quemados u obstruidos. Si estuvieran quemados, la SOLUCIÓN sería cambiar los contactos.
Si estuvieran obstruidos la SOLUCIÓN es limpiarlos.
- Si el motor no arrancara, la batería estuviera cargada, y el encendido estropeado, entonces hay que comprobar si funciona el mecanismo de explosión del coche. Si es así, la SOLUCIÓN es repararlo. Si no, la SOLUCIÓN es cambiar el conducto de la gasolina.
- Si con las anteriores comprobaciones no hemos sido capaces de dar una solución, entonces habrá que llevar el coche a un mecánico que sea más experto.

V. DIAGRAMA ESQUEMÁTICO

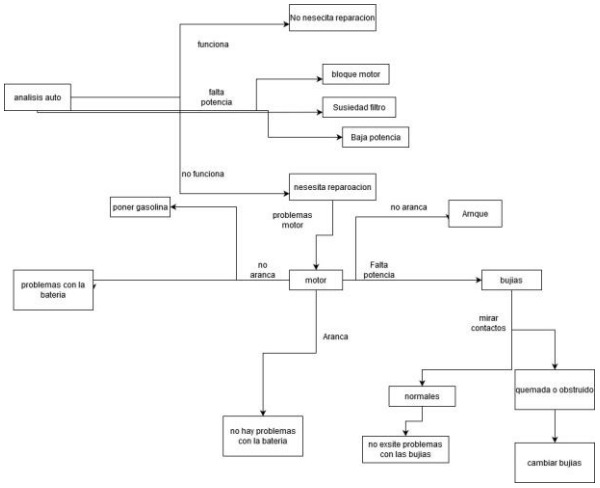

Fig. 2. Diagrama Esquematico

VI. DESARROLLO DEL PROBLEMA

A. Desarrollo en prolog

Lo concerniente a prolog se divide en dos parte una visual y una parte lógica la parte visual de prolog que mediante el uso de algunas librerías se pudo implementar una parte grafica del sistema se podrá visualizar en la siguiente imagen

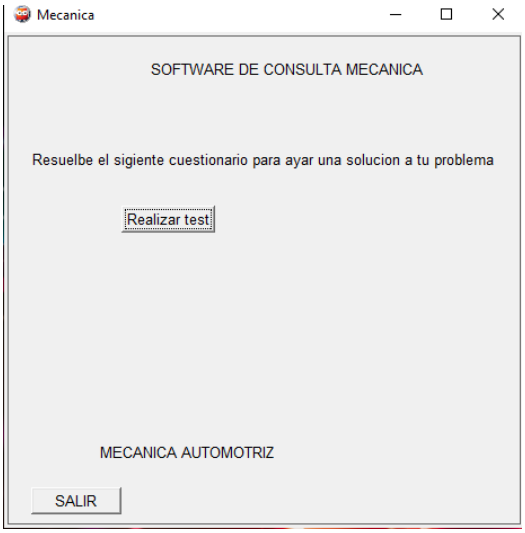

Fig. 3.Bentana principal

Resultados del sistema que da el sistema al responder el cuestionario de preguntas :

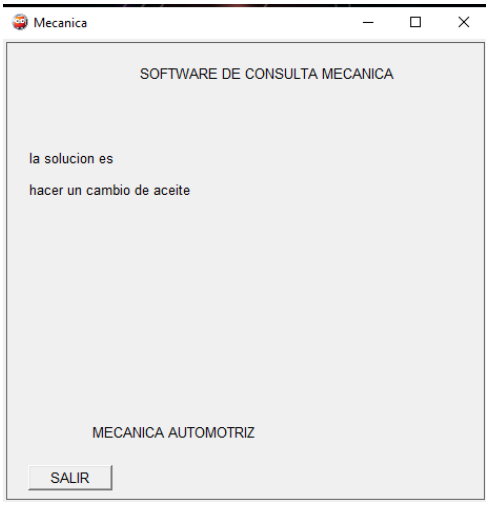

Fig. 5.Pagina de respuestas

A continuación se hablara de la parte lógica del sistema la siguiente sección el código contiene la solución a las fallas de acuerdo a las reglas de diagnóstico:

```
fallas('hacer un cambio de aceite'):-aceite,!.
```

Esta sección es la parte donde se resuelven las preguntas para resolver las fallas con su respectivo identificador de aceite:- cambio_aceite,
pregunta('tienes problemas de motor?').

A continuación se podrá apreciar la parte más importante del código la cual es el proceso del diagnóstico basado en preguntas de si y no, Donde el usuario dice si, se pasa a la siguiente pregunta del mismo ramo, si dice que no se pasa a la pregunta del siguiente ramo y así hasta dar con una solución o responder que no tiene solución.

```
:-dynamic si/1,no/1.
preguntar(Problema):- new(Di,dialog('Diagnostico
mecanico')),
    new(L2,label(texto,'Responde las siguientes preguntas')),
    new(La,label(prob,Problema)),
    new(B1,button(si,and(message(Di,return,si)))),
    new(B2,button(no,and(message(Di,return,no)))),

      send(Di,append(L2)),
          send(Di,append(La)),
          send(Di,append(B1)),
          send(Di,append(B2)),

          send(Di,default_button,si),
          send(Di,open_centered),get(Di,confirm,Answer),
          write(Answer),send(Di,destroy),
          ((Answer==si)->assert(si(Problema));
          assert(no(Problema)),fail).
```

B. Desarrollo en CLIPS

El desarrollo en clips fue desarrollado en consola de clips aunque cambian algunas preguntas sigue teniendo el mismo desarrollo planteado en el problema, Ejemplo de ejecución del problema:

```
CLIPS> (load "MecanicaClips.CLP")
Defining defrule: tipo-problemas_auto +j+j
Defining defrule: problemas-auto +j+j
Defining defrule: problemas-auto2 =j+j
Defining defrule: problemas-auto3 =j+j
Defining defrule: problemas-auto4 =j+j
Defining defrule: problemas-auto5 =j+j
Defining defrule: problemas-gasolina =j+j+j+j
Defining defrule: problemas-bateria =j=j+j+j
Defining defrule: problemas-bujias =j+j+j+j
Defining defrule: problemas-filtro =j=j=j+j
Defining defrule: problemas-Aranque =j=j+j+j
Defining defrule: problemas-consultar =j=j=j+j+j+j+j
TRUE
CLIPS> (run)
Tienes problemas con tu auto si/no?
si
el motor aranca(si/no)?
no
enciende luz(si/no)?
si
Falta Gasolina
```

Fig. 5.Ejecusion Programa

A continuación se podrá apreciar las partes mas importantes del código:
Esta sección es el inicio del codigo en clip:

```
(defrule tipo-problemas_auto
(initial-fact)
=>
(printout t "Tienes problemas con tu auto si/no?" crlf)
(assert (tipo-problema (read)))
)
```

En esta sección se plantea las preguntas que darán solución al problema:

```
(defrule tipo-problemas_auto
(initial-fact)
=>
(printout t "Tienes problemas con tu auto si/no?" crlf)
(assert (tipo-problema (read)))
)
```

En esta sección planteamos las respuestas que dará nuestro sistema

```
(defrule tipo-problemas_auto
(initial-fact)
=>
(printout t "Tienes problemas con tu auto si/no?" crlf)
(assert (tipo-problema (read)))
)
```

VII. OPINIONES Y RECOMENDACIONES

El desarrollo de el problema tanto en prolog como en clips tuvieron su grado de dificultad ya que cada uno tiene diferentes sintaxis y diferentes formas de ejecución en mi opinión el lenguaje prolog fu el lenguaje más completo e fácil de entender.

como recomendaciones recomienda revisar el git donde se encuentran los archivos además se recomienda tener conocimientos previos sobre el uso de estas herramientas para mejor comprensión.

IX. CONCLUSIÓN

Se puede decir que estos lenguajes de programación al igual que muchos otros lenguajes vasados en reglas son muy útiles en el desarrollo de programas que necesiten la aplicación de reglas en el sistema. Además de apreciar que cada programa tiene claras diferencias en cuanto a lo estructural y a las posibilidades que brinda cada uno y cada uno se adapta mejor a las necesidades del desarrollador. Es importante mencionar que se realizó un sin número de prácticas que se encuentran relacionadas con cada uno de los temas teóricos.

REFERENCES

[1] Méndez, M. (2013, 11 marzo). *Sistemas expertos, clips y prolog*. es2.slideshare.net. https://es2.slideshare.net/pucese/sistemas-expertos-clips-y-prolog

[2] Pérez N, F. E. R. N. A. N. D. O. (111-11-11). *Ingenierria del conosimiento*. fdoperez.webs.ull.es. https://fdoperez.webs.ull.es/doc/clips.pdf

[3] colaboradores de Wikipedia. (2020, 17 abril). *CLIPS*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/CLIPS

[4] Giarratano, J. C. (00-01-00). *CLIPS 6.4User's Guide*. http://clipsrules.sourceforge.net. http://clipsrules.sourceforge.net/documentation/v640/ug.pdf

[5] Adame, A. (2013, 28 mayo). *Sotfware de Programacion Lógica - CLIPS, PROLOG, LISP*. https://es2.slideshare.net. https://es2.slideshare.net/ayrtonaa/sotfware-de-programacion-lgica-clips-prolog-lisp

[6] https://github.com/edison123344/sistemasExpertos.git

[7] https://sdfsfsfosdf.blogspot.com/2020/