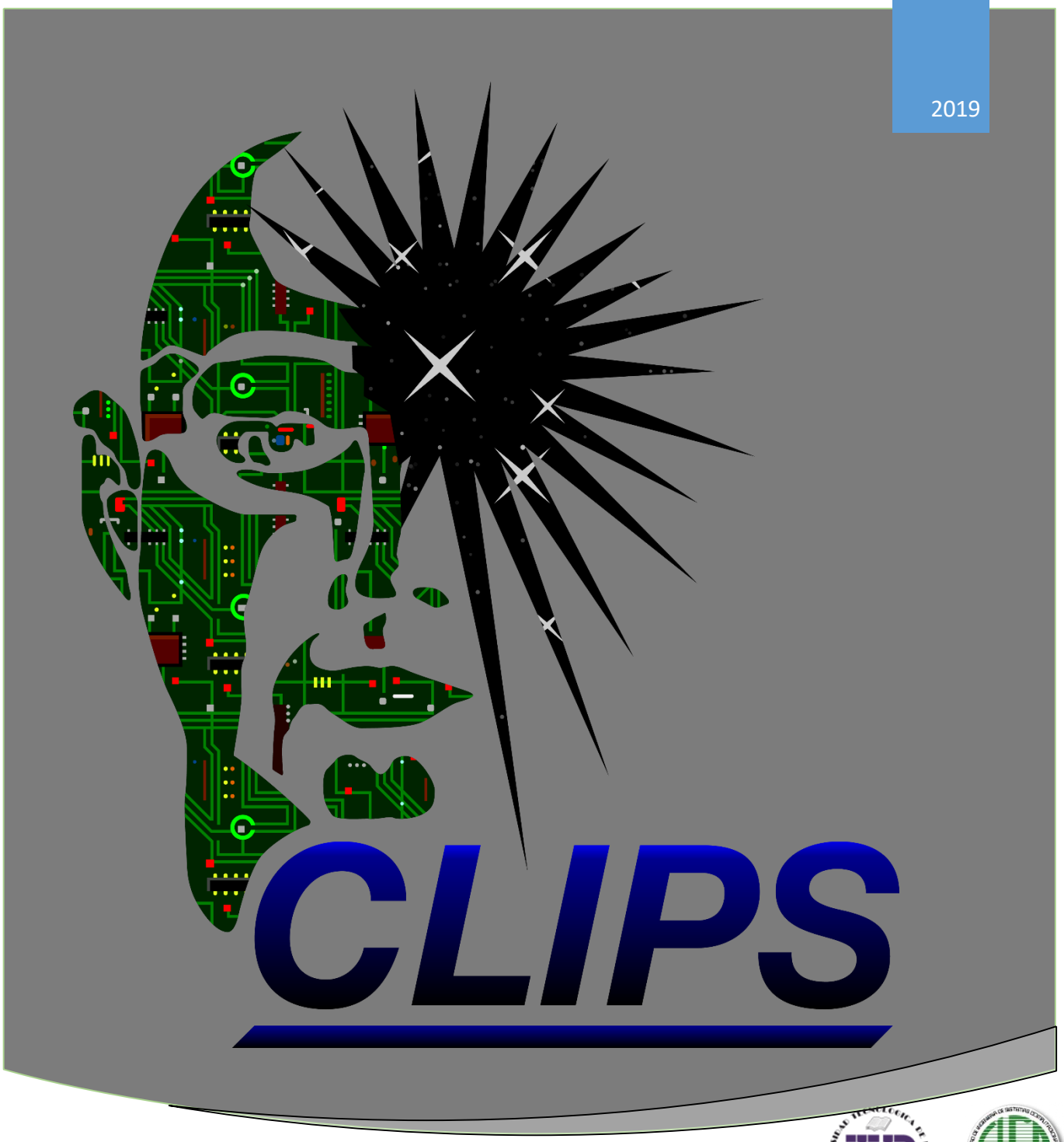
2019



JIAN KU

ÍNDICE

INTRODUCCIÓN

Los sistemas expertos son programas que reproducen el proceso intelectual de un experto humano en un campo particular, pudiendo mejorar su productividad, ahorrar tiempo y dinero, conservar sus valiosos conocimientos y difundirlos más fácilmente. Antes de la aparición del ordenador, el hombre ya se preguntaba si se le arrebataría el privilegio de razonar y pensar.

En la actualidad existe un campo dentro de la inteligencia artificial al que se le atribuye esa facultad: el de los sistemas expertos. Estos sistemas permiten la creación de máquinas que razonan como el hombre, restringiéndose a un espacio de conocimientos limitado; en teoría, pueden razonar siguiendo los pasos que seguiría un experto humano (médico, analista, empresario, etc.) para resolver un problema concreto.
Este tipo de modelos de conocimiento por ordenador ofrece un extenso campo de posibilidades en resolución de problemas y en aprendizaje. Su uso se extenderá ampliamente en el futuro, debido a su importante impacto sobre los negocios y la industria.

A través de cuatro capítulos, el objetivo de este documento es enseñar de forma practica el funcionamiento y construcción de un sistema experto con la herramientas CLIPS. Mediante su sistema de repaso al final de cada capítulo se reforzarán los conocimientos adquiridos a través de la lectura de esta guía.

CAPÍTULO 1 – HISTORIA Y FUNCIONAMIENTO DE LA HERRAMIENTA

1.1. INTRODUCCIÓN A LA HERRAMIENTA CLIPS

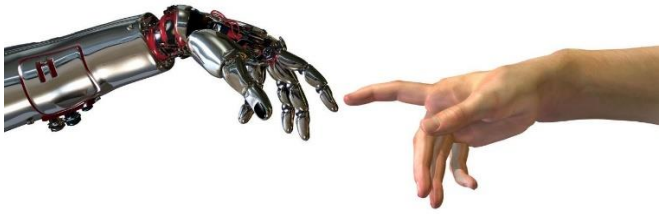
En 1984, se crea en el Lyndon B. Johnson Space Center de la Nasa una herramienta que provee un entorno de desarrollo para la producción y ejecución de sistemas expertos, la cual es conocida con el nombre de CLIPS. CLIPS es un acrónimo en inglés de Sistema de Producción Integrado en Lenguaje C.
Desafortunadamente, los fondos cesaron a principios de los años 1990, y hubo un mandato de la NASA para comprar el software comercial.

En la actualidad, entre los paradigmas de programación que soporta CLIPS se encuentran:

- ⇒ La Programación lógica
- ⇒ La Programación imperativa
- ⇒ La Programación Orientada a Objetos.

CLIPS es, probablemente, el sistema experto más ampliamente usado debido a que es rápido, eficiente y gratuito. Aunque es de dominio público, aún es actualizado y mantenido por su autor original, Gary Riley. [1]

1.2. EVOLUCIÓN DE CLIPS

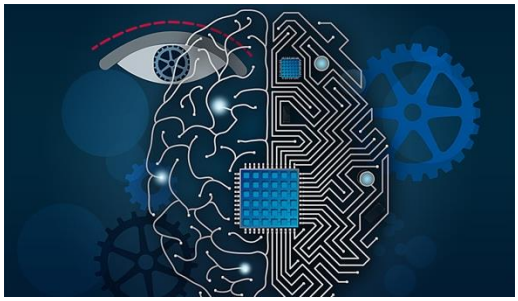


El primer prototipo de CLIPS fue desarrollado durante la primavera (boreal) de 1985, y tomó poco más de dos meses. Era compatible con todos los desarrollos hechos por la sección de Inteligencia Artificial, y su sintaxis estaba basada en la sintaxis de ART (otra herramienta para el desarrollo de sistemas expertos). Sin embargo,

CLIPS fue desarrollado sin tener acceso o haber conocido previamente el código fuente de ART.

Inicialmente, con el desarrollo de CLIPS se buscaba tener mayor conocimiento sobre la construcción de sistemas expertos y sentar las bases de un lenguaje para reemplazar las herramientas comerciales que estaban siendo usadas.

La versión 1.0 de CLIPS demostró que eso era posible. Después de un desarrollo adicional, se vio que el costo de CLIPS sería significativamente menor al de otras herramientas y que sería ideal para entrenamiento. Otro año de desarrollo y de uso interno sirvió para mejorar portabilidad, desempeño, funcionalidad y documentación de soporte. A mediados de 1986, CLIPS v3.0 estuvo disponible para grupos fuera de la NASA.



Otras funcionalidades agregadas transformaron CLIPS; ya no era una herramienta para entrenamiento sobre construcción de sistemas expertos, sino que ahora servía también para el desarrollo y ejecución de los mismos. Las versiones CLIPS v4.0 y v4.1 (1987) tenían significativas mejoras en cuanto a desempeño, integración con otros lenguajes, y capacidad de ejecución. CLIPS v4.2 (1988) fue una completa reescritura del código fuente con el fin de hacerlo más modular. Esta versión también incluyó un manual detallado de la arquitectura de CLIPS y una aplicación de ayuda para la verificación y validación de programas basados en reglas. Nuevas funcionalidades vinieron con la CLIPS v4.3 (1989).

Originalmente, CLIPS era un lenguaje de reglas basado en el Algoritmo Rete (Programación Lógica). CLIPS v5.0 (1991) introdujo dos nuevos paradigmas de programación: Programación Imperativa y Programación Orientada a Objetos (POO). El lenguaje POO dentro de CLIPS es llamado por sus siglas en inglés, COOL (Lenguaje Orientado a Objetos de CLIPS). CLIPS v5.1 (1991) ya soportaba las recientemente desarrolladas o mejoradas interfaces X Windows, MS-DOS y Macintosh. CLIPS v6.0 (1993) tenía nuevas funcionalidades relacionadas con el reconocimiento de patrones en objetos/reglas y soporte a Ingeniería de Software basada en reglas. CLIPS v6.1 (1998) soportaba compiladores C++, aunque ya no soportaba los viejos compiladores C no ANSI. También se agregaron comandos para llevar control del tiempo de desarrollo y para funciones definidas por el usuario. CLIPS v6.2 es soportada por diversos sistemas operativos, y tiene mejoras en su interfaz de desarrollo para Windows 95/98/NT y MacOS.

Actualmente, CLIPS es mantenido fuera de la NASA como software de dominio público.

1.3. FUNCIONAMIENTO DE CLIPS

Al tratarse de un sistema experto, posee al igual que otros, un lenguaje que estructura el conocimiento en hechos y permite representar reglas. Siendo los hechos, la información sobre el entorno que se usa para razonar, mientras que las reglas no son más que los elementos que permiten que el sistema evolucione, normalmente modificando hechos. Esa modificación se puede dar de dos maneras:

- Directa sobre la base de hechos almacenada
- Como consecuencia de cambios en el entorno

Por ejemplo; si una regla mueve un robot sus sensores proporcionarán hechos distintos en la siguiente lectura → Modificación como consecuencia de cambios en el entorno.

1.4. VENTAJAS Y DESVENTAJAS DE CLIPS

Ventajas	Desventajas
Modularidad Cada regla es una unidad del conocimiento que puede ser añadida, modificada o eliminada independientemente del resto de las reglas	Ineficiencia: la ejecución del proceso de reconocimiento de patrones es muy ineficiente. Opacidad: Es difícil examinar una base de conocimiento y determinar que acciones van a ocurrir.
Se puede desarrollar una pequeña porción de sistema, comprobar su correcto funcionamiento y añadirla al resto de la base de conocimiento. Uniformidad: Todo el conocimiento es expresado de la misma forma.	La división del conocimiento en reglas hace que cada regla individual sea fácilmente tratable, pero se pierde la visión global. Dificultad en cubrir todo el conocimiento: Aplicaciones como el control de tráfico aéreo implicarían una

	cantidad de reglas que no serian manejables.
Naturalidad: Las reglas son la forma natural de expresar el conocimiento en cualquier dominio de aplicación	Difícil de predecir el resultado de la competencia entre alternativas y depende mucho de los mecanismos de control.
Explicación: La traza de ejecución permite mostrar el proceso de razonamiento (watch rules)	Difícil expresar conocimiento mas estructurado y relaciones entre objetos.
Simplicidad	Decisiones basadas en información local limitada
Expresividad	Difícil implementar ciertas construcciones convencionales como ciclos y recursión.

[1]

PREGUNTAS DE REPASO, CAPÍTULO 1.	¿Qué es Clips?
	Mencione dos ventajas y dos desventajas de Clips.
	¿Qué puedo llevar a cabo con Clips?

CAPÍTULO 2 – CARACTERÍSTICAS DE CLIPS

2.1. CARACTERÍSTICAS PRINCIPALES

Si de algo destaca CLIPS ante los otros sistemas expertos del mundo es por su conjunto de características accesibles, seguras y enfocadas. [1]

2.1.1. Representación del conocimiento

CLIPS permite manejar una amplia variedad de conocimiento, soportando tres paradigmas de programación:

- ⇒ El declarativo
- ⇒ El imperativo
- ⇒ El orientado a objetos.

La programación lógica basada en reglas permite que el conocimiento sea representado como reglas heurísticas que especifican las acciones a ser ejecutadas dada una situación. La POO permite modelar sistemas complejos como componentes modulares. La programación imperativa permite ejecutar algoritmos de la misma manera que en C, Java, LISP y otros lenguajes.

2.1.2. Portabilidad

CLIPS fue escrito en C con el fin de hacerlo más por
tabla y rápido, y ha sido instalado en diversos sistemas operativos (Windows 95/98/NT, MacOS X, Unix) sin ser necesario modificar su código fuente. CLIPS puede ser ejecutado en cualquier sistema con un compilador ANSI de C, o un compilador de C++. El código fuente de CLIPS puede ser modificado en caso que el usuario lo considere necesario, con el fin de agregar o quitar funcionalidades.

WINDOWS 95 MAC OS X

2.1.3. Integralidad

CLIPS puede ser embebido en código imperativo, invocado como una sub-rutina, e integrado con lenguajes como C, Java, FORTRAN y otros. CLIPS incorpora un completo lenguaje orientado a objetos (COOL) para la elaboración de sistemas expertos. Aunque está escrito en C, su interfaz más próxima se parece a LISP. Pueden escribirse extensiones a CLIPS sobre C, y al contrario, CLIPS puede ser llamado desde C. CLIPS puede ser extendido por el usuario mediante el uso de protocolos definidos.

2.1.4. Desarrollo Interactivo

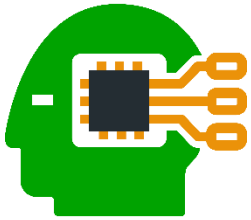
La versión estándar de CLIPS provee un ambiente de desarrollo interactivo y basado en texto; este incluye herramientas para la depuración, ayuda en línea, y un editor integrado. Las interfaces de este ambiente tienen menús, editores y ventanas que han sido desarrollados para MacOS, Windows 95/98/NT, X Windows, entre otros.

2.1.5. Verificación/Validación

A través de las características y funcionalidades que posee CLIPS se pueden realizar verificaciones o validaciones de sistemas expertos que permiten comprobar las reglas incluidas en el sistema experto que está siendo desarrollado, incluyendo diseño modular y particionamiento de la base de conocimientos del sistema, chequeo de restricciones estático y dinámico para funciones y algunos tipos de datos, y análisis semántico de reglas para prevenir posibles inconsistencias que evitarían la activación de las reglas o generarían errores.

2.1.6. Documentación

Aquí podrás encontrar una documentación especial y oficial de cada versión de la herramienta. Ya que , posee un paquete muy documentado, tanto a nivel de Manuales de Usuario como a nivel de Manuales de Referencia y de Código.



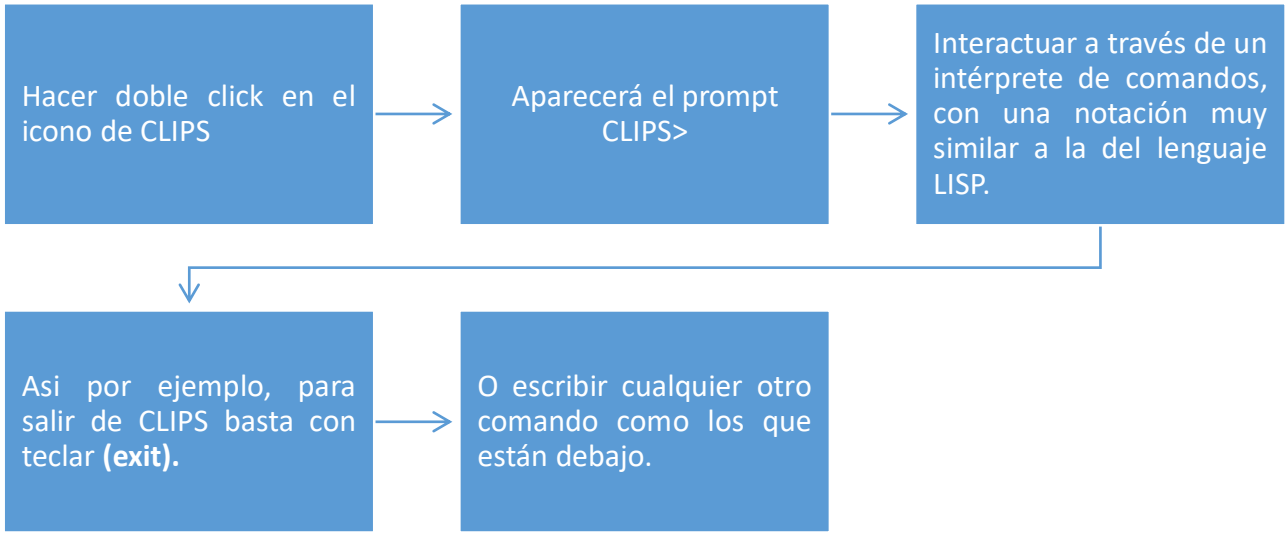
Documentación de la herramienta CLIPS.

PREGUNTAS DE REPASO, CAPÍTULO 2.	¿Porque se destaca Clips de otros expertos?
	Mencione las Características principales de esta herramienta.
	¿Cuáles son los 3 paradigmas de programación sobre los que CLIPS se apoya?

CAPÍTULO 3 - TUTORIAL DE CLIPS

3.1. TRABAJANDO CON CLIPS

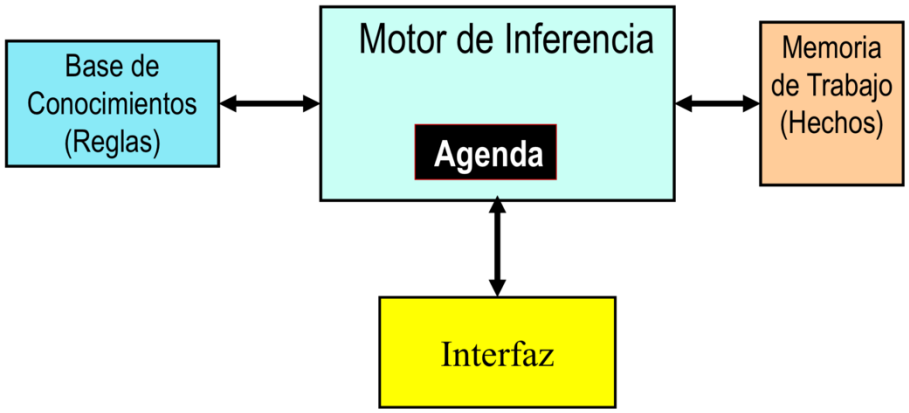
Para arrancar en CLIPS, simplemente se debe:



(exit)	Salir de CLIPS.
(clear)	Elimina datos y programas de la memoria, tal como si salieras y entraras en CLIPS.
(reset)	Elimina la información dinámica de la memoria y resetea la agenda.
(run)	Comienza la ejecución de un programa en CLIPS.

3.2. HECHOS Y REGLAS

En un su modo más sencillo, CLIPS opera manteniendo una lista de hechos **(facts)** y un conjunto de reglas **(rules)** con las que operar. Un dato curioso es que los hechos reposan almacenados en la "lista de hechos" dado que son necesarios para disparar o activar las reglas. [3]

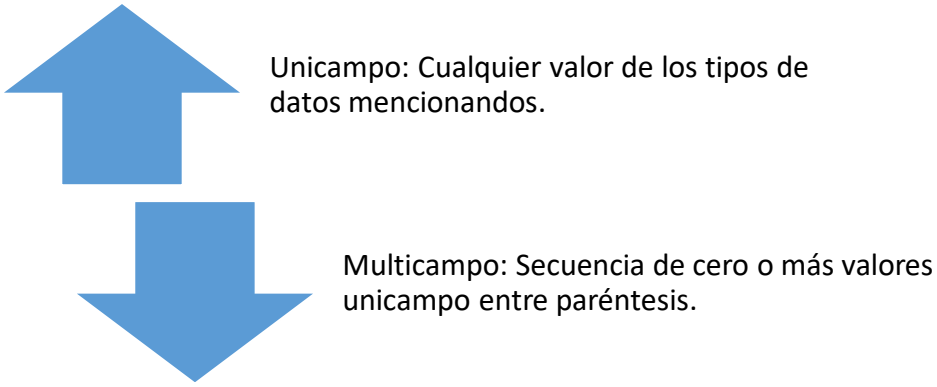


[4]

3.2.1. HECHOS

Un hecho es la forma mas sencilla de representar una parte de información o patrón tal como (**color verde**). Su campo puede ser numérico, simbólico, cadena, etc. Los espacios son utilizados para separar diversos símbolos, pero no debe ser confundido con la unión de dos espacios consecutivos. Por ejemplo: madre_de Jose. A su vez, se debe tener cuidado del uso de mayúsculas y minúsculas. [5]

Los campos de los hechos pueden ser uni o multicampo:



Por ejemplo: No es lo mismo el valor unicampo adiós que el valor multicampo (adios).

Es importante recalcar que un gran beneficio que nos brindan los hechos es la posibilidad de poder añadirse y eliminarse dinámicamente.

Para añadirlos se crean aseverando su existencia mediante el comando **assert command**. Un ejemplo, junto con la respuesta de CLIPS:

```
CLIPS> (assert (color verde))
<Fact-0>
```

Cada vez que escribamos facts tendremos acceso a la lista completa de hechos actuales. El texto **<Fact-0>** es la respuesta de CLIPS e indica que se ha creado un nuevo hecho **(fact number 0)** y ha sido añadido a la Memoria de Trabajo **(MT)**. El comando **(facts)** listará todos los hechos actuales almacenados en la MT:

```
CLIPS>(facts)
f-0 (color verde)
For a total of 1 fact.
```

Los hechos pueden ser *retractados* (eliminados) de la MT mediante el comando **retract**.

```
CLIPS>(assert (color verde))
<Fact-0>
CLIPS>(assert (color rojo))
<Fact-1>
```

Para retraerse del primer hecho es necesario conocer el *índice* de ese hecho, que es adjudicado por CLIPS, como se vio anteriormente:

```
CLIPS> (retract 0)
CLIPS>(facts)
f-1 (color rojo)
For a total of 1 fact.
```

Estos índices no se reutilizan, esto es después de borrar el hecho 0, éste no se le asignará a ningún hecho venidero.

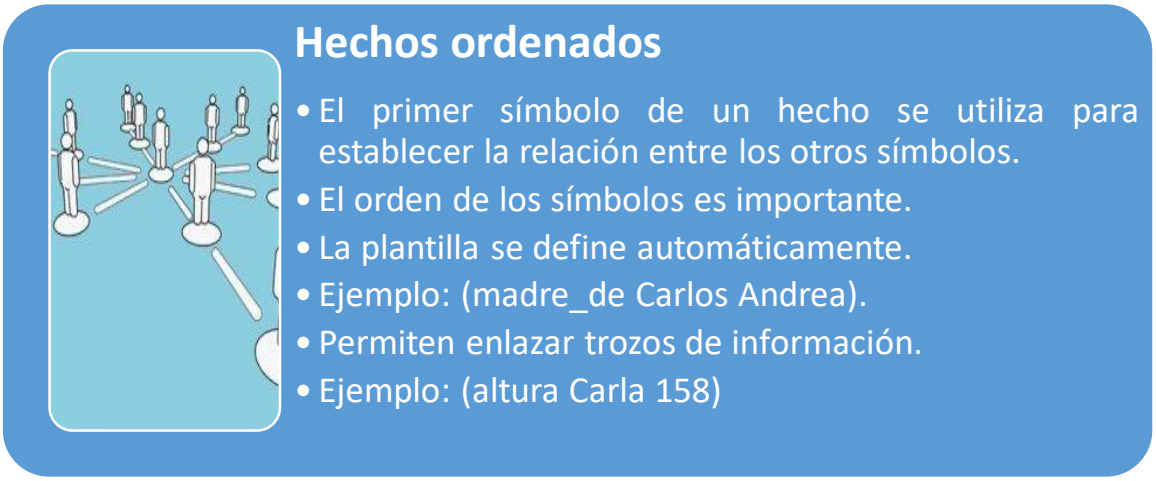
Tanto en los hechos como las reglas se recomienda introducirlos en varias lineas para obtener una mejor comprensión, sin olvidar presionar “ENTER” al acabar.

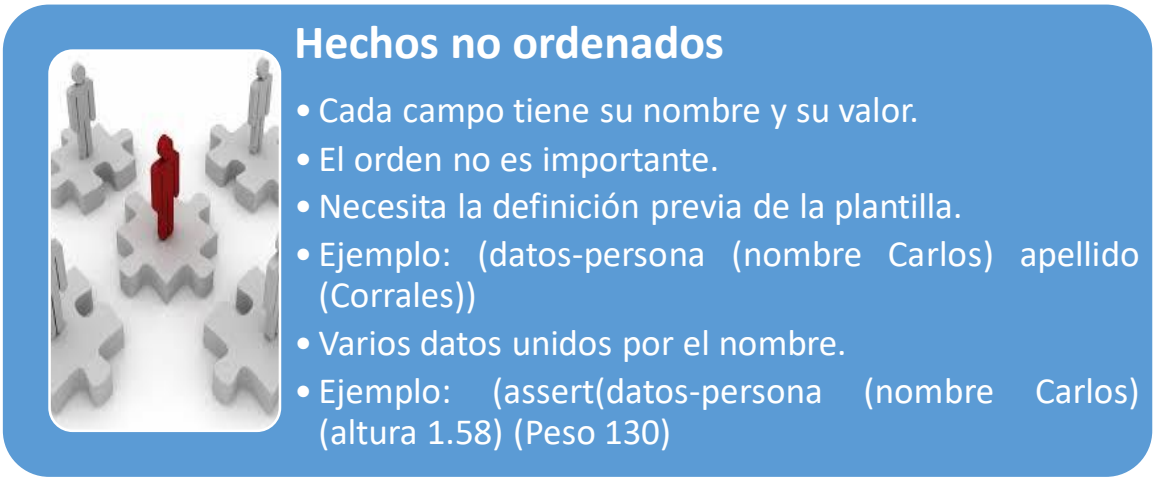
Por ejemplo:

```
CLIPS> (clear)
CLIPS>(assert(lista-color
rojo

amarillo
azul))

<Fact-0>

CLIPS>(facts)
```

```
f-0 (lista-color rojo amarillo azul)

For a total of 1 fact.
```

Los paréntesis son utiliados en los hechos para separar la información y pueden llegar a tener uno o varios símbolos.





Los hechos por sí mismo tienen una utilidad limitada.

3.2.2. REGLAS:

Ellas te permiten utilizar los hechos y con un futuro predeterminado, realizan programas de alguna utilidad. Es necesario recordarle que no puede haber dos reglas con el mismo nombre.

Muy similar a los hechos, ella también se encuentra en una especie de lista, llamada: base de conocimiento y pueden ser eliminadas. Los hechos son llamados a encajar en su lista, la cual atiende a un orden de prioridad.

En la parte izquierda de una regla pueden aparecer diferentes tipos de condiciones:

1. **Patrones Constantes:** con variables o con wildcards se instancian directamente con hechos en la base de hechos.

2. **Expresiones**: not, and, or, exist y forall con patrones.

3. **Tests de expresiones**: sobre las variables vinculadas.

La aplicación de reglas es necesaria para desarrollar un programa capaz de realizar alguna función útil. En general, una regla es expresado en la forma SI precondición ENTONCES postcondición. Esta clase de regla es conocida como *producción*, y de ahí que a CLIPS y sistemas similares se les llame *sistemas de producción*.

```
(defrule PERRO
(animal-es PERRO)
=>
(assert (sonido-es GUAU)))
```

Cada regla tiene tres partes:

La regla del ejemplo puede parafrasearse como: "si hay un hecho **(animal-es PERRO)** en la MT, entonces asevera otro hecho, **(sonido-es GUAU)**, en la MT".

ESTA PARTE ES INTERACTIVA ASI QUE PRUÉBALO.

1. **Limpia el sistema y teclea la regla tal cual.** Tecleando (rules) te dará la lista de reglas presente en la Memoria de Producción (una tan solo en este ejemplo).

2. En este momento no hay hecho alguno, así que vamos a **introducir uno, teclea (assert (animal-es PERRO))**. Si compruebas el contenido de la MT verás que sólo hay un hecho.

3. Para lanzar la regla, **teclea (run)**. Aparentemente no ocurre nada, pero si vuelves a comprobar el contenido de la MT, aparecerá un nuevo hecho, **(**sonido-es GUAU**)**, que ha sido inferido/producido por la regla.

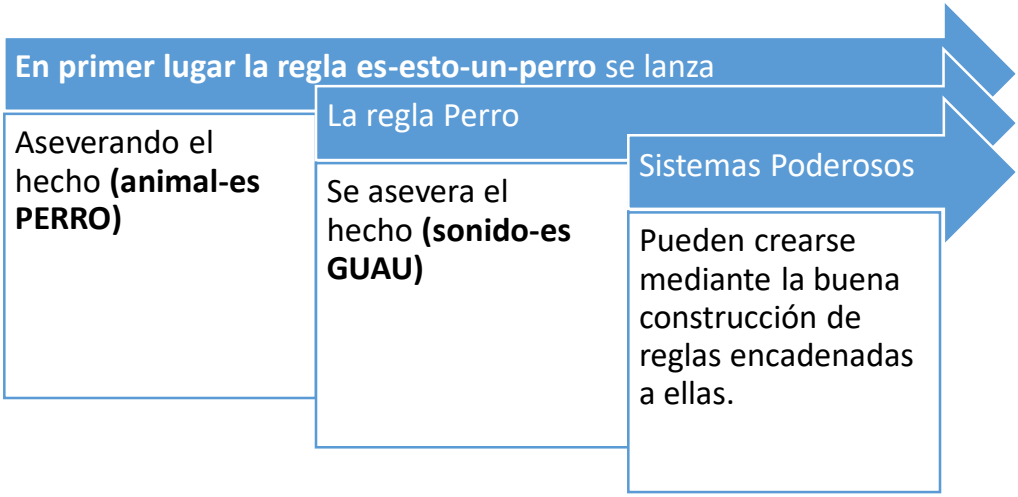
Esta es la fuerza de la programación basada en reglas: **la habilidad de hacer inferencias a partir de los datos**, siendo posible usar el conocimiento inferido para lanzar nuevas reglas.

Añade una segunda regla:

```
(defrule es-esto-un-peroo
  (animal-tiene dientes-puntiagudos)
  (animal-tiene cola-peluda)
  =>
  (assert (animal-es PERRO)))
```

Dado que esta regla tiene dos patrones, ambos deben satisfacerse para ser ejecutada la acción asociada.

Comprueba qué pasa en la MT si aseveramos los dos hechos que aparecen en el patrón de la regla. Si añades tales hechos, **(animal-tiene diente-afilados), (animal-tiene cola-peluda)** y luego ejecutas el programa (**(run)**), verás que aparecen cuatro reglas en la MT.



3.3. COMBINACIÓN DE ELEMENTOS CONDICIONALES CON AND, OR Y NOT

Para que una regla este activada todos los elementos condicionales de la precondición deben reconocer hechos de la memoria de trabajo. Por lo tanto, hasta ahora todas las reglas vistas tienen un **and** implícito entre los elementos condicionales.

CLIPS ofrece la posibilidad de representar **and** y **or** explícitos, así como la posibilidad de indicar mediante **not** que una regla será activada si no existe ningún hecho reconocido por un patrón.

La siguiente regla se activa en el caso de que no exista el hecho (animal-es pato):

```
(defrule noperoo
(not (animal-es PERRO))
=>
(assert (sonido-es noGUAU))
(printout t "Esto no es un perro" crlf))
```

***TENER EN CUENTA QUE*:** Para que **not** sea efectivo siempre, es necesario haber ejecutado en algún momento el **reset**.

Tecleando **(reset)**, entre otras cosas, se vacía la MT (las reglas están en la memoria de producción, no se toca) y se asevera el hecho **(initial-fact)**. Este hecho es el que se les presupone a las reglas cuyo patrón está vacío. y se mantiene en la MT hasta que se ejecute **(clear)** o se salga y entre de clips. En ocasiones, tal como el ejemplo

anterior, el uso de **not** conlleva que a esa regla se le añade el hecho **(initial-fact)** a su patrón, y es por ello que es necesario que en algún momento se haya invocado **(reset)** para que este tipo de reglas funcione correctamente.

3.4. DELIMITADORES DE SÍMBOLOS:

- Caracteres no imprimibles (espacio, retorno, tabulador, etc.)
- Comillad sdobles “
- Paréntesis ()
- Et &
- Barra vertical |
- Menor que, <, puede ser primer carácter de un símbolo.
- Virguililla
- Punto y coma : comienzo de comentario hasta un retorno. [5]

3.5. ALMACENAMIENTO DE REGLAS

Por convenio, los programas CLIPS tienen la extensión .CLP, si bien en estos archivos sólo se almacenan constructores (por el momento el único constructor que conocemos es defrule).

Es posible cargar uno de tales archivos mediante (load "nombre-archivo.clp").

También podemos siempre que queramos almacenar los constructores creados hasta ese momento mediante (save "nombre-archivo.clp").

Por el momento, los hechos deben ser tecleados en cada sesión.

3.6. UTILIDADES DE CLIPS

Las siguientes instrucciones te serán de utilidad:

a. **(dribble-on <nombre-fichero>), (dribble-off)**.	Equivalente a la función dribble de lisp. dribble-on registra las salidas del terminal y las entradas del teclado en el fichero especificado. dribble-off cierra el fichero.
b. **(agenda)**	Muestra las reglas instanciadas. Para cada instancia muestra la prioridad de la regla, el nombre de la regla y los hechos reconocidos por la regla.
c. **(facts)**	Muestra los hechos de la memoria de trabajo.
d. **(watch <parámetro>)**	Es útil para depurar programas. El parámetro puede ser uno de los siguientes símbolos: facts, rules, activations, statistics, compilations, focus, o all. Para la práctica te interesará utilizar (watch facts) y (watch rules), que indica con la secuencia **<==** cuando se elimina un hecho, y con la secuencia **==>**, cuando se inserta. Para que deje de dar esta información utiliza **(unwatch facts).** CLIPS> (assert (animal-es pato)) ==> f-0 (animal-es pato) <Fact-0> CLIPS> (run) FIRE 1 pato: f-0 ==> f-1 (sonido-es quack) Esto es un pato
e. **(matches <nombre-de-regla>)**.	Matches recibe como argumento el nombre de una regla y muestra los

	hechos reconocidos por cada patrón. Por ejemplo: CLIPS> (matches Estado-Inicial) Matches for Pattern 1 f-0 Activations f-0
Cualquier línea que comience con un punto y coma (;) no se interpreta. Son comentarios.	

PREGUNTAS DE REPASO, CAPÍTULO 3.	¿Cuál es la diferencia entre el hecho y la regla?
	¿Que comando utilizas para añadir un hecho y cuál para eliminarlo?
	¿Es necesario utilizar las combinaciones and, or, not? Si, No ¿Porqué?

CAPÍTULO 4 – VARIABLES Y HECHOS COMPLEJOS

4.1. HECHOS PERSISTENTES

Un gran beneficio que podemos obtener de CLIPS es la posiblidad de mantener ciertos hechos que usualmente deseamos que se conserven entre ejecución y ejecución, aún después de limpiar la memoria de trabajo. Esto puede conseguirse mediante el constructor deffacts.

Por ejemplo, si quiero aseverar que en mi BC hay un perro, un gato y un hurón, y esto quiero que conste cada vez que haga un (reset) se debe escribir:

(deffacts startup (animal perro) (animal gato) (animal hurón))

Una vez acertados (tras hacer un (reset)) estos son tres hechos más de la WS. Pueden ser eliminados de la WS, pero volverán a aparecer cuando se vuelva a ejecutar (reset).

4.2. PATRONES MÁS FLEXIBLES

Hasta ahora, los patrones usados eran más bien simples y restrictivos, ya que cada expresión lógica del patrón sólo podía ser satisfecha por un hecho. Frecuentemente interesa que una regla pueda ser lanzada por diversos hechos que satisfagan una determinada expresión. Para ello se pueden usar los comodines o *wildcards*. A modo de ejemplo, el patrón de la siguiente regla será lanzado por cualquier hecho que refiera a un animal, sea el hecho que sea:

```
(defrule animal
(animal ?)
=>
(printout t "animal found" crlf))
```

Así, si introducimos los animales de la anterior sección (haciendo un reset, si utilizamos obtenemos:

CLIPS>**(run)**
Animal found
Animal found
Animal found
Animal found
CLIPS>

IMPORTANTE: Es posible usar la “?” allá donde haga falta y tantas veces como se necesite, pero puede no ser legal poner “?” al principio de un hecho.

Por ejemplo: (child-of ? ?) es legal , pero (? ? hatchling) no lo es.

4.3. VARIABLES

4.3.1. Uso de variables

. ***ESTA PARTE ES INTERACTIVA ASI QUE PRUÉBALO***:

Para poder aprovechar todo el potencial del uso de comodines, es necesario ligar estos a una variable. La forma de ligarlos es simple:

1. Escribimos el nombre de la variable a continuación del comodín.

```
(defrule list-animales
(animal ? nombre)
=>
(printout t ? nombre "found" crlf))
```

Si probamos ahora una ejecución:

```
CLIPS>(run)
perro found
gato found
pato found
tortuga found
CLIPS>
```

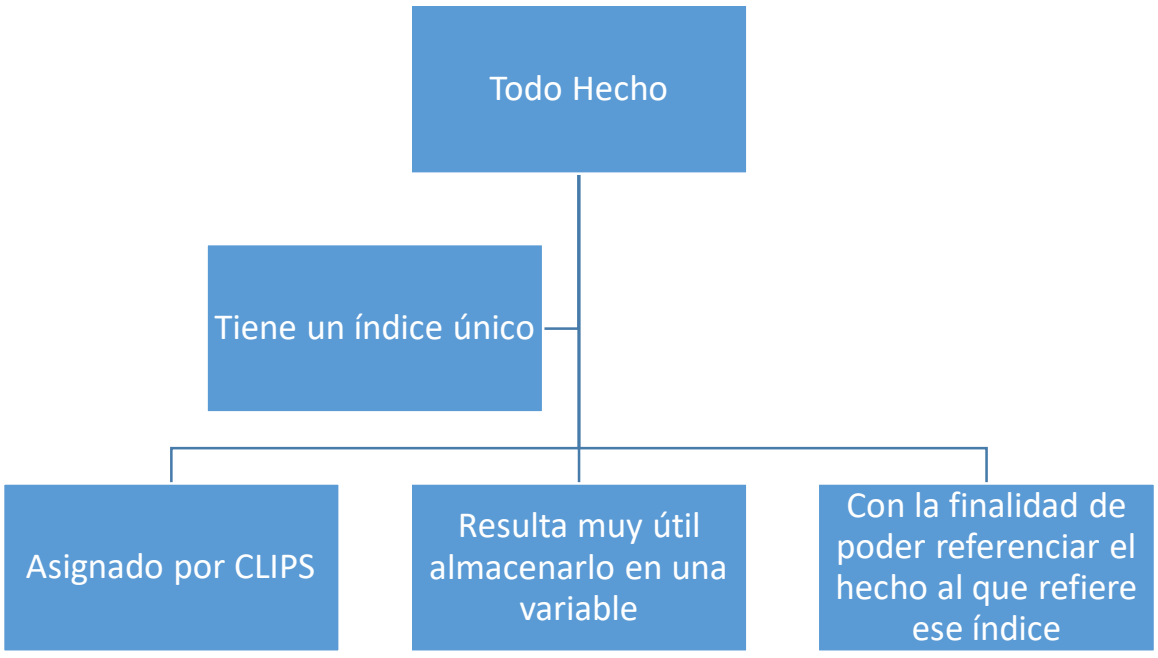
2 Conseguimos que se imprima cada uno de los cuatro hechos que casan con el patrón. Es posible utilizar tantas variables como se requiera, o utilizarla en varios sitios dentro del mismo patrón, que es lo más usual.

```
(defrule mamifero
  (animal ? nombre)
  (sangre caliente ? nombre)
  (not (pone - huevos ? nombre))
  =>
  (assert (mamífero ? name))
  (printout t ? name " es un mamífero" crlf))
```

3 Esta regla da el siguiente resultado:

```
CLIPS>(run)
perro es un mamífero
gato es un mamífero
CLIPS>
```

4.3.2. Variables para referirse a hechos



Por ejemplo, para eliminar ese hecho:

```
(defrule remove-mamifero
  ?fact <- (mamifero ?)
  =>
  (printout t "retracting " ?fact crlf)
  (retract ?fact))
```

En el patrón de esta regla, la variable ? fact almacena cada mamífero contenido en la BC (mamifero ?) . Eso es justamente lo que el operador de asignación (<-) hace. Si ejecutas esta regla obtendrás algo similar a esto (los índices pueden variar):

```
CLIPS>(run)
retracting <Fact-13>
retracting <Fact-14>
retracting <Fact-15>
retracting <Fact-16>
CLIPS>
```

Si ahora ejecutamos (facts) para listar los hechos almacenados en el WS encontraremos que no hay mamíferos.

4.3.3. Variables para almacenar expresiones

La sintaxis de CLIPS recuerda en gran medida a la de LISP, y como tal se usa notación prefija, tanto con operadores lógicos como aritméticos.

ESTA PARTE ES INTERACTIVA ASI QUE PRUÉBALO:

```
(defrule take-umbrella
(or (weather raining)
(weather snowing))
=>
(assert (umbrella required)))
```

1. Define una regla con un patrón que es una disyunción de dos hechos (si bien recuerda que denota un mejor estilo escribir dos reglas, una para cada operando).

2. De forma análoga si quieren hacen operaciones aritméticas:

```
CLIPS>(+ 5 7)
12
CLIPS> (- 5 7)
-2
CLIPS> (* 5 7)
35
CLIPS> (/ 5 7)
0.7142857142857143
CLIPS>
```

3. Prueba a reescribir 10+4*19-35/12 en notación propia de CLIPS y verifica que el resultado es 83.0833.

PREGUNTAS DE REPASO, CAPÍTULO 4.	¿A que se refieren con hechos peristentes?
	¿Para que se utilizan los comodines o wildcards?
	¿Cómo se liga un comodín a una variable?

CONCLUSIÓN

Esta guía es apta para enseñar de forma practica el funcionamiento y construcción de un sistema experto con la herramientas CLIPS , siendo esta de las mejore opciones al contar con las facilidades de ser rápida, eficiente y gratuita. El lenguaje que posee es universal y uno de los más conocidos. Esta guía ha expuesto los conocimientos claves para iniciar este recorrido de aprendizaje.

BIBLIOGRAFÍA

[1] G. Riley, «CLIPS Reference Manual,» Version 6.40 Beta, Panama, 2017.

[2] I. a. CLIPS, «Sebastián Ventura Soto,» Universidad de Córdoba, [En línea]. Available: http://www.uco.es/users/sventura/misc/TutorialCLIPS/TutorCLIPS01.htm. [Último acceso: 16 Diciembre 2019].

[3] L. D. H. Molinero, «TUTORIAL DE CLIPS,» Fac. Informática de Murcia, Murcia, España.

[4] I. a. CLIPS, «C Language Integrated Production System».

[5] E. L. CLIPS, «Javier Bejar,» [En línea]. Available: https://www.cs.upc.edu/~bejar/ia/transpas/lab/clips.pdf.