

El lenguaje CLIPS

- **CLIPS** es un lenguaje que permite **construir sistemas basados en reglas**
- Su sintaxis es tipo Lisp y expresa los componentes en forma de lista de símbolos
- Entre símbolos puede haber cualquier número de espacios en blanco
- Distingue entre mayúsculas y minúsculas
- Los símbolos pueden incluir: a-z A-Z 0-9 \$ * . = + / < > _ ? #
- No pueden empezar por: 0-9 ni \$? & (usos especiales)
- Símbolos especiales:
 - nil, TRUE, FALSE, crlf

Valores

- Los valores en CLIPS pueden ser:
 - Símbolos
 - Juan amarillo respuesta22 _ejemplo
 - Números
 - 56 47.8 5654L 6.0E4
 - Cadenas
 - “esto es un ejemplo”
 - Listas de símbolos, números, cadenas
 - (a b c d) (+ 3 5) (“Pregunta 1 “Nombre”) () (eq (3 5))
- Comentarios
 - Líneas que empiezan por ;
 - Si el comentario abarca varias líneas /* comentario */

Variables

- Cadenas que empiezan por ?
 - ?pregunta ?nombre ?edad
- **Las variables no son tipadas** aunque los valores lo sean
- Para asignar un valor a una variable se usa la función bind
 - (bind ?edad 18)
 - Las variables en las reglas se ligarán con los hechos en el proceso de matching
- Variables multivaluadas: empiezan por \$?
 - \$?apellido
- Variables globales. Se definen con la función
 - (defglobal ?*variable* = valor-por-defecto)
 - (defglobal ?*x* = 7)

A) – Hechos ordenados

- Secuencia de literales separados por espacios
 - Codifican la información según la posición
 - El primer literal suele representar una relación entre los restantes
 - Los restantes son como atributos o slots sin nombre

(convenio)
(alumnos Juan Luis Pedro)
(lista-de-la-compra pan leche arroz)
- Para incluirlos en la Base de Hechos se asertan (no se declaran)

```
(assert (alumnos Juan Luis Pedro))  
(assert (temperatura 25) )
```
- El encaje o *matching* con el LHS de una regla
 - Los literales deben estar en el mismo orden que en la regla
- Se usan para conceptos con poca información

B) - Hechos no ordenados: deftemplate

- Define un tipo de hecho, tiene varios slots (atributos) con nombre

```
(deftemplate persona
  (slot nombre (type SYMBOL))
  (multislot apellidos (type SYMBOL))
  (slot edad (type NUMBER)(default (+ 10 15)))
  (slot estado (type SYMBOL)(allowed-values soltero
    libre casado viudo)(default soltero)))
```
- Para cada slot se puede definir:
 - el tipo: type
 - (valores posibles: SYMBOL, FLOAT, INTEGER, STRING, NUMBER)
 - Valor por defecto: default (admite cualquier operación)
 - Valores permitidos: allowed-values
 - Slot multivaluados: multislot
- Se necesita establecer cada hecho con **assert**
- Se puede usar **deffacts** para hacer varios assert de los hechos iniciales de la memoria de trabajo.

1. Variables en las reglas CLIPS:

- Podemos poner **variables** en los patrones de la parte izquierda de una regla.
- El identificador de una variable comenzará por ?

CLIPS>

```
(deftemplate personaje (slot nombre) (slot ojos) (slot pelo))
```

CLIPS> (defrule busca-ojos-azules

```
    (personaje (nombre ?nombre) (ojos azules))
```

```
=>
```

```
    (printout t ?nombre " tiene los ojos azules." crlf))
```

CLIPS> (defacts gente

```
    (personaje (nombre Juan) (ojos azules) (pelo castagno))
```

```
    (personaje (nombre Luis) (ojos verdes) (pelo rojo))
```

```
    (personaje (nombre Pedro) (ojos azules) (pelo rubio))
```

```
    (personaje (nombre Maria) (ojos castagnos) (pelo negro)))
```

CLIPS> (reset)

CLIPS> (run)

Pedro tiene los ojos azules.

Juan tiene los ojos azules.

En esta regla hemos utilizado la función **printout**, que nos permite mostrar un mensaje por el dispositivo que fijemos (t = salida estándar).

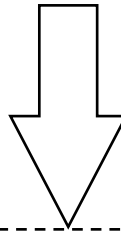
crlf es el salto de carro (observar que no pertenece a la cadena)

(defrule COMER
 (HAMBRIENTO ?PERSONA)
 (COMESTIBLE ?ALIMENTO)
->
 (assert (Come ?PERSONA el ?ALIMENTO)))

BASE de REGLAS

H1: (HAMBRIENTO PEDRO)
H2: (HAMBRIENTO PABLO)
H3: (COMESTIBLE MANZANA)
H4: (COMESTIBLE MELOCOTON)
H5: (COMESTIBLE PERA)

BASE de HECHOS



I1: (?PERSONA = PEDRO, ?ALIMENTO = MANZANA) (H1,H3)
I2: (?PERSONA = PEDRO, ?ALIMENTO = MELOCOTON (H1,H4)
I3: (?PERSONA = PEDRO, ?ALIMENTO = PERA (H1,H5)
I4: (?PERSONA = PABLO, ?ALIMENTO = MANZANA (H2,H3))
I5: (?PERSONA = PABLO, ?ALIMENTO = MELOCOTON (H2 ,H4))
I6 :(?PERSONA = PABLO, ?ALIMENTO = PERA (H2 ,H5))

2. Variables en las reglas CLIPS: Restricción por ligaduras consistentes

```
CLIPS> (undefrule *)
CLIPS>
(deftemplate busca (slot ojos))
CLIPS>
(defrule busca-ojos
  (busca (ojos ?color-ojos))
  (personaje (nombre ?nombre) (ojos ?color-ojos))
  =>
  (printout t ?nombre " tiene los ojos " ?color-ojos "." crlf))
CLIPS> (reset)
CLIPS> (assert (busca (ojos azules)))
<Fact-5>
CLIPS> (run)
Pedro tiene los ojos azules.
Juan tiene los ojos azules.
```


C) - Gestión de la Memoria de Trabajo (MT)

- (deffacts ...) define un conjunto de hechos iniciales que se cargan en la MT al hacer (reset)
- (assert <hecho>) añade hecho a la MT
- (retract <índice-hecho>) elimina hecho de la MT
- (facts) lista los hechos existentes en la MT
- (clear) elimina todos los hechos de la MT
- (reset)
 - elimina todos los hechos de la MT y las activaciones de la agenda
 - añade initial-fact y los hechos definidos con deffacts
 - añade las variables globales con su valor inicial
 - selecciona el módulo main

Estructura básica de un programa en CLIPS

; Primero la definición de plantillas

(deftemplate ...)

...

; Definición de hechos iniciales

(deffacts ...)

...

; definición de reglas

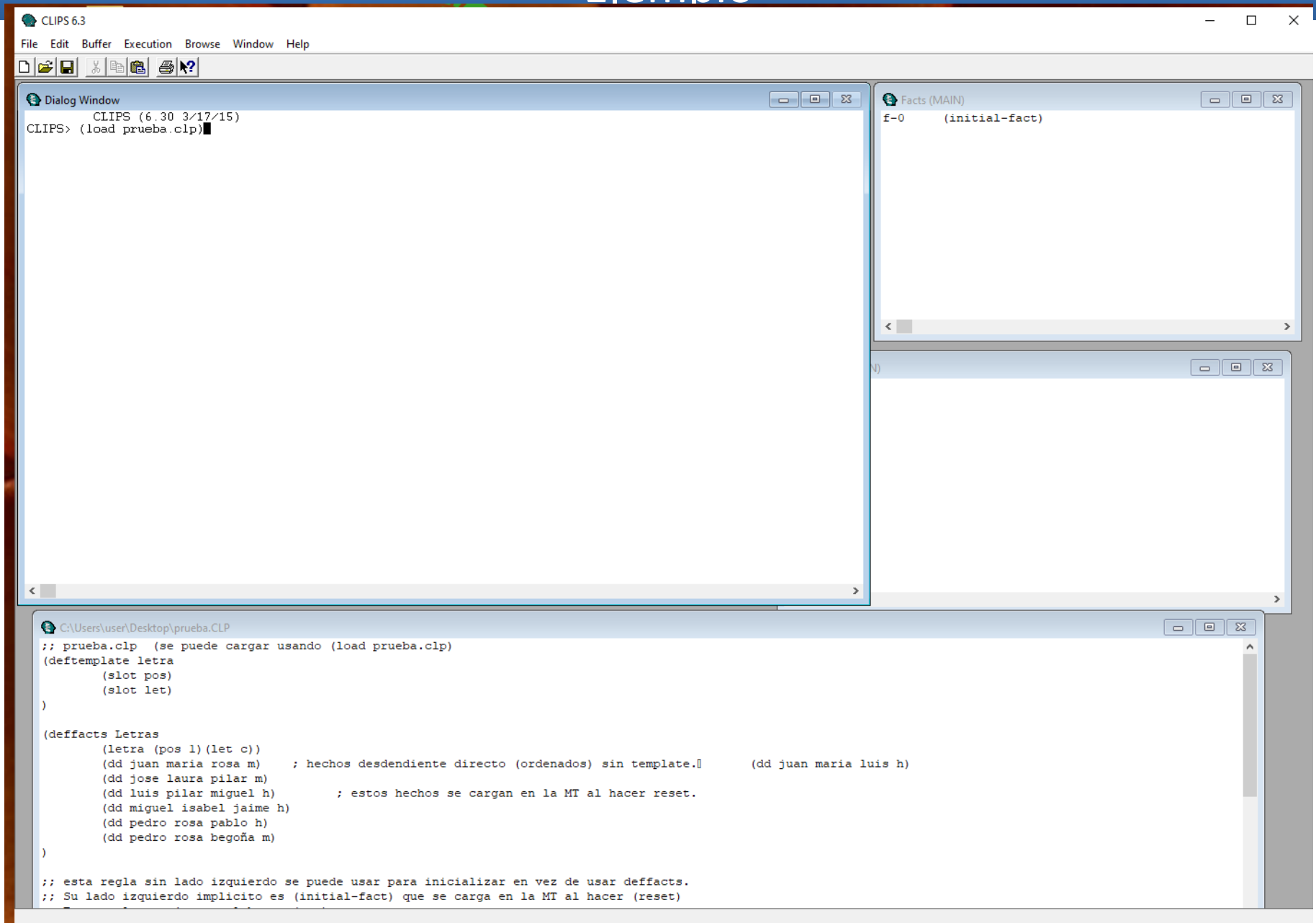
(defrule ...)

...

(reset) ; Inicializamos y cargamos hechos iniciales

(run) ; Ejecutamos el motor de inferencia.

Ejemplo



prueba.clp

```

;; prueba.clp (se puede cargar usando (load prueba.clp))
(deftemplate descendiente_directo
  (slot ascendiente1)
  (slot ascendiente2)
  (slot descendiente)
  (slot sexo)
)

(deffacts Letras
  (dd juan maria rosa m) ; hechos descendiente directo (ordenados) sin template.
  (dd juan maria luis h)
  (dd jose laura pilar m)
  (dd luis pilar miguel h) ; estos hechos se cargan en la MT al hacer reset.
  (dd miguel isabel jaime h)
  (dd pedro rosa pablo h)
  (dd pedro rosa begoña m)
  (descendiente_directo (ascendiente1 juan) (ascendiente2 maria)(descendiente luis) (sexo h))
  ;; usando hechos con templates.
)

;; esta regla sin lado izquierdo se puede usar para inicializar en vez de usar deffacts.
;; Su lado izquierdo implícito es (initial-fact) que se carga en la MT al hacer (reset)
;; Esta regla se ejecuta al hacer (run)

(defrule inicializar
  =>
  (assert (dd paco Lorena h))
)

```

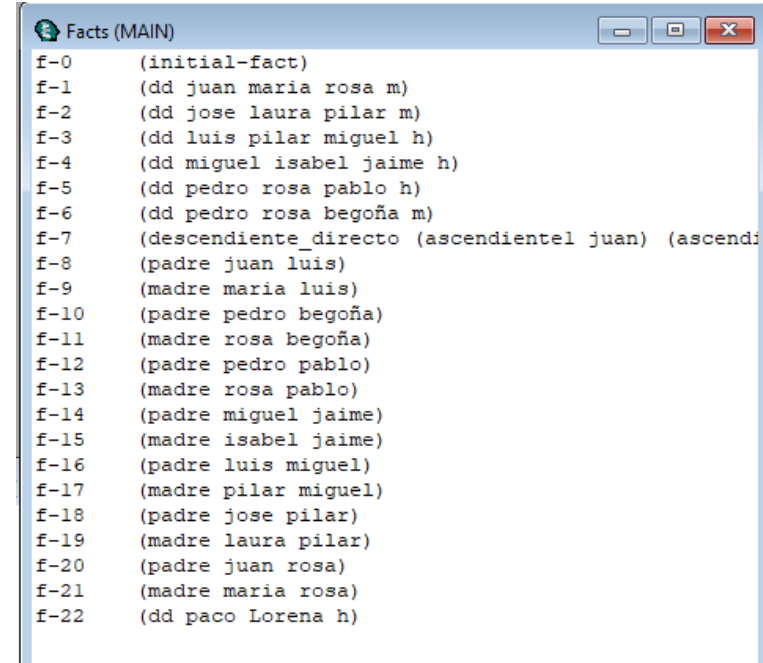
;; La siguiente regla (sintaxis CLIPS) permite asertar en la base de hechos predicados de la forma (padre Juan Rosa) o (madre María Rosa) a partir de los hechos iniciales dd.

```

(defrule padres
  (dd ?x ?z ?y ?)
  =>
  (assert (padre ?x ?y))
  (assert (madre ?z ?y)))

(defrule padres_bis
  (descendiente_directo (ascendiente1 ?x) (ascendiente2 ?z) (descendiente ?y) (sexo ?))
  =>
  (assert (padre ?x ?y))
  (assert (madre ?z ?y)))

```



```

Facts (MAIN)
f-0      (initial-fact)
f-1      (dd juan maria rosa m)
f-2      (dd jose laura pilar m)
f-3      (dd luis pilar miguel h)
f-4      (dd miguel isabel jaime h)
f-5      (dd pedro rosa pablo h)
f-6      (dd pedro rosa begoña m)
f-7      (descendiente_directo (ascendiente1 juan) (ascendiente2 maria)(descendiente luis) (sexo h))
f-8      (padre juan luis)
f-9      (madre maria luis)
f-10     (padre pedro begoña)
f-11     (madre rosa begoña)
f-12     (padre pedro pablo)
f-13     (madre rosa pablo)
f-14     (padre miguel jaime)
f-15     (madre isabel jaime)
f-16     (padre luis miguel)
f-17     (madre pilar miguel)
f-18     (padre jose pilar)
f-19     (madre laura pilar)
f-20     (padre juan rosa)
f-21     (madre maria rosa)
f-22     (dd paco Lorena h)

```



```
Dialog Window
CLIPS (6.30 3/17/15)
CLIPS> (load prueba.clp)
Defining deftemplate: letra
Defining deffacts: Letras
Defining defrule: inicializar +j+j
==> Activation 0      inicializar: *
Defining defrule: padres +j+j
TRUE
CLIPS> (reset)
<== Activation 0      inicializar: *
<== f-0      (initial-fact)
==> Activation 0      inicializar: *
==> f-0      (initial-fact)
==> f-1      (letra (pos 1) (let c))
==> f-2      (dd juan maria rosa m)
==> Activation 0      padres: f-2
==> f-3      (dd jose laura pilar m)
==> Activation 0      padres: f-3
==> f-4      (dd luis pilar miguel h)
==> Activation 0      padres: f-4
==> f-5      (dd miguel isabel jaime h)
==> Activation 0      padres: f-5
==> f-6      (dd pedro rosa pablo h)
==> Activation 0      padres: f-6
==> f-7      (dd pedro rosa begoña m)
==> Activation 0      padres: f-7
CLIPS>
CLIPS> (run 1)
FIRE      1 padres: f-7
==> f-8      (padre pedro begoña)
==> f-9      (madre rosa begoña)
CLIPS>
```

```
Facts (MAIN)
f-0      (initial-fact)
f-1      (letra (pos 1) (let c))
f-2      (dd juan maria rosa m)
f-3      (dd jose laura pilar m)
f-4      (dd luis pilar miguel h)
f-5      (dd miguel isabel jaime h)
f-6      (dd pedro rosa pablo h)
f-7      (dd pedro rosa begoña m)

f-8      (padre pedro begoña)
f-9      (madre rosa begoña)
```

```
Agenda (MAIN)
0      padres: f-7
0      padres: f-6
0      padres: f-5
0      padres: f-4
0      padres: f-3
0      padres: f-2
0      inicializar: *
```

```
C:\Users\user\Desktop\prueba.CLP
;; esta regla sin lado izquierdo se puede usar para inicializar en vez de usar deffacts.
;; Su lado izquierdo implicito es (initial-fact) que se carga en la MT al hacer (reset)
;; Esta regla se ejecuta al hacer (run)

(defrule inicializar
=>
(assert (letra (pos 2) (let b)))
(assert (dd paco Lorena h))
)

;; La siguiente regla (sintaxis CLIPS) permite asertar en la base de hechos predicados de la forma (padre Juan Rosa) o (madre Maria Rosa) a partir de los hechos
iniciales dd.

(defrule padres
(dd ?x ?z ?y ?)
=>
(assert (padre ?x ?y))
(assert (madre ?z ?y)))
```

Crear hechos iniciales con deffacts

● Ejemplo

```
(deffacts alumnos "mi clase"      /* -- hechos iniciales --*/
  (persona (nombre Pepe)(apellidos Gomez Garcia))
  (persona (nombre Juan)(edad 25)))
```

→ No olvidar hacer:

```
(reset) /*- borra todos los hechos de MT, añade los deffacts */
(facts) /*---- lista los hechos actuales en la M.T. --*/
f-0    (MAIN::initial-fact)
f-1    (MAIN::persona (nombre Pepe) (edad 25) (estado soltero) (apellidos Gomez
Garcia))
f-2    (MAIN::persona (nombre Juan) (edad 25) (estado soltero) (apellidos ))
```

● Si quiero volver a ejecutar deffacts, debo ejecutar reset de nuevo

Etiquetas temporales

- Son índices relativos al orden de creación de hechos
- $f - 0$ es el initial-fact, creado automáticamente por CLIPS
- Al resto de hechos se les van asignando índices sucesivos:
 - $f - 1$
 - $f - 2, \dots$
- Identifican de forma única cada hecho
- Cuando se elimina un hecho nunca se reasigna el índice a otro hecho
- Cuando se modifica un hecho se mantiene el mismo índice

Reglas

Sintaxis:

```

(defrule <nombre-regla>           ;; para eliminar : undefrule
[<documentación opcional>]       ;; Se pone entre " "
[(declare (salience <num>))]    ;; prioridad de ejecución
(patrón 1)
(patrón 2)
...
(patrón N)
=>
(acción 1)
(acción 2)
...
(acción M)
)

```

- Ver el contenido de una regla (ppdefrule calcular-precio)
- Una regla sin LHS se ejecuta solo cada vez que se ejecute el **reset**.
 - Reglas de inicialización

Parte izquierda de las reglas: Patrones

- La parte izquierda de las reglas suele incluir patrones:
 - Variables (?edad)
 - Variables anónimas (comodines, no importa su valor)
 - ? Se equipara con un valor
 - \$? Se equipara con múltiples valores
 - Expresiones con variables y conectivas lógicas
 - not (~), and (&), or (|)
 - Test de expresiones lógicas (test (< ?x 18))
 - Condiciones complejas precedidas de :
 - (persona (edad ?x&: (> ?x 18)))
- Las condiciones de la parte izquierda de una regla están implícitamente conectadas con and. Si necesitamos un or entonces hay que dividir la regla en dos.

Parte derecha de las reglas: Acciones

- Son acciones implícitamente conectadas con **and**
- Tipos de acciones:
 - Crear un hecho (**assert**)
 - Eliminar un hecho (**retract**)
 - Modificar un hecho (**modify**)
 - Llamar a una función
 - Asignar un valor a una variable (**bind**)
 - Entrada / Salida (**printout**, **read**, **readline**)
 - Parar la ejecución (**halt**)

Regla con variables

- Permite usar una regla con diferentes objetivos:
 - busco: que cumplan cierto estado y tengan menos de 30 años
- Al repetir `?est` fuerza que coincida su valor en los hechos que equiparen

```
(deftemplate busca (slot estado)) ;; un hecho para indicar qué busco
(defrule busca-candidato
  (busca (estado ?est)) ;; persona y busca han de tener el mismo ?est
  ?candidato <- (persona (nombre ?nom) (edad ?ed) (estado ?est))
                    ;; asigno un hecho a la variable ?candidato
  (test (< ?ed 30)) ;; solo ejecuto regla si cumple test
=>
  (modify ?candidato (estado libre)) ;; cambia el estado de candidato
  (assert (tengo candidato))
  (printout t "mi candidato es: " ?nom " estado anterior: " ?est crlf) )

(reset)
(assert (busca (estado soltero))) ;; decido buscar candidatos solteros
(run)
```

Regla con variables

- Ligar un hecho a una variable
- Usando el operador `<-` almacenamos las direcciones de los hechos en una variable

```
CLIPS> (deftemplate persona (slot nombre) (slot direccion))
CLIPS> (deftemplate cambio (slot nombre) (slot direccion))
CLIPS> (defrule procesa-informacion-cambios
  ?h1 <- (cambio (nombre ?nombre) (direccion ?direccion))
  ?h2 <- (persona (nombre ?nombre))
  =>
  (retract ?h1)
  (modify ?h2 (direccion ?direccion)))
CLIPS> (defacts ejemplo
  (persona (nombre "Pato Donald") (direccion "Disneylandia"))
  (cambio (nombre "Pato Donald") (direccion "Port Aventura")))
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-3      (persona (nombre "Pato Donald") (direccion "Port Aventura"))
For a total of 2 facts.
```

Ejemplo modificación hecho

- Ejecución condicional : para poner apellidos a quien no tenga

```
(defrule poner-apellidos ;; solo equiparan personas sin apellidos
  ?persona <-(persona (nombre ?nombre) (apellidos))
=>
  (printout t crlf "Introduce apellidos para " ?nombre ": " )
  (modify ?persona (apellidos (read))))
```



Busco los hechos que encajan y me quedo con su índice

Regla con variables

- **Variables locales y comodines**
- Podemos especificar en un patrón un comodín para sustituir un campo (?) o para sustituir más de un campo (\$?).
- Variable que no liga valor y reconoce cualquier valor: ?

```
CLIPS> (clear)
CLIPS> (deftemplate persona (multislot nombre) (slot dni))
CLIPS>
(deffacts ejemplo
  (persona (nombre Jose L. Perez) (dni 22454322))
  (persona (nombre Juan Gomez) (dni 23443325)))
CLIPS>
(defrule imprime-dni
  (persona (nombre ? ? ?Apellido) (dni ?dni))
  =>
  (printout t ?dni " " ?Apellido crlf))
CLIPS> (reset)
CLIPS> (run)
22454322 Perez
```

- Una regla puede dispararse varias veces con distintos hechos → más de un patrón
- (defrule misma-edad-diferente-nombre
 ?person1 <- (person)
 ?person2 <- (person (age ?person1.age) (lastName
 ~?person1.lastName))
=>
 (printout t "Se encontraron personas de " ?person1.age " años." crlf))

Ejemplo

```
:: EJEMPLO DE VARIABLES/COMODINES *
```

```
(def facts profesores
  (profesor Gregorio Fernández Fernández)
  (profesor Mercedes López Pérez)
  (profesor José Carlos González Cristóbal)
  (profesor Juan Ramón Velasco Pérez))

(def rule apellidos-iguales
  (profesor ?x ?y ?y)
  =>
  (printout t "D." ?x " tiene los apellidos iguales" crlf) ); no
  poner retorno en cadenas

(def rule buenos-dias
  (profesor ? ?x ?)
  =>
  (printout t "Buenos días, Sr./Sra." ?x crlf))

(def rule buenas-tardes
  (profesor $? ?x ?)
  =>
  (printout t "Buenas tardes, Sr./Sra." ?x crlf))
```


Regla con variables

- Comodín que reconoce cero o más valores de un atributo multivalor: $\$?$

CLIPS>

```
(defrule imprime-dni
  (persona (nombre  $\$?$ nombre ?Apellido) (dni ?dni))
=>
  (printout t ?dni "("  $\$?$ nombre ")" ?Apellido crlf))
```

CLIPS> (reset)

CLIPS> (run)

23443325 (Juan) Gomez

22454322 (Jose L.) Perez

- Operadores lógicos en patrones

- negación (operador ~): `(color ~rojo)`

- `(persona (nombre ?nombre) (pelo ~rubio))`

- disyunción (operador |): `(color rojo|amarillo)`

- `(persona (nombre ?nombre) (pelo castagno | pelirojo))`

- conjunción (operador &): `(color rojo&amarillo)`

Se puede utilizar en combinación con los anteriores para ligar valor a una variable

```
(defrule pelo-castagno-o-rubio
```

```
  (persona (nombre ?nombre) (pelo ?color&rubio|castagno))
```

```
  =>
```

```
  (printout t ?nombre " tiene el pelo " ?color crlf))
```

- Ejemplo:

`(habitación (plazas-libres ~0))`

`(habitación (plazas-libres 1|2|3))`

`(habitación (plazas-libres ~0 & ~4))`

`(habitación (plazas-libres ?p & ~0))`

`(habitación (plazas-libres ?p & 1|2))`

Restricciones sobre el valor de un atributo

- También podemos unir patrones con las relaciones lógicas or, and y not (por defecto, los patrones se unen con and).

```
(defrule no-cruzar
  (luz ~verde)
=>
(printout t "No cruce" crlf))

(defrule precaucion
(luz amarilla|intermitente)
=>
(printout t "Cruce con precaución" crlf))

(defrule regla-imposible
(luz verde&roja)
=>
(printout t "¡¡MILAGRO!!" crlf))

(defrule regla-tonta
(luz verde&~roja)
=>
(printout t "Luz verde" crlf)
) ; Se dispara con (luz verde)
```

```
(defrule precaucion
(luz ?color&amarillo|intermitente)
=>
(printout t "Cuidado luz " ?color
crlf))

(defrule no-cruzar
  (estado caminando)
  (or (luz roja)
      (policia dice no cruzar)
      (not (luz ?))) ; sin luz
=>
(printout t "No cruzar" crlf))
```

- **Predicados** sobre el valor de un atributo (precedidos por :)

```
(defrule mayor-de-edad
  (persona (nombre $?nombre) (edad ?edad&:(> ?edad 18))
  =>
  (printout t ?nombre " es mayor de edad. Edad:" ?edad crlf))
```

- Predicados sobre el valor de un atributo (igualdad)

```
(defrule mayor-de-edad
  (persona (nombre $?nombre) (edad =(+ 18 2))
  =>
  (printout t ?nombre
    " tiene dos años sobre la mayoría de edad." crlf))
```

- Asignación de valores a una variable: Función **Bind** (liga un valor no obtenido mediante reconocimiento)

```
- Sintaxis: (bind <variable> <valor>)
- Ejemplo:
(defrule suma
  (numeros ?x ?y)
  =>
  (bind ?r (+ ?x ?y))
  (assert (suma-es ?r))
  (printout t ?x " + " ?y " = " ?r crlf)
)
```

Comprobación de valores de los patrones

- Tenemos dos posibilidades:
 - (**test** <función-booleana> <arg>)
 - ?variable&:(<función-booleana> <arg>)
- Funciones booleanas:
 - lógicas: or, not, and
 - Comparación numérica: =,<>, >=, >, <=, <
 - cualquier tipo: eq, neq
 - funciones predicado: lexemep, stringp, numberp, evenp, symbolp, ...
- Ejemplo comprobación patrones:

```
(defrule mes-valido
(entrada ?numero)
(test (and (>= ?numero 1)(<= ?numero 12)))
=> (printout t "Mes válido" crlf))
(defrule mes-valido-equiv
(entrada ?numero&:(and (>= numero 1)(<= ?numero
12)))
=> (printout t "Mes válido" crlf))
```
- Cuando hay más de una variable que queremos comprobar en el patrón/patrones, utilizar **test**

Variables globales

- Permiten almacenar valores accesibles en reglas y funciones. Útiles para resultados.

- Se definen con `defglobal` y se eliminan con `undefglobal` (sin `?` ni `*`)

- Ejemplo:

```
(defglobal
?*num* = 3
?*suma* = (+ ?*num* 2)
?*cadena* = "hola"
?*lista* = (create$ a b c))
```

- Pueden estar en la parte izda de las reglas si no son utilizadas para asignar un valor, y su cambio no activa las reglas.

- No pueden utilizarse como parámetros de funciones ni métodos

```
(defrule regla-ilegal
(hecho ?*x*) =>)
(defrule regla-legal
(hecho ?y&:(> ?y ?*x*)) =>)
```

- Se puede visualizar el valor introduciendo su nombre en el prompt o:
 - `(show-defglobals [<nombre-modulo>])`

Estrategias de resolución de conflictos

- El motor de inferencia de CLIPS disparará las reglas aplicables por orden decreciente de prioridad (**salience**).
- Si hay varias reglas aplicables con la misma prioridad (o no se han definido prioridades) CLIPS utilizará **por defecto** la estrategia LIFO (**depth**): **disparar antes las reglas activadas más recientemente**.
- La estrategia FIFO (**breadth**) dispara las reglas de igual prioridad en el orden en que han sido activadas. **Primero las que antes se activaron**.
- (**set-strategy <estrategia>**) permite seleccionar la estrategia que utilizará el intérprete de CLIPS (**depth** o **breadth**)

El Motor de Inferencias en CLIPS

- El motor de inferencias trata de emparejar la lista de hechos con los patrones de las reglas.
- Si todos los patrones de una regla están emparejados se dice que dicha regla está **activada**.
- La **agenda** es la lista de todas las reglas activadas (es decir, cuyas condiciones han sido satisfechas y que todavía no han sido ejecutadas)
- Las activaciones se mantienen en la agenda, en que se disponen por orden de prioridad. La agenda se comporta como una pila en la que la regla en la cima será la primera en ser ejecutada.
- Cuando una regla es activada, se coloca en la agenda según los siguientes criterios:
 - Las reglas más recientemente activadas se colocan encima de las reglas con menor prioridad (valor **salience**), y debajo de las de mayor prioridad.
 - Entre reglas de la misma prioridad, se emplea la **estrategia de resolución de conflictos** fijada.
 - Si varias reglas son activadas por la aserción de los mismos hechos, y no se puede determinar su orden en la agenda según los criterios anteriores, se insertan de forma arbitraria.

Estrategias de resolución de conflictos en CLIPS

- CLIPS ofrece siete estrategias de resolución de conflictos: depth, breadth, simplicity, complexity, lex, mea y random.
- La estrategia actual se establece usando el comando **setstrategy** (que reordenará la agenda en función de la nueva estrategia).
- **Estrategia en profundidad (Depth)**
 - Es la estrategia por defecto.
 - Las reglas más recientemente activadas se colocan **sobre** todas las reglas de la agenda que tengan la misma prioridad.
 - Ejemplo:
 - El hecho A activa las reglas R1 y R2
 - El hecho B activa las reglas R3 y R4
 - Si A es asertado antes de B, las reglas R3 y R4 se colocan sobre R1 y R2 en la agenda.
 - La posición relativa de R1 y R2 (resp. R3 y R4) es arbitraria.
- **Estrategia en anchura (Breadth)**
 - Las reglas más recientemente activadas se colocan **debajo** de todas las reglas de la agenda que tengan la misma prioridad.
 - Ejemplo:
 - El hecho A activa las reglas R1 y R2
 - El hecho B activa las reglas R3 y R4
 - Si A es asertado antes de B, las reglas R1 y R2 se colocan sobre R3 y R4 en la agenda.

Estrategias de resolución de conflictos en CLIPS

- Estrategia de simplicidad/complejidad

- El criterio de ordenación es la especificidad de la regla, esto es, el número de comparaciones que deben realizarse en el antecedente. La siguiente regla tiene una especificidad de 5.

```
defrule example (item ?x ?y ?x)
                  (test(and(numberp ?x)(> ?x (+ 10 ?y))(< ?x 100))) =>
```

- **Estrategia de simplicidad**

Entre reglas de la misma prioridad, las reglas más recientemente activadas se colocan encima de las reglas con la misma o **mayor** especificidad.

- **Estrategia de complejidad**

Entre reglas de la misma prioridad, las reglas más recientemente activadas se colocan encima de las reglas con la misma o **menor** especificidad.

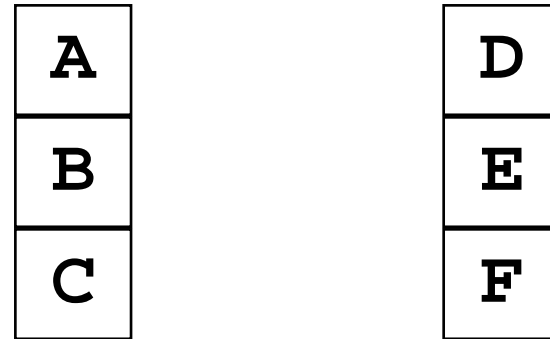
- Estrategia aleatoria:

- A cada activación se le asigna un número aleatorio para determinar su orden en la agenda. A las activaciones de la agenda se le asigna el mismo número en diferentes ejecuciones de la misma estrategia.

- Estrategia LEX

- Estrategia MEA

Ejemplo mundo de bloques

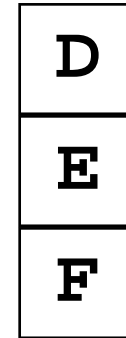
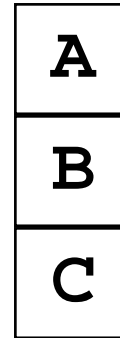


Representación sin templates

```
(def facts estado-inicial
  (bloque A) (bloque B) (bloque C)
  (bloque D) (bloque E) (bloque F)
  (estado nada esta-encima-del A)(estado A esta-encima-del B)
  (estado B esta-encima-del C)(estado C esta-encima-del suelo)
  (estado nada esta-encima-del D)(estado D esta-encima-del E)
  (estado E esta-encima-del F)(estado F esta-encima-del suelo)
  (objetivo C esta-encima-del E))
```

Mundo de bloques

- **Enunciado: Objetivo Poner C Encima de E**



❑ Otra representación

```
(def facts estado-inicial
  (pila A B C)
  (pila D E F)
  (objetivo C esta-encima-del E))
```

```

;;; Regla mover-bloque-sobre-bloque
;;; SI el objetivo es poner el objeto X encima del objeto Y y
;;; tanto X como Y son bloques y
;;; no hay nada encima del bloque X ni del bloque Y
;;; ENTONCES
;;; colocamos el bloque X encima del bloque Y y
;;; actualizamos datos

(defrule mover-bloque-sobre-bloque
  ?objetivo <- (objetivo ?objeto-1 esta-encima-del ?objeto-2)
  (bloque ?objeto-1)
  (bloque ?objeto-2)
  (estado nada esta-encima-del ?objeto-1)
  ?pila-1 <- (estado ?objeto-1 esta-encima-del ?objeto-3)
  ?pila-2 <- (estado nada esta-encima-del ?objeto-2)
=>
  (retract ?objetivo ?pila-1 ?pila-2)
  (assert (estado ?objeto-1 esta-encima-del ?objeto-2))
  (assert (estado nada esta-encima-del ?objeto-3))
  (printout t ?objeto-1 " movido encima del " ?objeto-2 "." crlf))

```

```
;;; Regla mover-bloque-al-suelo
;;; SI el objetivo es poner el objeto X al suelo
;;;     X es un bloque
;;;     no hay nada encima de X
;;; ENTONCES
;;;     movemos X al suelo y
;;;     actualizamos datos
```

```
(defrule mover-bloque-al-suelo
  ?objetivo <- (objetivo ?objeto-1 esta-encima-del suelo)
  (bloque ?objeto-1)
  (estado nada esta-encima-del ?objeto-1)
  ?pila <- (estado ?objeto-1 esta-encima-del ?objeto-2)
=>
  (retract ?objetivo ?pila)
  (assert (estado ?objeto-1 esta-encima-del suelo))
  (assert (estado nada esta-encima-del ?objeto-2))
  (printout t ?objeto-1 " movido encima del suelo. " crlf))
```

Agenda (MAIN)

```
0      ascendiente: f-12
0      hija: f-7
0      hermana: f-7,f-6
0      hermano: f-6,f-7
0      padre: f-6
0      madre: f-6
0      hijo: f-6
0      padre: f-5
0      madre: f-5
0      hijo: f-5
0      padre: f-4
0      madre: f-4
0      hijo: f-4
0      padre: f-3
0      madre: f-3
0      hija: f-3
```

Facts (MAIN)

```
f-0      (initial-fact)
f-1      (dd juan maria rosa m)
f-2      (dd juan maria luis h)
f-3      (dd jose laura pilar m)
f-4      (dd luis pilar miguel h)
f-5      (dd miguel isabel jaime h)
f-6      (dd pedro rosa pablo h)
f-7      (dd pedro rosa ana m)
f-8      (padre pedro ana)
f-9      (progenitor pedro ana)
f-10     (ascendiente pedro ana)
f-11     (madre rosa ana)
```

Load archivo

Visualizar la agenda y la MT

Materiales de referencia

- Documentación online sobre CLIPS
 - <http://clipsrules.sourceforge.net/OnlineDocs.html>
- Tutorial Universidad de Córdoba
 - <http://www.uco.es/users/sventura/misc/TutorialCLIPS/Reglas.htm>
- Otro tutorial
 - <https://www.csee.umbc.edu/portal/clips/tutorial/>