

INTELIXENCIA ARTIFICIAL

4º Grao Enxeñaría Informática

Práctica 2 (sesiones 4-5): diseño de SBC con CLIPS Curso 2022-23

Alberto J. Bugarín Diz

Departamento de Electrónica e Computación

Universidade de Santiago de Compostela

alberto.bugarin.diz@usc.es

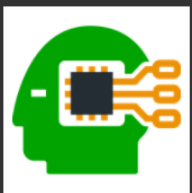
SBC con CLIPS

1. Instalación
2. Hechos y Reglas
3. Patrones y variables
4. Referencias

1. Instalación

- CLIPS: “C Language Integrated Production System”
- Sitio web oficial de CLIPS:

<https://sourceforge.net/projects/clipsrules/>

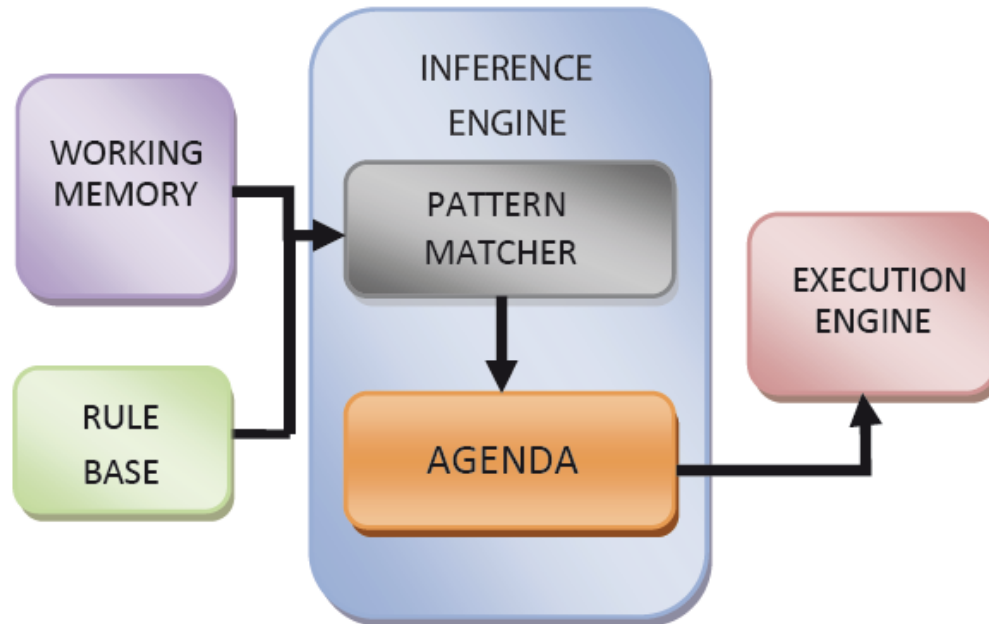


CLIPS Rule Based Programming Language

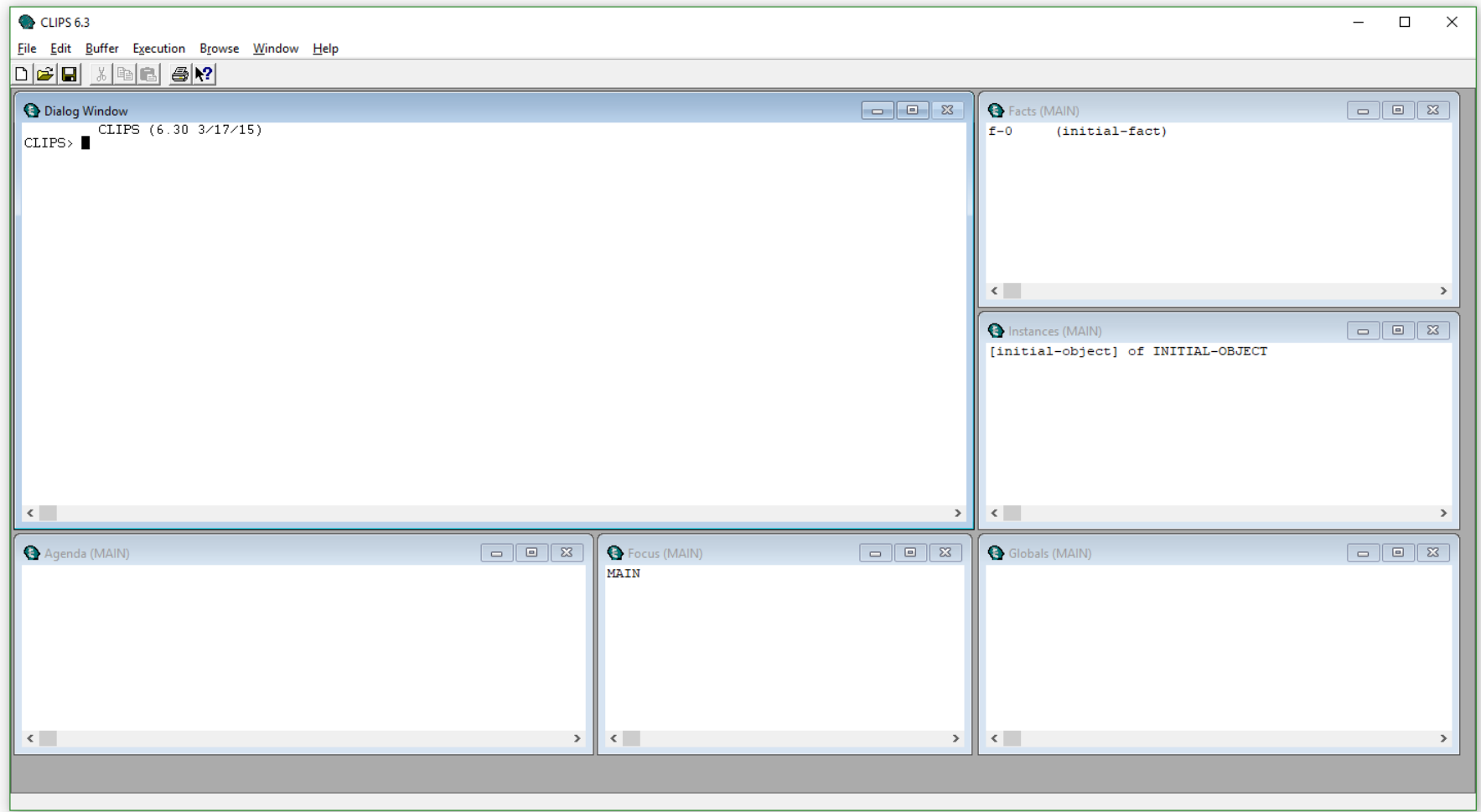
Expert System Tool

Brought to you by: [garyriley](#)

1. Instalación



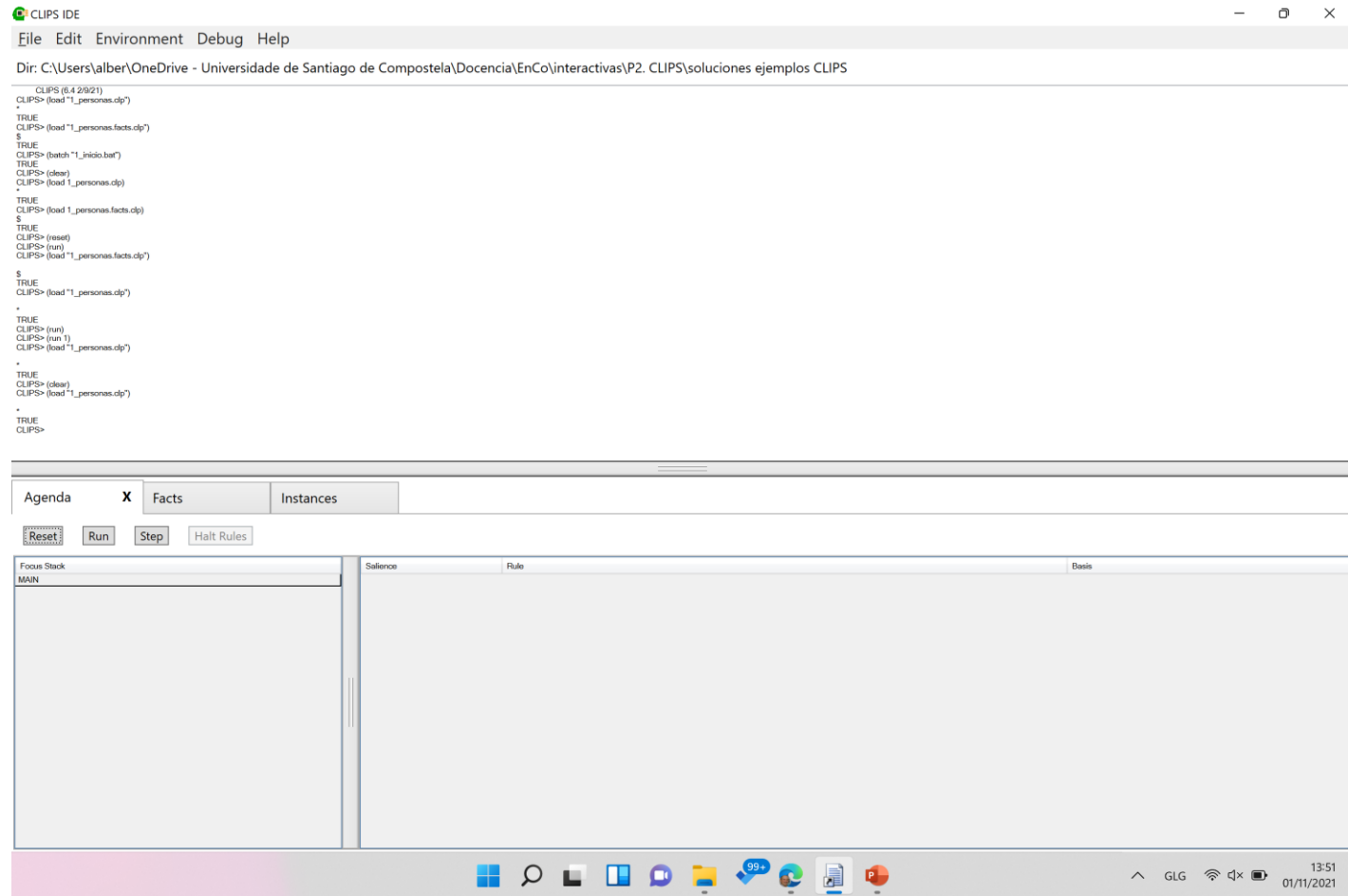
1. Instalación



- Memoria de trabajo (ventana Facts)
- “conjunto conflictivo” (ventana Agenda)

1. Instalación

- Versión 6.40



- Memoria de trabajo (ventana Facts)
- “conjunto conflicto” (ventana Agenda)

1. Instalación
2. **Hechos y Reglas**
3. Patrones y variables
4. Referencias

2. Hechos y reglas. Vectores.

- Para representar información, CLIPS nos permite utilizar:

1. Vectores ordenados de características.

(Pedro 45 V NO)

- Cada valor se refiere a un atributo predefinido: nombre, edad, género, e. civil, ...
- Importancia del **orden**

CLIPS> (assert (Pedro 45 V NO))

- CLIPS asigna un identificador de hecho, empezando desde f-0
- El hecho inicial, asertado con **reset**, es por defecto f-0
- CLIPS es sensible a mayúsculas y minúsculas

2. Hechos y reglas. Registros.

- Para representar información, CLIPS nos permite utilizar:

2. Registros.

- Agrupaciones (*templates*) con campos nombrados: **deftemplate**

```
CLIPS> (deftemplate Persona
```

```
          (field Nombre)
```

```
          (field Edad)
```

```
          (field Sexo)
```

```
          (field EstadoCivil) )
```

- Cada campo (“slot”) se refiere a un atributo predefinido

```
CLIPS> (assert (Persona
```

```
          (Nombre Juan)
```

```
          (Edad 30)
```

```
          (EstadoCivil casado)
```

```
          (Sexo V) )
```

- Si no se especifica un campo: CLIPS le asigna un valor nulo (**nil** en CLIPS)

2. Hechos y reglas. Registros.

- Para representar información, CLIPS nos permite utilizar:

2. Registros.

- Para que un campo sea un vector ordenado de características, usamos el identificador **multifield**:

```
CLIPS> (deftemplate Persona
```

```
          (multifield Nombre)
```

```
          (field Edad)
```

```
          (field Sexo)
```

```
          (field EstadoCivil) )
```

- Si un Cada campo (“slot”) se refiere a un atributo predefinido

```
CLIPS> (assert (Persona
```

```
          (Nombre Juan Carlos Cubero)
```

```
          (Edad 30)))
```

2. Hechos y reglas. Registros.

- Pueden añadirse hechos desde un fichero: **deffacts**

CLIPS> (load personas.clp)

- Para que pasen a formar parte de la memoria de trabajo ejecutamos:

CLIPS> (reset)

Borra los anteriores hechos, pero no los registros

Crea **f-0**

Crea los nuevos hechos

Tanto **load** como **reset** pueden ejecutarse desde el entorno (CTRL+L, CTRL+E)

personas.clp

; Datos de personas

```
(deftemplate Persona
  (field Nombre)
  (field Edad)
  (field Sexo)
  (field EstadoCivil)
)
```

```
(deffacts VariosHechos
  (Persona
    (Nombre JuanCarlos)
    (Edad 33))
  (Persona
    (Nombre Maria)
    (Sexo M))
)
```

```
(deffacts OtrosHechos
  (NumeroDeReactores 4)
)
```

2. Hechos y reglas

- Eliminación de hechos: (**retract** <índice de hecho>**+**)

CLIPS> (retract 1 2) Suprime los hechos con identificadores 1 y 2.

CLIPS> (retract *) Borra todos los hechos

- Modificación de hechos: `(modify <índice de hecho> (<nombre campo> <nuevo valor>))`

```
CLIPS> (modify 1 (Edad 27))
```

Cambia el identificador. Por ejemplo, pasa de **f-1** a **f-4**

- Copia de un hecho (con modificación en algunos campos):

```
CLIPS> (duplicate 1
          (Nombre "Pedro Pérez")
          (Edad 40))
```

- Eliminación completa de hechos (patrones, reglas, ...): (**clear**)

1. Instalación
2. Hechos
- 3. Reglas**
4. Patrones y variables
5. Referencias

3. Reglas

- Definición de reglas:

```
(defrule <nombre> <comentario>  
    <patrón antecedente> +  
    =>  
    <consecuente> +  
)
```
- Antecedente/LHS (**Left Hand Side**) => Consecuente/RHS (**Right Hand Side**)
- LHS:** uno o varios patrones, con las **condiciones** que han de darse sobre los elementos de la memoria de trabajo para que la **regla pueda activarse**
- RHS:**
 - cambio en la memoria de trabajo (**assert**) o (**retract**)
 - procedimiento: (**printout**)

3. Reglas

- Ejemplo:

reglas.clp

```
(deffacts VariosHechosVectores
  (Persona Pedro 45 V SI)
  (Persona Maria 34 M NO)
)
(defrule ECivilPedro_Soltero
  (Persona Pedro 45 V NO)
=>
  (printout t crlf "Pedro está soltero")
)
(defrule ECivilPedro_Casado
  (Persona Pedro 45 V SI)
=>
  (printout t crlf "Pedro está casado")
)
(defrule ECivilMaria_Soltera
  (Persona Maria 34 M NO)
=>
  (printout t crlf "Maria está soltera")
)
```

CLIPS> (clear)

CLIPS> (load datos.clp)

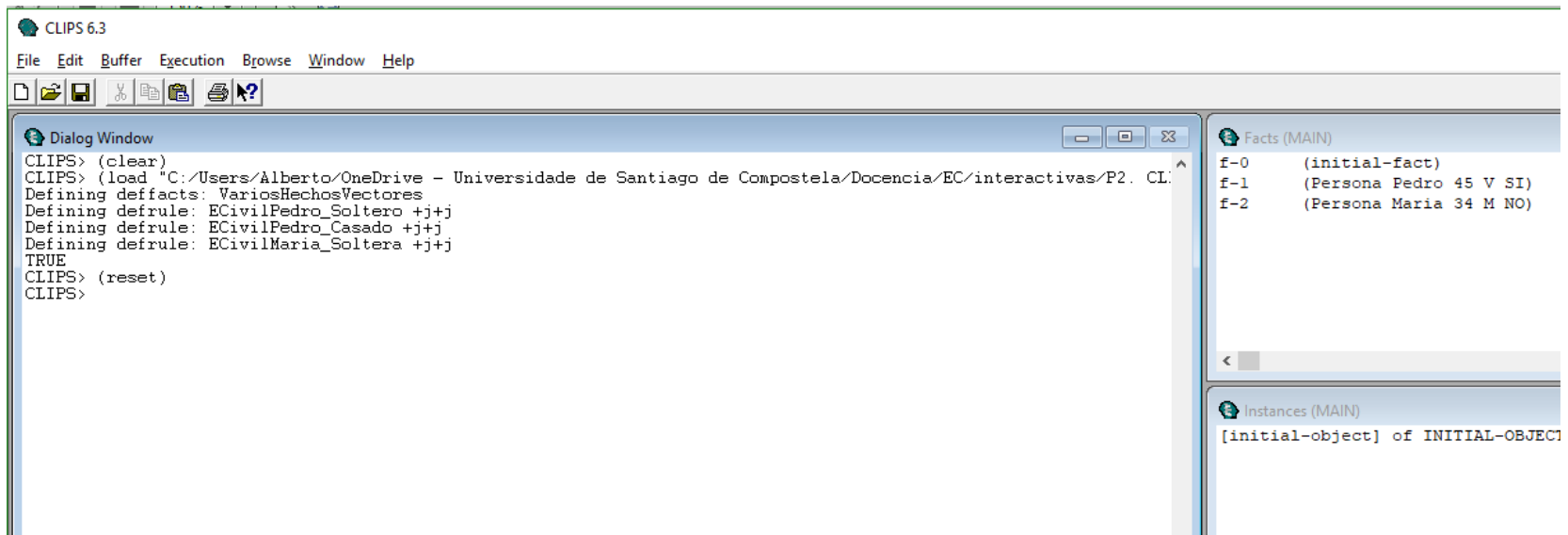
CLIPS> (reset)

t: stdout

crlf: CR + LF

3. Reglas

- Ciclo de control CLIPS



The screenshot shows the CLIPS 6.3 application window. The 'Dialog Window' on the left contains the following text:

```
CLIPS> (clear)
CLIPS> (load "C:/Users/Alberto/OneDrive - Universidade de Santiago de Compostela/Docencia/EC/interactivas/P2. CL
Defining deffacts: VariosHechosVectores
Defining defrule: ECivilPedro_Soltero +j+j
Defining defrule: ECivilPedro_Casado +j+j
Defining defrule: ECivilMaria_Soltera +j+j
TRUE
CLIPS> (reset)
CLIPS>
```

The 'Facts (MAIN)' panel on the right displays the following facts:

```
f-0      (initial-fact)
f-1      (Persona Pedro 45 V SI)
f-2      (Persona Maria 34 M NO)
```

Agenda (MAIN)

```
0      ECivilMaria_Soltera: f-2
0      ECivilPedro_Casado: f-1
```

Facts (MAIN)

```
f-0      (initial-fact)
f-1      (Persona Pedro 45 V SI)
f-2      (Persona Maria 34 M NO)
```

CLIPS> (run 1)

[CONTROL+T]

3. Reglas

CLIPS> (run 1)

[CONTROL+T]

Agenda (MAIN)

0 ECivilPedro_Casado: f-1

Facts (MAIN)

f-0 (initial-fact)
f-1 (Persona Pedro 45 V SI)
f-2 (Persona Maria 34 M NO)

CLIPS> (run 1)

Agenda (MAIN)

Facts (MAIN)

f-0 (initial-fact)
f-1 (Persona Pedro 45 V SI)
f-2 (Persona Maria 34 M NO)

Repetimos con: (run) ó CTRL+R

3. Reglas

- Cada iteración completa del ciclo de control de CLIPS:
 1. **Ejecución y refracción:** Se ejecuta la primera regla de la agenda y desaparece de la misma por el principio de refracción.
 2. **Emparejamiento:** Se emparejan los hechos de la memoria de trabajo con las reglas existentes.
 - Si en la parte derecha de la regla ejecutada se hubiese añadido algún elemento a la memoria de trabajo, algunas reglas nuevas podrían activarse y ser aplicables (→ agenda).
 - Si en la parte derecha de la regla ejecutada se hubiese suprimido algún elemento de la memoria de trabajo, algunas reglas que estaban en la agenda podrían desaparecer de esta.
 3. **Resolución de conflictos:** Según el criterio definido por CLIPS, se ordenan las reglas (activaciones) de la agenda y se selecciona una de ellas como la primera, que será la que se ejecute en el siguiente ciclo de control.

1. Instalación
2. Hechos
3. Reglas
4. **Patrones y variables**
5. Referencias

4. Patrones y Variables

- Emparejamiento en vectores ordenados de características: se pueden incluir variables en los patrones las reglas
- Sintaxis: `?<nombre de variable>`

- Ejemplo:

- Dado el hecho: `(Persona Pedro 45 V NO)`
- Y el patrón: `(Persona ?Nombre 45 V ?Casado)`
- Ambos se pueden emparejar con las sustituciones:

`?Nombre` por `Pedro`

`?Casado` por `NO`

- Para prescindir de una variable se usa la **variable anónima** `?`

```
(defrule ImprimeSolteros
```

```
  (Persona ?Nombre ? ? NO)
```

```
=>
```

```
(printout t crlf ?Nombre " está soltero"))
```

4. Patrones y Variables


- Consideraciones importantes sobre el funcionamiento:
 - Una **variable se liga** en la parte izquierda de la regla (LHS), en alguno de los **patrones** que aparezcan en su **antecedente**.
 - **Una vez ligada** a un valor (cuando se ha producido un emparejamiento con alguno de los hechos de la memoria de trabajo), **permanece** con dicho valor.
 - Aunque dos variables tengan el mismo nombre, si aparecen en **distintas reglas**, se consideran **variables diferentes** (como las variables locales en las funciones de un lenguaje de programación imperativo).

4. Patrones y Variables


- Una misma regla puede activarse con varios hechos distintos

```
(deffacts VariosHechosVectores
(Persona Pedro 45 V SI)
(Persona Juan 35 V NO)
(Persona Maria 34 M NO)
)

(defrule ImprimeSolteros
  (Persona ?Nombre ? ? NO)
=>
  (printout t crlf ?Nombre "está soltero")
)
```

 Facts (MAIN)

f-0	(initial-fact)
f-1	(Persona Pedro 45 V SI)
f-2	(Persona Juan 35 V NO)
f-3	(Persona Maria 34 M NO)

 Agenda (MAIN)

0	ImprimeSolteros: f-3
0	ImprimeSolteros: f-2

4. Patrones y Variables

- Más consideraciones:

- Las variables de la RHS tienen que estar instanciadas. De no estarlo, se produce un error

```
(defrule ReglaInvalida
  (Persona ?Nombre 45 V ?)
  =>
  (printout t crlf ?Nombre ?Casado))

Error: Undefined variable Casado referenced in RHS of defrule.
```

- Usar la variable anónima cuando no se desea instanciar una variable

```
(defrule ImprimeSolteros ; Demasiadas variables
```

```
  (Persona ?Nombre ?Edad ?Sexo NO)
```

```
=>
```

```
  (printout t crlf ?Nombre))
```

```
(defrule ImprimeSolteros ; Versión mejorada :-)
```

```
  (Persona ?Nombre ? ? NO)
```

```
=>
```

```
  (printout t crlf ?Nombre))
```

4. Patrones y Variables

- Más consideraciones:
 - No puede ponerse una variable como primer valor de un vector ordenado de características:

```
(defrule ReglaInvalida
  (?Relacion Pedro 45 V Si)
=>
  (printout t crlf ?Relacion))
```

Syntax error:
Check appropriate syntax for the first field of a pattern.

- El comodín \$?, representa cero, o más valores de un patrón


```
(defrule ImprimeNombresPersonas
  (Persona ?Nombre $?)
=>
  (printout t crlf ?Nombre))
```


4. Patrones y Variables


- El comodín `$?` Puede generar varias activaciones de la misma regla con el mismo hecho...

```
(deffacts Hechos
  (TiposEmergenciasActuales A B C)
)

(defrule ImprimeEmergencias
  (TiposEmergenciasActuales $? ?T $?)
=>
  (printout t "Emergencia -> " ?T crlf))
```

 Facts (MAIN)

f-0	(initial-fact)
f-1	(TiposEmergenciasActuales A B C)

 Agenda (MAIN)

0	ImprimeEmergencias: f-1
0	ImprimeEmergencias: f-1
0	ImprimeEmergencias: f-1

CLIPS> (run)

Emergencia -> A

Emergencia -> B

Emergencia -> C

4. Patrones y Variables

- En Reglas con varios patrones, se asume que implícitamente están conectados por **AND**: todos deben emparejarse con la BH para que la regla sea aplicable
- Si una misma variable aparece en varios patrones de la LHS...
 - ... la primera vez que aparece, simplemente se realiza una sustitución y toma un **valor concreto**
 - ... las siguientes veces que aparece, impone una **restricción** al patrón, debiendo tomar dicho valor
- Ejemplo

```
(defrule DiagnosticoEccema
  (FichaPaciente ?Nombre $?)
  (DatosExploracion ?Nombre $? picor $? vesiculas $?)
  =>
  (printout t crlf ?Nombre "tiene un eccema"))
```

4. Patrones y Variables

- En el emparejamiento con registros sólo se especifican los campos que se especifica una restricción

Fichero alarma.clp

```
(deftemplate Emergencia
  (field tipo)
  (field sector)
  (field campo)
)

(deftemplate SistemaExtincion
  (field tipo)
  (field status)
  (field UltimaRevision)
)

(defrule Emergencia-Fuego-ClaseB
  (Emergencia
    (tipo ClaseB))
  (SistemaExtincion
    (tipo DioxidoCarbono)
    (status operativo))
  =>
  |(printout t "Usar extintor CO2" crlf)
)
```

alarma.datos.clp

```
(deffacts HechosSistemaExtincion
  (SistemaExtincion
    (tipo DioxidoCarbono)
    (status operativo)
    (UltimaRevision diciembre))
)
```

```
CLIPS> (clear)
CLIPS> (load alarma.clp)
CLIPS> (load alarma.datos.clp)
CLIPS> (reset)

CLIPS> (assert (Emergencia
                  (tipo ClaseB)
                  (sector A)))
```

```
Agenda (MAIN)
0      Emergencia-Fuego-ClaseB: f-2,f-1
```

```
CLIPS> (run 1)
```

4. Patrones y Variables

- Ejemplo: diagnóstico de eccemas

```
(deftemplate FichaPaciente
  (field Nombre)
  (field Casado)
  (field Direccion))

(deftemplate DatosExploracion
  (field Nombre)
  (multifield Sintomas)
  (field GravedadAfeccion))

(defrule DiagnosticoEccema
  (DatosExploracion
    (Nombre ?N)
    (Sintomas $? picor $? vesiculas $?))
  (FichaPaciente
    (Nombre ?N))
  =>
  (printout t "Posible diagnóstico para el paciente " ?N ": Eccema " crlf)
)
```

- EJERCICIO: ¿cuáles serían los hechos necesarios para que la memoria de trabajo y la agenda de CLIPS queden como se muestra a continuación?

```
f-0      (initial-fact)
f-1      (FichaPaciente (Nombre Pedro)
f-2      (FichaPaciente (Nombre Juan)
f-3      (FichaPaciente (Nombre Maria)

f-4      (DatosExploracion (Nombre Pedro) (Sintomas
f-5      (DatosExploracion (Nombre Juan) (Sintomas :
f-6      (DatosExploracion (Nombre Maria) (Sintomas
```

```
Agenda (MAIN)
0      DiagnosticoEccema: f-5,f-2
0      DiagnosticoEccema: f-4,f-1
```

4. Patrones y Variables

- **Refracción:** cuando se ejecuta una regla, CLIPS la borra directamente de la agenda. Sólo si volvemos a añadir los hechos que la hicieron aplicable, entonces la regla volverá a entrar en la agenda, con un mayor valor del índice **f**
- Sobre el ejemplo anterior: `CLIPS> (retract 3 6)`

```
Agenda (MAIN)
0   DiagnosticoEccema: f-5,f-2
0   DiagnosticoEccema: f-4,f-1
```

`CLIPS> (retract 2)`

```
Agenda (MAIN)
0   DiagnosticoEccema: f-4,f-1
```

```
CLIPS>(assert (FichaPaciente
              (Nombre Juan)
              (Casado No)
              (Direccion "Santiago de Compostela"))
```

```
Agenda (MAIN)
0   DiagnosticoEccema: f-5,f-7
0   DiagnosticoEccema: f-4,f-1
```

4. Patrones y Variables

- Restricciones (lógicas) sobre campos:

not *~*

EJEMPLO: Mostrar las personas solteras cuyo color de pelo no sea marrón.

```
(defrule SolteroNoMarron
  (Persona
    (Nombre ?N)
    (ColorPelo ~Marron)
    (Casado No))
  =>
  (printout t ?N " no tiene pelo marrón" crlf))
```

and *&*

EJEMPLO: Mostrar las personas cuyo color de pelo no sea ni marrón ni negro.

```
(defrule PersonaNiMarronNiNegro
  (Persona
    (Nombre ?N)
    (ColorPelo ~Marron & ~Negro))
  =>
  (printout t ?N " no tiene pelo marrón ni negro" crlf))
```

or *|*

EJEMPLO: Mostrar las personas con el pelo de color marrón o negro.

```
(defrule PersonaMarronONegro
  (Persona
    (Nombre ?N)
    (ColorPelo Marron|Negro))
  =>
  (printout t ?N " tiene pelo marrón ó negro" crlf))
```

4. Patrones y Variables

- Restricciones (lógicas) sobre campos:
- & también se utiliza para asociar a una variable el valor que tenga un hecho en el campo sobre el que se está definiendo la restricción

```
(defrule PersonaMarronONegro
  (Persona
    (Nombre ?N)
    (ColorPelo ?Color & Marron|Negro))
  =>
  (printout t "El color de pelo de " ?N " es " ?Color crlf))
```

¡OJO! La variable (?) debe preceder a la restricción (&).

4. Patrones y Variables

- Los siguientes hechos, sobre las reglas anteriores...

```
Facts (MAIN)
f-0      (initial-fact)
f-1      (Persona (Nombre Alberto) (ColorPelo Marron) (Casado No))
f-2      (Persona (Nombre Maria) (ColorPelo Castaño) (Casado No))
f-3      (Persona (Nombre Ana) (ColorPelo Negro) (Casado No))
f-4      (Persona (Nombre Eloy) (ColorPelo Pelirroja) (Casado No))
```

```
Agenda (MAIN)
0      SolteroNoMarron: f-4
0      PersonaNiMarronNiNegro: f-4
0      SolteroNoMarron: f-3
0      PersonaMarronONegro: f-3
0      PersonaMarronONegro2: f-3
0      SolteroNoMarron: f-2
0      PersonaNiMarronNiNegro: f-2
0      PersonaMarronONegro: f-1
0      PersonaMarronONegro2: f-1
```

Eloy no tiene pelo marrón
Eloy no tiene pelo marrón ni negro

Ana no tiene pelo marrón
Ana tiene pelo marrón ó negro
El color de pelo de Ana es Negro

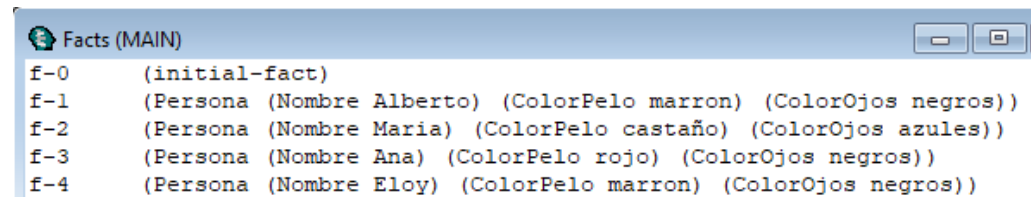
Maria no tiene pelo marrón
Maria no tiene pelo marrón ni negro

Alberto tiene pelo marrón ó negro
El color de pelo de Alberto es Marron

4. Patrones y Variables

- Ejemplo: mostrar el nombre y color de pelo de dos personas que:
 - Una de ellas tenga, o bien los ojos azules, o bien los ojos verdes, pero que no tenga el pelo negro.
 - La otra, que no tenga el mismo color de ojos que la primera y que tenga, o bien el pelo rojo, o bien el mismo color de pelo que la primera.

```
(defrule ParejaComplicada
  (Persona
    (Nombre ?N1)
    (ColorOjos ?Ojos1 & azul|verde)
    (ColorPelo ?Pelo1 & ~negro))
  (Persona
    (Nombre ?N2 & ~?N1)
    (ColorOjos ?Ojos2 & ~?Ojos1)
    (ColorPelo ?Pelo2 & rojo | ?Pelo1))
=>
  (printout t ? N1      " tiene los ojos " ?Ojos1
              " y el pelo " ?Pelo1 crlf
           ?N2      " tiene los ojos " ?Ojos2
              " y el pelo " ?Pelo2 crlf))
```



The screenshot shows a window titled "Facts (MAIN)" with a list of facts:

- f-0 (initial-fact)
- f-1 (Persona (Nombre Alberto) (ColorPelo marron) (ColorOjos negros))
- f-2 (Persona (Nombre Maria) (ColorPelo castaño) (ColorOjos azules))
- f-3 (Persona (Nombre Ana) (ColorPelo rojo) (ColorOjos negros))
- f-4 (Persona (Nombre Eloy) (ColorPelo marron) (ColorOjos negros))

```
CLIPS> (run)
Maria tiene los ojos azules y el pelo castaño
Ana tiene los ojos negros y el pelo rojo
```

5. Referencias

- Juan Carlos Cubero, Fernando Berzal (UGR) y col. Sistemas Inteligentes de Gestión. Tutorial de CLIPS. Universidad de Granada.
<http://elvex.ugr.es/decsai/intelligent/workbook/ai/CLIPS.pdf>
- María J. Taboada. “Sistemas Basados en Reglas”. En J.T. Palma, R. Marín. Inteligencia Artificial: técnicas, métodos y aplicaciones (Cap. 3). Ed. McGraw-Hill, 2008.